UNDERSTANDING COMPLEX SYSTEMS

Springer COMPLEXITY

Rolf P. Würtz
Editor

Organic Computing

Springer Complexity

Springer Complexity is an interdisciplinary program publishing the best research and academic-level teaching on both fundamental and applied aspects of complex systems – cutting across all traditional disciplines of the natural and life sciences, engineering, economics, medicine, neuroscience, social and computer science.

Complex Systems are systems that comprise many interacting parts with the ability to generate a new quality of macroscopic collective behavior the manifestations of which are the spontaneous formation of distinctive temporal, spatial or functional structures. Models of such systems can be successfully mapped onto quite diverse "real-life" situations like the climate, the coherent emission of light from lasers, chemical reaction-diffusion systems, biological cellular networks, the dynamics of stock markets and of the internet, earthquake statistics and prediction, freeway traffic, the human brain, or the formation of opinions in social systems, to name just some of the popular applications.

Although their scope and methodologies overlap somewhat, one can distinguish the following main concepts and tools: self-organization, nonlinear dynamics, synergetics, turbulence, dynamical systems, catastrophes, instabilities, stochastic processes, chaos, graphs and networks, cellular automata, adaptive systems, genetic algorithms and computational intelligence.

The two major book publication platforms of the Springer Complexity program are the monograph series "Understanding Complex Systems" focusing on the various applications of complexity, and the "Springer Series in Synergetics", which is devoted to the quantitative theoretical and methodological foundations. In addition to the books in these two core series, the program also incorporates individual titles ranging from textbooks to major reference works.

Understanding Complex Systems

Founding Editor: J.A. Scott Kelso

Future scientific and technological developments in many fields will necessarily depend upon coming to grips with complex systems. Such systems are complex in both their composition – typically many different kinds of components interacting simultaneously and nonlinearly with each other and their environments on multiple levels – and in the rich diversity of behavior of which they are capable.

The Springer Series in Understanding Complex Systems series (UCS) promotes new strategies and paradigms for understanding and realizing applications of complex systems research in a wide variety of fields and endeavors. UCS is explicitly transdisciplinary. It has three main goals: First, to elaborate the concepts, methods and tools of complex systems at all levels of description and in all scientific fields, especially newly emerging areas within the life, social, behavioral, economic, neuro- and cognitive sciences (and derivatives thereof); second, to encourage novel applications of these ideas in various fields of engineering and computation such as robotics, nano-technology and informatics; third, to provide a single forum within which commonalities and differences in the workings of complex systems may be discerned, hence leading to deeper insight and understanding.

UCS will publish monographs, lecture notes and selected edited contributions aimed at communicating new findings to a large multidisciplinary audience.

Rolf P. Würtz (Ed.)

Organic Computing

With 93 Figures and 1 Table

Springer

Editor

Dr. Rolf P. Würtz
Institut für Neuroinformatik
Ruhr-Universität Bochum
Room: ND03-32
D-44780 Bochum
Germany
Rolf.Wuertz@neuroinformatik.rub.de

ISBN: 978-3-540-77656-7 e-ISBN: 978-3-540-77657-4

Understanding Complex Systems ISSN: 1860-0832

Library of Congress Control Number: 2008922291

Cover design: WMXDesign GmbH

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Preface

In a nutshell, Organic Computing is a research field emerging around the conviction that problems of organization in complex systems in computer science, telecommunications, neurobiology, molecular biology, ethology, and possibly even sociology can be tackled scientifically in a unified way, by means of which progress in understanding aspects of organization in either field can be fruitful in the others. From the computer science point of view, the apparent ease with which living systems solve computationally difficult problems makes it inevitable to adopt strategies observed in nature for creating information processing machinery.

As an idea whose time simply has come, Organic Computing is growing from multiple roots. In November 2001, a Symposium "Organic Computing – Towards Structured Design of Processes" was held at the Heinz Nixdorf Museum in Paderborn, Germany, bringing together computer scientists and biologists to pursue the idea. Independently, the Technical Informatics Branch of the German Computer Science Society (GI) developed the concept in a series of workshops in 2002. The scope was broadened by the Organic Computing Initiative of GI at a workshop in Hannover in 2003, which outlined the scope of today's Organic Computing research. As a third root on the industrial side, Forrester Research presented a study in 2002, which proposed Organic IT as a strategy for information systems infrastructure.

In the meantime, Organic Computing is a powerful driving force for a whole spectrum of research. Most visibly in terms of academic funding, in fall 2004, the DFG issued a call for proposals for a priority program on Organic Computing, which started with 18 projects in August 2005 and is currently in its second phase. In January 2006, there was a first Dagstuhl seminar, which also attracted participants from overseas, a second one is scheduled for the spring of 2008.

In this book the major ideas behind Organic Computing are delineated, together with a sparse sample of computational projects undertaken in this new field. Many more can be found at the homepage of the Deutsche Forschungsgemeinschaft (DFG) priority research program 1183 "Organic Computing" at

http://www.organic-computing.de/spp and the rapidly growing literature list at http://kbs.cs.tu-berlin.de/∼parzy/oclc/.

To set the stage for the chapters to come I give an incomplete list of biological metaphors used in Organic Computing and show for each chapter which ones are applied and what applications are tackled. These metaphors include *evolution*, *neural networks*, *gene-regulatory networks* , *networks of brain modules*, *hormone system*, *insect swarms*, and *ant colonies*.

Chapter 1 is an introduction to goals and ideas, chapters 2 through 5 lay the theoretical foundations of Organic Computing. Chapter 6 describes the importance of the evolutionary metaphor together with modern developments in evolutionary optimization. Chapter 7 combines evolutionary approaches with neural network learning. Chapters 8 and 9 build on ontogenesis for system construction, with cross-references to neural networks. Chapters 10 and 11 use metaphors of insect swarms for applications in networking. Chapters 12 and 13 use procedures gleaned from the workings of the hormone system for networking applications. Finally, chapters 14 and 15 look at neural networks and interaction of modules in the mammalian brain for applications in computer vision.

Thanks go to the DFG for research funding in the priority program, the IEEE task force on Organic Computing for moral support, and to the Volkswagenstiftung for funding the website http://organic-computing.org.

I would like to thank all authors for their excellent contributions. Special thanks go to those who submitted early for their patience and those with very heavy schedules for finally submitting the missing articles.

Last not least, I thank my colleagues at the Institute for Neurocomputing and my wife and son for tolerating my negligence of other matters during the work on this book.

Bochum, November 2007 *Rolf Würtz*

Contents

List of Contributors

Wolfgang Banzhaf
Computer Science
Memorial University of Newfoundland
St. John's, NL, A1B 3X5, Canada
banzhaf@cs.mun.ca

Kirstie L. Bellman
Aerospace Integration Science Center
The Aerospace Corporation
Mail Stop M1/025, P.O. Box 92957
Los Angeles, California 90009-2957, USA
bellman@aero.org

Jürgen Branke
Institute AIFB
University of Karlsruhe
76128 Karlsruhe, Germany
branke@aifb.uni-karlsruhe.de

Uwe Brinkschulte
University of Karlsruhe, Bldg. 40.28
Engler-Bunte-Ring 8
76131 Karlsruhe, Germany
brinks@ira.uka.de

Ioan Buciu
Electronics Department
Faculty of Electrical Engineering and Information Technology
University of Oradea
Universitatii 1
410087 Oradea , Romania
ibuciu@uoradea.ro
Department of Informatics
Aristotle University
Box 451
GR-541 24, Thessaloniki, Greece

René Doursat
Institut des Systèmes Complexes (ISC), and Centre de Recherche en Epistémologie Appliquée (CREA)
CNRS and Ecole Polytechnique
57–59, rue Lhomond
75005 Paris, France
doursat@shs.polytechnique.fr

Falko Dressler
Dept. of Computer Science 7
University of Erlangen
Martensstr. 3
91058 Erlangen, Germany
dressler@informatik.uni-erlangen.de

Simon Harding
Computer Science
Memorial University of Newfoundland
St. John's, NL, A1B 3X5, Canada
simonh@cs.mun.ca

Christian Haubelt
University of Erlangen-Nuremberg
Am Weichselgarten 3
91058 Erlangen, Germany
haubelt@cs.fau.de

Christian Igel
Institut für Neuroinformatik
Ruhr-Universität
Universitätsstraße 150
44801 Bochum, Germany
christian.igel@
neuroinformatik.rub.de

Dirk Koch
University of Erlangen-Nuremberg
Am Weichselgarten 3
91058 Erlangen, Germany
dirk.koch@cs.fau.de

Maximilian Krüger
Institut für Neuroinformatik
Ruhr-Universität
Universitätsstraße 150
44801 Bochum, Germany
maximilian.krueger@
neuroinformatik.rub.de

Christopher Landauer
Aerospace Integration Science Center
The Aerospace Corporation
Mail Stop M8/080, P.O. Box 92957
Los Angeles, California 90009-2957,
USA
cal@aero.org

Klaus Mainzer
Institute of Interdisciplinary
Informatics
University of Augsburg
86135 Augsburg, Germany
klaus.mainzer@phil.uni-augsburg.de

Christoph von der Malsburg
Frankfurt Institute for Advanced
Studies
Max-von-Laue-Str. 1
60438 Frankfurt a. M., Germany
malsburg@fias.uni-frankfurt.de
Computer Science Dept., University
of Southern California
University Park
Los Angeles 90089-2520, USA

Daniel Merkle
Department of Computer Science
University of Leipzig
Johannisgasse 26
04103 Leipzig, Germany
merkle@informatik.uni-leipzig.de

Martin Middendorf
Department of Computer Science
University of Leipzig
Johannisgasse 26
04103 Leipzig, Germany
middendorf@informatik.uni-leipzig.de

Christian Müller-Schloer
Institute of Systems Engineering
University of Hannover
Appelstraße 4
30167 Hannover, Germany
cms@sra.uni-hannover.de

Phyllis R. Nelson
Electrical and Computer Engineering
Department
California State Polytechnic
University Pomona,
3801 W. Temple Ave.
Pomona, California, 91768, USA
prnelson@csupomona.edu

Mathias Pacher
University of Karlsruhe, Bldg. 40.28
Engler-Bunte-Ring 8
76131 Karlsruhe, Germany
pacher@ira.uka.de

Ioannis Pitas
Department of Informatics
Aristotle University
Box 451
GR-541 24, Thessaloniki, Greece
pitas@zeus.csd.auth.gr

Alexander von Renteln
University of Karlsruhe, Bldg. 40.28
Engler-Bunte-Ring 8
76131 Karlsruhe, Germany
renteln@ira.uka.de

Alexander Scheidler
Department of Computer Science
University of Leipzig
Johannisgasse 26
04103 Leipzig, Germany
scheidler@informatik.uni-leipzig.de

Hartmut Schmeck
Institute AIFB
University of Karlsruhe
76128 Karlsruhe, Germany
schmeck@aifb.uni-karlsruhe.de

Bernhard Sendhoff
Honda Research Institute Europe
GmbH
Carl-Legien-Str. 30
63073 Offenbach, Germany
bernhard.sendhoff@honda-ri.de

Bernhard Sick
Institute of Computer Architectures
University of Passau
Innstraße 33
94032 Passau, Germany
bernhard.sick@uni-passau.de

Thilo Streichert
University of Erlangen-Nuremberg
Am Weichselgarten 3
91058 Erlangen, Germany
streichert@cs.fau.de

Jürgen Teich
University of Erlangen-Nuremberg
Am Weichselgarten 3
91058 Erlangen, Germany
teich@cs.fau.de

Rolf P. Würtz
Institut für Neuroinformatik
Ruhr-Universität
Universitätsstraße 150
D-44801 Bochum, Germany
rolf.wuertz@organic-computing.org
rolf.wuertz@neuroinformatik.rub.de

1

Introduction: Organic Computing

Rolf P. Würtz

Institut für Neuroinformatik, Ruhr-Universität, 44780 Bochum, Germany.
rolf.wuertz@organic-computing.org, rolf.wuertz@neuroinformatik.rub.de

Complexity is observed everywhere. When we look around us, we see many systems that are mind-bogglingly complex. But as human beings we profoundly dislike complexity. We thrive in environments whose important traits are widely predictable, and we go to great lengths to prepare our environments precisely to that end. The most important way to do so is technologically and, despite occasional disappointments, this has been very successful. The complexity of Nature can be borne with humility as long as the preparation against the pitfalls of the natural environment can be carried out to relative satisfaction.

To make matters worse, the late twentieth century has seen the rise of difficult-to-handle complexity in *artifacts*. This means that the struggle against complexity has to cope with a new front brought about precisely by the methods devised to fight it in the first place. This kind of undesired artificial complexity is a real insult to engineers and technocrats and, at second glance, a true challenge.

In modern artifacts the highest complexity is in an embedded computer and the accompanying software. For practical purposes, the complexity such systems can attain is unlimited. I am typing this on a universal machine, which can do all sorts of information processing tasks a machine can possibly do (notwithstanding its finiteness, which infrequently causes trouble that can somehow be helped with hardware extension). This is good because it has been relatively expensive and I will not need a different one for a slightly different purpose. On the other hand, it means that my machine is capable of a huge universe of possible behaviors. The overwhelming majority of them I will never be willing to experience.

In the terms defined by von Foerster (e.g., [13]), we have come to the point where we are capable of building truly nontrivial machines. He defined a trivial machine as one showing a simple predictable response as opposed to a nontrivial machine with internal state, which also changes in response to conditions in the environment. My desktop computer periodically talks to other machines somewhere in the world to change its behavior according to

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_1,

the latest threats of infectious code floating around on the communication lines.

It is only a couple of years ago that rebooting would have put this machine into a reproducible and desirable state. Reverting to a defined previous state is still possible — but completely useless if the machine is supposed to be connected to the Internet, in which case the (formerly) desirable state will be corrupted to unusable or worse within hours if not minutes. Thus, all networked computers in the world are now bound to the wheel of progress in order to remain as useful as they used to be, and much more so if new functionality is required. And we have already passed the line, where the same becomes true for conceptually much simpler things such as telephones.

In this situation, our artifacts become conspicuously comparable to living beings. They are complex in themselves and have to cope with a complex, unpredictable, and in substantial parts malicious environment. Their interactivity may potentially even require computing models beyond the Turing machine. The classical computing models proceed from fixed input data to a fixed output. Although interrupts of any kind are causing their share of trouble in everyday computer use they do not seem to be properly reflected in theoretical computer science. One of several approaches to better account for this is given in [14].

Robotics and artificial intelligence are other domains of engineering where the interaction with the environment is infamously difficult. And indeed, for a household robot to be of any use it must be highly complex in order to be able to deal with commonplace difficulties like cluttered floors. But their user interface may not be very complex. It should be enough to advise a robot to "clean up that mess over there" and leave it to the machine to figure out the details. So the task for an engineer is to build highly nontrivial machines and trivialize their interfaces as best as possible.

The hope that this is possible rests on the observation that such systems exist around us. Our fellow humans, for example, can be advised very simply to do tasks like the one above, and the possible difficulties encountered can be overcome in various ways. It is therefore worthwhile to learn from the natural sciences, and especially biology, in order to build such machines.

In nature, things usually get the more complex the closer we look. In the history of physics, the closeness of look went from rigid bodies over indivisible atoms to a whole zoo of elementary particles. Still, there is hope that going to still higher energies, or looking even closer, will end the complexity and simplicity will prevail at the end of the endeavor. In biology, complexity is observed on all levels. The behavior of plants and simple animals is relatively predictable most of the time. By looking closer it is revealed that myriads of processes interact in very complex manners to keep the whole system stable and predictable. A rough but workable theory of how a bacterium will react qualitatively in many situations is relatively simple; a good theory of how the details of its cell work to produce this behavior is a very distant dream.

At this point, some clarification of the term "Organic Computing" is useful. The adjective "organic" has several meanings, Collin's dictionary lists eight of them. All somehow refer to living organisms, but the one relevant here is the most abstract one: "of or characterized by the coordination of integral parts; organized". As such the word is used in the collocation of *organic unity*, according to Encyclopædia Britannica "in literature, a structural principle, first discussed by Plato (in *Phaedrus*, *Gorgias*, and *The Republic*) and later described and defined by Aristotle. The principle calls for internally consistent thematic and dramatic development, analogous to biological growth, which is the recurrent, guiding metaphor throughout Aristotle's writings". Another use, also from literature theory is *organic form*, "the structure of a work that has grown naturally from the author's subject and materials as opposed to that of a work shaped by and conforming to artificial rules". These are directly relevant to Organic Computing, because of the demand that *artifacts* should be structured according to a biological paradigm and in their final form resemble the high degree of integration observed in living beings.

Within biology, the closest term to Organic Computing is *organicism*. Again, according to Encyclopædia Britannica, *organicism* is a branch of natural philosophy that "in biology, the theory that life and living processes are the expression of an activity that is possible only by virtue of of the living system rather than because of its individual components. As such, it is directly opposed to vitalism and mechanism". The latter distinction is a subtle one. On the one side, no special vital force is required for explanation, on the other, while reduction to physical laws may in principle be possible, higher levels of organization obey their own laws.

A central assumption of Organic Computing is that it is scientifically fruitful to interpret complex systems of interacting processes as computational systems and to study them as such. Consequently, they can in principle be reduced to or simulated by Boolean operations, but the study of their organizational structure should be carried out on a higher level. The second assumption, which is the computer science point of view, states that it is technically useful to apply the lessons learned from the study of natural systems to build computational systems with desired properties — complex in their inner structure, but relatively straightforward to interact with.

It is very clear that no magic is to be expected from Organic Computing in solving computational problems. Rather, the principles that guide the organic functioning of organisms must be known well enough to be applicable to the design of computing systems. At the current state of the art, this knowledge is rudimentary. This should not be seen as prohibitive, because it opens the possibility that lessons learned in the construction of well-organized machines have direct impact on biological theories.

From the incompleteness of biological theory it follows that computational principles derived from biology can not simply simulate life "as it is". Therefore, simplified metaphors are generally used. The most successful principle known in biology, in terms of the relation of explanatory power and com-

plexity of the principle itself, is of course *Darwinian evolution*. The most fascinating example of self-organization is the development of organisms from single cells, *ontogenesis*. In the life of higher organisms with a central nervous system, much of their information processing capabilities is acquired or refined by *learning*. These examples emphasize that self-organization happens concurrently on very different time scales.

Perhaps the most striking feature of these processes is that the systems show organized behavior by *themselves*, without any obvious planning or external control. Central to Organic Computing research are therefore phenomena of *self-organization*, accompanied by a set of effects known under the name *self-x properties*. The index of this book gives an impression of the formidable range this "x" can acquire. The focus on these properties is shared with *autonomic computing* [10, 6], which uses the autonomic nervous system as a metaphor and applies similar techniques and ideas to the organization of information systems.

For self-organization to be possible, the systems must contain ways of assessing themselves and modify their behavior or parameters, according to *metrics*, which measure the desirability or utility of a certain state. On a global level, such metrics can only be set by humans, because only those can set the goals for the machine and constrain its behavior. But the effect of human planning is very limited in complex systems, and therefore, performance metrics have to be applied on much lower levels, where they are not at all obvious. In the long run, these metrics must be also subject to evolution and learning. The hope here is to be able to find rather general meta-algorithms that can pick up even these evaluation metrics from the environment in a useful way.

There is currently no satisfactory theory for this sort of system behavior. The existing theory of self-organization, the study of certain types of nonlinear dynamical systems is comparable to the above described computational model in that it explains the development of static ordered structures from initial conditions. Such a dynamical system is comparable to a computing algorithm, which derives desired results from initial states. Like with computational models, this is too narrow to describe the self-organization of interacting *processes*. Here is a wide field for future research. The importance of creating and understanding networks of processes rather than static structures is stressed by Bellman et al in section 3.2.1.

The construction of nontrivial machines with trivial interfaces encounters more of these pairs of competing if not contradictory requirements. Machines must show as much *autonomy* as possible while not losing their *controllability* in important aspects. The same goes for *robustness* vs. *sensitivity*. Many dimensions of environmental parameters should be ignored, the systems should be robust against their change, or, in neural networks terms, *generalize* over them. It is still very difficult to design the dimensions of robustness or generalization into systems at will. Much more research and good theory is required to achieve this.

As the goals stated here are very general it is no surprise that Organic Computing is not monolithic and clearly separated from other fields but has significant overlap with many of them. To conclude, I shall list the most important ones with some pointers to the literature. By its goals as well as the systemic and organizational thinking, it is clear that Organic Computing is deeply rooted in *cybernetics* [15], and cybernetic terms and ways of reasoning actually permeate this book without being mentioned explicitly. On a more general level, ideas from *general system theory* [12] are used and lessons for general systems are hoped to be learned from studies in Organic Computing.

Biological metaphors like *artificial neural networks* [7, 3] and *evolutionary algorithms* [4, 11] are used as standard techniques and sources of inspiration and are also further developed in Organic Computing research. Within biology, there is the rapidly growing field of *systems biology* [9], which partly deals with similar problems as Organic Computing in studying the natural systems themselves. On the boundary between understanding natural systems and creating artficial ones is *theoretical neuroscience* [1], which sheds light on the adaptive and perceptual capabilities desired for artificial systems.

Computational intelligence [5] encompasses many of the heuristic techniques used in Organic Computing. The robotics-oriented branches of Organic computing overlap with *cognitive systems* and *behavior-based robotics* [2]. The idea of decentralized control ist most prominent in *multi-agent systems* [8].

Organic Computing builds on results from all of these disciplines and tries to add to the understanding of complex systems and their organization. The following chapters provide examples for this.

References

1. M. A. Arbib, editor. *The Handbook of Brain Theory and Neural Networks.* MIT Press, Cambridge, MA, 2nd edition, 2003.
2. R. C. Arkin. *Behavior-based Robotics.* MIT Press, Cambridge, MA, 1998.
3. C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.
4. J. Branke. *Evolutionary Optimization in Dynamic Environments.* Springer, 2002.
5. A. P. Engelbrecht. *Computational Intelligence. An Introduction.* Wiley & Sons, 2nd edition, 2007.
6. A. Ganek and T. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1):5–18, 2003.
7. S. Haykin. *Neural Networks – A Comprehensive Foundation.* Prentice-Hall, 1998.
8. N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.
9. H. Kitano, editor. *Foundations of Systems Biology.* MIT Press, 2001.
10. R. Murch. *Autonomic Computing.* Prentice-Hall, 2006.
11. H.-P. Schwefel. *Evolution and Optimum Seeking.* Wiley & Sons, 1995.
12. L. von Bertalanffy. *General System Theory.* George Braziller Inc., 1976.

13. H. von Foerster. Through the eyes of the other. In F. Steier, editor, *Research and Reflexivity*, pages 350–363. Sage Publications, London, 1991.
14. J. Wiedermann. Autopoietic automata: Complexity issues in offspring-producing evolving processes. *Theoretical Computer Science*, 383(2–3):260–269, 2007.
15. N. Wiener. *Cybernetics*. MIT Press, 1965.

2

The Organic Future of Information Technology

Christoph von der Malsburg[1,2]

[1] Frankfurt Institute for Advanced Studies, Max-von-Laue-Str. 1, 60438 Frankfurt a. M., Germany.
malsburg@fias.uni-frankfurt.de

[2] Computer Science Dept., University of Southern California, University Park, Los Angeles 90089-2520, USA.

Summary. Classically, programs are written with specific applications in mind. Organic computing will be based on a general architecture, which apart from libraries of standard algorithms will consist of generic mechanisms of organization. Users can then create specific applications by defining goal hierarchies, by instruction and the pointing out of examples. Systems will respond to these influences by adapting control parameters so as to direct the ontogenetic process of self-organization and by organizing sample material.

2.1 Introduction

We are all expecting great things to happen in information technology. The main theme may be the integration of information pools, the very essence of organization. When I drive my car through the countryside I expect my navigation system not only to lead me to my goal fast, given the current traffic pattern – updated minute by minute in the light of the movements of all the other cars with navigation systems –, but also to help me define my goal by providing information on food and gas and events, with opening times, prices, menus etc. We want our cars to become autonomous organisms, actively diagnosing and regulating themselves and adapting to traffic situations and to our personal needs. Augmented reality will glue important annotations to the things we see through our windshields or spectacles, telepresence will let us share in a meeting with others, eye contact and all, over thousands of miles. We expect our information technology to organize networking on a large scale, making, for instance, our digital identity portable, so that wherever we touch a keyboard and look onto a screen we are recognized and have immediate access to out digital belongings, down to customized key definitions. We would like our systems to be secure, conforming tightly to legal rights of access and monetary obligation. We would like our systems to be situation-aware, recognizing and modeling our needs and intentions, just as, or better than, the clerk at the check-in counter at the airport. We would like machines to be

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_2,

able to see and hear and to understand natural language. In short: we want our information technology to become intelligent if not conscious.

Originally, the word computing referred to nothing but numerical calculation. We now apply it in a much broader sense, circumscribed perhaps as data organization. Although for certain applications we have a direct interest in the algorithms to be executed, in most cases we care only for the final outcome and not for the underlying processes. Although algorithmic computing in the narrow sense will continue to play an important role, my discussion here is concerned with the broader field of data organization.

2.2 Computing power

Moore's law has for decades doubled the complexity of computing chips every 18 months, giving us very powerful computers on our desks or laps or palms indeed. This is to be multiplied with the number of computing chips being installed (19 out of 20 of which are actually embedded and invisible), resulting in humongous computing power available worldwide. Pushing forward VLSI technology to ever smaller dimensions has been expensive, but even more expensive was and is management of the growing complexity of processor chips. All their parts must work for the whole to work, creating a terrible yield problem, and, worse, making design and testing a nightmare. As a result, Moore's law may now be coming to a halt for economic reasons, and we may be entering a new era where chip complexity is no longer being pushed. As the price of high-end computing chips was determined mainly by their development cost, this can now be written off by mass production on a new scale, making computing chips dirt-cheap. That could finally lead to what has been predicted numerous times before (and has been prevented so far by the "killer micros" — single processor speed as the cheapest means to get faster): massively parallel systems, composed of thousands or even scores of thousands of cheap processors communicating with each other.

Today such systems already exist, but they are rare for two reasons. First, the hardware required to link that many chips is expensive, and, second, usage of these systems is restricted to problems of specific, explicitly data-parallel structure. In contrast, all of the above application domains require the integration of many heterogeneous and intensely communicating subprocesses. If these problems can be overcome, another "Moore era" may ensue, with systems combining large numbers of processors of limited complexity on homogeneous, cheap to produce physical platforms, the equivalent of wafer-scale integration, and based on a programming technology that manages to be data-parallel on its lowest level in spite of the heterogeneity of processes on a higher level. Nano-scale or molecular computing may then become a reality [7], leading to personal computers with the processing power of human brains.

2.3 Necessity of a new style of computing

To realize broad application of systems composed of large numbers of limited-complexity processing elements a new style of programming will be necessary, able to implement heterogeneous domains of data and processes in massively parallel systems, able to sustain faults in the system, and, above all, able to deal with complexity beyond the imagination of systems designers.

Random faults may be an inevitable consequence of pushing electronic technology down to molecular dimensions (although error-correction techniques may be able to shield us from that problem), but, more importantly, none of the assumptions made at system design time about an application domain and its data structures may be reliably met at execution time. The combinatorics of violated assumptions create complexity that grows exponentially with system size (and what is to be called a system has to span all the subsystems to be integrated with each other!), forcing the system designer to give up explicit consideration of modes of fault and to handle the problem in a generic way. The way to go may be to give up deterministic control altogether and formulate systems as probabilistic processes, such as modeled in belief propagation networks, for instance.

The classical computing model rests entirely on the insight of the programmer into the specific application of the program written. The programming paradigm of the future will be characterized by a total loss of such insight. The same way that programming the nodes of a communication system doesn't need any insight into the contents of the data streams to be handled, the future programmer will have to handle the organization of computation on an abstract level, without any detailed insight into the specific subject matter being processed.

2.4 The complexity barrier of computing

The computing power worldwide that is installed now or will be soon is arousing expectations as to what to do with it, creating tremendous market pull for complex software. Historically, the number of command lines in any large software venture, such as space programs, telephone exchanges, enterprise software, search engines etc., has been growing exponentially. New software projects often set themselves tasks that evidently are too complex to manage, leading to project failures, such as the American FAA Air Control project, the Denver Airport luggage handling system, the US's IRS or German Federal Tax software projects, which all failed without any tangible result. It is easy (and probably correct) to blame these failures on human management insufficiencies. But that only hides the fact that our computing paradigm is no longer adequate in view of the demands we put on it. The pool of available relevant human talent is already stretched to its limit.

The applications spoken of in the introduction may require for their realization an increase in the complexity of software by an order of magnitude. It is unimaginable that the existing workforce in the system development sector will be able to handle this complexity with present methods, or even given the pace at which these methods are currently evolving. Even if the growth in software productivity should have been 20% per year in the past — the most optimistic view I ever heard of — this would not be sufficient to handle that complexity increase, resulting only in a factor of 6 in 10 years, opening the scissor between supply and demand wider and wider. What we need is a new programming paradigm that leads to a quantum leap in productivity.

2.5 The classical programming paradigm

To understand the issue, we need to take a look at the classical paradigm of programming. It is based on detailed algorithmic control. This rests on a division of labor between human and machine, see figure 2.1. The machine is deterministic and blindingly fast, but is considered as totally clueless. Only the human programmer is in possession of all creative infrastructure, in the form of goals, methods, interpretation, world knowledge and diagnostic ability. In order to control the process in the machine, the human programmer needs detailed communication, the ability to look into the machine process, sometimes down to the switching of single bits. Modern computing systems have very ingenious means to make this detailed communication possible, involving, for instance, symbolic debuggers that permit the examination of individual processing steps in relation to the high-level language structures that gave rise to them. This requirement of detailed communication between domains so vastly different as the human mind and the digital process in the machine — different in speed by orders of magnitude, for instance — comes at a tremendous price and is a millstone around the neck for the computing process in the machine.

Also, detailed communication is made more and more difficult with growing system complexity. An illustrative case in point concerns heterogeneous parallel programming. It is notoriously difficult to know the actual execution times of programming steps. This isn't a problem when there is only one processing thread, but it is very much so when many different heterogeneous threads need to exchange data. One bad consequence of that is that processes start waiting around for data, and the potential efficiency of a parallel system is wasted. The solution for this must be to optimize the placement of processes in the network of communicating processors, presumably in a situation-dependent way. Software can still be designed for programmers to keep track of this relocation, in order to keep up detailed communication, but this will make the system even more complex and difficult to work with. The only natural way to solve this difficulty is to let the system autonomously

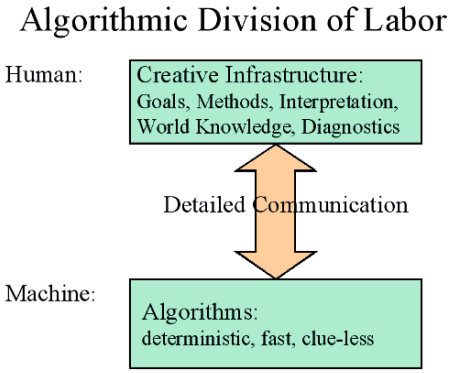


Fig. 2.1. Classic algorithmic computing entails a division of labor between man and machine. Creative infrastructure resides mainly in the human domain (upper box), the machine (lower box) blindly following commands. The two domains are coupled by detailed communication so that the programmer can inspect, understand and control the process in the machine in detail.

organize its internal structure, give up detailed communication and accept loss of insight.

In order to do that, we will have to endow systems with their own creative infrastructure enabling them to autonomously organize themselves, effectively creating their own programs. Our present style of programming sits on one side of a potential mound, the realm of algorithms; we need to get to the other side, where our systems become electronic organisms, see figure 2.2. What we have to achieve is the automation of automation. The millionfold execution of a few typed commands constitutes automation; when, however, the typing of commands itself becomes an excessive burden we need to automate even that.

Classically, the computer is programmed inside-out: we type imperative commands and then test what global, externally observable, behavior results. Anyone who has ever programmed knows that this process is fraught with surprises, and it often takes many iterations of debugging before the desired global behavior is achieved. We need to invert the process (as do declarative languages on a small scale) and limit ourselves to specifying the global behavior of the system, letting the system itself figure out how to achieve it — a process akin to education, which relies on example and encouragement instead of attempting to tamper with detailed brain mechanisms. Let the machine do the iterative debugging and automatically run the test cycles that it takes to align system details behind the set goals, see figure 2.3. The only component

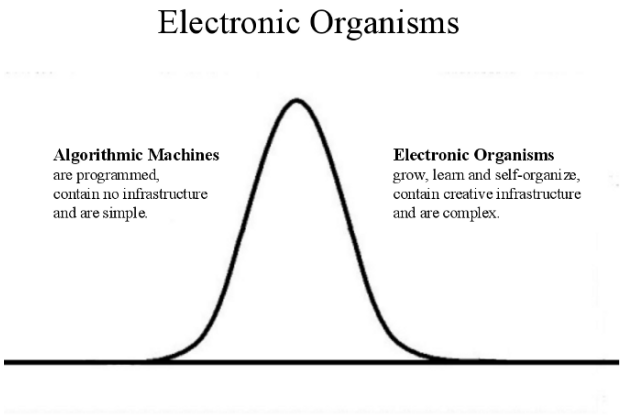


Fig. 2.2. To reach the realm of organic computing, a potential mound has to be crossed. Classically, software *has to be simple* in order to be intelligible by the human programmer, and it *may be simple*, containing little or no creative infrastructure. Electronic organisms contain much creative infrastructure and consequently *have to be complex*, but they *may well be complex*, autonomously regulating their inner structure without reliance on detailed communication with human programmers.

of the creative infrastructure that we humans want to hold onto (except in genuinely algorithmic applications) is setting the goals for our systems see figure 2.3.

Setting goals, devising contradiction-free task descriptions, is itself not a simple matter. It is common advice that any software project should start with an intensive goal definition phase, complete with (computer-simulated!) testing of all imaginable specific situations and weeding-out of design flaws on that abstract level, before even writing the first line of target code. Many large projects stumble apparently because this stage is not paid sufficient attention to. The pool of human intelligence involved with computing today will not become unemployed if code generation is automated, all brains being required to design clear abstract task descriptions. That workforce will become only that much more creative and potent.

Before going on I should admit here that the picture I am painting is all black-and-white. In reality, five decades of development in computer science have put the equivalent of a lot of creative infrastructure into the computer, see the section on architecture below. For many purposes we are already able to "program" on a high, abstract level. However, the systems permitting this had themselves to be programmed and debugged with the help of detailed communication and, above all, with the help of detailed planning of the kinds of tasks that the user will later be allowed to invoke. This style is extremely expensive, and permits only variations on a theme defined at system pro-

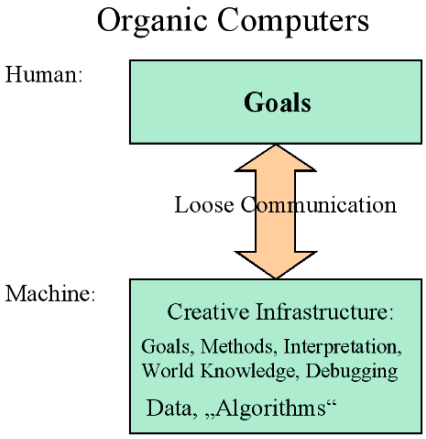


Fig. 2.3. In organic computing, the only task humans hold on to is the setting of goals. As the machine is autonomously organizing, detailed communication between programmer and machine is restricted to the fundamental algorithm, which is realizing system organization. Application-oriented mechanisms lose the status of algorithm and are treated as data, in analogy to the transcription factors in the ontogenetic toolbox.

gramming time. What is required now is to automate system development, invoking rather general mechanisms of search, pattern recognition, evolution and self-organization, such that the distinction between programmer and user will all but disappear.

2.6 Organic Computing

Let me summarize what I have said so far. We should take note that usage of the word computing has expanded by now and is embracing a wide range of applications characterized by data organization, erstwhile the exclusive domain of animals. We are heading for information technological applications that require no less than intelligence in the machine. Systems are becoming too complex to be programmed in detail any longer. The principles with which programmers formulate programs in their head have to be installed in the computer, so that it can program itself such as to conform to abstract, human-defined tasks.

No doubt this isn't just a pipe dream. Living systems, cells, organisms, brains, ecosystems and society are showing us the way. Living cells are not digital, are not deterministic, are not algorithmically controlled, yet are flexible, robust, adaptable, able to learn, they are situation-aware, evolvable and

self-reproducing. Organic computing advocates a view according to which organisms are computers and computers should be organisms. Realizing Organic Computing necessitates a broad research agenda. There are a number of fields that are already engaged in relevant activities, among them artificial life, artificial intelligence, belief propagation, Bayesian estimation, evolutionary and genetic programming, neural networks, fuzzy systems, machine learning and robotics. These fields need to be emboldened and coordinated. They need to be advanced from their peripheral position within departments to center stage, core courses and all, need to be forged into one coherent research venture. Moreover, the rich sources of relevant scientific information in the biological sciences and especially the neuro- and cognitive sciences need to be tapped, by the founding of interdisciplinary initiatives bringing together and develop the science of organization that is called for here and for understanding Life and human organization.

2.7 General aspects of organizing systems

Let me go over some of the themes that I believe will have to be developed in this context.

2.7.1 Architectures

In both the biosphere and in technology, specific systems are generally designed in two stages. First, an architecture is established that sets up a comparatively narrow universe of form, then in a second stage a specific structure is singled out from this universe. A prime example is the genetic toolkit that is widely shared in the animal kingdom. Relatively little information in the regulatory network of gene activation is able to select a specific animal species from the universe of forms defined by the bulk of the genetic machinery. Another example, in fact the one from which the name architecture derives, is the technology for creating buildings, where the universe of possible structures is defined by materials, design patterns and professional builders' skills, from which architects can select specific structures. VLSI is an architecture defining a range of electronic circuits, including digital computers, e.g., of von Neumann architecture. The generation of complex software systems is made possible by architectural frameworks including programming languages and structured and object-oriented programming. Also the neural and humoral machinery of our brain constitutes an architecture, defining the universe of mind functions and patterns.

Successful architectures manage to avoid the two dangers of bias and variance [3], of being too narrow or being too wide. The universe of processes defined by universal Turing machines is certainly wide enough, nobody being able to point out a specific bias that would exclude interesting processes from it, but on the other hand it contains too much variance and is so wide

that armies of programmers haven't been able yet to single out intelligent structures within it. On the other hand, artificial neural networks, as long as they are insisting on replacing programming with self-organization and learning, seem to fall prey to bias, defining too narrow a universe of patterns and processes. There is the so-called no-free-lunch theorem [14], which seems to suggest that there is no architecture fit to serve all structures of interest. This raises the question whether the application domain of intelligence is homogeneous enough to be captured by one coherent architecture.

There is reason to believe that the bias-variance dilemma and no-free-lunch theorems paint too pessimistic a view. They both rely on a rather narrow range of mechanisms used to single out specific structures from the originally defined universe, based on statistical estimation and optimization. Should there be more potent mechanisms of structure selection, the original architecture could be wider and still permit to define the structures of interest efficiently. The powers of self-organization haven't been sufficiently tapped to this purpose, especially for learning and adaptation.

2.7.2 Self-organization

A snow crystal constitutes a globally ordered structure both in terms of its microscopic molecular lattice and its overall dendritic shape. The forces that generate it are elementary interactions between molecules. In general, self-organization is the process by which a large number of simple elements interact by simple, naturally given forces, and out of a long and initially chaotic process of iterated interaction global order grows as a pattern of maximal harmony between these forces. Other often-cited examples of self-organization are regular convective cells, soap bubbles, the laser or self-assembled viruses [6, 9, 4, 12].

A self-organizing system defines an architecture, a universe of forms, plus a mechanism to select and create specific structures as attractor states. Such universes can still be very wide; the tremendous variety of solid materials demonstrates the richness of the universe created by atomic species and their interactions. Our goal in the context of organic computing is to define an architecture of data elements and their interactions, to be implemented in arrays of digital processors, so that iteration of the interactions lets the system gravitate towards (sequences of) globally ordered states. The challenge is to define this architecture on a very general level, without explicit reference to specific problems and applications. The latter is then to be achieved by installation in the system of appropriate initial states, an endowment of useful algorithms, and exposure to specific input patterns. The architecture, initial state and library of algorithms constitute the innate structure, based on which the exposure to specific input in education and learning prepares for specific tasks to be performed.

Self-organization is particularly important in noise-prone systems, such as the living cell or human brain or, in fact, the analog computer. The latter was brought down by the difficulty that when many elementary steps are

chained up, each one subject to some level of inaccuracy, the end result of a long computation is totally dominated by noise and useless. How does the cell or the brain avoid this error catastrophe? It all depends on the nature of the system dynamics realized by the interactions. If this dynamics is of the chaotic type, where small differences in initial state lead to large differences in final state, the system will be drowned in noise. If, on the other hand, the dynamics is of attractor type, such that sets of similar initial states lead to the same final state, then the error catastrophe is averted. The globally ordered states of self-organizing systems are attractor states. The task ahead of us in the present context is to define an architecture, a set of fundamental rules of interaction of active data elements, that turn functionally desirable system states into attractor states.

2.7.3 Cooperating pathways

Given the reliability and determinism of the digital machine, computation is customarily staged as a single sequence of transitions from initial to final state. The advantage of this is efficiency. The disadvantage is that definition of the pathways leading from problems to solutions has to come from outside the system, from a human programmer. If, on the other hand, we want the architecture of the system and the processes of self-organization to find those pathways without the benefit of a human programmer as *deus ex machina* to set it all right, we have to define good pathways in a principled way.

A general relevant principle is based on cooperative pathways. A result is a useful one if there are several mutually supportive ways to derive it. When we do mental calculation we routinely check the result by additional reasoning, like estimating the order of magnitude, or by comparison with previous calculations. In fact, whole mathematical systems, like Euclidean geometry or the natural number system, derive their absolute certainty from the mutual cooperativity – consistency – of all possible pathways of reasoning connecting facts. Even the rules of deduction derive their authority from their consistency with others and with facts. (It is remarkable to what extent mathematics ignores this background of its formal systems.)

An organized system, then, is to be seen as a large network of nodes and links, the nodes representing data items, the links interactions between them. Each data item is stabilized by the combined effect of a multiplicity of interactions, or lines of deduction, impinging on it. The system reorganizes its pathways and configurations of data elements such as to optimize the mutual consent or consistency between them. The quality of a given pathway is measured by its success in predicting or affecting its target data element, in which it succeeds only by cooperating and agreeing with other pathways.

Similarly, the decisions taken by a data element lead to consequences downstream of the pathways emanating from it, consequences that come back to either reinforce or contradict the original decision. There will in general be nested sets of such feedback loops of different length. On several time scales,

the system will thus organize configurations of pathways and data that are cooperative and mutually consistent in the way described. It is much more than a metaphor that societal decisions very explicitly are of this nature, individual decisions having to live with their own results down the line. As the saying goes, what goes around comes around.

Production systems [13] are a kind of declarative, non-imperative, programming languages, which are formulated in terms of rules or *productions*, each of which defines a *firing condition* and an *action*. Productions continually examine the content of a working memory, and when a rule recognizes the pattern defined in its firing condition to be present in the working memory it "fires" and performs its action. The neurons of the brain can actually be seen as the productions of such a system: when a neuron recognizes the pattern of activity impinging on its dendritic tree it fires its action, which consists of a pattern of excitation or inhibition on other neurons. (The working memory in this case is identical with the firing state of all the rules.) Production systems never became very popular, presumably because they were unable to overcome the problems surrounding the issue of conflict resolution — several rules firing simultaneously but contradicting each other. The nervous system obviously overcomes these problems by following multiple action pathways simultaneously and selecting successful ones, in the short run by letting contradiction annihilate itself by negative interference and consistency of alternate pathways prevail by positive interference, and in the long run by adaptively favoring such productions – neurons that are successful.

Let it be remarked that this style of computation is very wasteful in terms of processor cycles. If each result is the effect of 10 redundant pathways and has to wait for 10 iterations to be stabilized, a hundred elementary steps have gone into a calculation that could have been performed in one. I think we will have to live with this level of inefficiency as the price to be paid for autonomous system organization. What is expensive about computing, these days, is anyway no longer the computing power to realize it but the human effort going into its design. Moreover, once a successful computational structure has been found by self-organization, it can be simplified, by abolishing redundant pathways originally necessary to single out the correct ones, and by ceasing to wait for long feedback loops to come around. We observe this simplification and speed-up in our brain. Processes are first very slow, variable and unreliable when dealing with a new problem, but in long learning curves they become very efficient, reliable and fast. What we are observing is the gradual organismic growth of the (more or less distant) equivalent of algorithms.

2.7.4 Management of uncertainty

Computer science cannot deny being a child of mathematical logic. Mathematics is a world of absolute certainty. Let a single false statement creep in, and the whole edifice comes crashing down. In real life, all presumed facts

are uncertain. This is a problem for chains of logical deduction, for it means that uncertainty has to be propagated through them to be able to judge the reliability of conclusions. In consequence, the application of logic to real life is difficult. This may be the reason we usually avoid following long chains of explicit logical deduction (or when we do we easily fall into traps), although there is good reason to believe that implicitly our brain has mechanisms to handle uncertainty very well. For single, linear threads of reasoning uncertainty cannot but grow, and only by using meshes of interlocking and mutually supportive arguments can any reliability of conclusion be reached.

The currently active fields of Bayesian estimation and belief propagation are active at developing methodology to handle uncertainty. There are, though, serious problems still to be solved. One is to let a system figure out for itself what evidence to invoke for a given task. Another, how to estimate probability density functions, unavoidable in the absence of exhaustive observation. Still another, to learn more about appropriate structures of interlocking arguments; these are bound to contain loops, creating the problem how to avoid the pitfalls of logical circularity. Progress on this front is very important to the creation of systems that can autonomously operate in a world of uncertainty.

2.7.5 Differentiation

It is impossible to organize a system with a large number of degrees of freedom all changing at the same time, each guided only by a small number of neighboring ones, as dictated by the elementary interactions. The system would just be caught in local optima, with small collections of elements in mutual harmony within but discord between, if it converged at all. This happens when you rapidly cool a liquid below the freezing point and crystallization starts in many places simultaneously. The result is a large array of crystallites, small domains with different molecular lattice orientation. A recipe for getting global order is to start by organizing a few degrees of freedom, and let the order thus created spread to the rest of the system gradually, involving only a small number of degrees of freedom at a time. To get a globally ordered monocrystal, make sure that crystallization can start in only one place, at a nucleation seed, by suppressing nucleation centers anywhere else. The nucleation seed spreads its order, letting crystallization happen only at its surface, eventually incorporating all of the liquid in one coherent crystal order.

Embryogenesis starts with an egg that is small enough to be initially spanned by the organizing forces (to a large extent based on reaction-diffusion, see [10]). The first decisions taken establish global coordinate systems in the form of embryo-spanning chemical gradients, which act as signals controlling further processes, typically subdividing the embryo into smaller domains. These then undergo further differentiation into even smaller domains, and so on. The embryo gradually outgrows the range of the elementary interactions (or these are shrunk in relation to the embryo), such that more and more

degrees of freedom are opened, but they are fixated by self-organization as quickly as they arise [4]. The organic growth of companies can be described similarly: they start small and undifferentiated, the founder functioning in many roles initially. When the company grows beyond a certain size, it forms departments, which in turn may spawn subunits, and so on. This style is usually also realized in classical software development. The original idea, arising in one mind, is simple and coherent, and with time and growth, subproblems are spawned, giving work to more and more programmers.

Organic growth of a structure through differentiation ideally knows no backtracking. The sequence of decisions that fixate degrees of freedom form a tree that is traversed just once. This correspondingly is a very efficient process. Should, however, the final result not be successful, there is no rescue and the whole sequence has to be started over, the worst type of backtracking. In the course of evolution, Life creates an endless sequence of new organisms. Technology is driving its own version of evolution, spawning thousands of types of cars or computers, together paving the road to ever better products. Software technology will have to come round to adopt the same style. The labor-intensive way in which it is produced presently, line-by-line, makes it very painful to give up a software system once it has developed to some volume, forcing the mending of flaws and adaptation to new needs with the help of patches and compromises. This is what makes a software system gradually complicated, irrational, self-contradictory and incomprehensible, all due to the misadaptation of the original design to the final needs. This unsatisfactory state of affairs can only be overcome if growing a complex mature software system becomes easy, painless, fast and efficient. If the ontogenesis of organisms is any guide here, we will have to develop the equivalent of the genetic toolkit, which comprises general mechanisms to generate the coherent layout of an organism, plus a repertoire of morphogenetic mechanisms for the growth of particular tissues and appendages and of specific molecular functions which can be switched on where needed. On the basis of this architecture, Life is able to change an ape into a human quickly (on an evolutionary time scale), changing just a few control functions. The genome of the chimpanzee is said to be 98.5% identical to that of man. Likewise, a well-designed organic computing architecture should make it possible to create entirely new software systems by relatively light touches to the control of the process of differentiation.

2.7.6 Learning and instruction

The lion's share of information in my brain presumably is acquired by learning. Our whole genome (of which only a small part is specific to the brain) contains 1 GByte of information. Savants, who can absorb whole telephone books by leafing through them, put in evidence the enormous memory capacity of our brain. Normal humans' brains with all likelihood absorb as much information, although not normally being able to index it that explicitly. Research makes

it more and more clear that vision and language comprehension work by applying vast databases of acquired patterns. It is likely that this extensive reliance on learning is fundamental for brain function in general.

Information technology will be fundamentally transformed once the mechanisms of learning are understood and implemented. In spite of all efforts and claims made in various fields of study this has not been achieved yet. Input patterns beyond a size of a couple of hundred bits of information let learning times in terms of numbers of required examples grow astronomically. This problem is exacerbated if the input patterns are not all of the same context or the same learnable structure. Animals and especially humans make it clear that learning is possible from perceptual input fields (retina, cochlea etc.) in which patterns and pattern sequences contain hundreds of thousands of bits of information, sequenced hodgepodge from moment to moment. On the other hand, both animals and humans are restricted in their learning ability and can readily absorb only certain things[2].

Without this being the place to go into details, I would like to claim that the first step in any learning experience is a step of recognition. I first have to recognize a coherent pattern in my perceptual input in order to do two things: first, shut out the rest of the perceptual pattern as irrelevant for the moment, and second, categorize the input pattern so that I know where in my memory domain I can lay it down. This will then immediately permit me to find in my memory other, previously acquired patterns of the same sort and bring the newly acquired one in registry with them, part matching onto corresponding part. These corresponding parts of the same type can then form the ensembles of small patterns that are required as input to current statistical learning systems. An animal detects significant patterns in its environment with the help of abstract schematic descriptions, generated by evolution or by previous experience, which are mapped into the input by the recognition mechanism. Let's call this schema-based learning. A perhaps typical example is the schematic description of the human face infants seem to be born with [5], attracting their eyes to the mother's face minutes after birth, allowing them to quickly learn to recognize their mother, her mood and her focus of attention [1]. The necessity to prepare learning with the help of evolved schemas explains the restrictions of learning in animals to specific topics [2].

Let's assume the learning problem can be solved the way I just indicated. We could then program application systems by defining for them schematic descriptions of patterns that are significant for a given task. Only a very basic set of such patterns need to be programmed in any literal sense. If there is a sufficient critical mass of them, more could be created by human system instructors by pointing out examples that are simple enough and central enough to a theme so they can drive schema-based learning.

2.7.7 Abstraction – instantiation

It may well be that the centerpiece of intelligence is the brain's ability to establish and maintain the relationship between concrete, detailed situations and abstract, schematic descriptions of situations. Faced with a problem in a new situation I recognize in the situation a general pattern that relates it to situations I have seen before, and while modeling the situation at hand as a concrete instantiation of that pattern I can complement it with additional elements that constitute a solution. The main point on which humans are ahead of animals may be possession of a richer, higher and more abstract level of representation, most or all of it acquired culturally.

The abstraction-instantiation relationship is certainly central to computing. My high-level program is a relatively abstract description of the machine code that eventually is executed. The block diagrams with which I might start planning a program intend to be abstract descriptions of the concrete code I eventually write. Computing is based to a large extent on the mechanisms for traveling between abstraction levels. Many of these operations are performed automatically by appropriate algorithms, such as compilers or debuggers. But the majority of these operations are still going on only in the heads of people, such as applying general methods to concrete problems, or recognizing that a particular object class is appropriate to represent a particular problem.

The challenge to be met is to automate the processes of abstraction and instantiation by mechanisms that are general enough to work in new, unforeseen situations. Let's assume the abstract schemas to be applied are already resident in the system (generation of new abstract schemas is a very complicated issue in itself). Abstraction then amounts to a recognition process, recognition that the concrete situation contains a subset of elements that map to elements of the abstract schema under preservation of relations. In instantiation the challenge is to select for each of the elements or subpatterns of the abstract schema a concrete role filler from among a multiplicity of stored candidates, and to make all those choices in a coherent fashion so that the relations dictated by the abstract schema are actually realized in the instantiation. It is a complex and very important research subject to create an architecture whose initial configurations and mechanisms of self-organization can implement the processes of abstraction and instantiation on the very abstract level described just now, endowing the system with the ability to learn from examples to better and better navigate the abstraction hierarchy. Children demonstrate the feasibility of this, learning the skill after being taught abstract schemas together with a few relevant examples.

It is often predicted that we will be able to communicate with our computers by speaking to them. The difficulty of this is not the recognition of words from sound patterns. Although this is not a very easy task, it is routine by now. Apart from the difficulty of parsing and understanding complex sentences, a big problem is that natural language expresses things on a very abstract level. My car's navigation system is very good at turning maps of street arrange-

ments in front of me into coherent commands in natural language, but just think of the converse, me ordering my car to turn into the driveway to the right after the next intersection. Mapping that abstract description into the visual scene and into motor commands is quite a challenge.

2.7.8 Goal representation

As stated earlier, systems should be designed merely by the definition of goals. Definition of goals is a very complex business and must take place on as abstract a level as possible. Our task cannot be to tell the machine what to do in every possible concrete situation. Progress in programming efficiency has been very much progress in being able to formulate larger and larger classes of situations on an abstract level so as to treat them with one program. This trend will simply have to be accelerated decisively.

Again, Life will have to show us the way. Animals are born with drives and instincts to direct them purposefully through life. Ethologists have worked out a number of specific cases and have especially spent effort on finding out the innate patterns defining behavioral drives. It turns out that these innate patterns seem to be formulated schematically on a rather abstract level. The gosling is programmed to trust and follow mother goose, and its first task after hatching is to find out who mother goose is. According to Konrad Lorenz, the description of her is so abstract that he could imprint goslings to follow his yellow Wellington boots trough all their youth. The innate abstract pattern is just good enough to be triggered in one or several scenes, upon which learning mechanisms pick up more concrete details from those scenes, replacing or enriching and differentiating the original schema. Education is the process by which such learning experiences are chained up to map the originally very abstract definitions of innate behavioral patterns into real life situations.

We are animated by many goals, and they are related to each other in complex ways, being dependent on or in conflict with each other, and we spend a good part of our life doing nothing but sorting out what we like and want and should or should not do. Asimov originally believed the relationship of robots to people could be regulated by just three simple rules, but he later had to realize that those rules were by far not sufficient to deal with all the vicissitudes into which robots and people are likely to stumble. We will have to spend enormous amounts of effort to teach computers to behave, but in doing so we should not be bogged down with their digital details, any more than we ever lose time on the neural details of our children's' cortical gyri.

2.8 Conclusion

What I have called Aspects here is indeed to be realized simultaneously as different aspects of one fundamental system design. Thus, the recognition and pattern completion inherent in abstraction and instantiation are to be realized

as processes of self-organization that realize the required fine-grained homomorphic mappings between instance and abstract schema, developing from coarse to fine in a differentiating sequence. The data-and-process architecture of the system must naturally reflect and implement the structure of the organic world of which computing is to be an integral part and parcel. Learning and the implementation of goals are realized by one and the same mechanism of schema recognition, which focuses attention on significant segments of scenes.

My own interest is in understanding the brain, and it is my conviction that the best way to do so is by replicating one of its functions paradigmatically in the computer, that is, by acting as an engineer. We know the brain is realized as a network of simple elements, neurons, and their communication via electrical and chemical signals. Artificial neural networks (ANNs) attempt to model the brain's architecture. Taken as digital switches, as formulated in [8], they are a universal medium but don't self-organize. In analog form they can be made to learn and self-organize, but then they fall very short of anything to be called universal. The dynamic link architecture (DLA) [11] is an attempt to realize all the aspects of organic computing discussed here, and I am in the process of realizing one paradigmatic brain function on its basis, visual object recognition. Although much work is still to be done to reach full functionality and to take away all algorithmic crutches, no serious hurdle is in sight for this venture.

Organic computing may or may not be able to get off the ground in direct competition with solidly established software applications such as operating systems or enterprise software, and it may have to prove itself in novel fields that are too expensive to develop in classical programming style. Vision is such a field. Four decades of frustration made it clear that replicating vision on the computer is a very complicated thing, both in terms of processes and data. Mankind will never muster the resources to generate it while programming line-by-line. Full-fledged computer vision will only be realized with the help of organic growth, learning and instruction, that is, by organic computing.

In 20 years' time, large new information systems will be generated by starting with a widely adopted fundamental algorithm that defines the data-and-process architecture of an electronic organism, the equivalent of the genetic toolkit of animals. An initial state will be generated that defines basic schemas that implement goals, (thus directing the system towards a specific application field) and lay the groundwork for learning, and then a period of education and instruction will adapt the organism to the intended type of environment. Finally, users will train the system on particular jobs. This development will completely blur the distinction between natural and artificial systems.

Acknowledgments

I gratefully acknowledge funding by the Hertie Foundation and by the EU project FP6-2005-015803.

References

1. L. Cohen and M. Strauss. Concept acquisition in the human infant. *Child Dev.*, 50(2):419–424, 1979.
2. C. Gallistel. The replacement of general-purpose learning models with adaptively specialized learning modules. In M. Gazzaniga, editor, *The cognitive Neurosciences, 2d. ed.*, pages 1179–1191. MIT Press, Cambridge, MA., 2000.
3. S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.
4. B. Goodwin. *How the Leopard Changed Its Spots: The Evolution of Complexity.* Princeton University Press, 1994.
5. C. Goren, M. Sarty, and P. Wu. Visual following and pattern discrimination of face-like stimuli by newborn infants. *Pediatrics*, 56:544–549, 1975.
6. H. Haken. *Synergetics — An Introduction, Nonequilibrium Phase Transitions ind Self-Organization in Physics, Chemistry and Biology. 2nd enlarged edition.* Springer, Berlin, Heidelberg, New York, 1978.
7. D. Hammerstrom. Biologically inspired architectures. In R. Waser, editor, *Information Technology.* Wiley-Series: "Nanotechnology", 2006.
8. W. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bull. Math. Biophys.*, 5:115–133, 1943.
9. I. Prigogine (with I. Stengers). *Order Out of Chaos.* Bantam Books, New York, 1983.
10. A. Turing. The chemical basis of morphogenesis. *Phil. Trans. Roy. Soc.*, B237:37–72, 1952.
11. C. von der Malsburg. Dynamic link architecture. In M. Arbib, editor, *The Handbook of Brain Theory and Neural Networks, 2nd Edition*, pages 365–368. The MIT Press, 2002.
12. C. von der Malsburg. Self-organization in the brain. In M. Arbib, editor, *The Handbook of Brain Theory and Neural Networks, Second Edition*, pages 1002–1005. The MIT Press, 2002.
13. D. Waterman and F. Hayes-Roth. *Pattern-Directed Inference Systems.* Academic Press, New York, 1978.
14. D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.

3

Systems Engineering for Organic Computing: The Challenge of Shared Design and Control between OC Systems and their Human Engineers

Kirstie L. Bellman[1], Christopher Landauer[1], and Phyllis R. Nelson[2]

[1] Aerospace Integration Science Center, The Aerospace Corporation, P.O. Box 92957, Los Angeles, California 90009-2957, USA.
bellman@aero.org, cal@aero.org

[2] Electrical and Computer Engineering Department, California State Polytechnic University Pomona, 3801 W. Temple Ave., Pomona, California, 91768, USA.
prnelson@csupomona.edu

Summary. The term "emergence" is usually used to mean something surprising (and often unpleasant) in the behavior of a complex system, without further qualification. Designers of OC systems want to manage emergence in complex engineered systems so that it can contribute to, or even perhaps enable, accomplishing the system's performance goals. That is, OC designers aim to construct systems that are more flexible and adaptable in complex environments, to gain some of the advantages in robustness and adaptability that biological systems seem to gain from these phenomena. In this chapter we suggest some principles that we believe underlie the enormous flexibility and opportunistic adaptability of biological systems. We show how these principles might map to systems engineering concepts when they do, and what to do instead when they don't. We then describe five specific challenges for the engineering of OC systems, and how we think they might be addressed. We also discuss the key role played by language and representation in this view of designing and deploying an OC system. Finally, we describe our progress and prospects in addressing these challenges, and thus in implementing systems to demonstrate the capabilities that we have identified as essential for successful OC systems.

Key words: Biologically-inspired system architectures, computational reflection, layers of symbol systems, representational mechanisms, self-modeling systems, systems engineering

3.1 Introduction

Organic Computing (OC) would like to take advantage of one of the key attributes of biological systems; they adapt and change on multiple time scales as they evolve, develop, and grow, and they do so without external direction

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_3,

or control. However, such self-design and self-organization is at variance with any of our current engineering methods for designing and controlling complex systems. One of the central challenges to OC systems is that not only do we want somehow to create the foundations for biological-like system properties, but also that we must do so in a manner that allows us to monitor continually, manage and even further develop such systems while they are in operation. Hence, we can learn from biological systems, but in fact OC systems face a unique challenge: OC systems must remain closely linked to us, their designers, builders, and users. This chapter addresses the challenge of how OC systems engineering can be accomplished by providing specific capabilities that enable the system and its human developers and systems engineers to jointly shape system goals and behaviors.

In this chapter, we start with an emphasis on certain characteristics of biological systems and describe how such characteristics – if they were to be imitated in engineered systems – lead to several striking new challenges for the human systems engineer. These difficult tasks include how to share control with a somewhat autonomous system and how to change the traditional role of the systems engineer from attempting to determine and build all system characteristics to a new role of carefully building in key interaction points for evaluating, shaping, guiding, deterring, or preventing certain system behaviors. This new style of interaction between the human engineer and the system implies that there is also a fundamental shift in what we as the engineers believe we can design the system to do, and in how we evaluate what acceptable solutions are. This new methodology changes our notions of sufficiency, optimality and any other evaluation criteria we attempt to apply to the design and the performance of the engineered OC system. Furthermore, since any evaluation criteria will partly develop along with system capabilities, we must design a system that does not have elegant predefined responses, but rather can generate reasonable solutions on-the-fly.

Learning how to effectively share control between humans and partially autonomous systems is already familiar to the research community; it is just made more difficult by the degree of self-modification and self-organization in an OC system. After all, OC is a continuation of automation, except that instead of just responding autonomously, the OC system is able to self-design some of its response capabilities to the world, maybe even including its own sensory as well as "action" capabilities.

To be a systems engineer for a system with the resources to adapt over its operational life requires a redefinition of the concept of "optimal" that has driven traditional design. Specifically, the OC systems we are proposing must contain components and processes that are not optimized for the narrow *a priori* definition of system *specifications* that has traditionally formed the basis for design and validation. For example, new emergent features enable new strategies, and therefore by necessity will fall outside the specifications previously defined for the system. Therefore, we have to evaluate the systems design and performance, which includes developing new metrics, as well as

design the system components and processes so that they can be used and evaluated under unexpected – even unintended – circumstances in a monitorable way.

To be a systems engineer for any partly or fully automated system capable of adaptation and reconfiguration of its components and processes requires sharing control with that automated system, and hence a negotiation between the local and even private requirements of the autonomous system and the often more global perspective and requirements imposed by the system developer or user. These differences are not only in viewpoint (for example, how information is locally or globally understood and determined), but in the contexts for requirements and system capabilities. For example, the system developer may need to consider not only the system's operational context, but also legal, political, social and indeed moral contexts for potential uses of the system. Therefore, the human developers and the system may have very different purposes and goals. For example, immediate costs to the autonomous system may bias its reasoning processes and therefore its developmental processes, to the detriment of necessary long-term goals hoped for by the system developers and expected by the system users or owners. These negotiations between human system developer and partly self-determining OC systems lead to a number of distinct challenges for the human system developers and systems engineers.

In the sections that follow, we discuss many of the processes that are central to OC system capabilities. However, before we do so it is worth emphasizing here that there are three classes of processes that we discuss: first are those processes that we believe underlie the distinctive and remarkable properties of biological systems, for which we discuss how we might build analogs appropriate for OC systems; second are those processes that may or may not exist in biological systems, but certainly are critical to OC systems in order to make use of the biological-like processes; third are OC processes that are critical to our ability, as the human engineers, managers, users, and owners, to communicate with the OC system, to manage and to share control with it, and possibly to repurpose it. Because of the importance of system-human communication, we argue that meaningful and context-specific communication between the system and its designers, developers, and users is essential to this endeavor, and that therefore, the creation and use of appropriate and sharable language is fundamental to its success.

In section 3.2 we discuss such biological characteristics as permissive growth and development, how biological systems achieve controlled sources of variation, and the opportunistic nature of biological processes and systems. We close by emphasizing several differences between biological systems and engineered systems that will drive the challenges for systems engineers.

In section 3.3 we examine the systems engineering challenges of developing the above capabilities, focusing on five specific challenges. The first of these challenges is to create generative processes. That is, although traditional design methods include tools for adjusting an operating point within a known

parameter space, we will also need to develop processes for our OC systems that can efficiently create new and very different possibilities for the system. Secondly, because OC systems will adapt and change, the instrumentation that provides information about the system's current internal state will also need to rapidly adjust in a number of ways to the system's increasing complexity. This challenge also implies that we will also need to develop tools for creating evaluative processes that express the results of measurements in ways that are useful and understandable to both the system and its engineers, developers and users. The next challenge that we address is how to build the capabilities for reflection and direction that enable an OC system to identify and assess possible responses, and choose, implement, and adjust them as its context and understanding shift. Our fourth challenge is to enable our OC systems to utilize a portion of their resources to "actively experiment", discovering properties, relationships, attributes, and limitations of both their own capabilities and their ability to operate within different environments. The final engineering challenge is to combine the capabilities resulting from the previous four challenges to enable our OC systems to build models of their changing environment, and to use those models to identify unusual features of their situation. That is, we suggest that an OC system must achieve a situational awareness capability that directs its resources toward the aspects of its environment and internal state that present, at the current time, the most important threats or opportunities.

In section 3.4 we discuss processes that enable the OC system to share information and control with its human developers and managers. In order to build the basis for shared control, the system and the human must be able to communicate about system state and control decisions, and also to negotiate plans and goals. Hence we consider the difficult problem of developing shared representations and languages. We also discuss our progress and prospects along these lines.

In section 3.5, we wonder aloud if we could perhaps take advantage of some of the biological principles suggested in this chapter to better organize our own discovery processes as a community of OC researchers and to leverage off of each other's work as we together confront the challenges of achieving the potential of OC systems.

3.2 Key biological principles for an OC system

One of the fundamental goals for OC is to develop systems with key biological-like capabilities to adapt and change on multiple time scales, and to evolve, develop, and grow on their own in response to their current state, their context (including the goals and purposes of their designers, owners, and users), and their history. This goal is motivated by the astonishingly wide variability of responses that are observed in biological systems, as well as their remarkable robustness in responding to sudden large changes in their environment.

In this section we focus our attention on several principles that we believe are essential to providing the foundations of biological-like adaptation and robustness: building processes rather than building components, "permissiveness", generative processes and controlled sources of variation, and "opportunistic" processes.

Hence, in section 3.2.1, we emphasize that one of the most important aspects of biological processing may be, in fact, that biological systems build the processes that create and maintain biological structures, rather than building structures *per se.* Instead of attempting to achieve a particular structure or a particular result, the emphasis is instead on building processes which are analogs of factory floors or assembly lines, an image easily extended to cellular and genetic lines. As we discuss in this section, this approach means that basic elements are constantly rebuilt and renewed, which allows points of entry for all sorts of adaptive possibilities.

In section 3.2.2, we describe how the "assembly lines" of nature do not reproduce the precisely-constrained products that we strive for in engineered systems. Rather, numerous observations of variation of biological components and their further differentiation into new types point to a type of widespread permissiveness. The "permissiveness principle" allows all interactions, relationships, variations, actions and results *unless* any of these are shown to be deleterious or harmful. One could only use such a principle if there are methods for monitoring and discovering the results and effects of such variations. Clearly one can only follow such permissive strategies in the context of populations of elements. The building processes noted above provide both the populations of elements or events and many of the means for changing those elements or events.

The permissiveness principle results in the occurrence of many different kinds of unintended interactions, resulting in turn in side-effects and emergent phenomena (section 3.2.3). These sources of variation and of novelty are critical to enabling the types of changes in a biological system that, if used correctly, can become the basis for adaptive responses over the life of an individual cell or organism or, on a different time scale, over the evolution of a species. However, even though permissive processes provide many novel kinds of variations and occurrences, biological systems have found that the "hit-or-miss" quality of changes and variations stemming from emergence and side-effects is not persistent or consistent enough to meet the requirements for controlled sources of variation required by many adaptive processes. Therefore, biological systems have somehow created active processes that generate variations. In this section, we describe two qualitatively different types of generative processes: ones that create relatively well-defined, persistent, and constrained sources of variation and ones that change the nature of the solution space.

As clearly indicated by the above arguments, the resulting broad range of possible system behaviors could be exploited by adaptive processes. We call such processes "opportunistic" because they are designed to notice and then

take advantage of variations and events occurring due to emergence, side-effects or controlled sources of variations. In section 3.2.4, we present some examples of "opportunistic processes" in animal systems and describe some of the capabilities of biological systems that enable these processes.

Lastly, in section 3.2.5, we discuss the differences between OC processes and biological processes that we contend are essential because OC systems are artificial, engineered constructs rather than the result of the evolution of an entire ecology. Even with appropriate analogs of all these biological principles, an OC system requires additional capabilities and processes in order for us, the human developers and users, to monitor, shape, and negotiate with it. This last topic will lead us directly into section 3.3, which presents the challenges for OC systems engineering that we deduce from these biological principles.

3.2.1 Build processes not structures

One of the obvious properties of biological systems is that they grow and they develop. Growth and development are at once adaptive advantages for an organism because it can respond to its changing environment with its growth and development, and a necessary result of life: since there is no external designer and developer of a biological system, it must "bootstrap" itself into existence by this growth and development. This then is the essential reason for the principle that biological systems in fact build the processes that create and maintain biological structures, rather than building a structure directly. Any system that self-organizes and self-designs will require some bootstrapping processes. The interesting question here is whether the bootstrapping processes of our artificial systems will need to use the same biological strategy of building up a family of related elements that can then be differentiated and used by the system.

We begin by considering how biological systems emerged from a less differentiated universe of matter and energy in the first place. Without speculating about this evolution in detail, we draw three important ideas from such imaginings.

First, when persistent biological structures emerge from the dynamics of physical systems, they are indeed persistent and separate, but only in a relative sense. This is because they are created out of the *same materials* — and therefore share in many ways the same fundamental parameters at some basic level — as their surroundings. A cell wall is semi-permeable; a brain region is a recognizable region with sloppy boundaries and extents, and so it goes. One of the implications of this view is that the boundaries of a biological structure are always leaky and somewhat continuous with the world around it.

Another implication, emphasized below, is that because these structures share so much in common with their surroundings they continue to exist only because of active building and maintenance processes. Unlike algorithms or transistors, which one can consider to be "permanent" when viewed in terms of the operational lifetime of the system, there are many examples in

biological systems where biological "components" such as cell walls, muscle, or neural pathways go away or change if active maintenance processes change. A good example is what happens to bone and heart muscles in the zero-gravity environment of space [63, 15, 64].

Not only are there many biological examples of components disappearing upon disuse, but there are also many examples where biological systems appear to make use of these active building and maintenance processes to support crucial flexibility in systems. This "flexible modularity" is seen clearly in language and movement. As Bellman and Walter [13] state,

> We have overused the idea of built-in structures by being overly dependent on prewired patterning. This concept places the emphasis on the coherence and the "fixedness" of the assemblages. It largely ignores the means of introducing flexibility and variability into the combinations of elements used in assemblages. Yet the ability to recombine relatively independent elements and hence to decompose the assemblages is an equally important and complementary process to our ability to form those assemblages. Any word or movement can potentially be combined with a very large number of other words or movements to form a large number of sentences or acts. Hence, both language and movement are structurally coherent in the assemblages and are also generative. We use the word generative because it puts the emphasis on producing and originating new forms that conform to a body of rules. We also think of this generative quality as being acted out in an "on-line" fashion. That is, the animal is constantly generating new assemblages as it acts or speaks and as it adjusts for and monitors the context. Many of these assemblages could be temporarily formed for the moment's purpose, which places the emphasis on the processes that combine elements and not on the fixedness of the combinations.

In section 3.2.3.2 we discuss in more detail the types of adaptive behavior supported by such generative processes. Here we simply want to emphasize that structures and behaviors within an OC system will be more like these biological "assemblages". That is, active processes will continually recruit the necessary components, build useful assemblages, and maintain those assemblages, often doing so only for the duration of a specific current context.

The second key idea about the evolution of biological structures is that biological systems continue to use the dynamics among emerged structures to create and maintain new structures, hence building up many layers of structures with complex interactions. One of the results of the above viewpoint is that the emerged structure does not have to be made to fit with its surroundings. Rather, because it has emerged at all, it is *ipso facto* viable within its surroundings. In that sense there is a continual validation – in engineering terms – of the interface (but not necessarily validation of the performance or the functions of that component or set of relationships).

Another result is that the boundaries of the emerged structures may be less distinguishable from the rest of the system and its external environment than we might expect or desire. But since the boundaries are in our terminology more "leaky", they share more properties with the other parts of the system, including many layers of the system at once. Such shared parameters could, through "opportunistic processes", become the means by which the system both integrates across different system elements and adaptively controls parameters that have been found to vary meaningfully with critical differences in system behavior and the accomplishment of different system goals. This property results in complex side effects, but also provides tremendous opportunity for the use of shared characteristics in adaptive and integrative processes. Our design problem, described in section 3.3, is to help develop the types of discovery processes in the OC system so that it finds the ones that are most useful for our purposes.

As an example, an important characteristic often observed in biological systems is that a single control element such as a "master gene" or neurotransmitter can have multiple, diverse, and widely distributed impacts throughout the system's levels and processes. In a system whose structures are continuously maintained, created, and modified by processes that use the same basic raw materials, this distributed effectiveness of a control element allows local adaptations while helping to provide a basis for system-wide integration. In a system with leaky layers, one could imagine how serendipitous combinations of side-effects, if properly captured, could result in such wide-spread effects and help provide the basis for integrating across diverse kinds of elements and layers of elements. Thus, a third key concept we can learn from biology is that as increasingly complex processes and structures are developed some common control elements link them. Often these shared control elements are really families of elements related through the history of their development through common "building processes" and through retained common features. However, there will also be differences among the control elements within a family due to local specialization. That is, because many of these building processes are distributed throughout the entire biological system, their assembly lines can be impacted and specialized to local conditions. Hence the fact that there are building processes is key to both providing the similarities among families of elements and the "entry points" for the adaptations that will occur because of local requirements.

One of the most important consequences of shifting from building structures to building processes is that a system will have a much broader range of possibilities. The continual renewal of processes and structures gives the system a "safe" region of its "possibility space" within which it is relatively free to adjust, because its existing processes and structures are known to be successful in at least some "nearby" portions of its possibility space. At the same time, because these processes are shared and distributed across many parts of the system's hierarchy, some of the integrated responses can also enable "long leaps". This means that local adaptation at one level can have widespread

effects across the hierarchies of emerged structures and components. These long leaps also enable the recruitment of far-ranging and diverse components.

The biological style offers a very rich set of control options that include both controlling for processes and for outcomes. In cybernetic terms, Kreitman's conjecture [36], states that in an environment of arbitrary disturbances, at any particular time one can control for either the process or the outcome, but not both. Biology does both, though clearly at different times. There is a first emphasis on building processes that generate populations of imprecisely replicated and varying elements. In section 3.3.4 we discuss the other necessary processes where, through feedback and selection mechanisms of several sorts, the system refines and modulates these processes to get desired results. When these building processes operate in a permissive biological environment they produce a wealth of interactions and emergent structures that will be utilized by the biological system.

3.2.2 Permissive growth and development

In this section, we explore the implications of the "permissiveness principle", which we consider a fundamental biological principle, one that helps separate biological from engineered behaviors and capabilities. In order to introduce it, we will first start with a brief description of classical systems engineering, and then contrast that with biological "permissiveness".

Systems engineering for traditionally-constructed systems defines and locks in the performance requirements for the system and the interfaces among its components. During the early design and specification stage, often called conceptual design, the foundational mappings of functions onto specific components are identified. These choices become the basis for specifying the rest of the system, so that consideration of alternatives is often frozen out of the ensuing design process. This approach helps to organize and manage the design process, which is focused on the familiar and extremely useful representation of a system as a block diagram that details both the individual subsystems and their allowed interactions. However, the choices of the contents of the boxes (i.e., what hardware and operational capabilities will be grouped together) and the interfaces between them (i.e., what symbols they will exchange and in what directions) can have profound influence on the functionality of the final product. Hence, the concept of a system organizing itself seems not only foreign, but perhaps also a bit dangerous, especially given that unanticipated behaviors of these traditionally-designed systems often result in catastrophic failures.

The challenge for systems engineers in the traditional approach can thus be seen as one of finding the best partition of the system; that is, to define the blocks and their interfaces. However, a focus on the contents of the boxes and on their designed-in interfaces leads us to ignore a wide variety of small interactions with the expectation that they will not contribute to the behavior of the system as a whole. In the context of a system made of a very large number

of elements (for example, cutting-edge microprocessor designs have over 700 million transistors in an area less than 0.5 cm^2), there is an increasing potential for "small" interactions to lead to emergent behavior with unintended impacts, some immediately observable as detrimental; some detrimental over much longer time frames.

Let us now consider qualitatively how biological systems differ from this classic style of engineering a system, in which one specifies and designs specific components, engineered to be as uniform as possible, and specific interfaces with other components so that the system controls as much as possible the interactions among components and hence any side-effects. Although there are many critical biological processes for shaping, refining, and controlling the system's dynamics through monitoring, regulatory and feedback processes of many types, biological systems allow a great deal of variability and imprecision in their components.

Similarly, we already mentioned that there is a great deal of leakiness among biological components and between levels of components. One important aspect of biological leakiness and variability is that many – perhaps even most – of the system's interactions and structures aren't controlled for directly. In fact, unlike engineered systems, multiple parallel and overlapping processes and structures exist in biological systems. These parallel and overlapping pathways are another manifestation of the permissiveness principle.

Biological systems appear to be deeply permissive at all levels of organization. That is, anything goes so long as the organism or system hasn't learned (e.g., over evolution in populations of organisms or by feedback in terms of a single organism) that there is a harmful or deleterious effect. Thus, in a biological system anything that is not physically impossible can become part of its behavioral repertoire or feature set unless it is explicitly disallowed or prevented by processes within the organism. This permissiveness principle will produce a large number of unregulated features and relationships, some of which will be noticed at any time by monitoring and regulatory processes. Regulation could eventually decrease those features and relationships that negatively impact the system's viability and functioning and by the same mechanisms preserve and even enhance ones that support system's viability and functioning.

Permissiveness could be one of the essential principles that allows a system to generate emergent phenomena and side-effects and the processes that utilize them. That is, some of these serendipitous combinations of effects relating families of distributed command elements, described before, will later be the basis of the opportunistic processes that are able to make use of such emergent phenomena.

3.2.3 Emergence, side-effects, and controlled sources of variation

In the classic view of adaptive control, a system equipped with its unique combination of sensor, behavioral, and feedback processes assesses its current

state and the relevant features of its operational environment, decides what courses of actions are feasible, and then selects, plans and executes the needed control adjustments and behavioral actions. Sensors, feedback, control adjustments and actions all presuppose that the system is able to vary any aspect of itself as a response to its plans or to events in the environment. Those attributes a system is able to change are limited and shaped by its physical and intellectual capabilities. Together these form a space of possible variations, which we call the "possibility space." Any selection of control adjustments, sensor tasking, actions or plans will be a subset of this space. In this section, we suggest that the permissiveness principle creates the proper milieu for emergence and side-effects, and that these, in turn, are critical sources of variation that are captured and made use of by adaptive systems to extend their possibility spaces.

Although permissive processes provide many novel kinds of variations and occurrences, biological systems have found that the "hit-or-miss" quality of changes and variations stemming from emergence and side-effects is not persistent or consistent enough to provide the controlled sources of variation required by many adaptive processes. Therefore, somehow biological systems have created active processes that generate variations. In this section, we describe two qualitatively different types of generative processes: ones that are relatively well-defined, constrained sources of variation that are persistently produced and maintained, and a second type which changes the possibility space and expands the design envelope in unexpected ways.

3.2.3.1 Emergence and side effects

We leave to others the challenge of characterizing emergent phenomena[3] and developing methods for predicting emergence in complex systems. Instead, we accept that side-effects and emergence are known to be possible, and go on to consider how a complex system could develop processes that take advantage of such phenomena when they occur, and make use of them to be more robust and adaptive.

Although emergence and other side-effects continue to plague human-engineered systems, in biological systems it is clear that emergence and other unexpected phenomena are utilized by the system to provide needed sources of change and novelty. We believe that permissive processes promote emergence, since they allow unexpected or unplanned coincidences to reinforce each other.

Cellular automata and other dynamical explorations demonstrate that stable, long-lived structures can emerge from stochastically-generated initial conditions in a "flat" rule space [27]. Although such demonstrations are very

[3] We find the following working definition of emergence by Christian Müller-Schloer to be useful: "Emergent phenomena is where collections of individuals interact, without central control, to produce results that are NOT explicitly 'programmed' and which are perceived as 'orderly.'" [69]

important for developing our understanding of emergence, the conditions in many of them are quite different from the conditions for emergence in biological systems. By examining biological systems, we may learn new strategies for the management and utilization of emergence. Emergence in biological systems 1) does not start with a uniform distribution of identical elements; 2) involves a system with goals, intentions, and learning; 3) must be observed from inside the system by the system. That is, in dynamics demonstrations, the observer is outside the system, but in biology the system itself must be the observer.

In biological systems, emergence occurs in a system that has lots of existing structures, which embody the history of both the individual and its species. These structures and processes can be recruited to support an emerging behavior or structure. Unlike the homogeneous elements of many dynamical models that were used valuably to prove the existence of emergence from very simple and uniformly distributed elements, biological systems have heterogeneous components at many different functional layers. Such existing structures mean that across the biological system there will be regions with quite different ongoing dynamical processes. These existing structures both constrain and shape the dynamics that may lead to emergence and at the same time, become the basic components of a different level of dynamics in the system. For example, the human nervous system has meaningful dynamics occurring at the molecular, cellular, and organ levels, all of which may lead to emergent structures and behaviors with impacts across the different levels.

The second major difference in biological emergence is that biological systems have goals and intentions; they plan and learn on their own and from others. Hence the emergence of new patterns resulting from unexpected dynamics and the side-effects of widely distributed relationships among components both impacts and is impacted by the existence of purposive behavior in biology. In this sense, the central challenge of OC research is also the central challenge of biological systems: how to combine unanticipated patterns and events with intended patterns and events.

Emergence can appear very differently when viewed from inside the system or by an external observer. The whole emergent pattern may never be seen as such from the vantage point of internal observation, but instead look like a set of correlated differences or changes distributed throughout different locally monitored regions. Ian Stewart [81] presents many interesting points about the problems faced by a participant in a dynamical system. These include observing patterns and making decisions based on limited information from within the system and attempting to abstract decision rules from the behavior of the system. Some of these problems can be demonstrated with Langton's ant. These simulations have two simple rules assuming an initially all-white grid. Step into a square; paint the square you came from black if it was white and white if it was black; if the current square is white, turn right and if it is black, turn left.

Stewart describes the patterns generated by simulations of Langton's ant as first "symmetric", then "chaotic", and then the ant "builds a highway forever" in about ten thousand steps. If the simulation starts with a pattern rather than all white squares, apparently it still ends up in a "highway." Stewart's conclusion is that one cannot predict this pattern, but rather only do the simulation: there is no possibility to shortcut the process, which he considers a key characteristic of emergence. Furthermore he considered the calculation of the rules for the behavior as intractable. In his analogs of the ant game, there can be several billion steps before something interesting happens. And then such analogs cannot tell you why or even how such patterns came to be. There are many points in this work that should be considered by those of us building OC systems. However, biological systems may have sidestepped some of these difficulties because they have found ways to utilize emergence without having to perceive the whole pattern or predict the endpoints in some emerging process.

In biological systems, the local-only perceptions and decisions of Langton's ant can have far-ranging effects because of the shared assembly lines and common control elements noted earlier. Furthermore, biology has also developed strategies to combine and coordinate the perception, learning, and decisions of a community of players through diverse memory and communication processes.

Biological systems have developed communication methods that allow them to share their experiences and viewpoints. This is discussed further in section 3.4. The importance here is that, although this does not change the intractability of perceiving the end points in their own evolution, it may change the ability of a biological system to track its own long-term and emergent patterns because other members of its species can observe it from the outside and communicate those observations. For example, if one animal even at a simple level can inform another that it is too close to a dangerous situation, that can motivate that perceived system to discover what changes or clues in its environment or self it ignored and allow it to correct those insufficiencies. Parents, of many species, constantly do that with their offspring.

Recent work in distributed optimization and market-like decision processes also show that if the decision game is set up correctly – including the topology of the relationships among participating decision elements – then local decision-making elements can not only come up with coherent and useful global decisions and effects but, under many conditions, optimal ones [71].

As an example that summarizes all the points we have made here, consider walking. The biological system starts with a number of built-in structures, such as neural pattern generators that contribute to the timing of the gait and successful configurations among muscles, tendons, and bones that contribute to walking for that individual or species. However, the system does not have to predict or lay out the values of all of these muscles and neurons to walk. Rather, it appears to launch the behavior with some rather stereotypic gait patterns and an abstract goal of intended place or direction. It then

uses self-monitoring, feedback, reflection and other opportunistic processes to dynamically recruit and organize its components. It constantly adjusts for external conditions, e.g., the slight shifts in ground, and for that system's own performance capabilities. For example, fatigue will impact how high it lifts its limbs or the speed of its gait. Parents help train and shape their offspring in the important movement patterns of their species — and the offspring clearly learns as well through trial and error. The resulting walking pattern details and path could not have been predicted ahead of time and therefore in accordance with Stewart's definition are emergent. However, it was not important to the biological system to know in advance that level of detail in its planning or to predict such details.

Clearly a system's measurement capabilities are critical to its ability to self-monitor and therefore to respond opportunistically to emergent processes. If the appropriate opportunistic processes are available, and if they are coupled with sufficient instrumentation, then neither a system nor its engineers, developers, and users need to distinguish between emergence and other effects. Instead, all of the permissively-attempted paths, together with their results, become available for use in achieving the purposes and goals set for the system.

3.2.3.2 Generative processes

Permissiveness alone may not be enough for a biological system to be able to adapt to changing contexts because the needed amplitudes and types of variations may not be available. Hence, biology has developed generative processes. Some of these produce relatively well-defined and constrained sources of variation that are persistently produced and maintained; these are like the volume knobs or the tuners on a stereo. In some sense, they embody a parameterization of some key characteristics of the system that can be controlled by the system. A second type of generative processes change the nature of the possibility space. They are likely to be much rarer in occurrence and more often fatal to the system. However, a few will survive the test of viability with the rest of the system to provide very new attributes, and can be picked up by the system in surprising ways. Some of these changes could lead to very different ways of doing things and eventually, in higher cognitive processing, to very different choice and decision processes.

There are many examples of processes in biological systems that first generate new combinations of elements and then, after evaluation, incorporate the successful ones into some more persistently available form. Examples of these generative processes occur at all levels of biological systems. At the genetic level, we observe that both meiosis (the normal reshuffling of genes from both parents) and mutations (the "errors" resulting from the imprecise and inefficient genetic replication processes) provide variation in a population that is acted upon by natural selection to create species that are adapted

to an environmental niche and also robust enough to handle many unanticipated environmental stresses. Meiosis is a dependable source of variation in the well-defined space of current genes. Mutations are rarer, mostly deleterious or lethal, but once in a while, greatly successful. Meiosis gives rise to variations within a local safe region of the possibility space for the species, while mutation is a source of long leaps to possible new regions.

At the cellular level, neurophysiologists have studied and speculated on the "helpfulness" of variations to allow more resilient pattern generation and to prevent perseveration or over-recruitment among cells. A special case is the ability to generate variations that act like noise generators. As an example, Garfinkel [28] studied the use of such noise generation in the regulation of heart patterns and recovery from abnormalities,

> The extreme sensitivity to initial conditions that chaotic systems display makes them unstable and unpredictable. Yet that same sensitivity also makes them highly susceptible to control, provided that the developing chaos can be analyzed in real time and that analysis is then used to make small control interventions. This strategy has been used here to stabilize cardiac arrhythmias induced by the drug ouabain in rabbit ventricle. By administering electrical stimuli to the heart at irregular times determined by chaos theory, the arrhythmia was converted to periodic beating.

An example of generative processes at the behavioral level is behavioral merging. Behavioral merging is not only an example of generative processes but also an example of the way in which the resulting new combinations and variations in behavioral elements can be used to handle adaptively a common control problem. It demonstrates the resolution of conflict between competing goals, tasks, or requirements, and the ability to map actions to goals in highly flexible ways. As described by Bellman and Walter [13],

> A given instance of behavior can reflect several motivations and work toward several goals at once. Contrary to the usual emphasis in behavioral studies, in which an animal must choose between mutually exclusive acts, an animal in nature is rarely in the situation where it must engage in one behavior to the exclusion of other behaviors. Rather, an animal's movement frequently shows "behavioral merging," in which several motivational goals and action patterns are combined into one coherent pattern. In studying the merging of feeding and aggression behaviors in the lizard, an animal noted for the rigidity of its behavior patterns, Bellman found that when elements of feeding and aggression conflicted, other elements were selected and substituted, so that, overall, both feeding and aggressive patterns were combined into one fluid behavioral sequence.
>
> The behavioral sequence resulting from merging points to a particular type of flexibility in a movement system. A specific movement pattern

> can subserve a number of goals. If this is so, then a specific movement pattern is not necessarily linked to one goal any more than to any other goals (although there may be some kind of weighting, so that a given behavior is most often associated with one particular goal). This implies that a movement is not "released" as a necessary consequence of the occurrence of a particular motivational goal; rather it is "recruited" to serve that goal. Furthermore, from behavioral merging, we see that a whole action pattern need not be recruited but only those elements best fitting the circumstances.

This last point reinforces the importance of building processes, which provide the biological system with many points of entry for adaptive responses. These include the ability to drop steps in its processes if existing components of the system or features of the external operational environment permit it.

Lastly, in human language we clearly see generative processes that produce a large variety of phrases and patterns, while remaining consistent with both grammatical and semantic rules. This last example brings to the fore that these generative processes operate not only on physical and behavioral properties of the system, but also on the symbolic and representational capabilities of the system that underlies its cognitive and communicative capabilities.

Generative processes result in many different combinations of processes, structures, and symbols that can accomplish a particular outcome when combined with methods that build up or emphasize some relationships, processes, or symbols while de-emphasizing others. Such a variety of choices enables a system or structure to substitute for a "broken" method or less effective method, an ability that is at the core of the robustness of biological systems. Compare this to the usual engineered system, which carefully specifies and engineers away all sources of variation and interactions among components, and which is brittle in the sense that it usually "breaks" if presented with unanticipated contexts or interactions.

3.2.4 Opportunistic processes

As clearly indicated by the above arguments, the resulting broad range of possible system behaviors could be exploited by adaptive processes. We call such processes "opportunistic" because they are designed to notice and then take advantage of variations and events occurring due to emergence, side-effects, or controlled sources of variations.

One of the classic attributes of adaptive behavior in animals is their ability to take advantage of whatever is in the environment in order to accomplish their goals. Even simple creatures, known for rigid behavioral patterns, usually are able to break their behavioral patterns if something in the environment has made those parts of the behavior unnecessary. [4] Hence if a suitable hole

[4] Konrad Lorenz had some notable exceptions such as the egg rolling behavior of the gray goose or the nut storing behavior of the red squirrel, but such rarer

exists, the organism will not dig it, if a suitable object is nearby the organism will not go further to move another object, and so forth. In more advanced animals this ability to take advantage of aspects of the environment is extended into sophisticated problem-solving capabilities that allow the animal to make use of objects in novel ways. An early example comes from the famous "insight experiments" of the German psychologist Köhler, who demonstrated the ability of apes to use objects in the laboratory in novel ways [35]. One such experiment required the apes to reach a desired banana by discovering that they could link several sections of a pole together. [5]

Jakob von Uexküll, the prolific and observant German ethologist, was one of the first to consider the qualities that go into adaptive and opportunistic biological systems. Especially important was his insight that an animal's perceptions are deeply affected by its effectors (its capabilities to move and do things with some given object or within some given ecosystem). One of his early stories is a description of how a hermit crab's motivational state affects its perception of an empty shell [83, 75]. When the crab was molting and vulnerable, it backed into the shell for protection. When the crab was hungry, it approached the shell displaying hunting behavior. When the crab was mating, it approached the same shell as a potential competitor and showed aggressive displays.

This type of opportunism and adaptiveness appears to occur even in the behavior of single-cell animals. As Jennings [30] observed, unicellular animals are capable of many of the complex and adaptive behaviors of multicellular animals. They respond to the same classes of stimuli to which humans do, they have specialized receptive areas (although not yet specialized for different senses), and they frequently have specialized contractile parts whose action is coordinated. They exhibit spontaneous behavior, early trial-and-error behavior, habituation, and context-dependent responsiveness to stimuli. As Jennings concludes, "We do not find in the nervous system *specific qualities* not found elsewhere in protoplasmic structures. The qualities of the nervous system are the general qualities of protoplasm." [30, page 263]

Jennings provides several fascinating examples of the ability of single-cell animals to interact with the environment and with each other. One example is the predator-prey relationships among *infusoria* such as *Didinia* and *Paramecia* [30, page 186].

---

examples are usually understandable in light of the criticality of that behavior to the animal's survival: one could say that nature in that case has over-engineered the response.

[5] Parenthetically, the apes in this case, claims Köhler, devised a smarter solution than his. Instead of the apes using the extended pole to knock the banana down, they used the extended pole to polevault their way to the banana. The importance of this observation is that the possibility space is determined partly by the animal's unique capabilities. Clearly, Köhler was not agile or light enough to polevault, so he did not recognize this possibility.

His observations demonstrated several startling, adept adaptive capabilities, including the ability for a hunter to cease its hunt of one highly-elusive target and turn to a better target, even though the first target was still visible and potentially available. The hunter subsequently caught the second prey. It did not cease its chase of the first target because it was no longer hungry, or because it was too tired to continue, or because the target was out of the scope of its sensors. Rather, it apparently stopped because it had not been successful enough within a certain amount of time and effort.

These biological examples are behavioral, and hence observable to scientists, but there is no reason to suppose that this opportunistic style of processing is not carried out at all levels of biological systems. In fact we contend that they are. Although we can only touch on a few illustrative examples here, we also believe that it is instructive for OC researchers to study the myriad ways in which different biological systems adapt. Such examples can inform as well as inspire the OC field on the style and power of the opportunistic processes we would like to develop for our artificial systems.

One of the first things that is evident from these biological examples is how critical generalization, differentiation, and learning processes are to the ability of the system to notice, capture, incorporate, and adaptively control the co-occurrences and interactions among a rich set of different types of components. Babies demonstrate wonderful examples of opportunistic developmental processes that are tempered by discrimination and learning. At first, a baby attempts to fit anything and everything into its known behaviors and goals. Everything it can grasp goes into the mouth, is picked up, is dropped, and so forth. Gradually, through refinement, differentiation, and learning processes, the baby learns what is too hot, too acrid, too heavy or just right. Its explorations are carefully constrained by concerned parents.

Opportunistic processing will critically depend on the instrumentation available to the system, which will determine what variations it can perceive, capture and incorporate into its repertoire of repeatable capabilities or reproducible states. Again, at all levels of the system, from genetic, cellular to behavioral and cognitive, we see very different means by which correlations and co-occurrences are retained and eventually culled until the correct aspects of some complex set of events has been retained by the system for future manipulations.

The building processes discussed earlier enable several of these opportunistic strategies. For example, the observation that animals can take advantage of the existence of structures or events in their environment or in their own metabolic pathways happens partly because in the building processes one has developed a set of feedback and monitoring processes that support a sequence of operations by having a large number of steps that are initiated or not, depending upon the occurrence of pre-conditions such as chemical precursors or trigger events. In addition, permissiveness will guarantee a large number of relationships and side-effects, creating the very good chance that the system will often stumble on "shortcuts" in its sequencing of events.

Just as combinatorial processes can be taken advantage of, so can processes that appear to be oppositional. In this case, the manipulation of opposing effects results in good solutions. Examples of this are the ways that combinations of neurotransmitters with opposing effects combine to provide the desired states, as well as a means to modulate such states with very small adjustments. Similar examples occur in the tensions between flection and extension muscle groups for limb movement, as well as the balance between excitatory and inhibitory neural pathways. In all these cases, the benefits of such an arrangement seem to rest on the ease with which small control adjustments of either opposing effect can lead to big changes in the states.

The opportunistic processes feed a number of critical processes that evaluate, reflect on, and utilize the information that has been gathered about correlated states to drive the behaviors of the system. However, opportunistic processes cannot just depend on the co-occurrences resulting from external events to drive their opportunities for correlating, refining, and differentiating the drivers for complex states. Hence they also need to drive their exploration of correlates. In section 3.3 we examine how active experimentation is used by the system to constantly learn more about how its own components interact and affect each other.

3.2.5 Critical distinctions between biological processes and OC processes

It is tempting to concentrate only on those processes that enable the biological-like characteristics that interest us, e.g., those processes that can generate novel behaviors and responses, those that contribute to the discovery and utilization of emergent patterns, etc.. However, in OC systems it is also essential to consider what other capabilities and processes must be added in order to allow human engineers, managers, users, and owners to communicate with and guide the OC system not only during the initial design phase, but also throughout its operational life. Such considerations are especially important for large and costly systems that will experience shifting contexts during their design and life-cycle.

There are two driving differences between the needs of biological systems in general and OC systems. The first is a result of the essentially "alien" nature of the OC system we noted in the beginning of this section. Biological systems are not only somewhat continuous with their environment, but are also part of a complete ecosystem; that is, in collaboration and competition with other systems, all of which are linked to and part of a larger whole. In contrast, the very concept of engineering a system leads it to be disconnected and "alien"; it is developed by us, usually with materials very different from those in its operational environment, doing functions that may have little to do with those of any other system in the operational environment.

In addition, because the biological system is continually renewed and recreated from the raw materials of its surroundings, it may more readily adjust its

structure and composition because of small shifts in the available materials or the current conditions. Engineered systems on the other hand have predefined form and substance. Of course, this can have some advantages in an environment that suddenly no longer feeds the building processes of the biological system. However, it will not have the opportunities described above for adaptation through growth and development, and there will be no biological-like renewal of composition and structure.

One result of this is that OC systems require much more work in defining appropriate and deep enough context models of the operational environment, and in ensuring that there are appropriate interfaces for noticing all the needed attributes of the environment in coordination with the system's own sensing behavior and actions. One cannot depend on the "natural" shared parameters based on the physics and natural history of an ecological niche that occur with biological systems.

The second major requirement of OC systems that is different from biological systems is the need of the OC system to always remain tethered to us. They must always be monitored for and shaped by our goals and purposes. This does not mean that the OC system will not be able to act somewhat independently of us; in fact, we argue later that one of the chief modes of interaction with the OC system will be through negotiation and not through the usual fixed control methodologies used in other engineered systems. In so far as an OC system is self-organizing, it will in fact *be* one of the developers and one of the systems engineers of itself, in coordination with the human development team. But this need for entrée into the internal state of the OC system's sensors, effectors, and decision processes leads to the need for sufficient instrumentation, reflection and reasoning, and communication capabilities to work with us. As we discuss in section 3.4, the language of this collaboration and negotiation needs to be co-developed from the experience base of the OC system. We will require these systems to inform us of their state and intentions. Thus, an OC system must not only create appropriate symbols for its own use, but must also be able to explain them to us.

Whether structures, processes, and representations emerge or whether they are supplied at the beginning by the designers, what is important to us as OC systems engineers is to build in mechanisms for recognizing new possibilities, and for guiding the development process to some extent based on our (and the system's) growing understanding of the application problem domain and the processes and structures that have developed within the system in response to it. As much as possible, we also want to keep some of the means to "know" and to "find purpose or meaning" retained by the system, since the design engineers are unlikely to be available for the entire lifetime of the system. Thus, unlike a biological organism, an OC system will always retain a special type of differentiation from its surroundings in that it is deployed into an operational context to achieve purposes other than its own survival.

3.3 Systems engineering challenges

Our biological analogies have led us to suggest that OC system engineers will need to provide certain key capabilities that we believe enable the adaptability, opportunism, and robustness of biological systems. However, doing so presents significant challenges precisely because these capabilities require a radically different approach to engineering design, in which the processes that constantly rebuild, renew and expand the system, rather than the system's components and their interactions, are the central focus. Unlike the processes and resulting structures that we observe in nature, those created or managed by engineers will of necessity be artificial, if only because they are deployed to accomplish externally defined goals and purposes, and hence are in an important sense alien to their environment rather than having evolved as part of a complete ecology. Because we want biological capabilities in systems that have "non-organic" origins, we the developers will have to provide the means for detailed interactions with its operational environment and the starting structures and processes usually provided by evolution.

This means that a number of approaches and designs for such systems cannot work, because some basic assumptions and other design crutches are no longer available, no longer implicit in the approach. Instead, a new set of systems engineering methods and attitudes is needed, and this section begins our exploration of these issues.

The specific engineering challenges we will consider in this chapter are:

- creating appropriate generative processes (section 3.3.1);
- inventing appropriate instrumentation and evaluation processes (section 3.3.2);
- providing the capability for the system to analyze and reflect on the information it has gathered (section 3.3.3);
- enabling the system to actively experiment so that it improves its (and our) representations and models (section 3.3.4); and
- providing methods for the system to refine the symbols, representations, and models of both the system and its context, to create a kind of "situational awareness" (section 3.3.5).

We now discuss each of these challenges, as well as their implications, in more detail.

3.3.1 Creating generative processes

By generative processes we mean something much more interesting and much more challenging to construct than search processes, because the latter search a fixed and given space using predetermined parameters, whereas the former actively create both the parameterizations and the search space as they proceed. The additional capabilities of these generative processes as compared to traditional techniques are expected to yield some surprising results, desired

and undesired, that the system can discover, evaluate, and choose to exploit or suppress, as well as new possibilities that offer increased effectiveness in controlling for processes and qualities that support the system's purposes and goals. These generative processes thus need to be able to efficiently create new and very different possibilities for the system, as well as to discover and describe linkages that couple processes and symbols, enabling coordinated responses among multiple processes and functional units at lower levels of the system's hierarchy.

Designing generative processes thus means developing capabilities to recognize and coordinate coherent activity among subsets of the processes that build functional units, as well as methods for describing this coherence by creating new symbols or variables. Such an approach is fundamentally different from the traditional conception of searching a predefined space using predefined variables and predefined criteria for success. For example, in computing algorithms we usually assume a predetermined basis set of characteristic and well-behaved variables, write a generic search through a large space chosen because it is easy to describe in terms of those variables, and then apply the success constraints to eliminate large portions of the space. An alternative approach, which seems to be more like what humans do, is to create the search space and new characteristic descriptions of it as we go along. We start with the construction of the search space from the viewpoint of our purposes or goals, which lets us incorporate many of the constraints into the construction itself, so that we automatically avoid consideration of large but uninteresting portions of a generic search space, and concentrate on those regions that seem interesting or useful, even if the resulting possibility space is made up of disconnected and oddly-shaped portions of what would have been the generic search space, and even though the resulting space would have been difficult to describe in terms of a set of predetermined variables. We also organize our understanding of this search space by making up new interpretable descriptions of the various options we discover, a form of reparameterization that allows us to more easily use these new possibilities in other contexts.

The previous example suggests that purposes and goals are central to the efficient construction of generative processes, since they provide a ranking of high-level criteria for success. We saw many examples of this continuous mapping between the goals and the generative processes in the biological examples described in section 3.2.3.2.

When we humans solve problems we actually seem to use two basic strategies: we start with familiar possibilities and search for relevant or useful combinations, but we also use long leaps to radically different possibilities. This example of human problem-solving illustrates the two types of capabilities that we include in the overall description of "generative processes". One is more local and continuous, resembling refinement or "pushing the envelope", while the other involves disconnected leaps. In section 3.2.1, we speculated on how the assembly lines and building processes support these long leaps by producing families of distributed control elements that share features and

whose domains of action include simultaneously different levels of modules or components in the system. Generative processes as we conceive of them must thus include some methods that are able to efficiently "leap to" and evaluate regions of the system's possibility space that are "far" from those it has already experienced, as opposed to only searching for a new combination of established parameters. Although these exploratory leaps could be accomplished by random variation of basic units, such a random combinatorial search is not sufficiently efficient in terms of resources and time even at fairly modest levels of complexity. In fact, the size-, context-, and time-dependent structure of these possibility spaces makes it impossible even in principle for random searching to explore the space in any useful amount of time.

We envision the following characteristics for "leaps". They will produce relatively large changes from the current state, frequently "land" in "useful" final states, utilize previous discoveries of successful strategies, be closely coupled to evaluative, reflective and directive processes that result in the depreciation or elimination of unsuccessful results and the reinforcement of interesting ones, and be strongly context-dependent.

We suggest that generative processes in OC systems are likely to be based on the "modularity" of the higher-level processes and structures that the system knows so far, and perhaps even more particularly to be based on the representations of those processes and structures. Biology seems to take advantage of modularity to achieve many of those characteristics that we desire in our artificial systems. Deem [23] has described biological modularity, as well as the hierarchy of complexity that accompanies it, as follows.

> A modular structure to the molecules of life allows biological information to be stored in pieces. The existence of this modularity means that evolution need not proceed just by changes of one base of the genetic code or movement of one atom or amino acid at a time; rather, functional units can be exchanged among living organisms. For example, [...] proteins often comprise almost independent modules, and the genetic information that codes for those modules may be transmitted through evolution. The modular structure of proteins is hierarchical, with identifiable elements at the levels of atoms, amino acids, secondary structures, and domains. Hierarchical elements continue through the levels of proteins, multiprotein complexes, pathways, cells, organs, individuals, and species.

Thus, one way to produce both meiosis- and mutation-like generative processes is to mix previously-created and relatively high-level functional units in new ways (perhaps by rearrangement or reuse in a new internal context), because existing functional units embody known successful strategies. The results of such operations with higher-level units of the system hierarchy not only provide methods for expansion of the system's possibility space, but do so in a way that ties to methods of building new descriptive terms based on existing ones. Another way to understand the potential of modularity is to

recognize that the building processes that continually rebuild and renew the system's components and structures are also in a sense modular, with various entry points depending on context. This means that these same processes can be recombined in new ways using a context-dependent combination of sequences of entry points.

One potentially valuable way to model these effects may be the class of small world effect models currently studied in network science [84, 2]. These models are used to study the impacts of having local neighborhoods of links interspersed with a few long links.

Modularity-based approaches are likely to be much more efficient than those that modify or combine low-level processes and structures. If given sufficient resources, searches based on low-level components will certainly eventually discover not only the same "reshufflings" and "leaps" as modular schemes, but potentially much other useful information as well. However, we contend that approaches based on low levels of a system's hierarchy will generate many more unsuccessful proposals because they largely ignore the knowledge of the possibility space and trajectories through it that are represented by previous successful discoveries embodied in the modules.

Since search processes need a space to search in, part of the problem here is to construct that search space in a sufficiently flexible way to also allow sufficiently fast searching. We specifically do not use the term "efficient" here, since efficiency is the opposite of the robustness we want the system to exhibit. The modules themselves are like safe regions; that is, safe configurations with local allowable variation. The combination of modules is a way of combining disconnected safe regions in the possibility space for constructing a search space.

We have been discussing the advantages of modularity; this is one of the chief legacies that biological systems get from their evolution. That is, there will be many side-effects that are at best neutral or, as in the case of genetic mutation, largely deleterious. It takes a special narrow combination of constraints and coincidental events to show the benefits of an effect. One of the ways that a biological system benefits from being a member of a family of biological systems is the leverage of many failed possibilities being constrained away before it comes into existence. Many of the partial structures and properties of the biological system are in fact the embodied memory of these constraints now physically imposed by genetics, the structure of its physical and cognitive components. We as the human systems engineer working with the developing OC system will play the role of evolution through simulation and our experience across families of like systems. Providing some of this experience base and the constraints is in fact one of the chief jobs of the systems engineer for OC systems.

The ability of an OC system to create its own languages and representations is an important key to achieving the efficiency of modularity-based generative processes in part because these internally-produced symbols can be exchanged with other processes and structures both within a single system

and across multiple realizations of a system. Such exchanges of symbols in effect disseminate the knowledge of processes and structures that have been proven to succeed in at least some part of the system's possibility space, a point to which we will return in section 3.3.4.

All types of generative processes that we advocate are powerful tools for building adaptive systems, but they are not themselves sufficient to produce adaptability, robustness, opportunism, and other biological-like characteristics that we want in OC systems. The results of these generative "experiments", both those operating within a search space and those violating its boundaries, must be measured and then evaluated in terms of the symbols and languages known to the system (section 3.3.2).

The system and its engineers must also be able to reflect on the results of measurements, comparing them to models of internal operation and external context. These comparisons involve the critical (and difficult) ability to compare models and results on the fly, adjusting the models to fit new, and especially novel, results. In addition, the capability for reflection (which takes place at many levels of the system hierarchy) is needed to identify possible responses and project their consequences into the future. At any given time, the system must choose how it will respond. We call this "direction."

Coupling generative processes to instrumentation, evaluation, reflection, and direction in a strongly permissive milieu gives the system an ability to actively experiment (section 3.3.4) when the resources to do so are available. We contend that generative processes play an essential role in enabling effective active experimentation, and conversely, devising and evaluating active experiments is essential for the creation of effective generative processes. Therefore, because of these mutually enabled roles, we believe that research on strategies for designing these processes is essential to progress in engineering OC systems.

A final challenge related to developing generative processes is that, in addition to supplying the original generative processes, the raw materials which they manipulate must be built as part of the engineering process. We make no assumptions here about where these beginnings come from, but we expect that a lot of it will be imposed from the outside, by the designers, who will be making decisions concerning the raw materials, initial conditions and processes, basic symbols, evaluation and validation criteria, and other starting points.

3.3.2 Instrumentation and evaluation

The traditional block diagram view of a control system as shown in figure 3.1 implicitly assumes the ability to measure values of the reference input, actuating signal, manipulated variable, controlled variable, and feedback signal. The instrumentation for these measurements is implicit in the diagram, since each block uses one or more of these variables as its input. In this section we consider the challenge of implementing appropriate instrumentation in OC

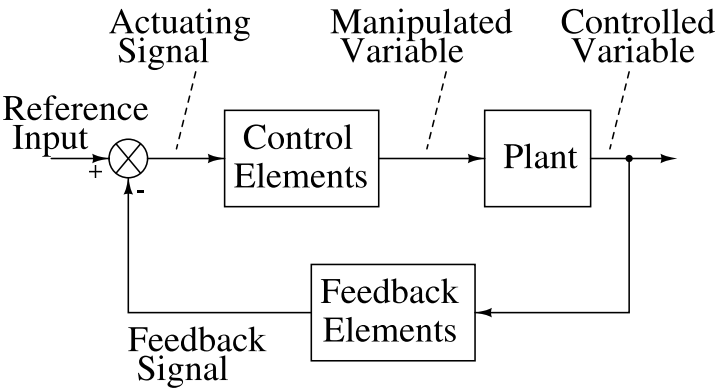


Fig. 3.1. Feedback control system block diagram

systems, where the system will need to adjust and even perhaps create its instrumentation in response to its changing processes, structures, symbols, representations, languages, models, and context. We also consider the closely related challenge of evaluation, which is the capability to use measurements obtained from the system's instrumentation, together with models of its state and trajectory, to achieve its purposes and goals.

Although the approach represented in figure 3.1 has supported an immense body of successful engineering analysis and design, the sense of simplicity and comfort we perceive from these successes is tempered by the knowledge that some engineered systems demonstrate complex and challenging phenomena such as bifurcation, chaos, and emergence. One of the goals for OC systems is to enable our engineered systems to recognize and, where possible, make use of these phenomena to achieve their purposes and goals. We are thus led to the questions of what to measure and how to measure in order to give our systems the information that enables this capability.

These instrumentation questions are not unique to OC systems. In fact, they resemble both theoretical and experimental challenges in physics having to do with modeling and measurement of systems that are discrete at the lowest scale but also have average effects at higher scales such as pressure and temperature that are *usually* sufficient to describe the collective behavior of a large number of these discrete elements in a particular region. Pippard [74] has captured one version of the physicist's view as follows:

> Can it be that the systematic reduction of complex processes to their basic constituents, obeying laws of marvelous simplicity, has left us with a body of knowledge whose usefulness is rather problematical? It has been the habitual claim of physicists that they could make predictions whose verification underpinned the laws and conferred on science a validity that no other branch of learning could aspire to.

> Was this a delusion? Of course not, but the claim may have been overoptimistically expressed.
> It was always recognized that complexity might preclude detailed prediction — no one ever hoped to follow the motion of each molecule in a gas. But long before statistical mechanics provided a theoretical foundation, it was clear that the average properties of pressure, velocity, temperature, etc., obeyed quite straightforward laws of thermodynamics and hydrodynamics. Straightforward though they were, the equations expressing them were still capable of yielding highly irregular solutions, and this time there is no molecular complexity to explain turbulence away — it is intrinsic to the equations. This should have been enough to alert us to the potential in almost any non-linear differential equation to surprise us by the diversity of its solutions.

The difficulties of predicting physical phenomena such as phase transitions and the turbulence transition in fluid flow, and especially the challenge of recognizing such changes in global behavior in real time from a set of local measurements made inside the medium, give us a concrete analogy for considering instrumentation and measurement in the context of complex systems, and particularly in OC systems.

Basically, we need to answer two questions: "How much instrumentation or information is good enough?" and "What kind of instrumentation or information is good enough?" In the examples that phase transitions and fluid dynamics demonstrate, instrumenting every process and structure down to the equivalent of the discrete molecules in a fluid does not necessarily ensure that our instrumentation or the information it produces is "good enough" in the sense that we can link it to useful descriptions, parameterizations and models, and, ultimately, to purposes and goals, even when the "molecules" have fixed properties.

Instrumenting everything is also not a useful approach for other reasons. For example, too much instrumentation may fundamentally alter what we instrument, or the instrumentation and measurement may consume so much of the available resources that it overwhelms the capability to process it, meaning that it is impossible to make use of the resulting data. For example, in wireless networks, as the number of mobile nodes increases the messages tracking their location can overwhelm the capacity of the network, precluding its ability to accomplish its goal of transmitting content messages.

Also, if we measure all the details then we need models at the same level of complexity. (Otherwise we might as well focus our measurement efforts on determining averages at appropriate scales.) Our notion of "particularity" is useful here. The enormous amount of detail that can be relevant in a complex dynamic environment for any system means that the system needs a lot of help in observing and describing it. Any proposed modeling method will suffer from the complete inability to reach a "critical mass" of information until there is enough descriptive detail, since the important interactions may include what

would otherwise seem to be minor effects, but which can sometimes combine to be dominant. However, the more detailed the model the more information it may require before the remaining uncertainties are small enough that the results are useful. What this means to us is that the system that is attempting to survive in and interact with its environment needs multiple methods in addition to multiple resolutions for describing it, and that verification and validation are essential (i.e., every hypothesis is tested, and every conclusion is provisional).

In fact, because there are very good higher-level continuum models that are almost always adequate for describing the overall state and trends of the system's operation, it seems far more useful to implement almost all instrumentation at levels of the system's hierarchy that inform these "continuum" models. This leaves the question of how to recognize those cases where these models are inadequate, which we address by implementing and exploiting the use of reflection and situational awareness, a topic to which we return in section 3.3.3 and 3.3.5. Notice how this instrumentation and these models will correspond to modules.

Another way to approach the question of designing instrumentation is to ask if there are characteristic scales that are especially significant for recognizing emergent phenomena. Again, to take the physicist's view,

> We know now that the invisible hand that creates divergences in some theories is actually the existence in these theories of a no man's land in the energy (or length) scales for which cooperative phenomena can take place, more precisely, for which fluctuations can add up coherently. In some cases, they can destabilize the physical picture we were relying on and this manifests itself as divergences. [24]

This conception that there may be some appropriate scale at which emergent phenomena first become significant gives hope that it could be possible to build instrumentation that identifies at least some emergent phenomena. Additionally, the modularity-based approaches we have suggested as a basis for generative processes may be helpful in addressing this challenge. As generative processes develop new possibilities for the system they in effect "parameterize" their descriptions of those possibilities to give new higher-level descriptions that are exactly correlated with useful regions of the system's possibility space, and include to some degree this sense of scale. From this view, the challenge of instrumentation can be restated as how to describe the disjoint regions of possibility space that have been found to be interesting, and how to represent the possibilities within each of those regions. Just as generative processes offer enormous advantages for simplifying the process of finding "good enough" strategies quickly, we suggest that instrumentation that is linked to the structure of the system's evolving set of possibilities will leverage the same simplifications.

Linking instrumentation to the modularity of processes and structures is particularly significant in the kinds of OC systems we are advocating because

their processes and structures are not fixed. Unlike the clearly delineated blocks, interface protocols, and variables of the system in figure 3.1, we expect that OC systems will have multiple and interacting processes using and affecting the same measured variables, precisely because these variables are linked to the structure of the system's possibility space. Thus, in OC systems as in biological ones, the same variables may be used in multiple models at various scales of resolution (levels of the system's hierarchy). Since we expect that OC systems will be used in changing contexts, they will also face measurement challenges related to resolution and dynamic range, as the required granularity of analysis depends on the operating context. It is therefore necessary to consider very carefully the problem of how the system gains its knowledge of internal and external events and processes, which leads us directly to instrumentation issues.

Instrumenting an OC system is in our view especially challenging because it is critically interdependent with the capability to continuously evaluate the measurements produced by that instrumentation. However, we see opportunity for addressing these very significant questions in both this interdependence and in the biological paradigm of introducing processes that create structures. The layered hierarchy and leaky, overlapping processes and structures of biological systems, together with the foundational concept that the system's structures are built from processes rather than being fixed for life as in figure 3.1 means that the system has the capability to adjust its instrumentation, or even to create new instrumentation, in response to changing needs. That is, such an approach satisfies the need for multiple descriptions coupled to verification and validation.

Such self-modification of instrumentation requires the capability to evaluate the available data to determine the system's position and trajectory within its possibility space. More specifically, we must invent new types of models that are able to continuously accept measurement data, identify and rank the importance of sources of uncertainty, and propose changes to the instrumentation that can reduce those uncertainties. This concept of dynamically integrating instrumentation and evaluation has been considered in the NSF Dynamic Data Driven Applications Systems (DDDAS) program.

> DDDAS is a paradigm whereby application/simulations and measurements become a symbiotic feedback control system. DDDAS entails the ability to dynamically incorporate additional data into an executing application, and in reverse, the ability of an application to dynamically steer the measurement process. Such capabilities promise more accurate analysis and prediction, more precise controls, and more reliable outcomes. The ability of an application/simulation to control and guide the measurement process, and determine when, where and how it is best to gather additional data, has itself the potential of enabling more effective measurement methodologies. Furthermore, the incorporation of dynamic inputs into an executing application invokes

> new system modalities and helps create application software systems that can more accurately describe real-world complex systems. This enables the development of applications that adapt intelligently to evolving conditions, and that infer new knowledge in ways that are not predetermined by startup parameters. [22]

The phrase "adapt intelligently" is significant here because it ties the changes in the system to purposes and goals. Thus, a strong guiding principle for instrumentation is that it supplies information in terms of representations of the state of the system that can be related to purposes and goals. It is this relationship of instrumentation, representations and models to purposes and goals that allows the system to evaluate their effectiveness and make choices of alternatives and refinements.

To summarize, we believe that we need to develop new types of models that support evaluative methods and processes for the OC system and the OC system engineer to shape, control, divert, and correct the generative processes through reflection on the information produced by feedback and instrumentation. We expect that multiple measurement and evaluative processes will be in operation at all times at various levels of the hierarchy of an OC system. Then reflection is used to analyze the effectiveness of the system's own processes in context, by continuously comparing these evaluations with models to estimate possible trajectories in the system's possibility space, ultimately resulting in the system choosing some combination of actions or directions that are expected to support achieving its goals and purposes.

3.3.3 Reflection and direction

Like others, we [49, 50] have argued that self-perception and self-monitoring are critical features for goal-oriented autonomous systems in order for them to move around their environments. In other words, one can imagine designing an organism or a robot with bumper-car feedback that hits a wall and stops or turns. In many ways, that can suffice for certain types of simple activities in very constrained environments like a room with four rectangular walls and a hard floor. But even in elementary creatures, such as crabs, lizards and crayfish, we see much more sophisticated adaptive mechanisms [13, 83]. Animals are very competent at knowing how high they can leap, how fast they can run, and what hiding places they can enter. This knowledge is only partly about their own capabilities, but more importantly about how their capabilities map into their environment. As Churchland said [18, page 74], "self-consciousness on this view is just a species of perception [...] self-consciousness is thus no more (and no less) mysterious than perception generally." He goes on to emphasize the considerable variety of "self-monitoring" [p. 185] that occurs at different levels. Recognition and perception of what is "oneself" and what is not oneself are difficult processes, but we readily can identify their occurrence in a number of biological systems, from single cells in immune systems [73] to

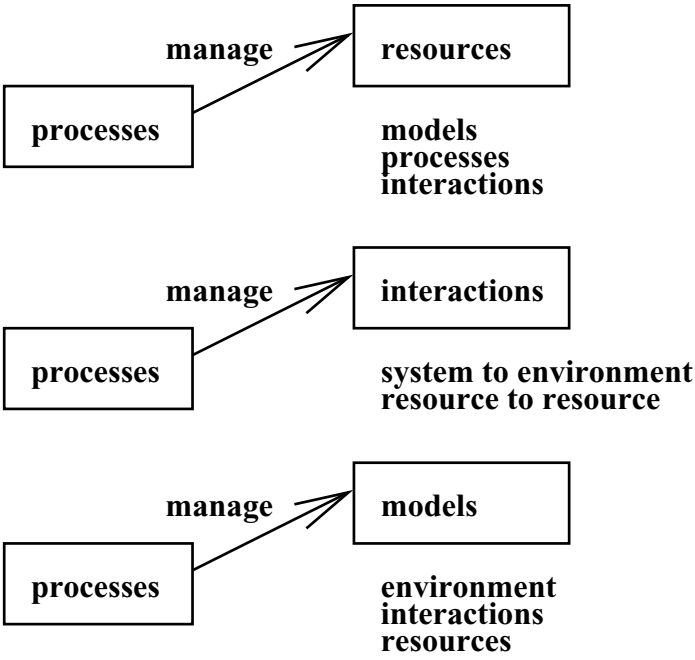


Fig. 3.2. Reflective software system processes

mammals. It is easy to imagine mechanisms that could make those perceptions available to higher level and more cognitive systems.

Self-monitoring capabilities are not the same as "self-reflection". For example, the means to monitor internal state and respond to that internal state are available to a thermostat. Ironically, although we have indisputable evidence for self-reflection in humans, our most concrete definitions of self-reflection capabilities come from the world of computer programs. Patti Maes [65] defines reflection as "the process of reasoning about and/or acting upon oneself." Practically speaking, in computers, computational reflection means having machine-interpretable descriptions of the machine's resources. We have found in our approach [42] that it is extremely useful to have not only state information available but also general meta-knowledge about the limitations and required context information for all the system's resources. There are then processes that can act on this explicit knowledge about capabilities and state in order to better control the system in its performance and maintenance [12, 42, 45]. It is clear from Damasio's discussions [19] that he is thinking of his "third type of image" as being available for both self-monitoring and self-reflection in a sense compatible with the ideas described here.

The engineering challenges we have considered so far will give us systems that discover new options (generative processes), and also measure and model their current state (instrumentation and evaluation). The next challenge that

we consider is how to implement processes that generate, identify, and evaluate possible responses based on the system's reflective capabilities. These reflective capabilities will explicitly reason about what relevant information and what relevant resources the system currently has, the effectiveness of the current strategies and approaches to reaching goals, and the current shortfalls in both capabilities and information. Then other processes can, DDDAS-like, task sensors to collect additional information or creatively use combinations of existing resources, or replan current approaches.

As we have discussed in the previous section, our proposed OC systems implement instrumentation and evaluation at multiple resolutions in possibility space and in time and at multiple levels of their hierarchy, and reflection and direction must be implemented over these diverse scales as well. To use a biological analogy, reflexes supply rapid responses in situations that are likely to be critical to an organism's survival. Our systems will undoubtedly need this same "rapid response" capability, as well as a means to identify the conditions that should trigger these responses. At the same time, just as with biological systems, OC systems will need to adjust their original responses as they obtain and evaluate more information. This continual reflection at different time scales and levels results in multiple responses to a particular situation. In addition, these responses may have different levels of complexity and sophistication, as well as being operative over different time scales.

A fully reflective system has processes that collectively and cooperatively manage all of the resources, including the models, processes, and their interactions; it has processes that manage all of the interactions, including system to environment and resource to resource, at multiple time, space, and conceptual scales; and it has processes that manage all of the models, including those of the environment, the resources, and their interactions [61]. These processes are illustrated in one way in figure 3.2, and again in figure 3.3 on page 57.

The reflective capabilities provide a great deal of the information and processes that will be needed by the human developers for their monitoring and evaluation of the OC system, including relatively detailed models of what the system perceives and knows about its goals, state, environment, and options.

3.3.4 Active experimentation

Traditional engineered systems are designed to respond to changes in conditions, but biological systems exhibit a much more active style. For example, engineered systems often take advantage of relatively stable conditions by shutting off many of their subsystems to conserve energy. Biological systems, on the other hand, present a much more complex approach to the utilization of their resources: they devote some of their capacity to actively experiment with their environment, their capabilities, and their limitations. That is, biological systems perform a much more complex overall optimization strategy in choosing how to respond that recognizes the potential of self-modifying systems to find new or alternative strategies that expand its possibilities so

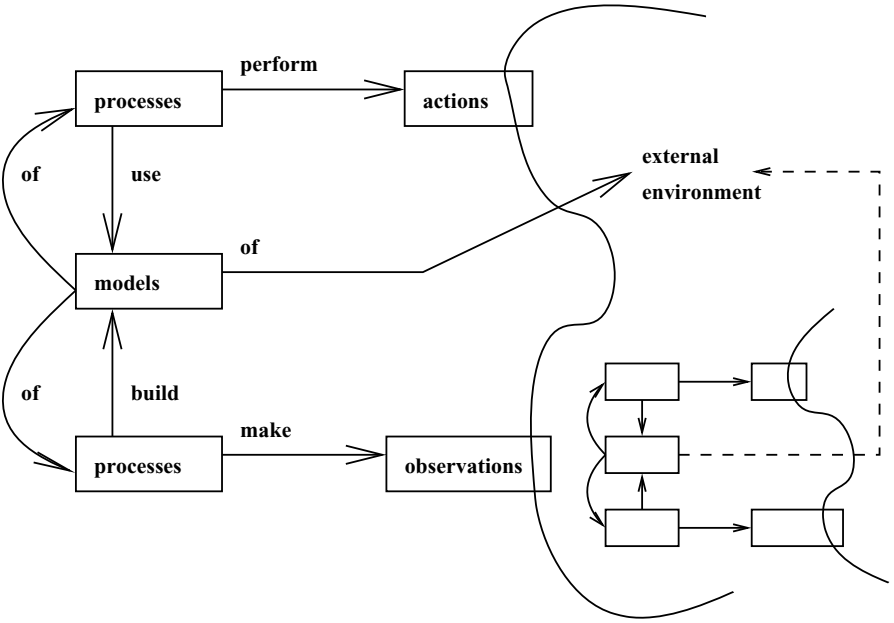


Fig. 3.3. Reflective system in environment

that it will have more options in the next "emergency". Also, if the system is already active, it may have some operating processes that can be adjusted to provide a very rapid response while other capabilities are recruited and the overall response is adjusted on a longer time scale. That is, if the system is already active, it doesn't have to start its response from scratch.

Active experimentation can be seen in part as a calibration effort in the sense that the system chooses to repeat previous actions in what it hypothesizes are similar conditions, or to perform similar actions in what it hypothesizes are the same conditions, evaluating the results in order to search for correlations and differences. Such active experimentation also provides opportunities to test the limitations of capabilities in safe situations (e.g., animal play) so that emergency responses can whenever possible be accomplished within those limitations.

Another use of active exploration is to improve models of the local conditions. Examples of this type of active exploration are well documented in the ethological literature, where an animal essentially moves randomly about its environment, discovering and building a cognitive map of features of that environment that are relevant to its size, abilities, etc.. These features can later become vital for rapid responsiveness. For example, we may observe that a lizard darts into a hole at our approach. The speed of this response is possible

because of the lizard's prior knowledge of the location of the hole and that it is of appropriate size and shape for a refuge.

In another application of active experimentation, which we have called continual contemplation, the same approach is applied to the mapping and organization of the internal states and capabilities of a system. We have previously described continual contemplation as "continual exploratory data analysis not only on external or domain knowledge introduced to the system through sensors or data sources, but also continual exploratory data analysis on the system's own state and its own use of its resources as it attempts to support user requests and to solve problems." [9]

Finally, active experimentation can be applied to all levels of system processes, up to and including its language and reasoning levels. These experiments make use not only of its own capabilities, but artifacts that it may have built in its environment. For example, human beings are using computer systems to expand our possibility space by leveraging the particular capabilities of computers to perform certain kind of operations faster, more uniformly, or more often than we do.

Each of the previous engineering challenges yields system capabilities that are utilized in active experimentation: the permissive application of generative processes, the ability to measure and evaluate in new ways, reflection that gives the system access to its own structure, language processes that can express and interpret new models, variables, and control strategies, and the ability to implement multiple responses at various scales and hierarchical levels. Biological systems seem to demonstrate that using these capabilities even when they are not immediately needed to respond to current conditions has long-term advantages. That is, the potential future efficiencies gained from discovering alternative strategies, testing correlations and limits, mapping and synthesizing information on local conditions and system capabilities, and shaping operating processes rather than starting new processes in response to an emergency all seem to give biological systems an overall benefit that outweighs the additional expenditure of resources.

3.3.5 Situational awareness and context modeling

Because reactive planning and response is not always fast enough, the system can gain a great advantage by advance planning. It must be remembered that almost all advance planning is not used, although it is clearly not useless because planned and rehearsed responses are much faster than new reactions. This means that a lot of advance planning is needed, much more than reactive planning, with the corresponding advantage in viability. However, in order to do advance planning, it is necessary to construct rich enough models of the "niche" for the system, or in our case, the operational context of the performing OC system. Because the OC system is not really built from the dynamics of the world, but through our view of it, it is not sufficient only to give it models and model-building capabilities based on our current view

of the environment and the composition of the system. Rather, the system will critically depend on the instrumentation we discussed earlier, as well as reflection, evaluation, and active experimentation capabilities that test the adequacy of its models within the current environment. One of the challenges, clearly exemplified by von Uexküll's story of the crab discussed earlier, is that under different modes of operation and with different goals, the same environment may in fact mean very different things.

Thus the context models for the operational environment are not completely predetermined but must rather be constructed with more attention paid to those parts of the environment needed for a given activity and with the appropriate "hooks" for monitoring the essential parameters determinable from that particular environment and relevant for the system at that time. In other words, as part of an action plan in its operational environment, a system will be dynamically recruiting not just the components to do things, but also the sensors that can provide the feedback, the analysis processes that can assess the feedback within its operational context, and the processes that reason about the feedback for its implications on the world and on the goals or performance of the system.

3.4 Representation and language

The ability of a system to use representations or even systems of representations, such as language, provides enormous advantages to the system in its capabilities. As we will show these advantages are so profound that one sees the use of representations occurring very very early in the development of living organisms. In this part of the chapter, we will first describe this early development of representation in biological systems, and then relate the use of representations and language to several critical capabilities within an OC system, e.g., representing goals and negotiation with other reasoning systems. In the last part of this section, we will describe how an OC system might start to build up its own set of representations and meaningful terminology, drawing on early work in Artificial Intelligence and describe why these capabilities are so critical to allowing us, the developers and users of an OC system, to continually monitor and shape the behavior, goals, and results of an OC system.

3.4.1 Representation in biological systems

It has not been widely appreciated in the computing community until recently just how complex biological systems are [4, 13]. In this subsection, we replay a compelling (at least for us) argument [10] that representations are the key to biological flexibility, and that they occur starting with the very smallest of animals.

Earlier, we presented evidence that unicellular animals have a rich repertoire of behaviors resulting from the coordination of body parts and internal structures. This internal coordination requires communication among the internal structures. Tomkins's [82] model of biological regulation assumes the use of internal symbols in unicellular and multicellular animals. If we ignore for a moment the biochemical details, his argument on the evolution of biological regulation is elegantly straightforward: even "ancient molecular assemblages" possessed cellular properties capable of self-replication. Nucleic acid and protein synthesis are endergonic reactions; hence primordial cells were required to capture energy from the environment. However, changes in the environment that diminished the supply of monomeric units necessary to polymer synthesis or altered the formation of adenosine tri-phosphate (ATP, an essential component of energy management and metabolism) were probably lethal. Therefore, survival would require regulatory mechanisms that maintain a relatively constant intracellular environment.

Tomkins divides this biological regulation into two modes. In simple regulation there is a direct chemical relationship between the "regulatory effector molecules" and their effects. As examples, he cites enzyme induction, feedback inhibition of enzyme activity, and the repression of enzyme biosynthesis. The critical point here is that in simple regulation, the control of the internal environment is tenuous at best, since the regulatory molecules are themselves important metabolic intermediaries. Therefore the animal's internal environment is still closely tied to the availability of essential nutrients. In complex regulation, there are metabolic "symbols" and "domains". To quote Tomkins [82, page 761], "The term *symbol* refers to a specific intracellular effector molecule which accumulates when a cell is exposed to a particular environment." As two examples, he cites adenosine 3'5'- cyclic monophosphate (cAMP), which in most microorganisms is a symbol of carbon source depletion, and guanosine 5'-diphosphate 3'- diphosphate (ppGpp), which is a symbol of nitrogen or amino acid deficiency. Importantly, "metabolic symbols need bear no structural relationship to the molecules that promote their accumulation in a nutritional or metabolic crisis ... cyclic AMP is not a chemical analog of glucose." [82, page 761] Tomkins also points out that metabolic lability is another attribute of intracellular symbols that allows their concentrations to fluctuate quickly in response to environmental changes. However, note that this lability is different from the troublesome lability of the simple regulation mechanisms. In the case of simple regulation, since the regulatory molecules are themselves metabolic intermediaries, they (and hence the internal environment) will fluctuate in a direct manner according to the supply of external nutrients and conditions. However, in the case of complex regulation, the symbols will respond rapidly to the external environment, leaving protected for some time the metabolic processes they control. This protected time is exactly the time in which the organism has the chance to make some adaptive response to the environment (e.g., swim away from the carbon-depleted region), and this, it turns out, is exactly what bacteria do. For example, carbon-starved *Escherichia coli* develop

flagella, which allow the bacteria to be motile; cAMP is critical to the development of flagella. By incorporating a symbol "level" the animal gains time in which it can protect its metabolic processes from external conditions. In Tomkins's examples of the necessity of cAMP to the development of flagella in *E.coli*, we see that the effects controlled by the symbol are not all metabolic but also include adaptive behavioral responses that will protect the metabolic processes. He also points out that many symbols may share in the control of a given process.

Later he extends his argument from single cells to multicellular animals and uses the slime mold, *Dictyostelium discodium*, as a model of transition of intracellular symbols to intercellular symbol use. In the slime mold, the cells exist as independent myxamoebas until starved. At this point, cAMP accumulates in the cells, similarly to *E.coli*, as a symbol of carbon depletion, but unlike *E.coli*, it is also released from the cells into the external medium where it acts as the attractant that causes myxamoebas to aggregate into one multicellular slime mold. As Tomkins states [82, page 762], "Cyclic AMP thus acts in these organisms both as an intracellular symbol of starvation and as a hormone which carries this metabolic information from one cell to another." But, as noted earlier, cAMP is labile and therefore, Tomkins argues, is not suitable for the long distance required for intercellular communication in large metazoa. He proposes that hormones, more stable chemical compounds, took over the role. As he emphasizes, the process in intercellular communication always begins and ends in the internal primary codes of individual cells.

Just as in the case of internal communication processes of unicellular animals, the intercellular communication processes of multicellular organisms are symbol-based. Note how these internal communication processes, in both unicellular and multicellular animals, make possible behavior or coordinated goal-directed movements. Movement is fundamentally a cooperative phenomenon, requiring communication among the organism's parts. As we saw in the example of motility in *E.coli*, even the most primitive movement is controlled and mediated by the use of symbols.

In other words, in the unicellular animal, we have a collection of symbols, like cAMP, which together with the way they affect the processes under their control and the way this collection of symbols affect each other, constitutes a primitive brain without nerves. This primitive brain without nerves is elaborated in multicellular animals in two ways. (1) The labile symbols of the unicellular animal are replaced by hormones, which are more stable chemical symbols, and by nerves, which provide more specific routes of information than chemical diffusion. (2) Layers of symbols develop to the point that "domain" (in Tomkin's sense) becomes not a set of body processes but rather a set of brain processes.

As stated in [10, page 918],

> An increased use of symbols disassociates the intracellular processes of unicellular organisms from the environment. This means that an event

> which occurs in the receptive space of the organism does not produce an immediate response. What it produces is an internal reaction that symbolizes the event in the environment. These symbols of external events then become part of the internal processes of the organism. The more an organism has the ability to symbolize external events and the greater its capacity to manipulate those symbols internally, the more it is freed from non-adaptive, direct responses to fluxes in the energy and matter surrounding it. It begins to have the capability to organize delayed actions, which give it the freedom to plan, simulate, and act when its own internal processes deem it appropriate; such actions can take place at greater and greater distances in time and space from the initial external event.

This "disassociation" is critical to representation, but also to the reflective processes described earlier. The separation enabled by the disassociation of symbols allows the time and the freedom from external "realities" within which the system can essentially "simulate" or do what-ifs and other types of reasoning about possibilities, e.g., different behaviors, different contexts, different results, before committing itself to the actual energy to perform actions.

Clearly, there is a careful and context-dependent trade-off between the *timeliness* of rapid responsiveness and the *timelessness* permitted by representation for reasoning and reflection. Biological systems typically begin to address this trade-off by taking advantage of the multiple layers of their responsiveness; while some layers of a system are doing immediate actions, other layers are doing longer term reasoning processes. This is clearly seen in the crayfish emergency "tail flip", which is one of the most rapid responses known: it is just 10 ms from the stimulation of the flight response to an undirected flip. Meanwhile, a slower system within the crayfish is carefully figuring out a trajectory for swimming away from the possible threat that led to the emergency response.

This trade-off between timeliness and timelessness is also seen in the generative processes, discussed in section 3.2.3.2, where reasoning and reflection are part of the system's ability to assess and alter its rapid construction and maintenance of current configurations or assemblages of components. That is, the generation of assemblages does not have to be perfect, but rather because of adaptive processes can be generated and then fine-tuned rapidly, depending on the changing circumstances and changing needs of the system. These adaptive processes clearly require reflection and reasoning. This point is especially relevant to "substitutability", which is the ability of a system to rapidly adapt by changing the processes and structures used to accomplish a given purpose or goal.

This brings us to the key topic of representing purpose, goals and meaning in OC systems. These terms are not intended to imply anything about consciousness or even awareness on the part of the OC system. Rather, through

explicit or implicit means the system must represent goals in order to evaluate the results of actions.

3.4.2 Purpose, goals and meaning

Part of the adaptive behavior of biological systems is seen in the sophisticated capabilities that animals have for developing and encoding "meaningful" representations about their environment and their own states, and for developing processes using these representations to plan actions that achieve desirable states. These "desirable states" will always be situated. In other words, they will always include a combination of features in the world in relationship to features exhibited within the biological system. For example, in Tomkins' examples above the feature of carbon availability in the external world was being represented in single-cell animals, as well as their own state of energy availability. These desirable states are the lowest level of "purpose" and "goals", and as pointed out in the section above, like other representations can become increasingly separated in time and space from external events. That is they can become increasingly "abstract", and with abstraction, impact more diverse parts of the total system.

Animals show sophisticated abilities to represent and process the actions needed to support diverse goals. One such ability is to satisfy multiple goals with a single course of action. This is called merging and was studied by one of the authors because of its possible use as a source of variation to help explain the amazing flexibility of responsiveness in biological systems. But it also has profound implications for the underlying adaptive decision processes available to animal systems and the way that goals are represented. One of the consequences of merging is that there can be multiple goals for any action and multiple actions for any goal. A given instance of behavior can reflect several motivations and work toward several goals at once.

Contrary to the usual emphasis in behavioral studies, in which an animal must choose between mutually exclusive acts, an animal in nature is rarely in the situation where it must engage in one behavior to the exclusion of other behaviors. Rather, an animal's movement frequently shows "behavioral merging", in which several motivational goals and action patterns are combined into one coherent pattern. In studying the merging of feeding and aggression behaviors in the lizard [4], an animal noted for the rigidity of its behavioral patterns, Bellman found that when elements of feeding and aggression conflicted, other elements were selected and substituted, so that, overall, both feeding and aggressive patterns were combined into one fluid behavioral sequence. The behavioral sequence resulting from merging points to a particular type of flexibility in a movement system. A specific movement pattern can subserve a number of goals. If this is so, then a specific movement pattern is not necessarily linked to one goal any more than to any other goal (although there may be some kind of weighting, so that a given behavior is most often associated with one particular goal). This implies that a movement is not

"released" as a necessary consequence of the occurrence of a particular motivational goal; rather it is "recruited" to serve that goal. Furthermore, from behavioral merging, we see that a whole action pattern need not be recruited but only those elements best fitting the circumstances.

Clearly, these abilities have advantages both in terms of expended effort and in terms of rapidity of response. However, as in many other qualities of biological systems, it is efficiency of a peculiar type. It is highly efficient in allowing future adaptiveness and in robustness, but not efficient or optimal for any single given set of actions. Part of the reason for this style of efficiency may have much to do with the type of complex multi-criterion optimization within a rapidly changing environment required by biological systems. That is, in conventional man-made systems, which are engineered to optimize their performance within carefully specified environments, one can develop planning processes that optimize the performance given a fairly fixed set of criteria. The emphasis can thus be on the efficiency of the fixed course of action. In biological systems, and in the systems we are trying to invent in OC, the complexity of the system's interactions and requirements and the changing environment require an emphasis on the ability to rapidly adapt and hence to change course or replan. This adaptiveness requires all sorts of properties that in single-purpose systems are redundant or excessive.

We have already noted in this paper many capabilities which support the ability of a system to change course and replan, including generative and opportunistic processes, reflection, and active experimentation. However, in order to develop appropriate evaluation methods for choosing the best solutions among combinatorial possibilities and to deal with the synthesis of the information we will have in the necessarily explicit models of an OC system, we will also need many new ideas on what we mean by optimization or even satisficing in these systems. As difficult as it will be to represent goals, it is even more difficult to state the evaluation criteria that will determine "goodness" and "fitness" for the OC system.

One shift will be away from any long-term or overall optimization or satisficing and toward strategies of local and short-term optimization with methods designed to rapidly capture and summarize the wide-spread impact of decisions. Having layers of relationships will help because small continuous change at one layer can have much wider and diverse impacts that can be monitored for from the standpoint of different levels of recruited modules.

It should be noted here that adaptation is not like optimization, and is not usefully implemented by optimization. It is always just satisficing, not optimizing, and usually the time constraints on decision making mean that even formal satisficing is not possible either, so some combination of experience and guessing is needed.

3.4.3 Negotiation with other reasoning systems

So far we have been emphasizing the importance of representations in the ability of animal systems to regulate themselves and then as a continuation of that self-regulation, to represent and reason about their goals and to adaptively plan actions in a dynamic environment. As is hinted at in Tomkins's examples of the slime mold, the same set of internal self-regulatory symbols that are used intercellularly to coordinate the actions of populations of cells can also be, by many mechanisms, made visible to other organisms in order to coordinate their behavior at the population level. Hence the symbols excreted, secreted, vocalized, enacted, etc., allow animals to coordinate their mating, hunting, fighting for territory, learning, and many many other types of needs for communication and coordination. That the animal kingdom displays such diversity in the types of symbols, the reasons for symbols, and the mechanisms for conveying symbols speaks to the enormous importance of shared representation in adaptive complex systems. Interestingly, in the animals with increasingly complex reasoning capabilities there is a correspondingly increasing complexity of communication capabilities, including those displaying the nuances of emotional state (which contains a wealth of information about the motivational state and likely goals and intentions) and social needs for negotiation and coordination. It is our belief that OC systems, because of their complexity and because of our need to monitor and shape their behaviors, require similarly sophisticated negotiation and coordination capabilities.

As we describe the needs of representation and communication in OC systems, it is important to note that in a complex system:

- there is a time delay, both for making up or choosing the information to convey, and for getting the information out; hence the need for some autonomy, and
- there is often a difficulty in characterizing complex states; think for a moment of how difficult it is for you, as a human, to describe your unobservable symptoms to a medical doctor.

These delays are one source of emergence: if there are arbitrary time delays in a communication process, or other feedback process, then that process can exhibit instabilities and emergences. However, communication is so important to both the individual and the group that there are multiple, overlapping, and even redundant symbols, representations, and modalities in order to ensure that critical information is conveyed.

One of the advantages we have in biological systems is the situatedness of our systems in the physics of the world and the common evolution among members of a species and even among all mammals. We understand because we are. Much of our ability to reason from the outside about another human's state and meanings has a lot to do with our commonalities as humans. For example, consider the deep problems that autistics have in communicating as an example of how even minor human variations can have vast impact

on our abilities to negotiate and communicate. On the other hand, there are some vocalizations that, like reflexes, act as immediate emergency responses: it is a curious feature, especially in mammals, that distress calls and other broadcasts are occasionally so important that they can be recognizable across species.

In OC systems we will be building an "alien" system with no shared evolution, with little shared constraining world (the dynamics, etc., will be different), and, potentially, little overlap in our operating environments. As we develop and use language in an OC system, we might want to consider how to develop co-evolutionary strategies, recognizing that both we and our systems will be changed by that co-evolution.

The important question here is how we will create livable systems. We certainly need to understand how an OC system negotiates with its human engineers and builds up a common set of symbols, etc., so that its communication may be understood and its effects will be appropriate. Here there are two meanings of "livable": for the system to be able to live in its environment and for us to be able to live with it. We mean both.

3.4.4 Use of language

An OC system needs to model its surroundings and its own behavior for self-assessment and self-improvement. Since the designers cannot know everything about the system's environment and development, the system will have to create new models or modify existing ones. It will therefore have to have ways to assess the efficacy of its models, and change them as it deems necessary.

More fundamentally, the languages in which the models are written may not be adequate for all development paths, so the system will also have to create new kinds of models and new languages in which to define them, and sometimes re-express its older models and processes in the new language.

We write "languages" in the plural because we do not believe that any one modeling language or paradigm can be sufficient, even in principle, to model all relevant or important aspects of a complex environment [14]. We therefore advocate the use of a collection of "little languages", instead of trying to fit everything into one big one. Of course, the collection of "little languages" has multiple underlying assumptions, and this multiplicity requires some integration process, but we have developed an integration mechanism that is well-suited to complex system integration, called "Wrappings", described in section 3.4.6 below.

We have argued here that the creation and use of language internal to the system is fundamental to the success of OC systems in complex environments. The study of this symbolic aspect of systems design and operation is called "Computational Semiotics", and it lies at the intersection of the edges of mathematics, linguistics, philosophy, logic, and computation [43, 44]. It is about the creation and use of symbol systems by constructed complex systems,

but we are trying to push it much farther in the direction of interaction with human language, since the system has to tell us about itself.

A nice introduction to the general topic can be found in [25] (if you can ignore the intrusions of justifiably annoyed comments about transformational grammar), with some comparisons to the early writings in language [76] and philosophy [85, 86]. There are also other approaches based in logic [79] and computation [70]. Another description from a different viewpoint can be found in [17].

Our attention to the use of language includes whatever programming or specification notations are to be used, since they are almost always too precise for what the designer knows about a system, so they require the designers to make too many decisions before it is possible to know enough to make those decisions properly [77].

We intend that the system will help the designers create the language, by operating for a while, so that the system can know enough to make some good choices, and that it can present enough information to the designers so that they can make other good choices.

In the most general terms, we can describe the operation of such a system as follows:

- system observes external and internal behavior
  - developers must provide initial languages
  - system use languages to record these observations
  - system assesses the adequacy of its own languages
  - system changes the languages or invents new ones as necessary
  - the process cycles back to the system's use of languages
- system creates models
  - developers must provide initial notations
  - system uses notations to record these models
  - system assesses the adequacy of the notations
  - system changes the notations or invents new ones as necessary
  - the process cycles back to the system's use of notations
- system inherits or creates goals
  - developers must provide initial goals
  - system reasons about the models in pursuit of its goals
  - system assesses the adequacy and consistency of the goals
  - system changes or replaces the goals as necessary, according to the results of negotiations with developers
  - the process cycles back to the system's use of goals

To describe these processes in more detail, and to explain how we expect these systems to work, we start with our emphasis on context, then proceed to symbol systems, language, and models. We end with a discussion of our progress and prospects for OC systems.

3.4.4.1 Communication and cooperation

A significant aspect of how we will cooperate with our complex computing systems is our ability to communicate with them and their ability to communicate with us. To that end, and since we are no longer expecting to design every aspect of these systems, they will have to be able to use symbols of their own devising, created to represent some meaning significant to them, which must also be conveyed to us. There are examples of early artificial intelligence systems that generate symbols to represent things that they have learned about the world and their own internal states [31, 72].

These systems will have to make new models of both their environment (context) and their internal processes and states, evaluate the effectiveness of those models, and revise them or build new models again as necessary. These systems will also have to communicate their internal models to us, so we can monitor their actions and predict their expected activity to ensure it is in line with our intentions for the system.

This is why we emphasize the semiotics of these systems; if we can understand enough about the processes of language formation and use, we can design systems that will be able to explain themselves to us. To do that, we need to understand the symbol systems, what they are used for, how they are defined (whether by the designer or internally), how a system can evaluate them in the context in which they are being used, and how a system can change its symbols appropriately and tell us what it did.

3.4.5 Symbol systems and representational mechanisms

We start with a description of our approach to representational systems, and show how engineered systems can be expected to create and use them.

It may seem as though we are starting from "too far back" in the design process, namely before the domain is well understood, but in our opinion we must start there, because the different philosophy on adaptation that we have developed above requires fundamental changes in the nature of our computing systems and devices, and the development processes that lead to them.

Besides, it is our opinion that every complex system design process actually starts (and usually finishes) before the domain is well understood, often long before, even though that fact is not generally known in advance (though we claim that it could and should be expected).

For us, a representational mechanism is the same as a modeling mechanism. That is, any computational scheme that derives a computationally accessible object (or process) to represent some phenomenon of interest (either external to the system or internal) is a representational mechanism.

The modeling scheme is better if the model is better. The better models capture more properties of the phenomenon (or at least more of the properties important to the modeler), and the representational mechanism is better when it can represent more phenomena of interest to the system.

There must be processes that identify the phenomena and create the models, processes that can transport the resulting models in time (memory) or space (communication), and processes that analyze the models for making decisions or convert them to other forms for further analysis (translation).

Symbol systems are one kind of representational mechanism, chosen for simplicity and ease of computation. A symbol system consists of a finite set of basic symbols and a finite set of combination methods. The analogue is a phrase-structure grammar with constraints [26].

There may be different types of symbols and structures, and different kinds of combination methods, but it is important that they be finitary, which means that each combination method can only combine a fixed number of structures at each use (each combination method can have various kinds of restrictions on what structures it can combine, which we take to have the power of context-sensitive grammars). All structures in a symbol system can be pictured as finite trees, and the combination methods are ways to combine trees into larger ones.

The "get-stuck" theorems tell us that the systems need to be able to evaluate and adjust not only their models, but their basic symbol systems and modeling mechanisms [53, 47]. Basically, for any given symbol system, we can compute the maximum number of expressions of each length, and then argue that adequate modeling of more complex environments eventually leads to expressions that are too large to process quickly enough: the system "gets stuck", and the only way out is to change the symbol system.

We also want to make it very clear that symbol systems are just one kind of representational mechanism, one that we have chosen because they are easy to use and analyze, not because they are necessarily the only or even the best choice for all modeling problems.

3.4.5.1 Language formation

As the system operates in a complex environment, it gathers information about what choices it has made and what activities it has seen in the environment and in its own internal operation. It is our intention to have these systems create private descriptive language for their own use, and explain it to their human users. To that end, the systems have to have many more empirical modeling capabilities than usual. As a first step, one can do very simple, straightforward syntactic analyses of language and language use [38]. With such methods, we have shown how language formation might occur, with identifying common or replicated patterns in the structures, the processes, or in the relationships [48].

Here the system can do some empirical invention. It can accumulate commonalities and replications of structure and process, in context. It can accumulate commonalities and replications of descriptions and relationships, and it can assign symbols to those clusters and recognize them when they occur again, gradually describing more over time.

Digital computers can only do three things: move and copy data, compare data, and interpret limited range models of arithmetic (computers do not do arithmetic). All of these operations are entirely syntactic. In a sense, we are trying to make these devices compute with semantics, which means defining syntactic representations of semantics relationships, and computing up the meaning hierarchy from data through information to knowledge [39].

For example, there is a kind of abstraction that is part of "continual contemplation" in reflective systems [46]. Any process or any structure can be decomposed, the parts abstracted with an attached context of their use in the combination, and then reassembled and reintegrated computationally. The parts then become process or structure components that can be put together in other ways, with other components, according to their Wrappings. This kind of abstraction occurs very often in mathematics, as proof steps become methods and sometimes subjects in their own right.

3.4.5.2 Model evaluation

After a system has built models, then it needs methods to assess them, and, if necessary, replace them. In particular, a system needs to be able to determine that a model is inadequate.

This process is called model-deficiency analysis, and it proceeds from two sources of information: intentional goals and observed behaviors, and most particularly, from places in which the behaviors do not match the goals. These will be described in terminology that is internal to the system, which also means that the language used must be adequate to describe them. This is another force towards development by the system of multiple little languages and also of better languages.

The notion of allowable variation, when applied to language, means to us that the system should use several different sets of foundations simultaneously, that is that the internal languages occur at different resolutions, with different local contexts and different interactions, so that their efficacy for particular problems can be compared. This choice is already known to be important to simulation systems [20, 68], at least at the level of temporal and spatial resolution. We assert that it is equally important in other domains.

Computational reflection offers important advantages in the evaluation of models. However, because reflection will explicitly represent system processes and structures, we encounter an interesting dualism. As soon as processes are made explicit, they become descriptive structures, and as soon as structures have interpreters, they become processes. This dual view allows both kinds of things, that is, both descriptive structures and processors, to be processed in different ways for different purposes.

3.4.6 Progress and prospects

In this subsection, we describe what we can do now in terms of realizing our (admittedly extremely ambitious) concepts for representations, symbol systems, languages, and model construction and evaluation.

Once we find the right mechanisms, we can implement these systems using our Wrapping approach to integration infrastructure. Wrappings also provide several useful notions for implementing and combining the little languages noted above. The *Problem Posing* programming paradigm [51] separates the *information service requests* in a program from the *information service providers*. It is always clear which is intended, and the distinction is known to the compilers and interpreters of the notation. We have shown that it applies to programming notations from all of the major programming paradigms: imperative, declarative, relational (constraint), functional (applicative), object (message), and others.

For example, function definitions are information service providers and function calls are information service requests. We normally associate the two by using the same name, but the names are in completely different name spaces. The Problem Posing interpretation allows us to break the direct association and reconnect them in much more interesting and flexible ways. We define *problems* as information service requests, and *resources* as information service providers, so that we can treat them separately. In particular, we can then study the notion of a *problem space* as an explicit representation of the goals and purposes of various processes in a particular application domain, without needing to specify *a priori* how those problems are to be addressed [14, 54].

One of the more interesting ways to connect problems to resources is with *Knowledge-Based Polymorphism*, that is, with a knowledge base that maps problems into resource uses in context. Our Wrapping approach to integration infrastructure takes this mapping as fundamental.

We start with the widely observed notion that declarative knowledge has the advantage of being analyzable. But declarative knowledge does not *do* anything; it needs an interpreter, and we need to make those interpreters explicit for study. The Wrapping approach is based on these two fundamental aspects of computation in constructed complex systems, the descriptions and the interpreters:

- *Wrapping Knowledge Bases* (WKBs) describe the uses of all resources, not just how, but also whether, when, and why to use a particular resource in a particular context.
- *Problem Managers* (PMs) interpret the WKBs to select and apply resources. PMs are also resources, are also Wrapped, and therefore also selectable.

Such a system has no privileged resources at all. Any part of the system can be replaced (actually superseded) by a corresponding provided part. This

flexibility allows all of the integration processes to be studied in the same system.

We mentioned above in the language summary that we expect the OC system to use a multiplicity of languages, and hence the Wrappings will have to integrate a multiplicity of languages. What is needed is to interpret the models or other situations in which language fragments are used as problems requiring certain semantic information, and the language fragments as resources providing some semantic information, and the Wrappings as a knowledge-based connection from needed to provided semantics.

We have described a Wrapping-based architecture for systems that have models of themselves [52, 61], which they can use to examine and change their own behavior. These systems have descriptive models of every process in them, and interpreters to produce that process behavior from the descriptions. The interpreters are also processes, and also have descriptions, so the system is completely self-describing.

We have built systems using Wrappings for small but difficult integrations, as well as for larger systems. One example was a system with 48 resources for evaluating the effects of new technology insertion into a situation management and rapid response system. The general point being made here is that even fairly complicated systems can be put together rapidly with Wrappings, provided that they and their expected behavior are well-understood. We have also built systems with models of (some of) their own behavior, and systems that create their own symbol systems (a word identification program using grammatical inference [3]), but not (yet) reflective ones. We have built systems that re-express parts of themselves, but so far only the descriptions, not (yet) the processes.

These applications lead us to believe that we can implement some of the appropriate computational resources for OC systems, including some version of the following capabilities, which will allow systems to

- manage their own computational processes. Wrappings identify the resources they have and the classes of problems they address.
- manage their own modeling processes [5, 41].
- are partially self-modeling and self-modifying, as shown in the previous section.
- manage their own symbol systems and invent new ones.

The hardest part of this approach is the symbol creation necessary for this kind of constructive semantics, that is, how a system can represent the connections between purely syntactic data (which is all that computers can contain) and semantic meanings (connections to the phenomena).

That boundary is the fundamental phenomenon in the use of symbols, often called the symbol grounding problem [29], which has generated a large amount of discussion, including claims that it is already solved [80]. In our view, for OC systems, the designers make the first choices of symbols, and that reduces the importance of this problem.

In any case, this is where all symbolic processing starts: feedback loops and other stable structures and correlations lead to symbols, e.g., a system moves and it sees a motion, or it speaks and it hears a sound, or any of a number of connections from simple outputs to multiple inputs.

In the absence of a good solution for this (hard) problem, all is not lost. We can still proceed under the assumption that our system has a fairly limited set of different kinds of actions that it can perform, and a limited set of different kinds of events or activities that it can recognize.

We will not expect the system to create new kinds of external interfaces, though it may be able to create new instances of many kinds of interface. We will expect the developers to provide a rich set of initial interfaces, so that the system has enough information to study its environment and enough capability to affect it appropriately.

The key to building these linguistically capable systems is to provide the right set of language producing and modifying mechanisms. Empirical statistics has discovered a large set of notations and methods that are useful in this regard, for representing and understanding large sets of correlated time series, and we expect each application domain to have its own special methods also. More methods and more different kinds of methods are needed, though, and we expect that developing the new methods needed for this detailed level of language use within the system is still hard. Integrating these methods is a challenge well addressed by Wrappings, and model-deficiency analysis holds great promise for future developments.

3.5 Conclusions

In this chapter we considered three major and mutually reinforcing types of developments in a successful adaptive system: The first is creating the "possibility space". This possibility space includes much more than the history and development of an individual or even a population of individuals; in fact, for biological systems the possibilities start in the physics of the environment which will become the system's ecological niche. The second set of developments could be thought of as creating processes that both enlarge and constrain the shape of this possibility space. The last set of developments is the more traditional concern of adaptive systems research: the control processes that enable the system to navigate through the possibility space. That is, given its goals and its current state within the possibility space, what exactly can the system perceive (of its possibilities), what can the system control, and what can it do.

In the case of each of these major sets of developments, we first described what we consider to be biological versions of these processes, and then what that might imply in terms of engineering OC analogs of such processes. But in addition to creating analogies to existing biological processes, we also discussed some unique challenges for OC systems; the greatest of which is that

these systems must always be accessible, monitorable and coordinated with our goals and intentions for the systems. This implied to us that OC systems require sophisticated instrumentation, self-monitoring and reflection capabilities as well as the ability to represent their states to us, communicate and negotiate with us, and hence share the development of its control and organization with us.

Aside from the small progress we have made, particularly on reflective processes described in the previous section, we feel that OC in order to make progress critically needs to develop several capabilities as a community of OC researchers. That is, in the systems engineering challenges we discussed the issues involved in exploring the possibility space in OC systems, and some of the strategies that a biological system uses to do its explorations. Two of the foremost strategies of biological systems are 1) to use a population of individuals to explore formidably large possibility spaces and do so with active experimentation, and 2) provide the means for communication and coordination among the entire group so that the perceptions and experience of individuals can be combined. So can we not in fact treat the community as a population of organisms determined to explore the possibility space of OC systems? And could we not, with coordinated efforts, act like an active experimentation process, carefully correlating across our different experiences? Of course in order to do this we need to deal with the reproducibility of our results within OC systems and we will need to work very hard on building up much better models of the goals and operational contexts for our individual demonstrations. Lastly, we would all, as a community of researchers, benefit from whatever methods are developed to give us overviews in many different ways of these complex systems.

References

1. Harold Abelson, Gerald Sussman, with Julie Sussman, *The Structure and Interpretation of Computer Programs*, Bradford Books, now MIT Press (1985)
2. Lada Adamic, "The Small-World Web", on the Web at URL "http:// www. hpl. hp. com/ research/ idl/ papers/ smallworld/ smallworldpaper. html" (availability last checked 15 July 2007)
3. Dana Angluin, Carl H. Smith, "Inductive Inference: Theory and Methods", *Computing Surveys*, Volume 15, Number 3, pp. 237-269 (September 1983)
4. Kirstie L. Bellman, *The Conflict Behavior of the Lizard, Sceloporus Occidentalis, and Its Implications for the Organization of Motor Behavior* (PhD Dissertation), 225 pages, University of California, San Diego (1979)
5. Kirstie L. Bellman, "An Approach to Integrating and Creating Flexible Software Environments Supporting the Design of Complex Systems", pp. 1101-1105 in *Proceedings of WSC'91: The 1991 Winter Simulation Conference*, 8-11 December 1991, Phoenix, Arizona (1991)
6. Kirstie L. Bellman, "Developing a Concept of Self for Constructed Autonomous Systems", pp. 693-698, Volume 2 in *Proceedings of EMCSR'2000: The 15th*

European Meeting on Cybernetics and Systems Research, Symposium on Autonomy Control: Lessons from the Emotional, 25-28 April 2000, Vienna (April 2000)
7. Kirstie L. Bellman, "Emotions: Meaningful Mappings Between the Individual and Its World", (to appear) in R. Trappl, P. Petta (eds.), *Emotions in Humans and Artifacts*, MIT Press (2001, expected)
8. Kirstie L. Bellman, "The Challenge for a New Type of Computational Semiotics: The Roles and Limitations of Diverse Representations in Virtual Worlds", *Proceedings of VWsim'01: The 2001 Virtual Worlds and Simulation Conference, WMC'2001: The 2001 SCS Western MultiConference*, 7-11 January 2001, Phoenix, SCS (2001)
9. Kirstie L. Bellman, "Peacemaker 2020: a System for Global Conflict Analysis and Resolution, a Work of Fiction and a Research Challenge" in Robert Trappl, ed., *Programming for Peace: Computer Aided Methods for International Conflict Resolution and Prevention*, Springer Dordrecht the Netherlands 2006.
10. Kirstie L. Bellman and Lou Goldberg, "Common Origin of Linguistic and Movement Abilities", *American Journal of Physiology*, Volume 246, pp. R915-R921 (1984)
11. Kirstie L. Bellman, Christopher Landauer, "Integration Science is More Than Putting Pieces Together", in *Proceedings of the 2000 IEEE Aerospace Conference (CD)*, 18-25 March 2000, Big Sky, Montana (2000)
12. Kirstie L. Bellman, Christopher Landauer, "Towards an Integration Science: The Influence of Richard Bellman on our Research", *Journal of Mathematical Analysis and Applications*, Volume 249, Number 1, pp. 3-31 (2000)
13. K. L. Bellman and D. O. Walter, "Biological Processing", *American Journal of Physiology*, Volume 246, pp. R860-R867 (1984)
14. Richard Bellman, P. Brock, "On the concepts of a problem and problem-solving", *American Mathematical Monthly*, Volume 67, pp. 119-134 (1960)
15. Patrick L. Barry, "Good Vibrations", NASA News 02 November 2001, on the web at URL "http:// science. nasa. gov/ headlines/ y2001/ ast02nov_ 1. htm" (availability last checked 02 June 2007)
16. Braitenberg, V., *On the Texture of Brains: an Introduction to Neuroanatomy for the Cybernetically Minded*, New York, Springer-Verlag, 1977, p 121.
17. Daniel Chandler, "Semiotics for Beginners", on the Web at URL "http:// www. aber. ac. uk /media /Documents /S4B" (availability last checked 02 June 2007)
18. Paul Churchland,. *Matter and Consciousness*, Cambridge, Mass.: The MIT Press, New York: Avon Books (1984)
19. Antonio Damasio, *Descartes' Error: Emotion, Reason, and the Human Brain*, New York: Avon Books (1994)
20. Paul K. Davis, "An Introduction to Variable-Resolution Modeling and Cross-Resolution Model Connection", pages 1-43 in [21]; revised version in RAND report R-4252-DARPA, RAND Corp. (1993)
21. Paul K. Davis, Richard Hillestad (eds.), *Proceedings of DARPA Variable-Resolution Modeling Conference*, 5-6 May 1992, Herndon, Virginia, Conference Proceedings CF-103-DARPA, RAND Corp. (March 1993)
22. "DDDAS: Dynamic Data Driven Applications Systems", on the web at URL *http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=13511* (availability last checked 05 July 2007)
23. Michael W. Deem, "Mathematical adventures in biology", *Physics Today*, January 2007, pp. 42-47.

24. Bertrand Delamotte, "A hint of renormalization", *Am. J. Phys.* vol. 72 (2004) 170-184
25. John M. Ellis, *Language, Thought, and Logic (Rethinking Theory)*, Northwestern University Press (1993)
26. J. M. Foster, *Automatic Syntactic Analysis*, American Elsevier (1970)
27. Martin Gardner, "The fantastic combinations of John Conway's new solitaire game 'life' ", *Scientific American*, Vol. 223, pp. 120-123 (October 1970)
28. A. Garfinkel, M.L. Spano, W.L. Ditto, J.N. Weiss. "Controlling cardiac chaos", Science 257 (5074), pp. 1230-1235 (Aug 28 1992)
29. Steven Harnad, "The Symbol Grounding Problem", *Physica D*, Vol. 42, pp. 335-346 (1990) on the web at URL "http:// www. ecs. soton. ac. uk/ harnad/ Papers/ Harnad/ harnad90. sgproblem. html" (availability last checked 02 June 2007)
30. Jennings, H. S., *Behavior of the Lower Organisms*, Bloomington, In., Indiana Univ Press 1906 (Republished with new foreword by C. King, 1976)
31. L. Kaelbling, M. Littman, and A. Cassandra, "Planning and Acting in Partially Observable Stochastic Domains", *Artificial Intelligence* 101(1-2): 99-134, 1988.
32. Catriona M. Kennedy, "A Conceptual Foundation for Autonomous Learning in Unforeseen Situations", Tech. Rpt. WV-98-01, Dresden Univ. Technology (1998); also on the Web at URL "http:// citeseer .ist .psu .edu /kennedy98conceptual .html" with abstract at URL "http:// citeseer .ist .psu .edu /kennedy98conceptual .html" (availability last checked 02 June 2007)
33. Catriona M. Kennedy, "Distributed Reflective Architectures for Adjustable Autonomy", in David Kortenkamp, Gregory Dorais, Karen L. Myers (eds.), *Proceedings of IJCAI-99 Workshop on Adjustable Autonomy Systems*, 1 August 1999, Stockholm, Sweden (1999); a similar note is on the web at URL "http:// citeseer .ist .psu .edu /kennedy99distributed .pdf" with abstract at URL "http:// citeseer .ist .psu .edu /251251 .html" (availability last checked 02 June 2007)
34. Gregor Kiczales, Jim des Rivières, Daniel G. Bobrow, *The Art of the Meta-Object Protocol*, MIT Press (1991)
35. Wolfgang Köhler, *Intelligenzenprüfungen an Anthropoiden*, (1917), translation *The Mentality of Apes* (1925)
36. Kevin B. Kreitman, "Kreitman's Conjecture and the Law of Requisite Heterarchy", *Symposium on Cybernetic Factors for Socio-Economic and Management Framework*, 14th International Congress on Cybernetics, 21-25 August 1995, Namur, Belgium (1995)
37. George Lakoff, *Women, Fire, and Dangerous Things*, U. Chicago Press (1987)
38. Christopher Landauer, "Syntax is More Than You Think", pp. 801-806 in *Proceedings of ISAS'98: The 1998 International Multidisciplinary Conference on Intelligent Systems and Semiotics*, 14-17 September 1998, NIST, Gaithersburg, Maryland (1998)
39. Christopher Landauer, "Data, Information, Knowledge, Understanding: Computing Up the Meaning Hierarchy", pp. 2255-2260 in *Proceedings of SMC'98: The 1998 IEEE International Conference on Systems, Man, and Cybernetics*, 11-14 October 1998, San Diego, California (1998)
40. Christopher Landauer, "Some Measurable Characteristics of Intelligence", Paper WP 1.7.5, *Proceedings of SMC'2000: The 2000 IEEE International Conference on Systems, Man, and Cybernetics (CD)*, 8-11 October 2000, Nashville Tennessee (2000)

41. Christopher Landauer, Kirstie L. Bellman, "Knowledge-Based Integration Infrastructure for Complex Systems", *International Journal of Intelligent Control and Systems*, Volume 1, Number 1, pp. 133-153 (1996)
42. Christopher Landauer, Kirstie L. Bellman, "Integration Systems and Interaction Spaces", pp. 161-178 in *Proceedings of FroCoS'96: The First International Workshop on Frontiers of Combining Systems*, 26-29 March 1996, Munich (March 1996)
43. Christopher Landauer, Kirstie L. Bellman, "Semiotics of Constructed Complex Systems", pp. 35-40 in *Intelligent Systems: A Semiotic Perspective*, *Proceedings of the 1996 International Multidisciplinary Conference*, *Volume I: Theoretical Semiotics*, *Workshop on Intelligence in Constructed Complex Systems*, 20-23 October 1996, Gaithersburg, Maryland (1996)
44. Christopher Landauer, Kirstie L. Bellman, "Mathematics and Linguistics", pp. 153-158 in *Intelligent Systems: A Semiotic Perspective*, *Proceedings of the 1996 International Multidisciplinary Conference*, *Volume I: Theoretical Semiotics*, *Workshop on New Mathematical Foundations for Computer Science*, 20-23 October 1996, Gaithersburg, Maryland (1996)
45. Christopher Landauer, Kirstie L. Bellman, "Model-Based Simulation Design with Wrappings", pp. 169-174 in *Proceedings of OOS'97: The 1997 Object Oriented Simulation Conference*, *WMC'97: The 1997 SCS Western MultiConference*, 12-15 January 1997, Phoenix, SCS International (1997)
46. Christopher Landauer, Kirstie L. Bellman, "Computational Embodiment: Constructing Autonomous Software Systems", pp. 131-168 in *Cybernetics and Systems: An International Journal*, Volume 30, Number 2 (1999)
47. Christopher Landauer, Kirstie L. Bellman, "Situation Assessment via Computational Semiotics", pp. 712-717 in *Proceedings ISAS'98: the 1998 International Multidisciplinary Conference on Intelligent Systems and Semiotics*, 14-17 September 1998, NIST, Gaithersburg, Maryland (1998)
48. Christopher Landauer, Kirstie L. Bellman, "Language Formation in Virtual Worlds", pp. 1365-1370 in *Proceedings of SMC'98: The 1998 IEEE International Conference on Systems, Man, and Cybernetics*, 11-14 October 1998, San Diego, California (1998)
49. Christopher Landauer, Kirstie L. Bellman, "Generic Programming, Partial Evaluation, and a New Programming Paradigm", Paper etspi02 in *Proceedings of HICSS'99: The 32nd Hawaii Conference on System Sciences (CD)*, *Track III: Emerging Technologies*, *Software Process Improvement Mini-Track*, 5-8 January 1999, Maui, Hawaii (1999); revised and extended version in [51]
50. Christopher Landauer, Kirstie L. Bellman, "Computational Embodiment: Agents as Constructed Complex Systems", Chapter 11, pp. 301-322 in Kerstin Dautenhahn (ed.), *Human Cognition and Social Agent Technology*, Benjamins (2000)
51. Christopher Landauer, Kirstie L. Bellman, "Generic Programming, Partial Evaluation, and a New Programming Paradigm", Chapter 8, pp. 108-154 in Gene McGuire (ed.), *Software Process Improvement*, Idea Group Publishing (1999)
52. Christopher Landauer, Kirstie L. Bellman, "New Architectures for Constructed Complex Systems", in *The 7th Bellman Continuum*, *International Workshop on Computation, Optimization and Control*, 24-25 May 1999, Santa Fe, NM (1999); in *Applied Mathematics and Computation*, Volume 120, pp. 149-163 (May 2001)

53. Christopher Landauer, Kirstie L. Bellman, "Symbol Systems in Constructed Complex Systems", pp. 191-197 in *Proceedings of ISIC/ISAS'99: The 1999 IEEE International Symposium on Intelligent Control*, 15-17 September 1999, Cambridge, Massachusetts (1999)
54. Christopher Landauer, Kirstie L. Bellman, "Lessons Learned with Wrapping Systems", pp. 132-142 in *Proceedings of ICECCS'99: the 5th International Conference on Engineering Complex Computing Systems*, 18-22 October 1999, Las Vegas, Nevada (1999)
55. "Virtual Simulation Environments", pp. 191-196 in *Proceedings of AISP'00: The 2000 International Conference on Artificial Intelligence, Simulation, and Planning*, 6-8 March, 2000, Tucson (2000)
56. Christopher Landauer, Kirstie L. Bellman, "Reflective Infrastructure for Autonomous Systems", pp. 671-676, Volume 2 in *Proceedings of EMCSR'2000: The 15th European Meeting on Cybernetics and Systems Research, Symposium on Autonomy Control: Lessons from the Emotional*, 25-28 April 2000, Vienna (April 2000)
57. Christopher Landauer, Kirstie L. Bellman, "Some Measurable Characteristics of Intelligence", Paper WP 1.7.5, *Proceedings of SMC'2000: The 2000 IEEE International Conference on Systems, Man, and Cybernetics (CD)*, 8-11 October 2000, Nashville Tennessee (2000)
58. Christopher Landauer, Kirstie L. Bellman, "Symbol Systems and Meanings in Virtual Worlds", *Proceedings of VWsim'01: The 2001 Virtual Worlds and Simulation Conference, WMC'2001: The 2001 SCS Western MultiConference*, 7-11 January 2001, Phoenix, SCS (2001)
59. Christopher Landauer, Kirstie L. Bellman, "Computational Infrastructure for Experiments in Cognitive Leverage", in *Proceedings of CT'2001: The Fourth International Conference on Cognitive Technology: Instruments of Mind*, 6-9 August 2001, Warwick, U.K. (2001)
60. Christopher Landauer, Kirstie L. Bellman, "Wrappings for One-of-a-Kind System Development", Paper STSSV04 in *Proceedings of HICSS'02: The 35th Hawaii International Conference on System Sciences (CD), Track IX: Software Technology, Advances in Software Specification and Verification Mini-Track*, 7-10 January 2002, Waikoloa, Hawaii (Big Island) (2002)
61. Christopher Landauer, Kirstie L. Bellman, "Semiotic Processing in Constructed Complex Systems", *Proceedings of CSIS2002: The 4th International Workshop on Computational Semiotics for Intelligent Systems, JCIS2002: The 6th Joint Conference on Information Sciences*, 08-13 March 2002, Research Triangle Park, North Carolina (2002)
62. Christopher Landauer, Kirstie L. Bellman, "Refactored Characteristics of Intelligent Computing Systems", *Proceedings of PERMIS'2002: Measuring of Performance and Intelligence of Intelligent Systems*, 13-15 August, NIST, Gaithersburg, Maryland (2002)
63. Michael E. Long, "Surviving in Space", National Geographic, January 2001, also on the web at URL "http:// www. nationalgeographic. com/ ngm/ 0101/ feature1/ index. html" (availability last checked 02 June 2007), with many useful links to more information
64. Barbara F. Lujan, Ronald J. White, "Human Physiology in Space", National Space Biomedical Research Institute, Houston, Texas, on the web at URL "http:// www. nsbri. org/ HumanPhysSpace/ index. html" (availability last checked 02 June 2007),

65. Patti Maes, "Computational Reflection", Technical Report 87-2, MIT AI Laboratory (1987)
66. P. Maes, D. Nardi (eds.), *Meta-Level Architectures and Reflection*, *Proceedings of the Workshop on Meta-Level Architectures and Reflection*, 27-30 October 1986, North-Holland (1988)
67. Colin McGinn, *The Mysterious Flame: Conscious Minds in a Material World*, New York: Basic Books (1999)
68. Alex Meystel, "Multiresolutional Architectures for Autonomous Systems with Incomplete and Inadequate Knowledge Representations", Chapter 7, pp. 159-223 in S. G. Tzafestas, H. B. Verbruggen (eds.), *Artificial Intelligence in Industrial Decision Making, Control and Automation*, Kluwer (1995)
69. Christian Müller-Schloer, "Quantitative Emergence." Talk presented at Dagstuhl Seminar on Organic Computing, January 2006, p. 8.
70. Alex Meystel, *Semiotic Modeling and Situation Analysis: An Introduction*, AdRem, Inc. (1995)
71. David C. Parks and Moshe Tennenholtz (eds.), *Games and Economic Behavior*, Special Issue Dedicated to the ACM Conference on Electronic Commerce (EC'07), Elsevier, 2008 (to appear).
72. H. Pasula, L. Zettelmoyer, and L. Pack Kaelbling, "Learning Symbolic Models of Stochastic Domains", *Journal of Artificial Intelligence Research* 29(2007).
73. Candace Pert et al., "Neuro-peptides and their receptors: a psychosomatic network", J. Immunology 135(2), pp. 820-826 (1985)
74. A. B. Pippard, *Response and Stability: An Introduction to the Physical Theory*, Cambridge University Press, Cambridge, 1985, p. 127.
75. Torsten Rüting, "History and significance of Jakob von Uexküll and of his institute in Hamburg", *Sign System Studies*, Vol. 32, Nos. 1/2, pp. 35-72 (2004)
76. Ferdinand de Saussure, *Cours de linguistique générale*, Payot, Paris (1916), translated by W. Baskin as *A Course in General Linguistics*, Fontana/Collins, Glasgow (1977)
77. Mary Shaw, William A. Wulf, "Tyrannical Languages *still* Preempt System Design", pp. 200-211 in *Proceedings of ICCL'92: The 1992 International Conference on Computer Languages*, 20-23 April 1992, Oakland, California (1992); includes and comments on Mary Shaw, William A. Wulf, "Toward Relaxing Assumptions in Languages and their Implementations", *ACM SIGPLAN Notices*, Volume 15, No. 3, pp. 45-51 (March 1980)
78. Brian Cantwell Smith, "Varieties of Self-Reference", In J.Y. Halpern (Editor), *Reasoning about Knowledge*, *Proceedings of TARK 1986*, AAAI Publication, pp. 19-43 (1986)
79. John Sowa, *Knowledge Representation*, Morgan Kaufmann (1999)
80. Luc Steels, "The symbol grounding problem is solved, so what's next?", in M. De Vega, G. Glennberg and G. Graesser (eds.), *Symbols, embodiment and meaning*, Academic Press, New Haven (2007)
81. Ian Stewart and Jack Cohen, *Figments of Reality: The Evolution of the Curious Mind*, Cambridge University Press (1999)
82. G. M. Tomkins, "The Metabolic Code", *Science* 189: 760-763, 1975.
83. Jakob von Uexküll, "Streifzüge durch die Umwelten von Tieren und Menschen", (1934); translated and edited by Claire H. Schiller as "A stroll through the worlds of animals and men", pp. 5-80 in Claire H. Schiller (eds.), *Instinctive Behavior: The Development of a Modern Concept*, International Universities

Press, New York (1957); also reprinted in *Semiotica*, Vol. 89, No. 4, pp. 319-391 (1992)
84. Duncan J. Watts and Steven H Strogatz, "Collective dynamics of 'small-world' networks", *Nature* 393 (1998)
85. Ludwig Wittgenstein, *Logisch-philosophische Abhandlung* Annalen der Naturphilosophie (1921), translated into English by C. K. Ogden as *Tractatus logico-philosophicus*, Routledge and Kegan Paul (1922); also on the Web at URL "http:// www. kfs. org/ jonathan/ witt/ tlph.html" (availability last checked 31 May 2007); also in Project Gutenberg at URL "http:// www. gutenberg. org/ etext/ 5740" (availability last checked 31 May 2007);
86. Ludwig Wittgenstein, *Philosophische Untersuchungen*, translated into English by G. E. M. Anscombe as *Philosophical Investigations*, Prentice Hall (1999), Blackwell (1953/2001) (which includes the original German text also)
87. Carl Zimmer, "From Fins to Wings", National Geographic, November 2006, also on the web at URL "http:// www. nationalgeographic. com/ ngm/ 0611/ feature4/ index. html" (availability last checked 02 June 2007), with many useful links to more information

4

Controlled Emergence and Self-Organization

Christian Müller-Schloer[1] and Bernhard Sick[2]

[1] Institute of Systems Engineering – System and Computer Architecture, University of Hannover, Appelstraße 4, 30167 Hannover, Germany.
cms@sra.uni-hannover.de

[2] Institute of Computer Architectures – Computationally Intelligent Systems Group, University of Passau, Innstraße 33, 94032 Passau, Germany.
bernhard.sick@uni-passau.de

Summary. Admiration of nature's ability to develop robust self-organizing and complex structures showing emergent behavior was the starting point for the Organic Computing (OC) endeavor. Although the concepts of self-organization and emergence have been subject to extensive investigations and discussions for more than 100 years, soon it became clear that we lack a quantitative assessment of these concepts as a basis for an implementation in technical systems. The main questions to be answered in this context are: Can we define emergence and self-organization (or sub-concepts thereof) compatible with a quantitative, experimental, and objectifiable method as required in natural science? Can we control self-organization and emergence without forcing their meaning? Are there generic architectures generally applicable to technical systems serving this purpose? In this chapter, we will try to give some answers to these questions. After an introduction and specification of the problem we will review some recent approaches to a definition of emergence and assess them with respect to their usability in our technical context. We will then introduce an architectural template, the Observer/Controller architecture, which seems to be a key feature in most OC systems. In addition to the general pattern – essentially constituting a higher-level control loop – this Observer/Controller architecture will be developed in some detail as a framework for own implementations. We present a quantitative approach for a technically relevant definition of emergence and self-organization, and propose a systematic approach to a ranking of various Observer/Controller architectures.

4.1 Introduction

Emergence and self-organization have been discussed extensively in the literature. In the context of Organic Computing (OC) we are mainly interested in a technical utilization of these concepts. At least in the context of intelligent technical systems, emergence is mostly the result of a self-organizing bottom-up process, which seems to be responsible for its fascination: Without external interference, a system is able to develop some higher form of order. But exactly

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_4,

this phenomenon makes emergent self-organizing processes somewhat suspect from a traditional engineering viewpoint. Engineers are used to setting an objective (a specification), starting a development process, and making sure that the specification is met. Hence, at the heart of Organic Computing we have to solve the problem of "controlled emergence", a seeming essential contradiction between bottom-up self-organized freedom and top-down enforced compliance with a preset specification [14].

Two steps are necessary to solve the problem: First we have to understand more deeply the phenomena of emergence and self-organization. Given the extensive covering of these terms in the literature, we do not claim to come up with a generally applicable, new definition. We rather want to restrict ourselves to certain aspects necessary for the technical utilization of these two concepts. This means, above all, that we are mainly interested in quantitative aspects of emergence and self-organization, moreover, only in those quantitative aspects that are accessible to an automatic evaluation in a (computer-based) system.

This leads to the second step: We have to investigate how emergence and self-organization can be fostered or even designed in a technical system while, at the same time, they are kept under control. We want to allow a maximum of "freedom" and "creativity" of the system itself, but only within a certain, well-defined area. For this purpose we have to define architectural superstructures that are able to keep emergent systems under control and guide them towards the desired objectives.

Organic Computing is inspired by nature. But this does not mean that OC tries to copy nature. We borrow certain – especially organizational – concepts from complex natural systems such as the brain, companies, or societies. One of those concepts is the Observer/Controller structure that can be found in many functioning (hence ordered) natural systems. For example, the brain seems to consist of a huge amount of control "circuitry" serving just the purpose of checking and double-checking the function of the acting subsystems. Although we try to learn from nature, we realize that within the foreseeable future we will not be able to come close to its complexity, functionality, and efficiency.

In section 4.2 of this chapter, we will review some of the more important concepts of emergence developed in the philosophy of mind. We will also take a look at various new definitions or notions of emergence in intelligent technical systems (e.g., multi-agent systems) and discuss whether they could be useful from our viewpoint. Then we will briefly introduce the generic architectural template of Observer/Controller architectures (section 4.3). In section 4.4 we will discuss an approach to a quantitative analysis of emergence. Section 4.5 offers an approach to a quantitative analysis of self-organization, again with respect to technical applications. In section 4.6 we will resume the discussion of Observer/Controller architectures begun in section 4.3 but now with the focus on different ways of an architectural implementation. In particular we are interested in the distribution of such architectures. This discussion leads

to a taxonomy or roadmap, which could help to classify OC architectures. Section 4.7 will summarize the major findings.

4.2 Emergent behavior of intelligent technical systems – an analysis of related work

In this section, we will analyze various definitions of emergence. Initially, we will take a look at philosophical notions of emergence and discuss whether they are useful for OC. Then, some very recent definitions of emergence are reviewed, published by Abbott, Stephan, Fromm, Gabbai et al., di Marzo Serugendo et al., and de Wolf et al. (see [14] for a more detailed discussion of some of the publications). We have selected newsworthy publications (2005 or later) that have a close relationship to technical systems (and, therefore, OC).

4.2.1 The notion of emergence in philosophy of mind

In philosophy of mind, the *emergent* behavior of more or less complex "systems" (being either natural, supernatural, or artificial) has been investigated for more than a hundred years and definitions of "weak emergentism" and "strong emergentism", for instance, have been provided (see [22] for a comprehensive review).

Weak emergentism is based on the three theses of (1) physical monism, (2) systemic (collective) properties, and (3) synchronous determinism. From a viewpoint of technical (i.e., artificial) systems, only the *thesis of systemic (collective) properties* is relevant. Basically, it says: "Emergent properties are collective (systemic), i.e., the system as a whole has this property but single components do not have properties of this type." Often, this sentence is cited this way: "The whole is more than the sum of its parts." But the meaning of these two sentences is significantly different; the former has much stronger requirements. Examples for systems that have emergent properties in such a weak sense are artificial neural networks (i.e., combinations of simple nodes and connections that are used for pattern matching), flocks or swarms of artificial animals (e.g., bird swarms that are able to avoid obstacles), or robots (e.g. playing soccer or building heaps of collected items). We can notice that terms at the component level are not sufficient to describe properties that arise at the system level, for example: "Rules", "patterns", "classes" are terms that are not used at the level of synapses or neurons. The thesis of systemic (collective) properties claims that single components do not have properties *of the same type* as the overall system. Therefore, the weight of a car – being the sum of the weights of its components – is not an emergent property in this sense. From the viewpoint of OC, weak emergence is certainly a necessary pre-condition, but not a sufficient one. There are many OC systems that are emergent in a weak sense but their emergent properties are not interesting.

A notion of emergence that adds very stringent requirements to weak emergence is *strong emergence*. Strong emergentism is based on the *thesis of irreducibility* that addresses the question why a system has a certain property. Basically, it says: "The macro-behavior of a system can in principle not be explained knowing the micro-behavior of components, their interaction rules, etc." A reductive explanation aims at explaining properties of a system using descriptions of components of a system, their properties, their arrangement, etc.

An example where we do not have an emergent behavior in the strong sense is car behavior: We can explain a car's drivability in bends because we know the properties of component parts and their interactions. All the examples mentioned in the previous paragraph are, therefore, not emergent in this strong sense. Moreover, there are no artificial (including OC) systems that can be termed to be emergent in this sense as they are all subject to the laws of nature. However, there is often an *explanatory gap*: Either we do not have the knowledge to explain expected behavior in advance or we did not specify the components or systems that we investigate in a sufficiently detailed way. Admittedly, it is mind-boggling whenever a complex behavior on the system level results from very simple rules on the component level but lack of knowledge alone should not qualify a phenomenon as emergent.

Altogether, from the viewpoint of OC, historical, philosophical definitions of emergence are either too weak or too strong. The former means that too many systems are termed emergent, the latter implies that no artificial (technical) systems are emergent. We need a technology-oriented notion of emergence (more than weak emergence) possibly depending on the type of OC systems we investigate and the type of questions we ask. In this sense, a system may be regarded as being emergent concerning one (objective) aspect and being non-emergent with respect to another.

4.2.2 Novel, technology-oriented notions of emergence

In his approach to explain emergence [1], Abbott defines emergence as a relationship between a phenomenon and a model, where a model is a collection of elements with certain interrelationships. Central to his definition of emergence is the concept of *epiphenomena*. An epiphenomenon is defined "as a phenomenon that can be described (sometimes formally but sometimes only informally) in terms that do not depend on the underlying phenomena from which it emerges" (cf. thesis of systemic properties). A phenomenon is called emergent over a given model if it is epiphenomenal with respect to that model. An emergent behavior is called *static* if its implementation does not depend on time. Thus, hardness as a property of a material (and not a property of isolated atoms) or the resonant frequency of a resonant circuit (and not of its components) could be attributed to this variety of emergence. In contrast, an emergent behavior is regarded as *dynamic* if it is defined "in terms of how

the model changes (or doesn't change) over some time". Dynamically emergent phenomena can additionally be subdivided into *non-stigmergic* dynamic phenomena and *stigmergic* dynamic phenomena. The former can be described by means of continuous equations, the latter involve autonomous entities with discrete states. As OC systems are artificial technical systems that are termed to be "intelligent" (i.e., able to adapt to a dynamic environment; cf. [4]), we are mainly interested in stigmergic, dynamic emergence.

In his work on emergentism [22], STEPHAN focuses on definitions of emergence that distinguish various causes for emergent behavior. His definitions see emergence

1. as a consequence of *collective self-organization* (i.e., interesting properties at the system level are realized by an interaction of identical or very similar components),
2. as a consequence of *non-programmed functionality* (i.e., systems interacting with their environment show a certain goal-oriented, adaptive behavior that is not a result of dedicated control processes or explicit programming),
3. as a consequence of *interactive complexity* (i.e., systemic properties, patterns, or processes are the result of a complex interaction of components; cf. [3]),
4. in the sense of *incompressible development* (i.e., a macro-state of a system with a certain microdynamics can be derived from the microdynamics and the system's external conditions, but only by simulation), and, finally,
5. in the sense of *structure-unpredictability* (i.e., the formation of new properties, patterns, or structures follows the laws of deterministic chaos).

All these types of emergence actually occur in OC systems, and sometimes it depends on the type of the posed question which of those definitions of emergence one should use in a certain context.

In contrast to STEPHAN's cause-oriented approach, FROMM sees emergence from a largely modeling-oriented viewpoint (focusing on multi-agent systems; cf. [8, 9]). The following – somehow recursive – definition is used as a starting point: "A property of a system is emergent, if it is not a property of any fundamental element, and emergence is the appearance of emergent properties and structures on a higher level of organization or complexity." The four ordered types or classes of emergence are *nominal*, *weak*, *multiple*, and *strong emergence* (where "weak" and "strong" are not equivalent to the corresponding terms used in philosophy of mind!). The four classes differ mainly in the type of the system (e.g., closed with passive components, open with active or multiple levels, or new levels), the roles of the components (e.g., fixed, flexible, or fluctuating), and the feedback mechanisms between components or levels (e.g., top-down feedback or multiple feedback). In particular the classes of weak and multiple emergence definitely can be found in many OC systems.

In the opinion of GABBAI ET AL. – who study emergence in the context of multi-agent systems [10] – "emergence is a sometimes negative phe-

nomenon found in complex systems, which can also be positively exploited to varying degrees. The full, or ultimate, positive exploitation of emergence is self-organization; a system aligns itself to a problem and is self-sustaining, even when the environment changes." Gabbai et al. explicitly regard self-organization as a specific form of emergence. While we do not agree with this constricted viewpoint (limited by the specific application field), we want to emphasize an important side note which can be found in [10]: *Entropy* could be utilized to measure *order*, which emerges, e.g., due to self-organization, and, hence, to measure emergence.

The need "to find new principles, theories, models, mechanisms and methodologies to engineer self-organizing systems with or without emergent phenomena" is also recognized by di Marzo Serugendo et al. in [7]. They focus on the design of multi-agent systems applying various self-organization mechanisms. Their viewpoint is that of *neo-emergentism* which distinguishes itself from the historical (i.e., philosophical) *proto-emergentism* by being less "miraculous". That is, an artificial system can be regarded as emergent even if its behavior can be understood and reproduced at least to a certain degree (i.e., if it is not a "black box"). In their opinion, *designable emergence* occurs in a narrow "space lying between conditions that are too ordered and too disordered." Again, *order* is mentioned as an important criterion.

In [5], De Wolf and Holvoet discuss the relationship of emergence and self-organization. They introduce the noun *emergent* for the result of a process – in contrast to the process itself – which leads to a certain macroscopic pattern: "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 w.r.t. the individual parts of the system." Properties, behavior, structure, or patterns, for instance, can be emergent. Self-organization is defined as "... a dynamical and adaptive process where systems acquire and maintain structure themselves, without external control." Structure can be spatial, temporal, or functional. Self-organization and emergence are seen as emphasizing different characteristics of a system. Both can, according to the authors, exist in isolation or together. From an OC viewpoint, these two concepts seem to be highly related: In intelligent technical systems, emergence typically is the result of an adaptive and self-organizing (in the broadest sense) process with many components. Self-organization without emergent behavior is possible, e.g. as the result of a predefined interaction of only a few components. In [6], De Wolf et al. describe an industrial application example of a self-organizing emergent system – an automated guided vehicle warehouse transportation system – where (amongst others) an entropy measure is applied to measure the system-wide behavior of a self-organizing emergent system.

While all these cause-oriented, process-oriented, or modeling-oriented concepts are valuable from the OC viewpoint, one important aspect has been neglected until recently: the measurement-oriented view. In our opinion, it is a must to reconceive emergence considering the analysis of OC systems.

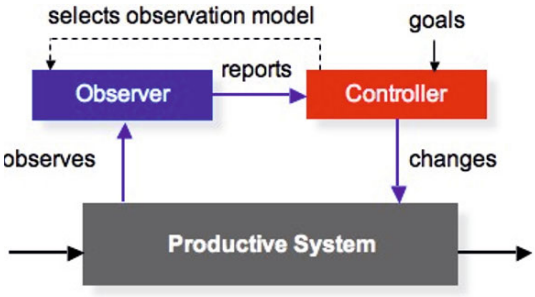


Fig. 4.1. Observer/Controller architecture.

It must be shown, for example, how the following types of emergence could be quantified: emergence due to interactive complexity (Stephan), multiple emergence (Fromm), or stigmergic dynamic emergence (Abbott). In OC systems, we want to "do" (design or allow) emergence but must, at the same time, keep it under control. That is, the emergent behavior of OC systems must be achieved by a balanced approach where the emergent processes are kept within pre-defined boundaries by certain control mechanisms. However, the other viewpoints have certainly to be considered when appropriate measures are defined, selected, or combined. A measurement-oriented notion of emergence (cf. section 4.4) may coexist with most of the existing definitions of emergence (in particular the definitions of "weak" and "strong" emergence used in philosophy of mind). It must be technically realizable and allow to determine a "degree" of emergence. Finally, it must definitely be objective, i.e., independent from the knowledge of the observer or the measurement techniques. We expect that there will be a variety of measures for different emergent phenomena and different system objectives, resulting in a collection of emergence "detectors". For each application we must determine the appropriate attributes that characterize emergence, e.g., measures for order, complexity, information flow, etc.

4.3 Observer/controller-architecture

OC systems consist of possibly large numbers of interacting and cooperating subsystems. Each of these subsystems might be relatively autonomous, pursuing its own goals. In order to assess the behavior of such a system and – if necessary – for a regulatory feedback to control its dynamics, we assume that a generic *Observer/Controller* architecture (O/C architecture) is required as depicted in figure 4.1 [14, 17, 20].[3]

[3] A similar architecture is used in IBM's Autonomic Computing project with the four steps Monitor, Analyze, Plan, and Execute (MAPE).

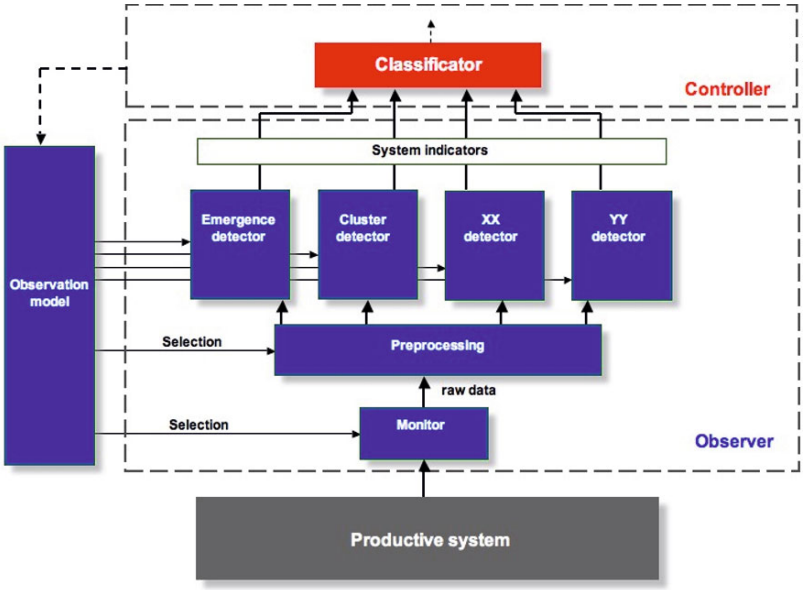


Fig. 4.2. Observer architecture.

The productive system, also called system under observation and control, does the "actual" work, i.e., it transforms the sensory input via fixed algorithms into appropriate output actions. Traditionally, *how* this task is performed by the production system remains unchanged over the lifetime of the system. In OC systems, however, the internal mechanisms of the production system can be adapted according to the current context. A higher-level authority is needed to make these changes. We call them observer and controller. They are responsible for an appropriate surveillance and feedback.

In [18] we have proposed a generic O/C architecture (O/C for observer/controller). The observer collects and aggregates information from the production system. The aggregated values (system indicators) are reported to the controller, who takes appropriate actions to influence the production system. The observer contains several specialized detectors to calculate the system indicators from the observed raw data (figure 4.2). The collection of attribute emergence values (the emergence fingerprint) is a part of the observation result as determined by the observer. 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.

It is important to mention that an organic system continues to work and does not break down if observer and controller stop working. In comparison to a classical system design, OC systems have the ability to adapt and to cope with emergent behavior for which they have not been programmed in

detail. The goal of OC is to build systems that perform their tasks by using (controlled) self-organization. However, this is independent of using centralized or decentralized 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.

We cannot discuss all the aspects of O/C architectures here in detail and refer instead to the literature. We will return to the aspects of distribution of observers and controllers in section 4.6.

4.4 Quantitative emergence

4.4.1 The measurement of order

The meaning of order as perceived by a human[4] observer is not clear without ambiguity. A homogeneous mixture of two liquids can be regarded as "orderly" (figure 4.3, right). Applying the concept of thermodynamic *entropy*, however, will result in low entropy values (i.e., higher order) for the left example of figure 4.3. Apparently, order depends on the selection of certain attributes by the (human) observer. If we are interested in the spatial structure we have to base our measurement on the positions of the molecules (figure 4.3, left), if we are interested in homogeneity we can use the relative distances between the molecules (figure 4.3, right). The emergence definition presented in this chapter is based on the statistical definition of entropy (which essentially can be explained as counting events or occurrences).

The computation of the entropy of a system S with N elements e_i is done as follows:

1. Select an attribute A of the system elements of S with discrete, enumerable values a_j.
2. Observe all elements e_i and assign a value a_j to each e_i. This step corresponds to a quantization.
3. Transform into a probability distribution (by considering the relative frequency as a probability) over the attribute values a_j (i.e., a histogram) with p_j being the probability of occurrence of attribute a_j in the ensemble of elements e_i.
4. Compute the entropy according to Shannon's definition

$$H_A = -\sum_{j=0}^{N-1} p_j \log_2 p_j \tag{4.1}$$

[4] Currently the only observers who make these decisions are human designers and researchers, but eventually one could in fact imagine a system that could make these decisions based on knowledge bases and experiments with a target system (e.g. trying out a set of likely candidate attributes etc.).

Fig. 4.3. Order perception: Both pictures could be perceived as high order (left: more structure, right: more homogeneity) depending on the objective of the observer.

If the attribute values are equally distributed (all p_j equal) we will obtain the maximum entropy. Any deviation from the equal distribution will result in lower entropy values (i.e., higher order). In other words: The more structure is present (unequal distribution), the more order is measured. The unit of measurement is bit/element. Thus, the entropy value can be interpreted as the information content necessary to describe the given system S with regard to attribute A. A highly ordered system allows a simpler description than a chaotic one.

4.4.2 Observation model

The resulting entropy value depends on two decisions taken by the observer: Which attribute A is measured? With what resolution (or quantization) is it measured? The quantization determines the information content of the system description but it is not a property of the system. Neither is the selection of a certain attribute A a system property. This means that a measured entropy value is only meaningful if we know the exact observation context. This context is subsumed by the observation model.

This reflects the fact that order is not an intrinsic property of the system. Perceived order depends on subjective decisions or capabilities of the observer. In living systems, the sensory equipment limits the selection of observable attributes and the resolution of the measurement. In addition, the brain directs attention to certain observables, which are relevant in the present situation, and masks other attributes or registers them with lower effort, i.e., lower resolution. Hence, order results from an interaction between the observer and the observed system guided by the observation model. The observation model depends on the capabilities of the sensory equipment and the utility of certain observations with regard to the purpose.

An observer might be interested in more than one attribute. In this case, we obtain a vector of entropy values (H_A, H_B, H_C, ...) with respect to attributes $A, B, C, \ldots$ We could add them into a total system entropy H_S. H_S denotes the information content of the total system description under the given observation model. It has the drawback of hiding or averaging the single attribute entropies. Therefore we prefer the single entropy values ("emergence fingerprint").

4.4.3 Emergence

Entropy is not the same as emergence. Entropy decreases with increasing order while we aim at an emergence value, which increases with order. As a first try we define emergence as the difference ΔH between the entropy at the beginning of some process and at the end:

$$\Delta H = H_{start} - H_{end} \tag{4.2}$$

In case of an increase of order this results in a positive value of ΔH. A process is called emergent if (1) $\Delta H > 0$ and (2) the process is self-organized. This definition has two problems:

1. The measurement of H depends on the observation model (e.g. the abstraction level). An observation on a higher abstraction level will lead to a lower entropy value H even when there is no change of S in terms of self-organized order.
2. Since the start condition of the system is arbitrary, ΔH represents only a relative value for the increase of order. It would be preferable to have a normalized emergence measure.

The first problem can be solved by determining the portion of ΔH, which is due to a change of abstraction level (ΔH_{view}), and subtracting it.

$$\Delta H = \Delta H_{emergence} + \Delta H_{view} \tag{4.3}$$

$$\Delta H_{emergence} = \Delta H - \Delta H_{view} \tag{4.4}$$

An example for an effect, which changes ΔH without an underlying emergent process ($\Delta H_{emergence} = 0$), would be the measurement of a coordinate with a lower resolution. If we first observe with a 32-bit resolution and later with a 2-bit resolution, this results in a reduction of description complexity by 30 bit, which is, however, not due to a self-organization process ($\Delta H_{view} = 0$). In other words: Equation (4.2) holds only if we have not changed the abstraction level or if ΔH_{view} can be determined and subtracted.

The second problem is solved by definition of an absolute reference as starting condition. Obviously, this starting condition could be the state of maximum disorder with an entropy of H_{max}. H_{max} corresponds to the equal probability distribution. This leads to the following definition of emergence:

$$M = \Delta H_{emergence} = H_{\max} - H - \Delta H_{view} \tag{4.5}$$

Absolute emergence is measured through the increase of order due to self-organized processes between the elements of a system S in relation to a starting condition of maximal disorder. The observation model used for both observations must be the same ($\Delta H_{view} = 0$) or ΔH_{view} must be determined and subtracted.

4.4.4 Limitations and challenges

From our technical point-of-view emergence loses some of its philosophical connotation. We realize that our model does not cover the "strong emergence" definition, which demands that emergence is an *in principle* unexplainable phenomenon . But this definition is inapplicable in the domain of engineering and computer science. On the contrary, we are only interested in a quantifiable phenomenon resulting in a (self-organized) increase of order. We *define* as emergence what we can measure. If this definition is too restrictive, excluding some unexplainable emergent effects, we could accept that what we measure with our method is termed just "*quantitative emergence*" and constitutes only a certain form of emergence meaningful in intelligent technical systems. This definition leaves room for wider definitions of emergence in a more general sense.

Our quantitative definition of emergence is based on the assumption that emergent phenomena can always be observed in terms of patterns (observable, e.g., in space and/or time) consisting of large ensembles of elements. The resonance frequency of a resonant circuit does not constitute an emergent pattern but is rather a property of such a pattern. Order can also be determined in the time or frequency domain. Therefore, we can apply our emergence definition to the resonance frequency example if we observe the system behavior after a Fourier analysis. This extends the above definition of the observer model: Any type of preprocessing can also be a part of the observer model. This corresponds quite well to the operation of the animal (and human) perception[5]. The reader interested in a more detailed discussion is referred to [13].

4.5 Quantitative self-organization

There is a variety of definitions of self-organization. We start with the following one[6]: "Self-organization is a process in which the internal organization of a system, normally an open system, increases in order without being guided or managed by an outside source" [23]. The term self-organization is frequently

[5] In the cochlea, the sound moves hair bundles, which respond to certain frequencies. The brain therefore reacts to preprocessed signals [9].

[6] The original term "complexity" has been replaced by "order".

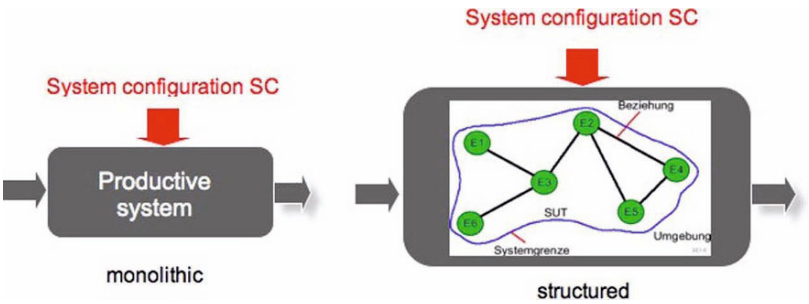


Fig. 4.4. Monolithic and structured productive systems, which can be defined by changing the system configuration *SC*.

used in a somewhat "magical" sense implying that something unknown within a black box happens, which leads to a certain kind of order. We want to take a pragmatic and more technical viewpoint and analyze the process of self-organization by asking *who* organizes *whom* (remembering that there is no effect without a cause, even if this cause might be distributed).

Organization is a process as well as the result of a process. To avoid confusion, let us call the result of an organization process a "system" (in a certain current state which can change over time). A system can be described in terms of structure and behavior. To change a system we can change its elements or their relationships (or both) by (re-)defining its structural description (e.g. a VHDL description of an integrated circuit). We can also modify its behavior by changing parameters (e.g. the threshold value for temperature control). In any case, to change a system, some active entity must be able to change structure and/or behavior of the system. This system might be *active* in its role as a productive system (e.g. transforming sensory inputs into control outputs according to a built-in algorithm) but it is *passive* under the aspect of changing its own structure and behavior.

In order for a system to be (re-)organized, it must be *adaptable,* i.e. it must allow for changes of its structure and/or behavior. In technical systems this means that certain control parameters must be visible and changeable at the outside of the system. Let us call the set of these parameters system configuration *SC* (figure 4.4). *SC* is a bit string and might be as short as one bit (to change the working mode) or as long as a full system description (a program or a VHDL file). Each (legal) value of *SC* defines a point in the system's configuration space. The configuration space comprises all possible configurations of the system. Further, we can define the size of the configuration space, its *variability*, as the number of bits of *SC*. Systems with only a few possible configurations ("modes") are called monolithic because they do not display structure to the outside world. In general, *SC* can define the

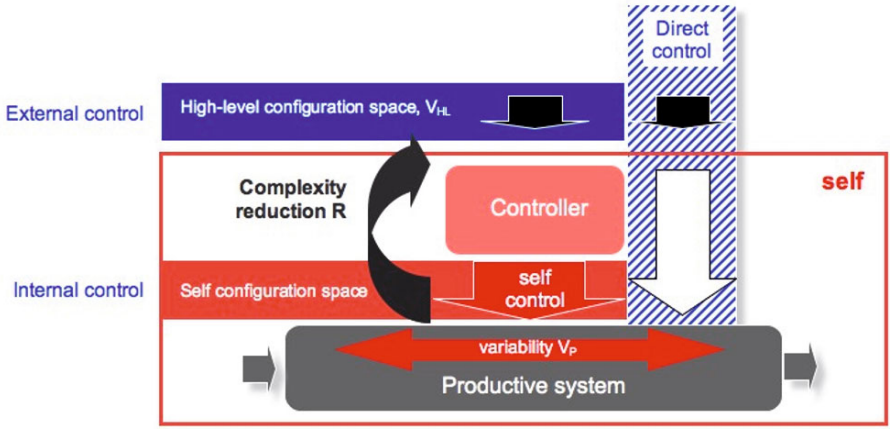


Fig. 4.5. A productive system is controlled by a system-internal active entity (controller) constituting self-control. The controller itself is controlled by high-level controls (high level configuration space). In addition there might be control signals acting directly upon the productive system (direct control).

constitutive elements and their relationships ("structured system") as well as parameters influencing the system's behavior.

Whether we can call a system self-organizing or not then depends on *where* the active entity sits that changes SC. The traditional way to control SC is from the outside by an engineer or a designer at design time. In Organic Computing , apparently the active entity must be part of the system itself because changes are made at run time. We can assign this role of active run time control of the system configuration SC to the controller (who needs some sensory input – provided by the observer – in order to make decisions).

The definition of *self*-organization as opposed to *outside* organization then boils down to the trivial statement that a self-organizing system contains active components or mechanisms that change the system's structure and/or behavior. Whether this active component is just one entity as depicted in figure 4.5 or distributed over many elements is open at this point and will be discussed in section 4.6. The term "self" is just a matter of definition: A controller inside the system is part of the system itself (*intrinsic control*), an outside controller would constitute *extrinsic control*.

It might be argued that this definition of self-organization includes also primitive cases such as the setting of just one bit to switch a system configuration between two modes, which will hardly be accepted as self-organization. Self-organization in the commonly accepted sense seems to imply larger numbers of elements rearranging themselves. This is certainly true but it makes just a quantitative difference. A system switching modes by itself would then be just a marginal case of a more general multi-element system modifying its structure and behavior.

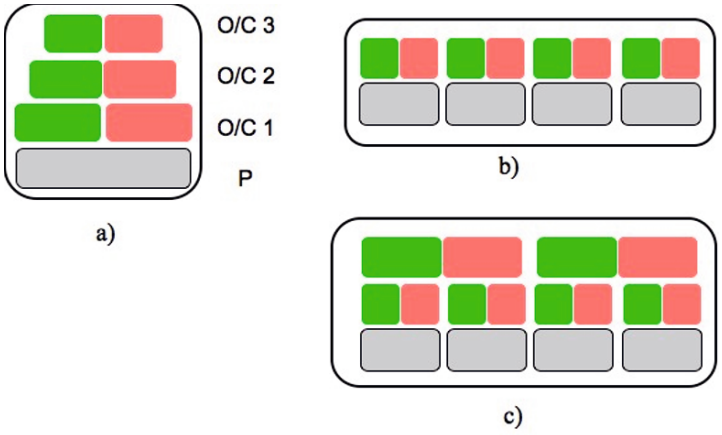


Fig. 4.6. O/C building blocks (a) vertically stacked, (b) cooperating horizontally in a fully distributed architecture, (c) in a mixed horizontal/vertical architecture.

In general, *SC* can at the same time be controlled from the outside and from the inside. We call the control inputs applied from the inside (i.e., by the controller) "self-control" (defining the self-configuration space). Since the controller itself must be controlled (or guided) as well, we need also high-level controls applied from the outside (defining the high-level configuration space, see figure 4.5). In addition there might be control signals acting directly upon the productive system (direct control). The notion of configuration spaces allows us to formulate self-organization quantitatively in terms of a degree of autonomy. If we measure the variability of the high level configuration space as V_{HL} (high-level control) and the variability of the internal configuration space (i.e. the variability of the productive system) as V_P, self-organization will always result in a *complexity reduction* $R = V_P - V_{HL} > 0$. A self-organizing system will switch autonomously between configurations of its internal configuration space displaying a smaller configuration space (lower complexity, smaller variability) to the outside. We can define the ratio of R and V_P as the (static) degree of autonomy S:

$$S = \frac{R}{V_P}. \tag{4.6}$$

Without complexity reduction there is no autonomy. Providing no external control ($V_{HL} = 0$) results in full autonomy ($S = 1$). S and V_P span a diagram (figure 4.7) illustrating the trend of OC system development: We will witness an increasing complexity of productive systems (higher variability V_P), which makes it increasingly challenging to reach large complexity reduction R and hence high degree of autonomy S. We will use this diagram in section 4.6 to position different O/C architectures.

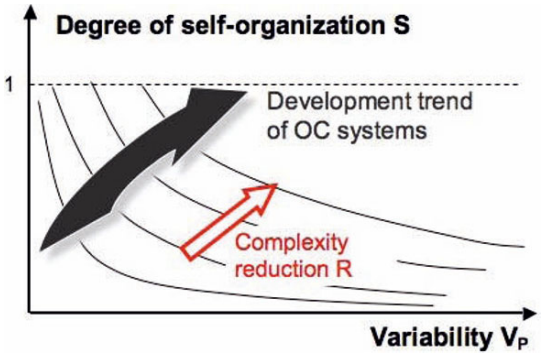


Fig. 4.7. Degree of self-organization S as a function of the variability of the productive system, V_P.

Looking more closely, however, it becomes clear that due to its static nature the above definition of S is not yet adequate to characterize the self-organizing and autonomous behavior of a system. The variability measures the size of a configuration space but makes no statement about the frequency of control interactions to change the configuration. Therefore it seems more adequate to measure the *flow of control information* over a certain period of time $t_2 - t_1$. This results in a dynamic definition of the degree of autonomy s (now in lower case). Let $h(t)$ and $l(t)$ be the high-level and low-level flow of control information, respectively, then we can define the dynamic complexity reduction r over the time window $t_2 - t_1$ as

$$r = \int_{t_1}^{t_2} (l - h)\, dt \tag{4.7}$$

and the dynamic degree of autonomy s as

$$s = \frac{\int_{t_1}^{t_2} (l - h)\, dt}{\int_{t_1}^{t_2} l\, dt} \tag{4.8}$$

A high degree of dynamic autonomy s means that over a period of time only a small amount of control information h needs to be fed into the system from the higher level: The manager needs to interfere only occasionally if the system runs smoothly. This does not preclude the internal self-organization mechanisms (the controller) to adapt the system frequently if necessary.

The above definition (static as well as dynamic) still has a practical drawback: The description complexity is not well-defined since descriptions

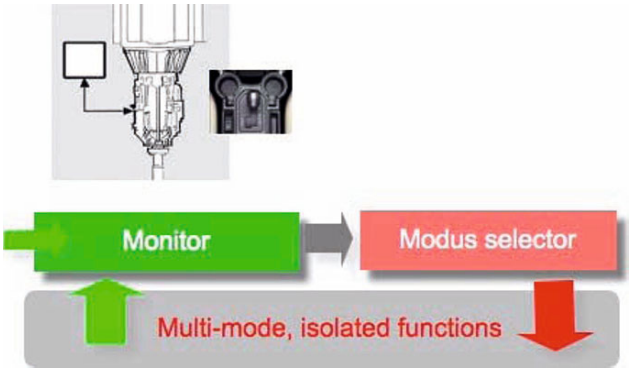


Fig. 4.8. Adaptive gearbox (multi-mode system).

can contain redundant information. Therefore we require minimal description length[7].

4.6 Towards an OC roadmap

4.6.1 The arrangement of O/C building blocks

It is an apparent shortcoming of the above discussion that it seems not to cover truly distributed self-organizing systems, the prototypes of self-organizing (and emergent) systems. In this section, we will show that our systematic approach towards self-organization, which has been exemplified with a single O/C, can be extended to a multitude of architectural options.

The Observer/Controller/Productive-system loop constitutes a basic building block of OC systems. This building block has all four characteristic properties of a holon (as defined by KOESTLER [24]), i.e., it is (1) a self-contained autonomous unit, (2) communicates with other holons on the same level of order (3) to build higher level "organs", which in turn have the properties of a holon. Their constituent holons can stop cooperating which (4) leads to their decay.

As shown in figure 4.6, these building blocks can be arranged in different architectures. We can

1. stack O/C levels vertically in a hierarchical manner, where the higher-level O/C observes and controls the next lower level, decreasing in the level of detail as we go to higher levels,

[7] In a subsequent paper [15] we differentiate the notions of autonomy (as defined in this chapter) and the more architectural aspect of self-organization. There we introduce also a degree of self-organization taking into account the distribution of self-control mechanisms.

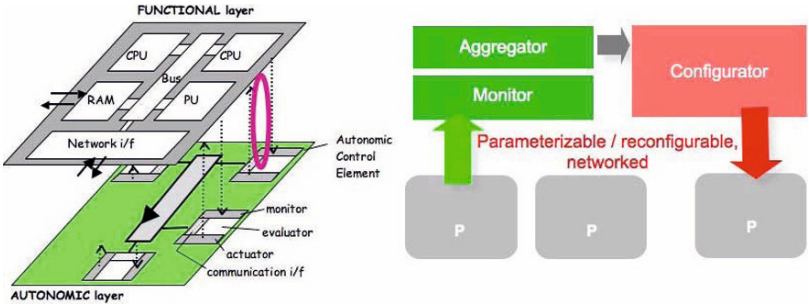


Fig. 4.9. Reconfigurable system.

2. juxtapose the building blocks horizontally to form a fully distributed O/C structure, or
3. mix these methods, e.g., to span a cluster of juxtaposed O/Cs with one or more higher-level O/Cs.

In the following, we want to illustrate this O/C architecture with a few existing or hypothetical technical examples. We will denote each variant with a tuple of identifiers $O/C(h, v)$, where $h = \{single, multiple\}$ and $v = \{1, 2, 3 \ldots\}$ denote the stacked O/C levels, not counting the productive system P. The four examples can be arranged in increasing order with respect to their degree of autonomy S and variability V_P:

1. Multi-mode system: $O/C(single, 1)$
2. Reconfigurable system: $O/C(multiple, 1)$
3. Brokered system: $O/C(multiple, 2)$
4. Distributed system: $O/C(multiple, 3)$

4.6.2 Multi-mode system

A multi-mode system is exemplified by an adaptive gearbox (cf. figure 4.8). The productive system has a few working modes (such as economy, sport, winter, etc.). The observer is a simple monitor collecting external sensor data and the driver's behavior, classifies these inputs and derives a situation indicator. The controller works as a modus selector setting the gearbox into the desired mode. Since we have one productive system and one level of O/C, this is a system of type $O/C(single, 1)$.

4.6.3 Reconfigurable system

A reconfigurable system is exemplified by a *system on a chip* (SoC) such as the experimental system developed by Herkersdorf and Rosenstiel [2] (figure 4.9). It displays a number of subsystems such as CPUs, peripheral units,

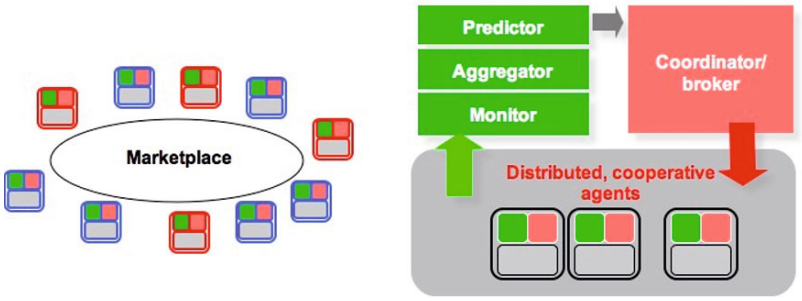


Fig. 4.10. Brokered system.

or memory modules that can be rearranged by switching some of them off or on. It allows for the parameterization of buses adapting the bandwidth to the present needs. The productive system (called functional layer in figure 4.9) is monitored and controlled by an O/C layer (called autonomic layer). The observer not only monitors load data but it has to aggregate these data over time in order to determine the load situation. The controller works as a configurator changing directly the working modes of the subsystems. It could even redefine whole subsystems by loading FPGA data into the circuit. In contrast to the multi-mode systems coexisting without interaction, the subsystems are closely connected. This system can be classified as $O/C(multiple, 1)$.

4.6.4 Brokered system

A *Brokered System* consists of semi-autonomous elements such as controllers in an automotive electronics architecture, where each controller is responsible for a certain function. To build a higher-level function, an active agent with the "idea" of this function in "mind" searches for cooperation partners, assesses their capabilities, makes a contract, and supervises their performance. Such an architecture has been proposed by Hofmann (Evoarch [11]) and further developed on the basis of ontologies by Mahmoudi [12].

The broker plays the role of a central matching and assessment service enabling a marketplace mechanism. It might be implemented as a centralized or a distributed entity. It monitors the agents, aggregates their states and predicts future demands. From this, a viable solution (optimal under the given circumstances, fulfilling at least the minimum requirements) is derived and set into action (figure 4.10). This OC system can be classified as $O/C(multiple, 2)$.

4.6.5 Autonomous self-organizing agents

A decentralized traffic control system (cf. figure 4.11) as developed by Rochner, Prothmann et al. [19] consists of cooperating and learning

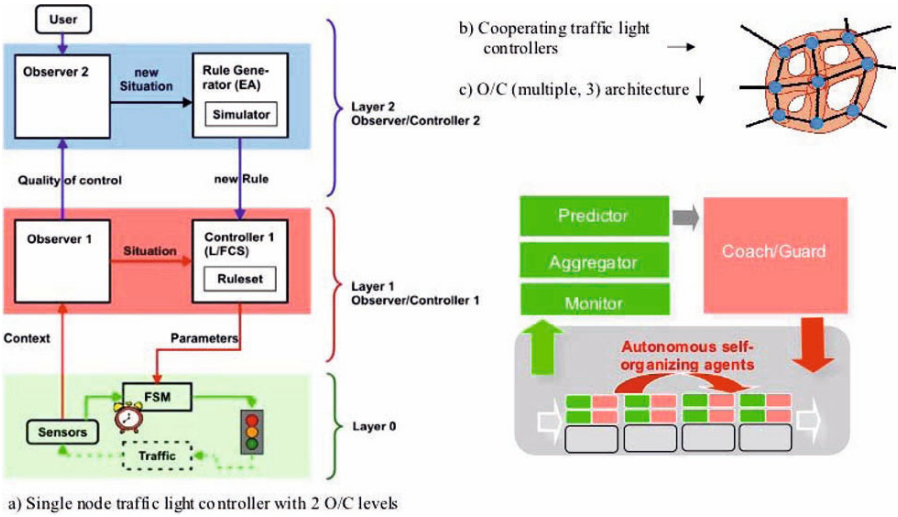


Fig. 4.11. Autonomous self-organizing agents, cooperating in a distributed traffic control system.

traffic light controllers. Each controller is built as an $O/C(single, 2)$ system. I.e., a traditional traffic light controller – as the productive system – is controlled and modified by a rule-based level-1 O/C, which in turn receives new rules from a level-2 model-based O/C (figure 4.11a).

While the single node controller has already been implemented in a prototypical version, the cooperation of the single node controllers is still subject of research. It is planned to realize a distributed scheme of data exchange between the nodes with a locally limited horizon (figure 4.11b). The level-3 O/C can be a physically centralized entity as shown in figure 4.11c or partially or fully distributed over the nodes. It has the task to monitor, aggregate, and predict the collective behavior of all traffic light controllers, assess their proper function and intervene, whenever necessary. The fully implemented architecture will be of type $O/C(multiple, 3)$.

4.6.6 Roadmap

We can use the (S, V) diagram to sketch a roadmap for the future development of OC systems (figure 4.12, with the example architectures discussed above). The variability of the systems will grow with the degree of distribution. This means at the same time that the task to control them will become increasingly difficult. The only possibility to manage this complexity will be a hierarchical approach with multiple levels of O/Cs, each of them responsible for its local or regional subsystem. As we climb up to higher O/C levels their tasks will become less intervening and more guarding, correcting, and limiting – very much like the high-level management of a company that normally will not

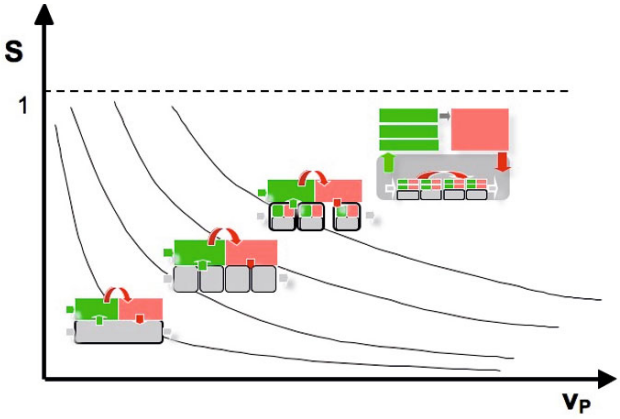


Fig. 4.12. (S, V) diagram as a roadmap for future OC systems, with the example architectures discussed above.

interfere with low-level decisions, as long as they conform to the high-level directives.

4.7 Summary and conclusion

In this chapter we have discussed various historical and recent technical concepts of emergence, have described a generic architectural template for the observation and control of adaptable systems, and have proposed a possibility for a quantitative assessment not only of emergence but also of self-organization. We can state that our technology-oriented notion of neo-emergence [7] is compatible with existing definitions: It is somewhere between weak and strong emergence in the sense of philosophy of mind. It is typically due to a self-organizing process [5] with various reasons [21, 22], is either weak or multiple in the sense of Fromm [8], and can be measured by means of order [10]. With the measures for emergence and self-organization proposed in this chapter it becomes possible to formulate a roadmap for the future development of OC systems and to classify them according to a simple description scheme. Future research will focus on the practical implications of these concepts.

Acknowledgments

This chapter is meant as an overview of the state of the art. Therefore it contains some work already published elsewhere. Much of this work originates from the many discussions within the DFG Priority Program "Organic Computing", especially within the projects "Quantitative Emergence", "Organic Traffic Control", and "On-line Fusion of Functional Knowledge within Distributed Sensor Networks". Key researchers involved in these projects are, in

addition to the authors of this chapter, H. Schmeck, J. Branke, U. Richter, M. Mnif, F. Rochner, H. Prothmann, and O. Buchtala.

References

1. R. Abbott. Emergence explained: Getting epiphenomena to do real work. Technical Report; online: `http://cs.calstatela.edu/abbott/`, 2005.
2. A. Bouajila, J. Zeppenfeld, W. Stechele, A. Herkersdorf, A. Bernauer, O. Bringmann, and W. Rosenstiel. Organic Computing at the System on Chip Level. In *Proceedings of the IFIP International Conference on Very Large Scale Integration of System on Chip (VLSI-SoC 2006)*. Springer, October 2006.
3. O. Buchtala and B. Sick. Functional knowledge exchange within an intelligent distributed system. In: P. Lukowicz, L. Thiele, G. Tröster (Eds.): *Architecture of Computing Systems - ARCS 2007*, volume 4415 of LNCS pages 126–141, Springer, 2007.
4. H. Cruse, J. Dean, and H. Ritter. Die Entdeckung der Intelligenz oder Können Ameisen denken? Deutscher Taschenbuch Verlag, München, 2001.
5. T. De Wolf and T. Holvoet. Emergence versus self-organisation: Different concepts but promising when combined. In S. Brueckner, G. di Marzo Serugendo, A. Karageorgos, R. Nagpal, editors, *Engineering Self-Organising Systems, Methodologies and Applications*, volume 3464 of LNCS, pages 1–15. Springer, 2005.
6. T. De Wolf, G. Samaey, T. Holvoet, and D. Roose. Decentralized autonomic computing: Analysing self-organising emergent behavior using advanced numerical methods. In: *Proceedings of the Second International Conference on Autonomic Computing (ICAC 2005), Seattle*, pages 52–63, 2005.
7. G. di Marzo Serugendo, M. P. Gleizes, and A. Karageorgos. Self-organisation and emergence in MAS: An overview. *Informatica*, 30(1):45–54, 2006.
8. J. Fromm. Types and forms of emergence. Technical Report; online: `http://arxiv.org/abs/nlin.AO/0506028`, 2005.
9. J. Fromm. Ten questions about emergence. Technical Report; online: `http://arxiv.org/abs/nlin.AO/0509049`, 2005.
10. J. M. E. Gabbai, H. Yin, W. A. Wright, and N. M. Allinson. Self-organization, emergence and multi-agent systems. In: *Proceedings of the International Conference on Neural Networks and Brain (ICNN&B '05), Beijing*, volume 3, pages 1858–1863, 2005.
11. P. E. H. Hofmann and S. F. Leboch. Evolutionäre Elektronikarchitektur für Kraftfahrzeuge: Selbstorganisierende Strukturen im Automobil. *it– information technology*, special issue on Organic Computing, 47:188-200, Oldenbourg Verlag, 2005.
12. G. Mahmoudi and C. Müller-Schloer. Towards Ontology-based Embedded Services. In L. T. Yang, H. Jin, J. Ma, T. Ungerer, editors, *Autonomic and Trusted Computing – Proceedings of the 3rd International Conference on Autonomic and Trusted Computing (ATC-06), Wuhan and Three Gorges*, volume 4158 of LNCS, pages 100–112, Springer, 2006.
13. M. Mnif M and C. Müller-Schloer. Quantitative Emergence. In: *Proceedings of the 2006 IEEE Mountain Workshop on Adaptive and Learning Systems (IEEE SMCals 2006)*, pp 78-84, Logan, 2006.

14. C. Müller-Schloer. Organic Computing – On the Feasibility of Controlled Emergence, In: *CODES + ISSS 2004 Proceedings, September 8-10, 2004*, pages 2–5, ACM Press, 2004.
15. C. Müller-Schloer, M. Mnif, E. Cakar, H. Schmeck, and U. Richter. Adaptive and Self-organising Systems, Submitted to: *ACM Transactions on Computational Logic*.
16. C. Müller-Schloer and B. Sick. Emergence in organic computing systems: discussion of a controversial concept. In L. T. Yang, H. Jin, J. Ma, T. Ungerer, editors, *Autonomic and Trusted Computing – Proceedings of the 3rd International Conference on Autonomic and Trusted Computing (ATC-06), Wuhan and Three Gorges*, volume 4158 of LNCS, pages 1–16, Springer, 2006.
17. C. Müller-Schloer, C. v.d. Malsburg, R. P. Würtz. Organic Computing, *Informatik Spektrum* 27(4):332–336, 2004.
18. U. Richter, M. Mnif, J. Branke, C. Müller-Schloer, and H. Schmeck. Towards a generic observer/controller architecture for Organic Computing, In: *Informatik 2006, GI-Jahrestagung, 02.-06.10.06, Dresden, Proceedings*, volume 1, pages 112–119, Springer, 2006.
19. F. Rochner, H. Prothmann, J. Branke, C. Müller-Schloer, and H. Schmeck. An Organic Architecture for Traffic Light Controllers. In: *Informatik 2006, GI-Jahrestagung, 02.-06.10.06, Dresden, Proceedings*, volume 1, pages 120–127, Springer, 2006.
20. T. Schöler and C. Müller-Schloer. An Observer/Controller Architecture for Adaptive Reconfigurable Stacks, In: *Proceedings ARCS 05*, pages 139–153, Springer, 2005.
21. A. Stephan. *Emergenz: Von der Unvorhersagbarkeit zur Selbstorganisation*. 2nd edition, Mentis, Paderborn, 2005.
22. A. Stephan. Zur Rolle des Emergenzbegriffs in der Philosophie des Geistes und in der Kognitionswissenschaft. In: D. Sturma, editor, *Philosophie und Neurowissenschaften*, pages 146-166. Volume 1770 in Taschenbuch Wissenschaft. Suhrkamp, Frankfurt am Main, 2006.
23. www.wikipedia.org, Website of Wikipedia Encyclopedia (online: `http://en.wikipedia.org/wiki/Self-organization`)
24. www.wikipedia.org, Website of Wikipedia Encyclopedia (online: `http://de.wikipedia.org/wiki/Holon`)

5

Organic Computing and Complex Dynamical Systems – Conceptual Foundations and Interdisciplinary Perspectives

Klaus Mainzer

Chair for Philosophy of Science, Institute of Interdisciplinary Informatics, University of Augsburg, 86135 Augsburg, Germany.
klaus.mainzer@phil.uni-augsburg.de

Summary. Complex computing systems begin to overwhelm the capacities of software developers and administrators. Self-organization has been a successful strategy of evolution to handle the increasing complexity of organisms with the emergence of novel structures and behavior. Thus, self-organization and emergence are fundamental concepts of organic computing. But these concepts are often used in a more or less intuitive manner. In the theory of complex systems and nonlinear dynamics, self-organization and emergence can be mathematically defined. Actually, these concepts are independent of biological applications, but universal features of dynamical systems. We get an interdisciplinary framework to understand self-organizing complex systems and to ask for applications in organic computing. In technology, the emergence of order and structures displays desired and undesired synergetic effects. Thus, controlled emergence is a challenge of computational systems simulating self-organizing organic systems of evolution. The question arises how far can we go in simulating high dimensional complex systems and avoiding uncontrolled risks.

Key words: Complex systems, nonlinear dynamics, controlled emergence, self-organization, computational dynamics

5.1 From linear to nonlinear dynamics

A dynamical system is a time-depending multi-component system of elements with local states determining a global state of the whole system. In a planetary system, for example, the state of a planet at a certain time is determined by its position and momentum. The states can also refer to moving molecules in a gas, the excitation of neurons in a neural network, nutrition of organisms in an ecological system, supply and demand of economic markets, the behavior of social groups in human societies, routers in the complex network of the Internet, or units of a complex electronic equipment in a car. The dynamics of a system, i.e. the change of the system's states depending on time, is represented

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_5,

by linear or nonlinear differential equations. In the case of nonlinearity, several feedback activities take place between the elements of the system. These many-bodies problems correspond to nonlinear and non-integrable equations with instabilities and sometimes chaos [9].

From a philosophical point of view, mathematical linearity means a strong concept of causality with similar causes or inputs of a dynamical system leading to similar effects or outputs: small changes in the parameters or small perturbations added to the values of the variables produce small changes in subsequent values of the variables. Further on, composed effects of linear systems can be reduced to the sum of more simple effects. Therefore, scientists have used linear equations to simplify the way in which we think about the behavior of complex systems. The principle of superposition has its roots in the concept of linearity. But, in the case of nonlinearity, similar causes lead to exponentially separating and expanding effects: small changes in the parameters or small perturbations added to the values of the variables can produce enormous changes in subsequent values of the variables because of the sensitivity to initial conditions. In this case, the whole is more than the sum of its elements.

The mathematical theory of nonlinear dynamics distinguishes different types of time-depending equations, generating different types of behavior, such as fixed points, limit cycles, and chaos. In a top-down approach of model building, we start with an assumed mathematical model of a natural or technical system and deduce its behavior by solving the corresponding dynamical equations under certain initial conditions. The solutions can be represented geometrically as trajectories in the phase space of the dynamical system and classified by different types of attractors. But, in practice, we often must take the opposite way of a bottom-up approach. Physicists, chemists, biologists, physicians, or engineers start with data mining in an unknown field of research. They only get a finite series of measured data corresponding to time-depending events of a dynamical system. From these data they must reconstruct the behavior of the system in order to guess its type of dynamical equation. Therefore, the bottom-up approach is called time series analysis. In many cases, we have no knowledge of the system from which the data came. Time series analysis then aims to construct a black box, which take the measured data as input and provides as output a mathematical model describing the data [13]. In practice, the realistic strategy of research is a combination of the top-down approach with model building and the bottom-up approach with time series analysis of measured data.

In classical measurement theory, measurement error is analyzed by statistical methods, such as correlation coefficient and autocorrelation function. But these standard procedures are not able to distinguish between data from linear and nonlinear models. In nonlinear data analysis, the measured data are used in a first step to reconstruct the dynamics of the system in a phase space. Nonlinear dynamical systems generating chaos must be determined by at least three equations. For example, a three-dimensional attractor is gen-

erated in a phase space with three coordinates $x(t)$, $y(t)$, and $z(t)$, which are determined by three time-depending nonlinear differential equations. But, in practice, it is often difficult to distinguish several variables of a system. Nevertheless, if only one variable can be measured, an attractor with a finite number of dimensions can be reconstructed from the measured time series with great similarity to the original attractor of the system. We must only assume that we can also measure the derivative of that variable, and further higher order derivatives up to some finite level d. Then, if the dimension of the system is less than d, we have enough information to completely describe the system with d differential or difference equations d derivatives is equivalent to measuring the system at d different time intervals. Therefore, according to Taken's theorem, the measured time series of a variable can be embedded in a reconstructed phase space with d dimensions. The sequence of points created by embedding the measured time series is a reconstructed trajectory of the original system, generating an attractor with great similarity to the original one of the system.

In practice, decisions about chaotic dynamics are rather difficult. How can we decide that a time series of measured data is not generated by noisy irregularity but by highly structured chaotic attractors? A chaotic attractor is determined by a trajectory in a bounded region of a phase space with aperiodic behavior and sensitive dependence on initial conditions. These criteria – determinism, boundedness, aperiodicity, and sensitivity – can be checked by several techniques of time series analysis. In the case of noise, the trajectories spread unbounded all over the phase space. A chaotic attractor is finite and always bounded in a certain region of the phase space. Aperiodicity means that the states of a dynamical system never return to their previous values. But values of states may return more or less to the vicinity of previous values. Thus, aperiodicity is a question of degree which can be studied in recurrence plots of measured points. Such plots depict how the reconstructed trajectory recurs or repeats itself. The correlation integral defines the density of points in a recurrence plot where the measured time series are closer than a certain degree of distance.

If a time series is generated by a chaotic system, the trajectory of the time series, which is reconstructed from the measurement data of embedding, has the same topological properties as the original attractor of the system, as long as the embedding dimension is large enough. Taken proved a method for finding an appropriate embedding dimension for the reconstruction of an attractor. But this method yields no procedure for finding a chaotic attractor, because its existence has been already assumed in order to determine its dimension from the measurement data.

Another way to characterize chaotic dynamics is to measure the strength of their sensitive dependence on initial data. Consider two trajectories starting nearly the same initial data. In chaotic dynamics only a tiny difference in the initial conditions can result in the two trajectories diverging exponentially quickly in the phase space after a short period of time. In this case,

it is difficult to calculate long-term forecasts, because the initial data can only be determined with a finite degree of precision. Tiny deviations in digits behind the decimal point of measurement data may lead to completely different forecasts. This is the reason why attempts to forecast weather fail in an unstable and chaotic situation. In principle, the wing of a butterfly may cause a global change of development. This "butterfly effect" can be measured by the so-called Lyapunov exponent. A trajectory $\mathbf{x}(t)$ starts with an initial state $\mathbf{x}(0)$. If it develops exponentially fast, then it is approximately given by $|\mathbf{x}(t)| \sim |\mathbf{x}(0)|e^{\lambda t}$. The exponent is smaller than zero if the trajectory is attracted by attractors, such as stable points or orbits. It is larger than zero if it is divergent and sensitive to very small perturbations of the initial data.

An attractor is typically finite in the phase space. Sensitivity to initial conditions means that any nearby points on the attractor in the phase space diverge from each other. They cannot, however, diverge forever, because the attractor is finite. Thus, trajectories from nearby initial points on the attractor diverge and are folded back onto the attractor, diverge and are folded back, etc. The structure of the attractor consists of many fine layers, like an exquisite pastry. The closer one looks, the more detail in the adjacent layers of the trajectories is revealed. Thus, the attractor is fractal. An attractor that is fractal is called strange. There are also chaotic systems that are only exponentially sensitive to initial conditions but not strange. Attractors can also be strange (fractal), but not chaotic. The fractal dimension of an attractor is related to the number of independent variables needed to generate the time series of the values of the variables. If d is the smallest integer greater than the fractal dimension of the attractor, then the time series can be generated by a set of d differential equations with d independent variables. For example, a strange attractor of fractal dimension 2.03 needs three nonlinear coupled differential equations generating its trajectory.

In summary, dynamical systems can be classified by attractors with increasing complexity from fixed points, periodic and quasi-periodic up to chaotic behavior. This classification of attractors can be characterized by different methods, such as typical patterns of time series, their power spectrum, phase portraits in a phase space, Lyapunov exponents, or fractal dimensions. A remarkable measure of complexity is the KS (Kolmogorov-Sinai) entropy, measuring the information flow in a dynamical system [3, 9]. A dynamical system can be considered an information processing machine, computing a present or future state as output from an initial past state of input. Thus, the computational efforts to determine the states of a system characterize the computational complexity of a dynamical system. The transition from regular to chaotic systems corresponds to increasing computational problems, according to the computational degrees in the theory of computational complexity. In statistical mechanics, the information flow of a dynamical system describes the intrinsic evolution of statistical correlations between its past and future states. The KS-entropy is an extremely useful concept in studying the loss of predictable information in dynamical systems, according to the complex-

ity degrees of their attractors. Actually, the KS-entropy yields a measure of the prediction uncertainty of a future state provided the whole past is known (with finite precision).

In the case of fixed points and limit cycles, oscillating or quasi-oscillating behavior, there is no uncertainty or loss of information, and the prediction of a future state can be computed from the past. Consequently, the KS-entropy is zero. In chaotic systems with sensitive dependence on the initial states, there is a finite loss of information for predictions of the future, according to the decay of correlations between the past states and the future state of prediction. The finite degree of uncertainty of a predicted state increases linearly to its number of steps in the future, given the entire past. In the case of chaos, the KS-entropy has a finite value (larger than zero). But in the case of noise, the KS-entropy becomes infinite, which means a complete loss of predicting information corresponding to the decay of all correlations (i.e., statistical independence) between the past and the noisy state of the future. The degree of uncertainty becomes infinite.

5.2 Self-organization and controlled emergence in nature

How can the knowledge of chaos be applied in order to avoid or control risky situations? This question will be a challenge for organic computing. It seems to be paradoxical that chaotic systems, which are extremely sensitive to the tiniest fluctuations, can be controlled. But nowadays the control of chaos has been realized in chemical, fluid, and biological systems. In technology, for example, the intrinsic instability of chaotic celestial orbits is routinely used to advantage by international space agencies that divert spacecraft to travel vast distances using only modest fuel expenditures. All techniques of chaos control make use of the fact that chaotic systems can be controlled if disturbances are countered by small and intelligently applied impulses. Just as an acrobat balances about an unstable position on a tightrope by the application of small correcting movements, a chaotic system can be stabilized about any of an infinite number of unstable states by continuous application of small corrections.

Two characteristics of chaos make the application of control techniques possible. First, chaotic systems alternatively visit small neighborhoods of an infinite number of periodic orbits. The presence of an infinite number of periodic orbits embedded within a chaotic trajectory implies the existence of an enormous variety of different behaviors within a single system. Thus, the control of chaos opens up the potential for a great flexibility in operating performance within a single system.

A second characteristic of chaos important for control applications is its exponential sensitivity. It follows that the state of chaotic system can be drastically altered by the application of small perturbations. Therefore, uncontrolled chaotic systems fluctuate wildly. But, on the other side, controlled

chaotic systems can be directed from one state to a very different one using only very small controls. Obviously, controlling strategies require that the system state lie close to the desired state. In such a case, the system dynamics can be linearized, making control calculations rapid and effective. In chaotic systems, ergodicity ensures that the system state will eventually wander arbitrarily close to the desired state. But in higher-dimensional or slowly varying systems, the time taken for the state to move on its own from one state to another can be prohibitive. In this case, fully nonlinear control strategies have been devised that use chaotic sensitivity to steer the system state from any given initial point to a desired state. Since chaotic systems amplify control impulses exponentially, the time needed to steer such a system can be quite short. These strategies have been demonstrated both in systems in which a large effect is desired using very modest parameter expenditures (e.g., energy and fuel) and in systems in which rapid switching between states is needed (e.g., computational and communication applications).

Nonlinear dynamics does not only yield chaos and noise, but also order. The emergence of order and structures in nature can be explained by the dynamics of attractors in complex systems [9]. They result from collective patterns of interacting elements in the sense of many-bodies problems that cannot be reduced to the features of single elements in a complex system. Nonlinear interactions in multicomponent ("complex") systems often have synergetic effects, which can neither be traced back to single causes nor be forecast in the long run or controlled in all details. Again, the whole is more than the sum of its parts. This popular slogan for emergence is precisely correct in the the sense of nonlinearity.

The mathematical formalism of complex dynamical systems is taken from statistical mechanics. If the external conditions of a system are changed by varying certain control parameters (e.g., temperature), the system may undergo a change in its macroscopic global states at some critical point. For instance, water as a complex system of molecules changes spontaneously from a liquid to a frozen state at a critical temperature of zero degrees Celsius. In physics, those transformations of collective states are called phase transitions. Obviously they describe a change of self-organized behavior between the interacting elements of a complex system.

According to E. Landau, the suitable macrovariables characterizing the change of global order are denoted as "order parameters". For example, the emergence of magnetization in a ferromagnet is a self-organized behavior of atomic dipoles that is modeled by a phase transition of an order parameter, the average distribution of microstates of the dipoles, when the system is annealed to the Curie-point. The concept of order parameters can be generalized for phase transitions when the system is driven away from equilibrium by increasing energy. If, for example, the fluid of a stream is driven further and further away from thermal equilibrium, by increasing fluid velocity (control parameter), then fluid patterns of increasing complexity emerge from vortices of fixed points, periodic oscillations to chaotic turbulence. Roughly speaking,

we may say that old structures become unstable, broken down by changing control parameters, and new patterns and attractors emerge.

More mathematically, nonlinear differential equations are employed to model the dynamics of the system. At first, we study the behavior of the elements on the microlevel in the vicinity of a critical point of instability. In a linear stability analysis, one can distinguish stable and unstable modes which increase to the macroscopic scale, dominating the macrodynamics of the whole system. Thus, some few unstable modes become the order parameters of the whole system. From a methodological point of view, the introduction of order parameters for modeling self-organization and the emergence of new structures is a giant reduction of complexity. The study of, perhaps, billions of equations, characterizing the behavior of the elements on the microlevel, is replaced by some few equations of order parameters, characterizing the macrodynamics of the whole system. Complex dynamical systems and their phase transitions deliver a successful formalism to model self-organization and emergence. Further on, the knowledge of characteristic order parameters and critical values of control parameters open a chance to influence the whole dynamics and to create desired states of technical systems by self-organization. The formalism does not depend on special, for example, physical laws, but must be appropriately interpreted for biological and technical applications.

According to the general scheme of nonlinear dynamics, biological organisms function on many levels that have emerged step by step during evolution. It is a question of granulation how "deep" we like to lay the initial layer of microdynamics. As far as we know at least atomic dynamics influence states of living organisms. During prebiotic evolution, interacting atoms and molecules created complex biomolecules (e.g., proteins) by catalytic and autocatalytic processes which are the building blocks of cells. Interacting cells achieved complex cellular systems like organs or organisms which are elements of populations. Further on, interacting populations became elements of ecological networks as examples of complex systems. Thus, from the nonlinear dynamics at each level, there emerge new entities that are characterized by order parameters. Examples of order parameters are characteristic macroscopic features of phenotypes which are determined by the genotype of an organism on the microlevel. The macrodynamics of these order parameters determine the microdynamics of the new entities, providing the basis of macrodynamics on the following level. In principle, the dynamics of each level can be modeled by appropriate nonlinear differential equations. In this case, the succeeding hierarchical level can be mathematically derived from the previous one by a linear stability analysis.

How can order be regulated and controlled in living organisms? This is a key-question for applications in organic computing. Bio-oscillators are an important example since they can be considered order parameters of life. Nature abounds with rhythmic behavior that closely intertwines the physical and biological sciences. The diurnal variations in dark and light give rise to circadian physiological rhythms. But the rhythmic nature of biological pro-

cesses is not only controlled by external processes. It often arises from the intrinsic dynamics of complex nonlinear networks. Since all biological systems are thermodynamically open to the environment they are dissipative, that is, they dispense energy to their surroundings in the form of heat. Thus, for the oscillator to remain periodic, energy must be supplied to the system in such a way as to balance the continual loss of energy due to dissipation. If a balance is maintained, then the phase space orbits become a stable limit cycle, that is, all orbits in the neighborhood of this orbit merge with it asymptotically. Such a system is called a bio-oscillator, which left to itself begins to oscillate without apparent external excitation. The self-generating or self-regulating features of bio-oscillators depend on the intrinsic nonlinearity of the biological system.

How can perturbations of such systems be used to explore and control their physiological properties? This question does not only inspire new therapeutic methods in medicine, but also technical applications in organic computing. An example of a complex cellular system is the heart, which can be considered a bio-oscillator [1]. In the simple case of an embryonic chick heart, a cardiac oscillator can be described by a system of ordinary differential equations with a single unstable steady state and displaying an asymptotically stable limit cycle oscillation that is globally attracting. After short perturbations, the pulses return quickly to the limit cycle. The dynamics can be studied in the corresponding time series of ECG-curves. The dynamics of a mammalian heart is much more complex. The question arises if observed fluctuations are the result of the oscillations being unpredictably perturbed by the cardiac environment, or are a consequence of cardiac dynamics given by a chaotic attractor, or both. In healthy patients, the heart rate is modulated by a complex combination of respiratory, sympathetic, and parasympathetic regulators. For ill patients, the ideas of chaos control can be incorporated into therapeutic situations. Control is attempted by stimulating the heart at appropriate times. Repeated intervention prevents the rhythm from returning to the chaotic mode.

Obviously, the total and global chaos of a system is dangerous. But local chaotic fluctuations are physiologically advantageous. Sustained periodicities are often unhealthy. To maintain health, the variables of a physiological system must be able to extend over a wide range to provide flexible adaptation. Healthy people have greater variability in their heart rates than those with heart disease. Thus, local chaotic fluctuations may provide the plasticity to cope with the exigencies of an unpredictable and changing environment.

Chaotic systems can be controlled more finely and more quickly than linear systems. In linear systems, the response of the output depends linearly on the input. Small changes in a parameter of a linear system produce only small changes in the output. The variable controlling a chaotic physiological response may need to change only a small amount to induce the desired large change in the physiological state. Moreover, a chaotic physiological system can switch very rapidly from one physiological state to another. Natural self-control and self-organization of complex physiological systems open a wide range of medical and engineering applications.

The traditional notion of health is one of homeostasis and is based on the idea that there exists an ideal state in which the body is operating in a maximally efficient way. In this opinion, illness is considered to be the deviation of the body from this state, and it is the business of the physician to assist the patient in regaining this state again. The nonlinear dynamics of biological systems suggest to replace homeostasis by homeodynamics allowing a more flexible view of how the systems work and making room for the concept of systems with complex responses, even to the point of inherent instability. The mammalian organism is composed of multiple nested loops of nonlinear interacting systems on the physiological level. How much greater are the possibilities for complex behavior at the psychic levels of the brain.

The coordination of the complex cellular and organic interactions in an organism needs a new kind of self-organizing controlling. That was made possible by the evolution of nervous systems that also enabled organisms to adapt to changing living conditions and to learn from experiences with its environment. The hierarchy of anatomical organization varies over different scales of magnitude, from molecular dimensions to that of the entire central nervous system (CNS). The research perspectives on these hierarchical levels may concern questions, for example, of how signals are integrated in dendrites, how neurons interact in a network, how networks interact in a system like vision, how systems interact in the CNS, or how the CNS interact with its environment. Each stratum may be characterized by some order parameters determining its particular structure, which is caused by complex interactions of elements with respect to the particular level of hierarchy.

In order to model the brain and its complex abilities, it is quite adequate to distinguish the following categories. In neuronal-level models, studies are concentrated on the dynamic and adaptive properties of each nerve cell or neuron, in order to describe the neuron as a unit. In network-level models, identical neurons are interconnected to exhibit emergent system functions. In nervous-system-level models, several networks are combined to demonstrate more complex functions of sensory perception, motor functions, stability control, etc. In mental-operation-level models, the basic processes of cognition, thinking, problem-solving, etc. are described.

In the complex systems approach, the microscopic level of interacting neurons should be modeled by coupled differential equations modeling the transmission of nerve impulses by each neuron. The Hodgkin-Huxley equation is an example of a nonlinear reaction-diffusion equation with an exact solution of a traveling wave, giving a precise prediction of the speed and shape of the nerve impulse of electric voltage. In general, nerve impulses emerge as new dynamical entities like ring waves in BZ-reactions or fluid patterns in nonequilibrium dynamics. In short: they are the "atoms" of the complex neural dynamics. On the macroscopic level, they generate a cell assembly whose macrodynamics is dominating order parameters. For example, a synchronously firing cell assembly represents some visual percept of a plant, which is not only the sum of its perceived pixels, but characterized by some typical macroscopic features like

form, background or foreground. On the next level, cell assemblies of several percepts interact in a complex scenario. In this case, each cell assembly is a firing unit, generating an assembly of cell assemblies, whose macrodynamics is characterized by some order parameters. The order parameters may represent similar properties between the perceived objects.

In this way, we get a hierarchy of emerging levels of cognition, starting with the microdynamics of firing neurons. The dynamics of each level is assumed to be characterized by differential equations with order parameters. For example, on the first level of macrodynamics , order parameters characterize a visual percept. On the following level, the observer becomes conscious of the percept. Then the cell assembly of perception is connected with the neural area that is responsible for states of consciousness. In a next step, a conscious perception can be the goal of planning activities. In this case, assemblies of cell assemblies are connected with neural areas in the planning cortex, and so on. They are represented by coupled nonlinear equations with firing rates of corresponding cell assemblies. Even high-level concepts like self-consciousness can be explained by self-reflections of self-reflections, connected with a personal memory, which is represented in corresponding cell assemblies of the brain. Brain states emerge, persist for a small fraction of time, then disappear and are replaced by other states. It is the flexibility and creativeness of this process that makes a brain so successful in animals for their adaptation to rapidly changing and unpredictable environments.

5.3 Self-organization and controlled emergence in computational, information, and communicating systems

Organic Computing applies principles of natural dynamical systems to technical systems. Dominating principles in the complex world of evolution are self-organization and self-control. How can they be realized in technical systems? Computational automata are a nice test bed for all kinds of technical systems. There is a precise relation between self-organization of nonlinear systems with continuous dynamics and discrete cellular automata. The dynamics of nonlinear systems is given by differential equations with continuous variables and a continuous parameter of time. Sometimes, difference equations with discrete time points are sufficient. If even the continuous variables are replaced by discrete (e.g., binary) variables, we get functional schemes of automata with functional arguments as inputs and functional values as outputs. There are classes of cellular automata modeling attractor behavior of nonlinear complex systems, which is well-known from self-organizing processes. But in many cases, there is no finite program, in order to forecast the development of random patterns. Thus, pattern emergence of cellular automata cannot be controlled in any case.

Cellular automata (CA) are only a theoretical concept of computational dynamics. In electrical engineering, information and computer science, the

concept of cellular neural networks (CNN) has recently become an influential paradigm of complexity research and is being realized in information and chip technology [2, 9]. CNNs has been made possible by the sensor revolution of the late 1990s. Cheap sensors and MEMS (micro-electro-mechanical system) arrays have become popular as artificial eyes, noses, ears, taste and somatosensory devices. An immense number of generic analog signals have been processed. Thus, a new kind of chip technology, similar to signal processing in natural organisms, is needed. Analog cellular computers are the technical response to the sensor revolution, mimicking the anatomy and physiology of sensory and processing organs. A CNN is their hard core, because it is an array of analog dynamic processors or cells.

In general, a CNN is a nonlinear analog circuit that processes signals in real time. It is a multi-component system of regularly spaced identical units, called cells that communicate directly with each other only through their nearest neighbors. The locality of direct connections is a natural principle which is also realized by brains and cellular automata (CA). Total connectivity would be energetically too expensive with the risk of information chaos. Therefore, it was selected by evolution of the brain and not applied in technology. Unlike conventional cellular automata, CNN host processors accept and generate analog signals in continuous time with real numbers as interaction values. The dynamics of a cell's state are defined by a nonlinear differential equation (CNN state equation) with scalars for state, output, input, threshold, and coefficients, called synaptic weights, modeling the intensity of synaptic connections of the cell with the inputs and outputs of the neighbor cells. The CNN output equation connects the states of a cell with the outputs.

CNN arrays are extremely useful for standards in visual computing. Examples are CNNs that detect patterns in either binary (black-and-white) or grayscale input images. An image consists of pixels corresponding to the cells of CNN with binary or grayscale. From the perspective of nonlinear dynamics, it is convenient to think of standard CNN state equations as a set of ordinary differential equations. Contrary to the usual CA approach with only geometric pattern formation of cells, the dynamical behavior of CNNs can be studied analytically by nonlinear equations. Numerical examples deliver CNNs with limit cycles and chaotic attractors. For technical implementations of CNNs, such as silicon chips, complete stability properties must be formulated, in order to avoid oscillations, chaotic, and noise phenomena. These results also have practical importance for image processing applications of CNNs. As brains and computers work with units in two distinct states, the conditions of bistability are studied in brain research, as well as in chip technology.

CNNs are optimal candidates to simulate local synaptic interactions of neurons generating collective macro phenomena. Hallucinations, for example, are the results of self-organizing phenomena within the visual cortex. This type of pattern perception seems to be similar to pattern formation of fluids in chemistry or aerodynamics. Pattern formation in the visual brain is due to local nonlinear coupling among cells. In the living organism, there is a

spatial transformation between the pattern perception of the retina and the pattern formation within the visual cortex of the brain. First simulations of this cortico-retinal transformation by CNNs generate remarkable similarities with pattern perceptions that are well-known from subjective experiences of hallucinations. Perceptions of a spiraling tunnel pattern have been reported by people who were clinically dead and later revived. The light at the end of the tunnel has sometimes been interpreted as religious experience.

CNNs with information processing in nanoseconds and even at the speed of light seem to be optimal candidates for applications in neurobionics. Obviously, there are surprising similarities between CNN architectures and, for example, the visual pathway of the brain. An appropriate CNN approach is called the "Bionic Eye", which involves a formal framework of vision models combined and implemented on the so-called CNN universal machine. Like a universal Turing machine, a CNN universal machine can simulate any specialized CNN and is technically constructed in chip technology. Visual illusions which have been studied in cognitive psychology can also be simulated by a universal CNN chip. The same architecture of a universal machine can not only be used to mimic the retinas of animals (e.g., of a frog, tiger salamander, rabbit, or eagle) but they can also be combined and optimized for technical applications. The combination of biological and artificial chips is no longer a science fiction-like dream of cyborgs, but a technical reality with inspiring ramifications for robotics and medicine.

In epileptology, clinical applications of CNN chips have already been envisaged. The idea is to develop a miniaturized chip device for the prediction and prevention of epileptic seizures. Nonlinear time series analysis techniques have been developed to characterize the typical EEG patterns of an epileptic seizure and to recognize the phase transitions leading to the epileptic neural states. These techniques mainly involve estimates of established criteria such as correlation dimension, Kolmogorov-Sinai-entropy, Lyapunov exponents, fractal similarity, etc. Implantable seizure prediction and prevention devices are already in use with Parkinson patients. In the case of epileptic processes, such a device would continuously monitor features extracted from the EEG, compute the probability of an impending seizure, and provide suitable prevention techniques. It should also possess both high flexibility for tuning to individual patient patterns and high efficacy to allow the estimation of these features in real time. Eventually, it should have low energy consumption and be small enough to be implemented in a miniaturized, implantable system. These requirements are optimally realized by CNNs , with their massive parallel computing power, analog information processing, and capacity for universal computing.

In complex dynamical systems of organisms monitoring and controlling are realized on hierarchical levels. Thus, we must study the nonlinear dynamics of these systems in experimental situations, in order to find appropriate order parameters and to prevent undesired emergent behavior as possible attractors. From the point of view of systems science, the challenge of organic computing is controlled emergence.

A key-application is the nonlinear dynamics of brains. Brains are neural systems which allow quick adaptation to changing situations during the lifetime of an organism. In short: They can learn, assess and anticipate. The human brain is a complex system of neurons self-organizing in macroscopic patterns by neurochemical interactions. Perception, emotions, thoughts, and consciousness correspond to these patterns. Motor knowledge, for instance, is learned in an unknown environment and stored implicitly in the distribution of synaptic weights of the neural nets. Technically, self-organization and pattern emergence can be realized by neural networks, working like brains with appropriate topologies and learning algorithms. A simple robot with diverse sensors (e.g., proximity, light, collision) and motor equipment can generate complex behavior by a self-organizing neural network. In the case of a collision with an obstacle, the synaptic connections between the active nodes for proximity and collision layer are reinforced by Hebbian learning: A behavioral pattern emerges, in order to avoid collisions in future [12]. In the human organism, walking is complex bodily self-organization, largely without central control by brain and consciousness: It is driven by the dynamical pattern of a steady periodic motion, the attractor of the motor system.

What can we learn from nature? In unknown environments, a better strategy is to define a low-level ontology, introduce redundancy – there is a lot in the sensory systems, for example – and leave room for self-organization. Low-level ontologies of robots only specify systems like the body, sensory systems, motor systems, and the interactions among their components, which may be mechanical, electrical, electromagnetic, thermal etc. According to the complex systems approach, the components are characterized by certain microstates generating the macrodynamics of the whole system.

Take a legged robot. Its legs have joints that can assume different angles, and various forces can be applied to them. Depending on the angles and the forces, the robot will be in different positions and behave in different ways. Further, the legs have connections to one another and to other elements. If a six-legged robot lifts one of the legs, this changes the forces on all the other legs instantaneously, even though no explicit connection needs to be specified. The connection is implicit: They are enforced through the environment, because of the robot's weight, the stiffness of its body, and the surfaces on which it stands. Although these connections are elementary, they are not explicit and included if the designer wished. Connections may exist between elementary components that we do not even realize. Electronic components may interact via electromagnetic fields that the designer is not aware of. These connections may generate adaptive patterns of behavior with high fitness degrees (order parameter). But they can also lead to sudden instability and chaotic behavior. In our example, communication between the legs of a robot can be implicit. In general, much more is implicit in a low-level specification than in a high-level ontology . In restricted simulated agents, only what is made explicit exists, whereas in the complex real world, many forces exist and properties obtain, even if the designer does not explicitly represent them. Thus, we must

study the nonlinear dynamics of these systems in experimental situations, in order to find appropriate order parameters and to prevent undesired emergent behavior as possible attractors.

But not only "low level" motor intelligence, but also "high level" cognition (e.g., categorization) can emerge from complex bodily interaction with an environment by sensory-motor coordination without internal symbolic representation. We call it "embodied cognition": An infant learns to categorize objects and to build up concepts by touching, grasping, manipulating, feeling, tasting, hearing, and looking at things, and not by explicit representations. The categories are based on fuzzy patchworks of prototypes and may be improved and changed during life. We have an innate disposition to construct and apply conceptual schemes and tools.

Moreover, cognitive states of persons depend on emotions. We recognize emotional expressions of human faces with pattern recognition of neural networks and react by generating appropriate facial expressions for non-verbal communication. Emotional states are generated in the limbic system of the brain, which is connected with all sensory and motor systems of the organism. All intentional actions start with an unconscious impulse in the limbic system, which can be measured before their performance. Thus, embodied intentionality is a measurable feature of the brain [4]. Humans use feelings to help them navigate the ontological trees of their concepts and preferences, to make decisions in the face of increasing combinatorial complexity: emotions help to reduce complexity.

The embodied mind is obviously a complex dynamical system acting and reacting in dynamically changing situations. The emergence of cognitive and emotional states is made possible by brain dynamics, which can be modeled by neural networks. According to the principle of computational equivalence [9], any dynamical system can be simulated by an appropriate computational system. But, contrary to Turing's AI-thesis, that does not mean computability in every case. In complex dynamical systems, the rules of locally interacting elements (e.g., Hebb's rules of synaptic interaction) may be simple and programmed in a computer model. But their nonlinear dynamics can generate complex patterns and system states which cannot be forecast in the long run without increasing loss of computability and information. Thus, artificial minds could have their own intentionality, cognitive and emotional states which cannot be forecast and computed like in the case of natural minds. Limitations of computability are characteristic features of complex systems.

In a complex dynamical world, decision making and acting is only possible under conditions of bounded rationality. Bounded rationality results from limitations on our knowledge, cognitive capabilities, and time. Our perceptions are selective, our knowledge of the real world is incomplete, our mental models are simplified, our powers of deduction and inference are weak and fallible. Emotional and subconscious factors effect our behavior. Deliberation takes time and we must often make decisions before we are ready. Thus, knowledge representation must not be restricted to explicit declarations. Tacit background

knowledge, change of emotional states, personal attitudes, and situations with increasing complexity are challenges of organic computing.

In a dramatic step, the complex systems approach has been expanded from neural networks to global computer networks like the Worldwide Web. The Internet can be considered as a complex open computer network of autonomous nodes (hosts, routers, gateways, etc.), self-organizing without central mechanisms. Routers are nodes of the network determining the local path of each information packet by using local routing tables with cost metrics for neighboring routers. These buffering and resending activities of routers can cause congestions in the Internet. Congested buffers behave in surprising analogy to infected people. There are nonlinear mathematical models describing true epidemic processes like the spread of malaria as well as the dynamics of routers. Computer networks are computational ecologies.

However, complexity of global networking does not only mean increasing numbers of PCs, workstations, servers, and supercomputers interacting via data traffic in the Internet. Below the complexity of a PC, low-power, cheap, and smart devices are distributed in the intelligent environments of our everyday world. Like GPS in car traffic, things in everyday life could interact remotely controlled by sensors. The real power of the concept does not come from any one of these single devices. In the sense of complex systems, the power emerges from the collective interaction of all of them. For instance, the optimal use of energy could be considered as a macroscopic order parameter of a household realized by the self-organizing use of different household goods according to confine consumption of electricity during special time-periods with cheap prices. The processors, chips, and displays of these smart devices don't need a user interface like a mouse, windows, or keyboards, but just a pleasant and effective place to get things done. Wireless computing devices on small scales become more and more invisible to the user. Ubiquitous computing enables people to live, work, use, and enjoy things directly without being aware of their computing devices.

A challenge of the automobile industry is the increasing complexity of electronic systems. If we consider the electronic cable systems of automobiles from the beginning through to today, there will be a surprising similarity to neural networks of organisms which increase in complexity during evolution. Contrary to biological evolution, electronic systems of today are rigid, compact, and flexible. In an evolutionary architecture (EvoArch) the nervous system of an automobile is divided into autonomous units (carlets) which can configure themselves into cooperative functions, in order to solve intelligent tasks [5]. They are the macroscopic features realized by interacting units in a complex system. Examples are the complex functions of motor, brake, and light, wireless guidance systems like GPS, smart devices for information processing, and the electronic infrastructure of entertainment. In EvoArch, there are several "self-x-features" with great similarity to self-organizing organic systems in biological evolution: Self-healing demands self-configuration and self-diagnosis. Self-diagnosis means error recognition and self-reflection, etc.

5.4 Perspectives of organic computing

Organic computing aims at the construction of self-organizing computing systems that display desired emergent behavior like organisms in natural evolution [6, 7, 11]. Emergence refers to a property of a system that is not contained in any of its parts. In the sense of nonlinear dynamical systems, the whole is more than the sum of its parts. In robotics, it concerns behavior resulting from the agent-environment interaction whenever the behavior is not preprogrammed. It is thus not common to use the term if the behavior is entirely prespecified like a trajectory of a hand that has been precalculated by a planner. Agents designed using high-level ontologies have no room for emergence, for novel behaviors. A domain or high-level ontology consists of a complete representation of the basic vocabulary, the primitives that are going to be used in designing the system. These are the only components that can be used: everything is built on top of these basic elements. The domain ontology remains constant for an extended period of time, often for the entire life of the system. A well-known example is the bounded knowledge representation of an expert system. High-level ontologies are therefore used whenever we know precisely in what environments the systems will be used, for traditional computational systems as well as for factory robot systems. In unknown environments, a better strategy is to define a low-level ontology and to introduce redundancy with a great variety of self-organization.

In the dynamical systems approach, we first need to specify what system we intend to model and then we have to establish the differential or difference equations. Time series analysis and further criteria of data mining help to construct the appropriate phase spaces, trajectories, and attractors. In organic computing, one approach would be to model an agent and its environment separately and then to model the agent-environment interaction by making their state variables mutually dependent [12]. The dynamical laws of the agent A and the environment E can be described by simplified schemes of differential equations $dx_a/dt = A(x_a, p_a)$ and $dx_e/dt = E(x_e, p_e)$, where x represents the state variables, such as angles of joints, body temperature, or location in space, and p parameters like thresholds, learning rates, nutrition, fuel supply and other critical features of change. Agents and environment can be coupled by defining a sensory function S and a motor function M. The environment influences the agent through S. The agent influences its environment through M. S and M constitute the agent-environment coupling, i.e. $dx_a/dt = A(x_a, S(x_e), p_a)$ and $dx_e/dt = E(x_e, M(x_a), p_e)$, where p_a and p_e are not involved in the coupling. Examples are walking or moving robots in environments with obstacles. In this case, the basic analysis problem can be stated in the following way: given an environment dynamics E, an agent dynamics A, and sensory and motor functions S and M, explain how the agent's observed behavior is generated.

One of the controllers of the dynamics evolves when the agent's angle sensors are turned off and cannot sense the position of its legs. In this case, the

activation levels of the neurons exhibit a limit cycle that causes the agent's single leg to stand and swing rhythmically. By that, it causes the robot to walk. The system's state repeatedly changes from the stance phase with the foot on the ground to the swing phase with the foot in the air and back. This example illustrates that the dynamical systems approach can be applied in a synthetic way in order to to design and to construct robots and their environments. But, in general, the dynamical systems approach is used in an analytical way: it starts from a given agent-environment interaction, which is formalized in terms of differential equations. The complex variety of behavior can be analyzed by solving, approximating, or simulating the equations, in order to find the attractors of dynamics. The dynamical attractors of the interacting system can be used to steer an agent or to let it self organize in a desired way.

Obviously, self-organization leads to the emergence of new phenomena on sequential levels of evolution. Nature has demonstrated that self-organization is necessary to manage the increasing complexity on these evolutionary levels [8]. But nonlinear dynamics can also generate chaotic behavior, which cannot be predicted and controlled in the long run. In complex dynamical systems of organisms monitoring and controlling are realized on hierarchical levels. There is still no final and unified theory of organic computing. We only know parts of biological, neural, cognitive, and social systems in the framework of complex dynamical systems. But even in physics, we have no unified theory of all physical forces. Nevertheless, scientists work successfully with an incomplete patchwork of theories. In order to know more about it, we need an interdisciplinary cooperation of technical, natural, computer, and cognitive science, and last but not least the humanities. The goal of organic computing is the construction of self-organizing computing systems as service for people, in order to manage a world of increasing complexity and to support a sustainable future of human infrastructure.

References

1. J. B. Bassingthwaighte, L. S. Liebovitch, and B. J. West. *Fractal Physiology.* Oxford University Press, New York, 1994.
2. L. O. Chua and T. Roska. *Cellular Neural Networks and Visual Computing. Foundations and Applications.* Cambridge University Press, Cambridge, 2002.
3. G. Deco and B. Schürmann. *Information Dynamics. Foundations and Applications.* Springer, New York, 2000.
4. W. J. Freeman. How and why brains create meaning from sensory information. *Int. J. Bifurcation and Chaos* 14:515–530, 2004.
5. P. H. Hofmann, et al. *Evolutionäre E/E-Architektur.* DaimlerChrysler, Esslingen, 2002.
6. P. Horn. *Autonomic Computing: IBM's Perspective on the State of Information Technology.* IBM, New York, 2001.
7. J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer Society* 1: 41-50, 2003.

8. K. Mainzer. *Symmetry and Complexity. The Spirit and Beauty of Nonlinear Science*. World Scientific Publisher, Singapore, 2005.
9. K. Mainzer. *Thinking in Complexity. The Computational Dynamics of Matter, Mind, and Mankind*. 4th enlarged edition, Springer, New York, 2004.
10. K. Mainzer. *KI – Künstliche Intelligenz. Grundlagen intelligenter Systeme*. Wissenschaftliche Buchgesellschaft, Darmstadt, 2003.
11. C. Müller-Schloer, C. (Ed.), 2005, Schwerpunktthema: Organic Computing – Systemforschung zwischen Technik und Naturwissenschaften. *it Information Technology* 4, 2005.
12. R. Pfeifer, and C. Scheier. *Understanding Intelligence*. MIT Press, Cambridge MA, 2001.
13. M. Small: *Applied Nonlinear Time Series Analysis. Applications in Physics, Physiology, and Finance*. World Scientific Publisher, Singapore, 2005.

6

Evolutionary Design of Emergent Behavior

Jürgen Branke, Hartmut Schmeck

Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany.
branke@aifb.uni-karlsruhe.de, schmeck@aifb.uni-karlsruhe.de

Summary. Most technical systems envisioned in organic computing are assumed to be complex, consisting of a large number of interacting components, self-organizing and exhibiting *emergent* behavior. As is argued in this chapter, a system's emergent properties surface only after realization or during a simulation of all interacting components. Thus, the usual "top-down" and "bottom-up" design paradigms have severe limitations when it comes to emergence. Instead, the use of evolutionary computation is advocated for the automated, simulation-based design of organic computing systems with emergent behavior.

Key words: Emergence, organic computing, evolutionary computation, design principles

6.1 Introduction

As a result of the continuing technological and scientific progress, the systems created by engineers, computer scientists and others become larger and larger, consisting of more and more highly interconnected and heterogeneous components. In short, they become ever more complex. On the other hand, the demand for reliability, adaptability and cost effectiveness remains or even increases. Being able to design and control such complex systems thus becomes a competitive necessity.

The organic computing initiative envisions computer systems of the future to be more flexible and adaptive, more autonomous, and with a stronger focus on user needs. In short, organic computers are self-organizing systems, able to dynamically adapt to a changing environment. They exhibit the so-called "self-x" properties, including self-organization, self-configuration, self-healing, self-adaptation, and self-protection. To achieve these goals, a life-like structure is envisioned, with many interacting, more or less autonomous subsystems, self-organizing into a coherent global system. This can be viewed to be analogous to complex natural systems, as e.g. societies of ants, termites, or wasps. An

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_6,

observer-controller structure [63, 22] is supposed to supervise and control the overall system.

On the one hand, such a structure relieves the designer from the need to foresee all possible events and rigidly program all system responses in every detail, as for example small faults can be self-repaired, the system can self-adapt to a variety of environments and users, and self-protect even against threats not yet existing at design time. On the other hand, the emergent behavior of such collections of interacting, self-adapting components poses new challenges to the designer.

Usually, emergence is defined as some global behavior of the overall system that can not be observed, and often not even deduced, from looking at the components individually, see, e.g., [46, 28]. Examples include traffic jams, the finding of shortest paths by ants, or the complex functionality of the brain consisting of many rather simple interconnected neurons. Note that the emergent behavior may be desired, as in the example of the brain, or undesired, as in the example of traffic jams.

Since emergent behavior only occurs if components are working together, the system's characteristics can not be derived by analyzing the individual parts, and analytical models of such complex systems usually don't exist[1]. But this also means that the usual "top-down" design principle is no longer appropriate. It is by definition impossible to know how design choices made on the component level affect the overall system behavior. Therefore, designing a set of interacting subsystems such that they generate a desired emergent global behavior , while preventing undesired emergent behavior, is a very challenging task.

This predicament has been discussed also by others, see e.g. [29, 34, 5, 53]. Most researchers agree that the design of complex, self-organizing systems with emergent properties is necessarily an iterative, step-wise refinement process, with extensive use of simulations and experiments. In fact, simulation seems to be the only practicable method of developing an understanding of the properties of organic systems, and we therefore conjecture that any promising design process has to involve simulation to evaluate a system's quality. Some tools have been proposed to support such a process, see, e.g., [9]. Nevertheless, the proper design methodology is still a heavily debated research issue. Eventually, the design process is often driven by trial-and-error and the intuition and ingenuity of an engineer. For inspiration, people often turn to principles from nature when designing self-organizing systems, see [57, 54] for compilations of such principles.

In this paper, we argue that simulation-based design, in its extreme form, basically corresponds to a black box search in the design space, using simulation to evaluate solutions. Although many real-world problems may not be

[1] Exceptions are perhaps physical systems, where the global behavior can be described by appropriately chosen differential equations

a complete black box, there are powerful black box optimization algorithms that could be used to automate simulation-based design of complex systems.

Design and optimization based on simulation have become practical only recently due to the immense increase in computational power. In the realm of simulation, entire complex systems can now be modeled realistically to the point of allowing accurate conclusions about the real world. And in the realm of design and optimization, it allows to develop and run iterative nature-inspired metaheuristics like evolutionary algorithms (EAs), which have proven to be successful in a wide variety of problem domains. Together, this provides, for the first time, a means to automatically optimize and design complex systems despite of the appearance of emergent phenomena.

Simulation-based design, however, poses a number of challenges:

1. **Evaluating a solution is time consuming.** Because simulating a complex organic system usually involves the simulation of a large number of interacting components, such a simulation is time consuming and computationally expensive.
2. **Evaluation is stochastic.** In many cases, a complex system contains some random component, be it only the environment it interacts with. As a consequence, the simulations used for evaluating complex systems involve a pseudo random number generator, and thus the observed performance is stochastic and depends on the random number seed. Such uncertainty in evaluation is a major challenge for design and optimization.
3. **Typical applications involve multiple objectives.** Although this aspect is not restricted to organic systems, many practical complex systems are supposed to satisfy a multitude of usually conflicting criteria. Since no single system is optimal with respect to all criteria, a compromise solution has to be found, which represents a proper trade-off of the different objectives.

Nature-inspired metaheuristics form one group of optimizers able to successfully tackle black box optimization problems. Nature-inspired optimization is a very active field of research, encompassing a number of different optimization approaches inspired by different natural phenomena. Among the most prominent representatives are simulated annealing [1], evolutionary algorithms [32], ant colony optimization [12], tabu search [40], or particle swarm optimization [33]. Their suitability for black box optimization alone would make them promising candidates for the design and optimization of complex systems. But, as will be demonstrated in this chapter, EAs are also able to address all the other challenges involved in the design and optimization of complex organic systems.

As a closely related field, simulation-based optimization has received increasing interest over the past years, good overviews can be found e.g. in [39, 67, 43]. So far, however, the area has been mostly concerned with the uncertainty of evaluations. A typical representative would be stochastic approximation [66], which is a variant of gradient search explicitly taking into

account stochastic evaluation functions. Here, we will show how EAs can be adapted to handle such problems, and will additionally look at all the other aspects mentioned above and considered important when designing or optimizing complex organic systems.

This chapter is structured as follows. The next section will provide a brief introduction to EAs. Then we discuss the different challenges with respect to the optimization of complex organic systems, and how EAs can address them. First, in section 6.3, we discuss ways to run the algorithm in reasonable time despite of the usually large time to evaluate a single solution. In particular, we discuss parallelization and the use of approximate models. Then, Section 6.4 looks at ways to allow EAs to cope with stochastic evaluations. The consideration of multiple objectives and the algorithm's ability to focus on the most interesting solutions is treated in Section 6.5. Section 6.6 briefly describes two exemplary applications. The chapter concludes with a summary and some ideas for future work.

6.2 A brief introduction to evolutionary computation

A detailed description of evolutionary algorithms is out of the scope of this chapter, and the interested reader is referred to, e.g., [32, 35]. However, a brief outline of the algorithms' main features shall be provided here.

Evolutionary algorithms are randomized heuristic search methods based on the principles of natural evolution, or more specifically, on Darwin's theory of the survival of the fittest. The two driving forces of EAs are selection and diversification. Starting with a set of candidate solutions (population), in each iteration (generation), new individuals are created based on the current population (diversification). The two primary construction operators are crossover, which combines information from two solutions to form a new solution, and mutation, which modifies an existing solution locally. In the next step, out of this larger set of parents and offspring, the new set of individuals allowed to reproduce is selected. By continually selecting good solutions for reproduction and then creating new solutions based on the knowledge represented in the selected individuals, the solutions "evolve" and become better and better adapted to the problem to be solved, just like in nature, where the individuals become better and better adapted to their environment through the means of evolution.

There are four different main streams of evolutionary computation that have originally been developed independently and focused on different aspects, namely genetic algorithms [42], evolution strategies [58, 64], evolutionary programming [38], and genetic programming [50]. While all can be used for design and optimization, the latter may be particularly interesting for open-ended design, because it allows for variable-length descriptions of solutions, e.g. in the form of LISP-expressions, i.e. it imposes fewer restrictions on the search space.

As all metaheuristics, EAs are more or less black box optimization techniques and thus don't impose any constraints on the optimization problem, e.g., the fitness function need not be differentiable.

6.3 Timely execution despite expensive evaluations

If during the course of optimization, many candidate solutions have to be evaluated, and each evaluation involves a time-consuming simulation, the overall time required for optimization may be excessively long. Therefore, methods are needed to speed up the optimization.

We consider here two fundamental ways to achieve that goal: either the execution itself is accelerated, or the algorithm is modified such that it can work with fewer evaluations (and thus requires fewer simulation runs). The former can be achieved by, e.g., parallelization, the latter by replacing the time-consuming simulation with approximate evaluations. These aspects are discussed in the following subsections.

6.3.1 Parallelization

Parallelization can be implemented on different levels: The lowest level is the level of a single evaluation or simulation. However, parallelization on this level is very problem specific and, in particular, if the interaction between system components is not restricted locally, could turn out to be quite challenging.

The highest level would be to run several instances of the EA in parallel on different processors. Since EAs are randomized search algorithms, they would generate different solutions when started with different random seeds on the different processors. The final solution would then be the best solution found by either of the parallel runs. Although that sort of high level parallelization would be very easy to realize, intuitively, it is not very efficient, as the different runs don't exchange any information.

Parallelizing a single run on the algorithm level seems most promising. Luckily, EAs are relatively easy to parallelize, since the time consuming evaluation of a solution can be done in parallel and independently for different solutions, for surveys see, e.g., [3, 26, 61].

Clearly, all individuals created in one generation can be evaluated independently on different processors. Also, mutation and crossover could be done in parallel. Only for selection, a solution's quality has to be judged relative to to the quality of all other solutions in the population, and thus global knowledge is required. This can be achieved in a master-slave fashion, where the master process maintains the population and selects parents, but sends out the parents for crossover, mutation and evaluation to the slave processes. This very straightforward parallelization scheme incurs significant communication overhead, as individuals have to be sent out and recollected in every iteration. Therefore, the EA community has developed algorithmic variants with

a more local selection, alleviating the need for global control. Most prominent among those are the island model and the diffusion model. In the island model, the population is divided into a number of subpopulations, which can run independent EAs on different processors. Only at regular intervals the subpopulations exchange some (usually the best) individuals with their neighbors in a so-called migration step. Communication is thus reduced to occasional migration. In the diffusion model, individuals are distributed spatially. In every generation, each individual selects a mating partner from its local neighborhood. The model is called diffusion model because the neighborhoods are overlapping, and a very good individual can spread only slowly (diffuse) from one local neighborhood to the next. The island model is ideally suited for workstation clusters with powerful processors and slow communication. The diffusion model with a local interaction structure is particularly well suited for massively parallel computers with a very fast local interconnection network.

Either model localizes selection, thereby creating temporary niches, in which also inferior individuals have a chance to survive for some time. As experience has shown, this effect is so beneficial that many people today actually implement either the island or the diffusion model even on a single processor.

All the approaches above more or less assume a dedicated parallel computer with equally powerful processors. However, in recent years, computer grids, i.e., the combination of available computers connected via Internet to form a virtual supercomputer, became a much cheaper and more accessible alternative. The power of computer grids has been demonstrated, e.g., by the project [65], which connected thousands of computers to search for extraterrestrial life, and companies like [56] or [74] commercialize the idea of networked computing. Clearly, computer grids have the potential to resolve the problem of high computer resources required by EAs, and will help pave the way to their still more widespread use.

However, the above parallelization schemes have to be adapted to a heterogeneous computer architecture with processors of vastly different power (with processor power actually varying over time, as only the computers' idle cycles are utilized), and slow and unreliable communication, see [20] for first steps in this direction.

6.3.2 Use of approximate models

Another way to reduce computational complexity and to speed up the procedure is to replace the usual evaluation by an approximate one. Such an approximate evaluation can be obtained through, e.g., response surface modeling. In the simplest case, methods from experimental design are used to determine a suitable set of potential solutions, called *design points*, that are evaluated and then used to construct a *metamodel*, a simple approximate

model of the true evaluation function. Typical types of approximation models include regression, kriging, or artificial neural networks.

Given such a metamodel, the application of EAs is straightforward. However, such a two-step process of first constructing a metamodel and then using it for optimization assumes that a good solution with respect to the metamodel also represents a good solution with respect to the true evaluation. The validity of this assumption clearly depends on the quality of the metamodel. The dilemma is that constructing an accurate metamodel for the whole search space may require even more evaluations than running the optimizer directly on the original evaluation function.

A promising alternative to that two-step process is to interweave model construction and optimization: in the beginning of the optimization process, a rather crude model may be sufficient, and later on, information from the optimization run can be used to identify the most promising regions of the search space, where the model can be repeatedly refined. In most cases solutions are generated by the optimizer and evaluated using the metamodel. Then, it is decided which of the solutions should be evaluated accurately, and finally, the information gained by evaluating some solutions accurately is used to update the metamodel. EAs are particularly suitable for combination with approximation models because they are black box optimizers (and thus the accurate evaluation can easily be exchanged with an approximate model on a solution-by-solution basis), and because they repeatedly resample promising regions of the search space, thereby gathering information over time in the most interesting regions, which can be used to refine the model exactly there. Comprehensive overviews of this rather large research area can be found e.g. in [48, 47].

The use of a metamodel is particularly helpful in the context of applications with noisy evaluation functions, or when searching for robust solutions. In those cases, in order to obtain sufficiently accurate estimates of a solution's quality, repeated evaluation of each solution is required, which makes optimization particularly costly. Here, metamodels can help reduce the number of evaluations per solution, see, e.g., [25, 55, 59]. These issues are discussed in more detail in the following section.

6.4 Stochastic fitness

EAs rely on an appropriate balance between exploration and exploitation, i.e., between testing new regions of the search space and concentrating the search on the most promising regions. The primary operator for exploitation is selection, which is a prerequisite for the advancement of search. In EAs, there are two potential selection steps: Good individuals have a higher probability to be selected as parents (parent selection), and bad individuals are removed from the population to make way for new individuals (usually termed environmental selection).

Selection implies the ability to discriminate alternative solutions accurately by their quality, in order to separate the good from the bad. However, as has been stated above, when designing complex organic systems, evaluation is done by randomized simulation, and thus selection is subject to uncertainty. This obviously impacts the algorithm's ability to select. Many authors have addressed this issue, and, in this section, we discuss ways that allow nature-inspired metaheuristics to work despite the noise.

There are a number of reasonable optimization goals in the presence of uncertainty, ranging from worst case performance over expected performance to the probability of being above a specified level. The by far most thoroughly studied and arguably most important criterion is expected performance, which will be assumed for the remainder of this section.

In most of the literature, the issue of uncertain evaluation is divided into two categories:

1. **Noisy evaluation** generally assumes an underlying objective function $f(x)$, which is unknown and disturbed by additive noise, i.e., the observed fitness function can be described as $F(x) = f(x) + \delta$, with δ a random variable (usually normally distributed with zero mean). In this case the expected fitness is the underlying (unknown) function: $E(F(x)) = f(x)$.
2. **Search for robust solutions** usually assumes that the underlying objective function $f(x)$ can be accurately evaluated during optimization, but the final solution is subject to noise when it is implemented, and the fitness obtained is thus $F(x) = f(x + \delta)$. Such a setting is typical in the case of, e.g., manufacturing tolerances. However, even if the probability distribution of δ is assumed to be known, it is not possible to calculate the expected fitness $E(F(x)) = \int_{-\infty}^{+\infty} f(x + \delta)p(\delta)d\delta$, because an analytically closed form of the underlying fitness function is not available.

The above categorization makes sense, as it addresses different application areas, but the border between the categories is blurred, and very similar techniques have been successfully applied for both scenarios. The main differences are that, in the case of noisy evaluation, the noise is assumed to be uncontrollable, and it is impossible to evaluate without noise, while in the case of searching for robust solutions, the disturbances applied *during optimization* can be chosen deliberately, and only the final solution is subject to uncontrollable noise. Controllability allows for the use of statistical variance reduction techniques. Furthermore, the distribution of *fitness* values for a particular solution is often assumed normally distributed in noisy optimization, while it is usually quite irregular (skewed and non-normal) when searching for robust solutions (because the noise enters the fitness function). In the following, we will discuss the issue of uncertain evaluation in general, and specify the assumptions underlying all approaches. For a survey see, e.g., [48].

6.4.1 Multiple samples

The simplest way to reduce uncertainty is by evaluating a solution repeatedly and using the average as an estimate for the true mean fitness. Sampling n times reduces a random variable's standard deviation by a factor of $\sqrt{n}$, but on the other hand increases the computation time by a factor of n. This is a generally perceived trade-off: either one can use relatively exact estimates but evaluate only a small number of individuals (because a single estimate requires many evaluations), or one can let the algorithm work with relatively crude fitness estimates and allow for more evaluations (as each estimate requires less effort). For examples of papers using this simple approach see, e.g., [44, 75, 71]. Depending on the application, variance reduction techniques like common random numbers or Latin hypercube sampling can help improving the estimates [15].

6.4.2 Implicit averaging

Instead of removing the noise by averaging over multiple samples, one might just let the algorithm cope with the uncertainty. Already many years ago, researchers have argued that EAs should be relatively robust against noise, see e.g., [37], and recently a number of publications have appeared which support that claim at least partially [6, 7, 8]. In [52] it is shown that for infinite population size proportional[2] selection is not affected by noise. Similarly, in [73] it was shown that a genetic algorithm with random perturbations applied to the design variables behaves identically to a genetic algorithm working on the expected fitness values if the population size is infinite.

The reason is that promising areas of the search space are sampled repeatedly by the EA, and the population usually contains many similar solutions. When the population is large, the noise in evaluating an individual is very likely compensated by that of a similar individual. This effect has been termed "implicit averaging" [48]. A natural question is whether explicit averaging in the form of re-sampling or implicit averaging in the form of a larger population size would be more efficient, given a fixed total number of fitness evaluations per generation. Conflicting conclusions have been drawn in different investigations [37, 10, 45]. In [51] and [52], some simplified theoretical models are developed, which allow to simultaneously optimize both population and sample size.

6.4.3 Response surface modeling

Instead of implicitly averaging over neighboring solutions, one can explicitly exploit information about previously evaluated similar solutions. And since

[2] With proportional selection, an individual's probability to be selected is proportional to its quality relative to the sum of qualities of all other individuals in the population.

nature-inspired optimization heuristics repeatedly sample promising regions of the search space, such data is usually available. In the simplest case of noise applied to the decision variables, neighboring solutions can be regarded as samples, and a weighted average over neighboring individuals can approximate the integral over possible disturbances [13]. Assuming a locally smooth fitness landscape, this idea has recently been extended in [55], where local metamodels are constructed based on previous evaluations in the neighborhood. Then, numerical integration over the metamodel can be used to approximate the expected fitness. In the case of noise applied to the fitness values, smooth metamodels, which average out the noise, can also be applied. In [25], we have successfully used local regression for that purpose, similar ideas can also be found in [59, 60]. Such techniques improve the fitness estimates without requiring additional samples.

6.4.4 Statistical ranking and selection techniques

The probability of erroneous selection depends not only on the uncertainty, but more on the signal-to-noise ratio, i.e., fitness difference relative to fitness variance. If the signal-to-noise ratio is large, selection is unlikely to make any errors. If it is rather small, selection is very uncertain. Thus, it seems promising to adapt the effort spent reducing the noise by repeated sampling to the uncertainty in a particular selection decision, rather than using a fixed number of samples. Consequently, it has been suggested to use a higher sample size for individuals with higher estimated variance [2]. Similarly, [68] bases the sample size on an individual's probability to be among the best (and thus to survive to the next generation).

While these attempts certainly represent improvements over the simple strategy of sampling each solution a constant number of times, they ignore the huge and well-developed field of statistically sound ranking and selection techniques. The primary difference between ranking and selection procedures and optimization procedures is that the former assume a given, usually small set of systems that are exhaustively examined, while the latter attempt to search efficiently through a search space too large for exhaustive search. But selection among a given small set of alternatives is done in every iteration of nature-inspired optimization, namely when the memory is updated. A survey of the most important selection techniques, together with an extensive comparison and demonstration of the respective strengths and weaknesses is provided in [16]. They are all based on the idea of sequential sampling, i.e., they take some samples, and then iteratively decide which systems should be sampled next until a termination criterion is met. One first effort to integrate methods from ranking and selection into EAs can be found in [11], where sequential sampling techniques are used to divide the individuals in the population into groups of similar quality, which then receive the same probability to be selected as parents.

In [24], we have integrated a selection technique, KN++ [49], into an EA's tournament selection. Also, we have shown how to numerically derive an even better selection procedure tailored to a specific noise level. In our test environment, both approaches, KN++ as well as our new procedure, showed approximately the same performance as the standard procedure, but required only half the number of samples.

6.4.5 Noise-adapted selection

In [23], we followed a completely different approach based on the observation that many standard EA variants include a form of randomized selection. For example, in rank-based selection, the probability to select an individual as parent is proportional to its rank in the population. In stochastic tournament selection, two individuals are compared and the better one is selected with a probability $p > 50\%$. Randomized selection is usually motivated by the wish to maintain diversity and to escape local optima. Noise has a similar effect as stochastic selection, namely that the inferior solution is selected with some probability. Thus, it should be possible to accept the noise inherent in the optimization problem and to use it to (at least partially) replace the randomness in the optimization algorithm. This has been achieved with our noise-adjusted tournament selection (NATS) presented in [23]. In NATS, the probability to accept a solution depends on the observed fitness difference between the two solutions. We used bootstrapping to generate suitable acceptance probabilities such that the expected acceptance rate is as close as possible to the desired acceptance rate.

6.4.6 Further issues

In [2] it was probably first suggested that the sample size, and thus the accuracy of evaluation, should be increased over the run. [14] looks at this problem more closely, and, in an extensive computational study, observes that it is best to have high accuracy in the beginning of a run (presumably because in that phase, the algorithm selects a subregion of the search space to work on), and towards the end of the run (when local fine-tuning and selection of the final solution require more precision). [4] look at a slightly different problem, but also conclude that the sample size should increase over the run.

For the case of multiple uncertain objectives, [70] modifies the usual Pareto dominance criterion to take uncertainty into account.

6.5 Multiple objectives

Design and optimization of complex organic systems often involves the consideration of multiple, usually conflicting objectives. There is usually not one

solution which is optimal with respect to all objectives, but a set of alternative solutions with different trade-offs. These solutions are generally called *Pareto optimal* or *efficient* whenever it is impossible to improve on such a solution in any criterion without suffering in at least one other criterion. Which of these solutions is the desired one depends on the preferences of the *decision maker* (DM). If these were known beforehand, e.g. in the form of a weighted combination of the objectives, the problem could be transformed into a single objective problem and solved in a standard way. However, very often the DM is unable to specify his or her preferences before the alternatives are known. It is therefore very convenient to have an optimization method capable of generating the whole set of Pareto-optimal solutions and to allow the DM to select among those afterwards.

EAs are population-based methods and thus capable of searching for all or a representative subset of the Pareto-optimal solutions in one run. In recent years, the field of evolutionary multi-objective optimization (EMO) has seen a dramatic rise in interest with thousands of papers published, a dedicated conference, and several books. For a comprehensive survey on the field, the reader is referred to [31]. An extensive and frequently updated repository of references can be found online [36].

The application of EAs to multi-objective problems is more or less straightforward. The main challenges are

1. to ensure convergence towards better solutions, and
2. to maintain a representative set of good alternatives.

To ensure convergence, one has to be able to determine when one solution should be preferred over another. Here, most approaches rely on the concept of non-dominance. A solution A is said to *dominate* a solution B, if A is at least as good as B in all objectives, and better in at least one objective. To ensure diversity along the front, among the non-dominated solutions, those in sparse areas are favored over those in crowded areas.

While generating the whole front of Pareto-optimal solution ensures that a DM's preferred solution is part of the solution set, such a solution set can contain a large number of alternative solutions, and selecting one may be tedious. Generating so many solutions is usually also time-consuming and would require rather large populations.

If we assume that for practical reasons the number of generated solutions should be small, one immediate question is how they should be selected from the large set of Pareto-optimal solutions, and whether the search can be focused on the most interesting solutions from the beginning. Most multi-objective approaches assume that the best representative set is equally distributed along the Pareto-optimal front. But although usually the DM can not a priori specify his or her preferences completely, some vague information is often available. Integrating this information into the optimization

and thereby biasing search towards the most interesting solutions holds the promise of reducing computation time and generating more relevant solutions.

There have been several attempts in this direction, see, e.g., [30, 21, 17]. Furthermore, it is possible to interactively learn user preferences during the optimization run, see, e.g., [72].

But even if no information about the DM's preferences is available, some solutions are more likely to be useful to the DM than others. In [18], we have proposed to evaluate solutions according to their expected marginal utility to a virtual DM. That is, we calculate how much additional value a solution provides according to a DM's utility function, averaged over an assumed distribution of possible utility functions. As a result, the algorithm using this diversity preservation mechanism exhibits a clear bias towards "knees", i.e., regions of the Pareto-optimal front with strong curvature. It has been argued before that these solution have particularly high practical relevance [27] because even a small improvement in either objective requires a significant worsening of the other objectives.

6.6 Exemplary applications

In this section, we will briefly discuss two exemplary applications where EAs have been successfully used to design complex systems.

The first example is the design of en-route caching strategies. In the Internet, document requests are routed from the requesting node point-to-point through the network to the node storing the document, then the document is sent all the way back to the requesting node. When hubs in the network become over-utilized, slowdowns and timeouts can disrupt the process. It is thus worthwhile to think about ways to minimize these effects. Caching, i.e., storing replicas of previously-seen objects for later reuse, has the potential to generate large bandwidth savings and in turn a significant decrease in response time. With en-route caching, each router in the network is equipped with a cache and may opt to store copies of some documents for future reuse [69]. The rules used for such decisions are called "caching strategies". Designing such strategies is a challenging task, because the different nodes interact, resulting in a complex dynamic system. The quality of a caching strategy can only be determined by simulation. [19] have demonstrated that genetic programming can be used successfully to design new effective caching strategies. The newly discovered caching strategy significantly outperformed all other state-of-the-art caching strategies tested.

Another example is the design of traffic light controllers. For simple controllers, where different phases are given fixed time intervals, theoretical analysis may still be possible. However, for adaptive controllers that adjust the signals based on traffic sensor data, a rigorous analysis does not seem to be possible and traffic planners usually rely on simulation to evaluate the quality of a traffic light controller. Goldate [41] has used EAs and traffic simulation to design a traffic light controller in a multi-objective setting, attempting to minimize travel time as well as the number of stops. Although his diploma

thesis can only be seen as a feasibility study, the resulting adaptive controller outperformed a controller designed by a human expert. The nature-inspired design of traffic light controllers is now explored with more rigor in the DFG project on "Organic Traffic Control"[3], including aspects like dynamic adaptation to changing (macro-)traffic patterns and interplay between the traffic light controllers of neighboring crossroads.

6.7 Conclusion

Organic computer systems with life-like structure are advocated as a way to handle the increasingly complex adaptive systems created by engineers and computer scientists. Consisting of a large number of dynamically interacting components, these systems exhibit emergent global behavior, which is not deducible from the local actions of a single component. Organic Computing thus calls for a methodology to determine appropriate local actions leading to the desired global behavior. Another major challenge for the design of organic systems is to control this emergent global behavior such that undesired effects of emergence can be prevented. In this chapter, we have argued that due to the emergent behavior, simulation seems to be required to evaluate a system's quality, which makes simulation a central component of design.

As has been demonstrated, EAs are particularly suitable for simulation-based design and optimization for the following reasons:

- The quality of a complex adaptive system can usually only be evaluated by simulation, as no closed analytical description is available. Since EAs are black box optimization heuristics, they do not impose any restrictions on the fitness function and naturally satisfy this requirement.
- EAs can be run efficiently in parallel, even on heterogeneous hardware platforms like computer grids. Furthermore, it is possible to partly substitute costly simulations by approximate models that are easier to compute. These two aspects allow to significantly reduce the running time if necessary.
- Even standard EAs can cope well with uncertainty in evaluation (e.g. due to stochastic simulations). Furthermore, they can be enhanced with advanced statistical methods from, e.g., ranking and selection to perform even better.
- EAs maintain a population of solutions throughout the run. As such, they are particularly suitable to handle multiple objectives and uncertainty about user preferences by searching for different trade-offs in parallel.

However, even though the suitability of this methodology has been demonstrated in several exemplary applications, more work is needed to further refine the methods and to make them ready for widespread easy use. Also,

[3] The project is part of the DFG priority program SPP 1183

while most of the above aspects have been examined independently so far, integrating them into one method seems a natural next step.

Finally, it seems promising to integrate the engineer's knowledge more closely into the process, e.g., by using interactive EAs [5].

Acknowledgments

The authors gratefully acknowledge the funding of their research on organic computing by the German Science Foundation (DFG) under SCHM752/12-1 and SCHM752/14-1.

References

1. E. Aarts and J. Korst. *Simulated annealing and Boltzmann machines*. Wiley, 1989.
2. A. N. Aizawa and B. W. Wah. Scheduling of genetic algorithms in a noisy environment. *Evolutionary Computation*, pages 97–122, 1994.
3. E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–461, 2002.
4. L. A. Albert and D. E. Goldberg. Efficient evaluation genetic algorithms under integrated fitness functions. Technical Report 2001024, Illinois Genetic Algorithms Laboratory, Urbana-Champaign, USA, 2001.
5. C. Anderson. Creation of desirable complexity: strategies for designing self-organized systems. In D. Braha et al., editors, *Complex Engineered Systems*, pages 101–121. Springer, 2006.
6. D. V. Arnold and H.-G. Beyer. Efficiency and mutation strength adaptation of the $(\mu/\mu_i, \lambda)$-ES in a noisy environment. In Schoenauer et al. [62], pages 39–48.
7. D. V. Arnold and H.-G. Beyer. Local performance of the $(\mu/\mu_i, \lambda)$-ES in a noisy environment. In W. Martin and W. Spears, editors, *Foundations of Genetic Algorithms*, pages 127–142. Morgan Kaufmann, 2000.
8. D. V. Arnold and H.-G. Beyer. A comparison of evolution strategies with other direct search methods in the presence of noise. *Computational Optimization and Applications*, 24:135–159, 2003.
9. C. Bernon, V. Camps, M.-P. Gleizes, and G. Picard. Tools for self-organizing applications engineering. In G. D. Serugendo et al., editors, *Engineering Self-Organizing Systems*, volume 2977 of *LNAI*, page 283.298. Springer, 2004.
10. H.-G. Beyer. Toward a theory of evolution strategies: Some asymptotical results from the $(1 \dagger \lambda)$-theory. *Evolutionary Computation*, 1(2):165–188, 1993.
11. J. Boesel. *Search and Selection for Large-Scale Stochastic Optimization*. PhD thesis, Northwestern University, Evanston, Illinois, USA, 1999.
12. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: From natural to artificial systems*. Oxford University Press, 1999.
13. J. Branke. Creating robust solutions by means of an evolutionary algorithm. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature*, volume 1498 of *LNCS*, pages 119–128. Springer, 1998.

14. J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer, 2001.
15. J. Branke. Reducing the sampling variance when searching for robust solutions. In L. S. et al., editor, *Genetic and Evolutionary Computation Conference (GECCO'01)*, pages 235–242. Morgan Kaufmann, 2001.
16. J. Branke, S. Chick, and C. Schmidt. Selecting a selection procedure. Technical report, Fontainebleau, 2005.
17. J. Branke and K. Deb. Integrating user preferences into evolutionary multi-objective optimization. In Y. Jin, editor, *Knowledge Incorporation into Evolutionary Algorithms*, pages 461–478. Springer, 2004.
18. J. Branke, K. Deb, H. Dierolf, and M. Osswald. Finding knees in multi-objective optimization. In *Parallel Problem Solving from Nature*, number 3242 in LNCS, pages 722–731. Springer, 2004.
19. J. Branke, P. Funes, and F. Thiele. Evolving en-route caching strategies for the internet. In *Genetic and Evolutionary Computation Conference*, volume 3103 of *LNCS*, pages 434–446. Springer, 2004.
20. J. Branke, A. Kamper, and H. Schmeck. Distribution of evolutionary algorithms in heterogeneous networks. In *Genetic and Evolutionary Computation Conference*, volume 3102 of *LNCS*, pages 923–934, 2004.
21. J. Branke, T. Kaußler, and H. Schmeck. Guidance in evolutionary multi-objective optimization. *Advances in Engineering Software*, 32(6):499–508, 2001.
22. J. Branke, M. Mnif, C. Müller-Schloer, H. Prothmann, U. Richter, F. Rochner, and H. Schmeck. Organic computing - addressing complexity by controlled self-organization. In *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*. ACM, 2007.
23. J. Branke and C. Schmidt. Selection in the presence of noise. In E. Cantu-Paz, editor, *Genetic and Evolutionary Computation Conference*, volume 2723 of *LNCS*, pages 766–777. Springer, 2003.
24. J. Branke and C. Schmidt. Sequential sampling in noisy environments. In X. Yao et al., editor, *Parallel Problem Solving from Nature*, volume 3242 of *LNCS*, pages 202–211. Springer, 2004.
25. J. Branke, C. Schmidt, and H. Schmeck. Efficient fitness estimation in noisy environments. In L. Spector et al., editors, *Genetic and Evolutionary Computation Conference*, pages 243–250. Morgan Kaufmann, 2001.
26. E. Cantu-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer, 2000.
27. I. Das. On characterizing the 'knee' of the pareto curve based on normal-boundary intersection. *Structural Optimization*, 18(2/3):107–115, 1999.
28. T. De Wolf and T. Holvoet. Emergence versus self-organization: Different concepts but promising when combined. In S. A. Brueckner et al., editors, *Engineering Self-Organising Systems: Methodologies and Applications*, number 3464 in LNCS, pages 1–15. Springer, 2005.
29. T. De Wolf and T. Holvoet. Towards a methodology for engineering self-organising emergent systems. In H. Czap et al., editors, *Self-Organization and Autonomic Informatics*, pages 18–34. Springer, 2005.
30. K. Deb. Solving goal programming problems using multi-objective genetic algorithms. In *Congress on Evolutionary Computation*, volume 1, pages 77–84. IEEE, 1999.
31. K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, 2001.

32. K. A. DeJong. *Evolutionary Computation.* MIT Press, 2002.
33. R. C. Eberhart and Y. Shi. *Swarm Intelligence.* Morgan Kaufmann, 2001.
34. B. Edmonds. Using the experimental method to produce reliable self-organised systems. In S. Brueckner et al., editors, *Engineering Self-Organising Systems*, volume 3464 of *LNAI*, pages 84–99. Springer, 2005.
35. A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* Springer, 2003.
36. Repository on multi-objective evolutionary algorithms. online, `http://www.lania.mx/$\sim$ccoello/EMOO/`.
37. J. M. Fitzpatrick and J. J. Grefenstette. Genetic algorithms in noisy environments. *Machine Learning*, 3:101–120, 1988.
38. L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley, 1966.
39. M. C. Fu. Optimizationn for simulation: Theory vs. practice. *INFORMS Journal of Computing*, 14(3):192–215, 2002.
40. F. Glover. Tabu search - part I. *ORSA Journal of Computing*, 1(3):190–206, 1989.
41. P. Goldate. Optimierung einer Ampelsteuerung mit Hilfe von evolutionären Algorithmen. Master's thesis, Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany, August 2003.
42. D. E. Goldberg. *Genetic Algorithms.* Addison-Wesley, 1989.
43. A. Gosavi. *Simulation-based optimization.* Kluwer Academic, 2003.
44. H. Greiner. Robust optical coating design with evolutionary strategies. *Applied Optics*, 35(28):5477–5483, 1996.
45. U. Hammel and T. Bäck. Evolution strategies on noisy functions, how to improve convergence properties. In Y. Davidor, H. P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature*, volume 866 of *LNCS*. Springer, 1994.
46. J. Holland. *Emergence - From chaos to order.* Addison-Wesley, 1998.
47. Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9:3–12, 2005.
48. Y. Jin and J. Branke. Evolutionary optimization in uncertain environments – a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.
49. S.-H. Kim and B. Nelson. A fully sequential procedure for indifference-zone selection in simulation. *ACM Transactions on Modelin and Computer Simulation*, 11(3):251–273, 2001.
50. J. R. Koza. *Genetic Programming.* MIT Press, 1991.
51. B. L. Miller. *Noise, Sampling, and Efficient Genetic Algorithms.* PhD thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1997. available as TR 97001.
52. B. L. Miller and D. E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4(2):113–131, 1996.
53. A. A. Minai, D. Braha, and Y. Bar-Yam. Complex engineered systems: A new paradigm. In D. Braha et al., editors, *Complex Engineered Systems*, pages 1–21. Springer, 2006.
54. R. Nagpal. A catalog of biologically-inspired primitives for engineering self-organization. In G. D. Serugendo et al., editors, *Engineering Self-Organizing Systems*, volume 2977 of *LNAI*, pages 53–62. Springer, 2004.
55. I. Paenke, J. Branke, and Y. Jin. Efficient search for robust solutions by means of evolutionary algorithms and fitness approximation. *IEEE Transactions on Evolutionary Computation*, 10(4):405–420, 2006.

56. Parabon Inc. Company homepage. Online. http://www.parabon.com.
57. H. V. D. Parunak. "go to the ant": Engineering principles from natural multi-agent systems. *Annals of Operations Research*, 75:69–101, 1997.
58. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipen der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.
59. Y. Sano and H. Kita. Optimization of noisy fitness functions by means of genetic algorithms using history of search. In Schoenauer et al. [62], pages 571–580.
60. Y. Sano and H. Kita. Optimization of noisy fitness functions by means of genetic algorithms using history of search with test of estimation. In *Congress on Evolutionary Computation*, pages 360–365. IEEE Press, 2002.
61. H. Schmeck, U. Kohlmorgen, and J. Branke. Parallel implementations of evolutionary algorithms. In A. Zomaya, F. Ercal, and S. Olariu, editors, *Solutions to Parallel and Distributed Computing Problems*, pages 47–66. Wiley, 2000.
62. M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors. *Parallel Problem Solving from Nature*, volume 1917 of *LNCS*. Springer, 2000.
63. T. Schöler and C. Müller-Schloer. An observer/controller architecture for adaptive reconfigurable stacks. In M. Beigl and P. Lukowicz, editors, *International Conference on Architecture Of Computing Systems*, volume 3432 of *LNCS*, pages 139–153. Springer, 2005.
64. H.-P. Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Technische Universität Berlin, Germany, 1975.
65. Seti@home. Project homepage. Online. http://setiathome.ssl.berkeley.edu/.
66. J. C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 31:332–341, 1992.
67. J. C. Spall. *Introduction to stochastic search and optimization*. John Wiley and Sons, 2003.
68. P. Stagge. Averaging efficiently in the presence of noise. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature V*, volume 1498 of *LNCS*, pages 188–197. Springer, 1998.
69. X. Tang and S. T. Chanson. Coordinated en-route web caching. *IEEE Transactions on Computers*, 51(6):595–607, 2002.
70. J. Teich. Pareto-front exploration with uncertain objectives. In E. Zitzler, K. Deb, L. Thiele, C. A. C. Coello, and D. Corne, editors, *Evolutionary Multi-Criterion Optimization*, volume 1993 of *LNCS*, pages 314–328. Springer, 2001.
71. A. Thompson. On the automatic design of robust elektronics through artificial evolution. In M. Sipper, D. Mange, and A. Peres-Urike, editors, *International Conference on Evolvable Systems*, pages 13 – 24. Springer, 1998.
72. D. S. Todd and P. Sen. Directed multiple objective search of design spaces using genetic algorithms and neural networks. In W. B. et al., editor, *Genetic and Evolutionary Computation Conference*, pages 1738–1743. Morgan Kaufmann, San Francisco, California, 1999.
73. S. Tsutsui and A. Ghosh. Genetic algorithms with a robust solution searching scheme. *IEEE Transactions on Evolutionary Computation*, 1(3):201–208, 1997.
74. United Devices. Company homepage. Online. http://www.ud.com.
75. D. Wiesmann, U. Hammel, and T. Bäck. Robust design of multilayer optical coatings by means of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 2(4):162–167, 1998.

7

Genesis of Organic Computing Systems: Coupling Evolution and Learning

Christian Igel[1] and Bernhard Sendhoff[2]

[1] Institut für Neuroinformatik, Ruhr-Universität Bochum, 44780 Bochum, Germany.
christian.igel@neuroinformatik.rub.de
[2] Honda Research Institute Europe GmbH, Carl-Legien-Str. 30, 63073 Offenbach, Germany.
bernhard.sendhoff@honda-ri.de

Summary. Organic computing calls for efficient adaptive systems in which flexibility is not traded in against stability and robustness. Such systems have to be specialized in the sense that they are biased towards solving instances from certain problem classes, namely those problems they may face in their environment. Nervous systems are perfect examples. Their specialization stems from evolution and development. In organic computing, simulated evolutionary structure optimization can create artificial neural networks for particular environments. In this chapter, trends and recent results in combining evolutionary and neural computation are reviewed. The emphasis is put on the influence of evolution and development on the structure of neural systems. It is demonstrated how neural structures can be evolved that efficiently learn solutions for problems from a particular problem class. Simple examples of systems that "learn to learn" as well as technical solutions for the design of turbomachinery components are presented.

7.1 Introduction

Technical systems that continuously adapt to a changing natural environment and act (quasi-) autonomously have not been designed so far. Several fundamental challenges have to be met. First, more flexibility is required on the software and possibly even on the hardware level. Second, this flexibility must not be traded in against system stability and robustness. Minimal performance must be guaranteed under all circumstances and degradation must be gradual and controlled. Third, the system must be expandable and sustainable.

Biological neural systems usually have such properties while their technical counterparts do not yet meet these requirements. Nevertheless, we believe that artificial neural networks (NNs) provide a computing paradigm whose potential has not yet been fully exploited. Our approach to address the abovementioned challenges and to tap the potential of artificial neural systems is

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_7,

to tune them for particular classes of problems and particular patterns of processing.

In nature, such specialization stems from evolution and development. Both design and shape structures ready to accommodate learning and self-organizing processes, which we see as the driving forces behind the capability of neural systems, see figure 7.1. We think that understanding the biological "design techniques" for nervous systems – evolution, development, and learning – paves the way for the design of artificial adaptive systems competitive with humans.

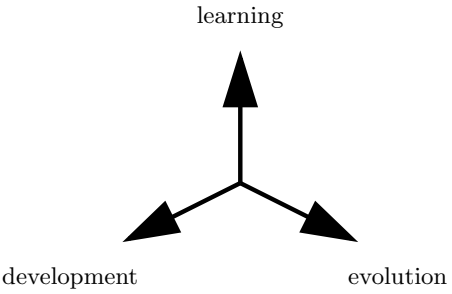


Fig. 7.1. Dimensions of natural design.

When designing adaptive systems, appropriate specialization (bias) and invariance properties are important, partially conflicting objectives. The "No-free-lunch theorems" for learning and optimization imply that it is fruitless to try to build universal adaptive systems. All systems have to be biased towards particular problem classes. This bias can be induced by evolved structures, on which learning and self-organizing processes operate. In this chapter, we review trends and recent results in combining evolutionary and neural computation. We will highlight synergies between the two fields beyond the standard examples and emphasize the influence of evolution and development on the structure of neural systems for the purpose of adaptation. We demonstrate how neural structures can be evolved that efficiently learn particular problem classes. We present simple examples of systems that "learn to learn" as well as technical solutions for the design of turbomachinery components.

The next section provides some background in artificial NNs and evolutionary algorithms (EAs). We put an emphasis on theoretical limitations and perspectives of these computing paradigms. We briefly describe simple NNs based on integrate-and-fire neurons, introduce EAs in the framework of stochastic search, and summarize the No-free-lunch theorems for learning and optimization. In section7.3, we discuss evolutionary structure optimization of neural systems and review some more recent trends in combining EAs and neural systems. Finally, we demonstrate how neural structures can be evolved that efficiently learn solutions for problems from a particular problem class.

7.2 Background

In this section, we provide short introductions to NNs and EAs and review "No-free-lunch" results for learning and optimization.

7.2.1 Neural computation

In the following, we briefly introduce basic ideas of NNs on the basis of the simple *rate-coded leaky integrator neuron* model. A detailed introduction to the broad field of neural computation is far beyond the scope of this article. A good starting point for reading is [5], recommendable introductory books on NNs for technical applications are [8, 33, 9] and on modeling nervous systems [17].

Neural systems can be described on different levels of abstraction. Many models, including those usually adopted for technical applications, can be derived from the *leaky integrator neuron*. This model is based on the assumption that the basic units of computation in nervous systems are single neurons. A model neuron i is situated in time t and its state is described by the *membrane potential* $u_i(t)$ governed by the differential equation

$$\tau_i \frac{\partial u_i(t)}{\partial t} = -u_i(t) + \sum_j w_{ij}\sigma_j[u_j(t)] + \sum_k w'_{ik}s_k(t) + \theta_i$$

with time constant τ_i. The neuron computes a weighted linear sum of the inputs it receives (see [55] for a review of more detailed models of single neurons). The first sum runs over all neurons j providing input to i, the second over all external inputs $s_k(t)$, which are gathered in the vector $\mathbf{s}(t)$, to the system. The weights w_{ij} and w'_{ik} describe the strengths of the synaptic connections. In the absence of input the membrane potential relaxes to the resting level θ_i. It is assumed that the only communication in a network of these units is through spikes of electrical activity traveling between the neurons. A neuron emits a spike when its membrane potential exceeds a certain threshold. Real spikes are discrete events, but in the model a rate code describing the average spiking frequency is assumed to capture the essence of the signals. This rate can either be viewed as an ensemble average across a population of neurons with the same properties, or as the frequency of spikes of a single neuron in some time interval. The activation function σ_i, which is usually sigmoidal (i.e., nondecreasing and bounded), maps the membrane potential u_i to the corresponding spiking frequency. Forward Euler approximation, $\partial u_i(t)/\partial t \approx (u_i(t + \Delta t) - u_i(t))/\Delta t$, with $\Delta t = \tau_i = 1$, leads to the basic discrete-time equation $u_i(t + 1) = \sum_j w_{ij}\sigma_j[u_j(t)] + \sum_k w'_{ik}s_k(t) + \theta_i$.

The structure or architecture of the NN can be described by a graph in which the nodes correspond to the neurons and there is an edge from i to j if neuron j gets input from neuron i. If the network graph contains no

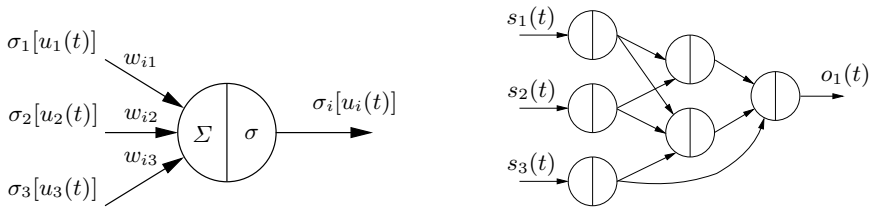


Fig. 7.2. Simple computational model of a single neuron, left, and a neural network graph, right.

cycles, we speak of a feed-forward NN. If we number the neurons such that node i only receives input from units j with $j < i$ and update the neurons in increasing order, the discrete-time network equation can be written as a static function $u_i(\mathbf{s}) = \sum_{j<i} w_{ij}\sigma_j[u_j] + \sum_k w'_{ik}s_k + \theta_i$. Often, some of the neurons are dedicated *output neurons* whose spike rates are gathered in the vector $\mathbf{o}(t)$ and the neural system can be viewed as a functional mapping input sequences $\mathbf{s}(t)$ to output sequences $\mathbf{o}(t)$. In case of a feed-forward NN, the mapping reduces to a static function assigning an output $\mathbf{o}$ to an input $\mathbf{s}$, see figure 7.2.

Models of NNs based on leaky integrator neurons, in principle exhibit universal approximation and computation properties under mild assumptions (see, e.g., [87, 89, 91]). However, the general question of how to design an appropriate neural system efficiently for a given task remains open and complexity theory reveals the need for using heuristics (see, e.g., [88]) — here these heuristics are the major organization principles of biological NNs, evolution, development, and learning.

Supervised learning of an NN means adapting the weights w_{ij}, w'_{ik} such that, given some input $\mathbf{s}(t)$, the output neurons show a predefined behavior $\mathbf{y}(t)$, which is described by sample (training) input-output sequences. A feed-forward NN learns a static function h based on sample input-output patterns $\{(\mathbf{s}_1, \mathbf{y}_1), \ldots, (\mathbf{s}_\ell, \mathbf{y}_\ell)\}$. This is usually done by gradient-based minimization of the (squared) differences between the targets $\mathbf{y}_i$ and the corresponding outputs $\mathbf{o}_i$ of the NN given the input $\mathbf{s}_i$. The ultimate goal is not to simply memorize the training patterns, but to find a statistical model for the underlying relationship between input and output data. Such a model will generalize well, that is, it will make good predictions for cases other than the training patterns. Therefore, a critical issue is to avoid overfitting during the learning process: The NN should just fit the signal and not the noise. This is usually achieved by restricting the effective complexity of the network, for example by regularization of the learning process [3].

In the context of feed-forward NNs, generalization can for example be formalized in the framework of statistical learning theory as follows. Let the goal be to learn a function from some input space S to some output space Y and

let $h : S \to Y$ be the function realized by the NN. Based on some input-output patterns drawn independently from the same distribution P on $S \times Y$, the goal of generalization is to minimize $\int_{S\times Y} P(\mathbf{s},\mathbf{y})L(h(\mathbf{s}),\mathbf{y})\mathrm{d}\mathbf{s}\,\mathrm{d}\mathbf{y}$, where $L : Y \times Y \to \mathbb{R}_0^+$ denotes a loss function. The distribution P is usually unknown and defines the learning problem at hand. The value $L(a, b)$ quantifies the cost or regret of predicting a instead of b and returns zero if its arguments are equal. For example when learning a one-dimensional real-valued function, $S = Y = \mathbb{R}$ and $L(a, b) = (a - b)^2$ is a typical choice.

In this chapter, we focus on the architecture of feed-forward neural networks, however, most of our findings and discussions apply equally well to recurrent neural systems, which also have been used successfully in applications in the past (in particular for time series prediction, e.g., [93, 61]).

7.2.2 Evolutionary computation

Evolutionary algorithms can be regarded as a special class of global random search algorithms. Let the search problem under consideration be described by a quality function $f : \mathcal{G} \to \mathcal{Y}$ to be optimized, where $\mathcal{G}$ denotes the search space (i.e., the space of candidate solutions) and $\mathcal{Y}$ the (at least partially) ordered space of cost values. The general global random search scheme can be described as follows:

① Choose a joint probability distribution $P_{\mathcal{G}^\lambda}^{(1)}$ on $\mathcal{G}^\lambda$. Set $t \leftarrow 1$.
② Obtain λ points $\mathbf{g}_1^{(t)}, \ldots, \mathbf{g}_\lambda^{(t)}$ by sampling from the distribution $P_{\mathcal{G}^\lambda}^{(t)}$. Evaluate these points using f.
③ According to a fixed (algorithm dependent) rule construct a new probability distribution $P_{\mathcal{G}^\lambda}^{(t+1)}$ on $\mathcal{G}^\lambda$.
④ Check whether some stopping condition is reached; if the algorithm has not terminated, substitute $t \leftarrow t + 1$ and return to step ②.

Random search algorithms can differ fundamentally in the way they describe (parameterize) and alter the joint distribution $P_{\mathcal{G}^\lambda}^{(t)}$, which is typically represented by a semi-parametric model. The scheme of a canonical EA is shown in figure 7.3. In evolutionary computation, the iterations of the algorithm are called *generations*. The search distribution of an EA is given by the *parent population*, the *variation operators*, and the *strategy parameters*. The parent population is a multiset of μ points $\tilde{\mathbf{g}}_1^{(t)}, \ldots, \tilde{\mathbf{g}}_\mu^{(t)} \in \mathcal{G}$. Each point corresponds to the *genotype* of an *individual*. In each generation, λ *offspring* $\mathbf{g}_1^{(t)}, \ldots, \mathbf{g}_\lambda^{(t)} \in \mathcal{G}$ are created by the following procedure: Individuals for reproduction are chosen from $\tilde{\mathbf{g}}_1^{(t)}, \ldots, \tilde{\mathbf{g}}_\mu^{(t)}$. This is called *mating selection* and can be deterministic or stochastic (where the sampling can be with or without replacement). The offspring's genotypes result from applying variation operators to these selected parents. Variation operators are deterministic or partially stochastic mappings from $\mathcal{G}^k$ to $\mathcal{G}^l$, $1 \le k \le \mu, 1 \le l \le \lambda$. An operator with $k = l = 1$

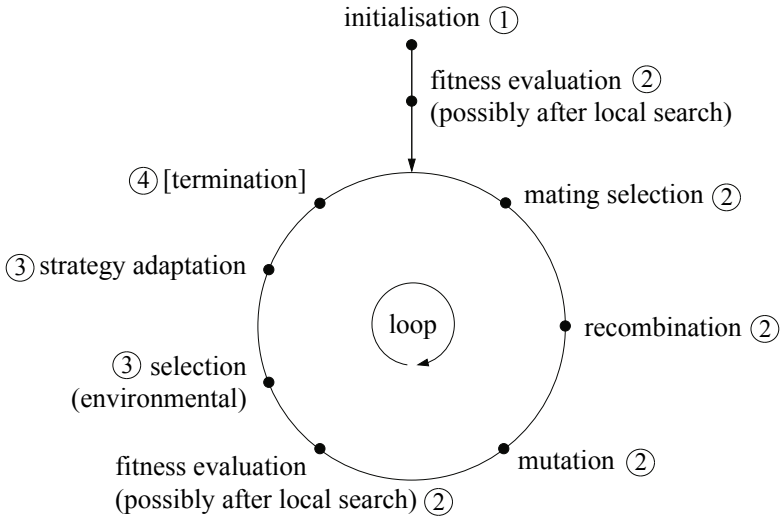


Fig. 7.3. Basic EA loop. The numbers indicate the corresponding steps in the random search scheme. When optimizing adaptive systems, the local search usually corresponds to some learning process.

is called *mutation*, whereas *recombination* operators involve more than one parent and can lead to more than one offspring. Multiple operators can be applied consecutively to generate offspring. For example, an offspring $\mathbf{g}_i^{(t)}$ can be the product of applying recombination $o_{\text{rec}} : \mathcal{G}^2 \to \mathcal{G}$ to two randomly selected parents $\tilde{\mathbf{g}}_{i_1}^{(t)}$ and $\tilde{\mathbf{g}}_{i_2}^{(t)}$ followed by mutation $o_{\text{mut}} : \mathcal{G} \to \mathcal{G}$, that is, $\mathbf{g}_i^{(t)} = o_{\text{mut}}\left(o_{\text{rec}}\left(\tilde{\mathbf{g}}_{i_1}^{(t)}, \tilde{\mathbf{g}}_{i_2}^{(t)}\right)\right)$. Evolutionary algorithms allow for incorporation of *a priori* knowledge about the problem by using tailored variation operators combined with an appropriate encoding of the candidate solutions.

Let $P_{\mathcal{G}^\lambda}^{(t)}(\mathbf{g}_1^{(t)}, \ldots, \mathbf{g}_\lambda^{(t)}) = P_{\mathcal{G}^\lambda}\left(\mathbf{g}_1^{(t)}, \ldots, \mathbf{g}_\lambda^{(t)} \,|\, \tilde{\mathbf{g}}_1^{(t)}, \ldots, \tilde{\mathbf{g}}_\mu^{(t)}; \boldsymbol{\theta}^{(t)}\right)$ be the probability that parents $\tilde{\mathbf{g}}_1^{(t)}, \ldots, \tilde{\mathbf{g}}_\mu^{(t)}$ create offspring $\mathbf{g}_1^{(t)}, \ldots, \mathbf{g}_\lambda^{(t)}$. This distribution is additionally parameterized by some *external strategy parameters* $\boldsymbol{\theta}^{(t)}$, which may vary over time. In some EAs, the offspring are created independently of each other based on the same distribution. In this case, the joint distribution $P_{\mathcal{G}^\lambda}^{(t)}$ can be factorized as $P_{\mathcal{G}^\lambda}^{(t)}(\mathbf{g}_1^{(t)}, \ldots, \mathbf{g}_\lambda^{(t)}) = P_{\mathcal{G}}^{(t)}(\mathbf{g}_1^{(t)}) \cdot \ldots \cdot P_{\mathcal{G}}^{(t)}(\mathbf{g}_\lambda^{(t)})$.

Evaluation of an individual corresponds to determining its fitness by assigning the corresponding cost value given by the quality function f. Evolutionary algorithms can — in principle — handle optimization problems that are non-differentiable, non-continuous, multimodal, and noisy. They are easy to parallelize by distributing the fitness evaluations of the offspring. In single-

objective optimization, we usually have $\mathcal{Y} \subset \mathbb{R}$, whereas in multi-objective optimization, see section 7.3.2.1, vector-valued functions (e.g., $\mathcal{Y} \subset \mathbb{R}^k, k > 1$) are considered. In co-evolution (see section 7.3.2.2), individuals interact to affect each other's adaptations. Therefore, the fitness values are not determined for each individual in isolation, but in the context of the current population (i.e., a more appropriate description of fitness assignment is $f : \mathcal{G}^\lambda \to \mathcal{Y}^\lambda$ or even $f : \mathcal{G}^{\lambda+\mu} \to \mathcal{Y}^{\lambda+\mu}$ if the parents are also involved in the fitness calculation). The interaction of individuals may be competitive or cooperative. As the fitness function is not fixed, co-evolution allows for "bootstrapping" the evolutionary process and "open-ended" evolution.

Updating the search distribution corresponds to *environmental selection* and sometimes additional *strategy adaptation* of external strategy parameters $\boldsymbol{\theta}^{(t+1)}$. The latter is extensively discussed in the context of optimization of NNs in [40, 45]. A selection method chooses μ new parents $\tilde{\mathbf{g}}_1^{(t+1)}, \ldots, \tilde{\mathbf{g}}_\mu^{(t+1)}$ from $\tilde{\mathbf{g}}_1^{(t)}, \ldots, \tilde{\mathbf{g}}_\mu^{(t)}$ and $\mathbf{g}_1^{(t)}, \ldots, \mathbf{g}_\lambda^{(t)}$. This second selection process is called environmental selection and may be deterministic or stochastic. Either the mating or the environmental selection must be based on the objective function values of the individuals and must prefer those with better fitness — this is the driving force of the evolutionary adaptation process.

It is often argued that evolutionary optimization is not well understood theoretically — ignoring the tremendous progress in EA theory during the last years. Although there are only a few results for general settings (e.g., convergence [76]), there exist rigorous expected runtime analyses of simplified algorithms on restricted, but important classes of optimization problems, see [46, 20] and references therein. The article [7] provides a good starting point for reading about EA theory.

7.2.3 The need for specialization: No-free-lunch

It is not only intuitive, but also proven that it is not possible to design an universal adaptive system that outperforms other systems across all possible problems. This is formally expressed by the No-free-lunch (NFL) theorems going back to the work of Wolpert and Macready [104, 105]. Coarsely speaking, the NFL theorems for learning state that without an assumption of how the past (training data) is related to the future (test data), prediction is impossible. In other words, without an a priori restriction of the possible phenomena that are expected, it is impossible to generalize and thus no algorithm is superior to another. Even worse, any consistent algorithm (i.e., any algorithm converging to the Bayes optimal classifier almost surely when the number of training patterns, drawn independently from the distribution describing the problem, approaches infinity) can have arbitrarily poor behavior when given a finite, incomplete training set [104, 19, 10].

These results carry over to general search and optimization algorithms. The NFL theorem for optimization formalizes that averaged over the set $\mathcal{F}$

of all possible objective functions defined between a finite search space $\mathcal{X}$ and a finite set $\mathcal{Y}$ of cost values all optimization algorithms have the same performance. It is assumed that the algorithms never visit a search point twice and that the performance measure just depends on the objective function values of the visited search points [20, 105, 82, 42, 43, 106]. More generally, the following holds for any probability distribution P over $\mathcal{F}$. If and only if $\mathcal{F} = \bigcup_i \mathcal{F}_i$, every $\mathcal{F}_i$ is closed under permutation, and $f, g \in F_i$ implies that f and g have the same probability $P(f) = P(g)$ to be the objective function, then all optimization algorithms have the same performance averaged over $\mathcal{F}$ w.r.t. P [43]. Closure under permutation of a set $\mathcal{F}_i$ means that for every bijective function $\pi : \mathcal{X} \to \mathcal{X}$ it holds that $f \in \mathcal{F}_i$ implies $f \circ \pi \in \mathcal{F}_i$. These assumptions for an NFL result to hold are rather strict, and fortunately problem classes relevant in practice are likely to violate them [42, 43].

Nonetheless, only if we consider restricted problem classes, in which the assumptions of the NFL theorems are not fulfilled, we can design efficient adaptive systems. It is important to make this bias towards a problem class explicit in the design process. In nature, such a bias stems from the evolved structures on which learning and self-organizing processes operate. In organic computing, simulated evolutionary structure optimization can create systems biased towards relevant problem classes.

7.3 Evolutionary computation and neural systems

Both artificial evolution and artificial neural systems have long histories, which in many respects resemble each other. In their beginnings, both were met with considerable skepticism from the biological as well as from the technological communities. For the first, their simplifications and abstractions meant throwing over board years of carefully accumulated details on how biological systems operate, develop, learn, and evolve. For the second the new type of distributed, stochastic, and nonlinear processing was equally hard to accept. During their maturation both fields met a couple of times, but not as often as one might expect bearing in mind that their philosophies to extract principles of biological information processing and apply them to technical systems are so similar.

Although not directly aimed at the formation of neural systems, the design of intelligent automata was among the earliest applications of EAs and may be traced back to the 1950s, see [22]. However, it took another 30 years until first papers were published describing explicitly the application of EAs to NNs and in this context — albeit more hesitantly — to learning [52, 63]. Then the subject quickly received considerable interest and several articles were published in the early nineties concentrating on optimization of both the network architecture and its weights. Although nowadays NNs and EAs are used frequently and successfully together in a variety of applications, the desired

breakthrough, that is, the evolution of neural systems showing *qualitatively* new behavior, has not been reached yet. The complexity barrier may have been pushed along but it has not been broken down. Nevertheless, many important questions on the architecture, (e.g., modularity) the nature of learning, (e.g., nature vs. nurture) and the development of neural systems (e.g., interactions between levels of adaptation) have been raised and important results have been obtained.

There are still only few works connecting current brain research with evolutionary computation, but first attempts have been promising, as we will see in section 7.3.2.5. Here, on a more general note, we argue that combining evolutionary development with brain science is more than just optimizing models of biological neural systems. The brain is a result of the past as much as of the present. That means that learning (the present) can only operate on an appropriate structure (the past). The current structure reflects its history as much as its functionality. Flexibility and adaptability of the brain are based on its structural organization, which is the result of its ontogenetic development. The brain is not one design but many designs; it is like a cathedral where many different parts have been added and removed over the centuries. However, not all designs are capable of such continuous changes and the fact that the brain is, is deeply rooted in its structural organization.

In this section, we discussed selected aspects of combining neural and evolutionary computing. More comprehensive surveys, all having slightly different focuses, can be found in [65, 71, 83, 107].

7.3.1 Structure optimization of adaptive systems

Although NNs are successfully applied to support evolutionary computation (see section 7.4.2), the most prominent combination of EAs and NNs is evolutionary optimization of adaptive neural systems.

In general, the major components of an adaptive system can be described by a triple $(\mathcal{S}, \mathcal{A}, \mathcal{D})$, where $\mathcal{S}$ stands for the structure or architecture of the adaptive system, $\mathcal{A}$ is a learning algorithm that operates on $\mathcal{S}$ and adapts flexible parameters of the system, and $\mathcal{D}$ denotes sample data driving the adaptation. We define the *structure* as those parts of the system that cannot be changed by the learning/self-adaptation algorithm. Given an adaptation rule $\mathcal{A}$, the structure $\mathcal{S}$ determines

- the set of solutions that can be realized,
- how solution changes given new stimuli/signals/data, partial failure, noise, etc.,
- the neighborhood of solutions (i.e., distances in solution space),
- bias (specialization) and invariance properties.

Learning of an adaptive system can be defined as goal-directed, data-driven change of its behavior. Examples of learning algorithms for technical

NNs include gradient-based heuristics (see section 7.2.1) or quadratic programming. Such "classical" optimization methods are usually considerably faster than pure evolutionary optimization of these parameters, although they might be more prone to getting stuck in local minima. However, there are cases where "classical" optimization methods are not applicable, for example when the neural model or the objective function is non-differentiable (e.g., see section 7.3.2.2). Then EAs for real-valued optimization provide a means for adjusting the NN parameters. Still, the main application of evolutionary optimization in the field of neurocomputing is adapting the structures of neural systems, that is, optimizing those parts that are not altered by the learning algorithm. Both in biological and technical neural systems the structure is crucial for the learning behavior — the evolved structures of brains are an important reason for their incredible learning performance: "development of intelligence requires a balance between innate structure and the ability to learn" [6]. Hence, it appears consequential to apply evolutionary methods to structure adaptation of neural systems for technical applications, a task for which usually no efficient "classical" methods exist.

A prototypical example of evolutionary optimization of a neural architecture on which a learning algorithm operates is the search for an appropriate topology of a multi-layer perceptron NN, see [103, 36, 29] for some real-world applications. Here, the search space ultimately consists of graphs, see section 7.2.1. When using EAs to design NN graphs, the key questions are how to encode the topologies and how to define variation operators that *act* on this representation. In the terminology of section 7.2.2, operators and representation both determine the search distribution and thereby the neighborhood of NNs in the search space. Often an intermediate space, the phenotype space $\mathcal{P}$, is introduced in order to facilitate the analysis of the problem and of the optimization process itself. The fitness function can then be written as $f = f' \circ \phi$, where $\phi : \mathcal{G} \to \mathcal{P}$ and $f' : \mathcal{P} \to \mathcal{Y}$. The definition of the phenotype space is to a certain degree arbitrary. The same freedom exists in evolutionary biology [60] and is not restricted to EAs. The probability of a certain phenotype $p \in \mathcal{P}$ to be created from a population of phenotypes strongly depends on the representation and the variation operators. When the genotype-phenotype mapping ϕ is not injective, we speak of neutrality, which may considerably influence the evolutionary process (see [41] for an example in the context of NNs). We assume that $\mathcal{P}$ is equipped with an extrinsic (i.e., independent of the evolutionary process) metric or at least a consistent neighborhood measure, which *may* be defined in relation to the function of the individual. In the case of NNs, the phenotype space is often simply the space of all possible connection matrices of the networks. Representations for evolutionary structure optimization of NNs have often been classified in "direct" and "indirect" encodings. Roughly, a direct encoding or representation is one where (intrinsic) neighborhood relations in the genotype space (induced by $P_{\mathcal{G}^\lambda}$) broadly correspond to extrinsic distances of the corresponding phenotypes. Note that such a classification only makes sense once a phenotype

space with an extrinsic distance measure has been defined and that it is only valid for this particular definition. (This point has frequently been overlooked because of the implicit agreement on the definition of the phenotype space, e.g., the graph space equipped with a graph editing distance). This does not imply that both spaces are identical. In an indirect encoding the genotype usually encodes a rule, a program or a mapping to build, grow or develop the phenotype. Such encodings foster the design of large, modular systems. Examples can be found in [54, 32, 25, 83, 84].

7.3.2 Trends in combining EAs and neural computation

In the following, we review some more recent trends in combining neural and evolutionary computing. Needless to say that such a collection is a subjective, biased selection.

7.3.2.1 Multi-objective optimization of neural networks

Designing a neural system usually requires optimization of several, often conflicting objectives. This includes coping with the bias-variance dilemma or trading classification speed against accuracy in real-time applications. Although the design of neural systems is obviously a multi-objective problem, it is usually tackled by aggregating the objectives into one scalar function and applying standard methods to the resulting single-objective task. However, this approach will in general fail to find all desired solutions [16]. Furthermore, the aggregation weights have to be chosen correctly in order to obtain the desired result. In practice, it is more convenient to make the trade-offs between the objectives explicit (e.g., to visualize them) after the design process and select from a diverse set of systems the one that seems most appropriate. This can be realized by "true" multi-objective optimization (MOO, [48]). The MOO algorithms approximate the set of Pareto-optimal tradeoffs, that is, those solutions that cannot be improved in any objective without getting worse in at least one other objective. From the resulting set of systems the final solution can be selected after optimization. There have been considerable advances in MOO recently, which can now be incorporated into machine learning techniques. In particular, it was realized that EAs are very well suited for multi-criterion optimization and they have become the MOO methods of choice in the last years [13, 18]. Recent applications of evolutionary MOO to neural systems address the design of multi-layer perceptron NNs [1, 2, 49, 103, 29, 11, 48] and support vector machines (SVMs) [39, 97].

7.3.2.2 Reinforcement learning

In the standard reinforcement learning (RL) scenario [100, 95, 74], an agent perceives stimuli from the environment and decides which action to take based

on its policy. Influenced by the actions, the environment changes its state and possibly emits reward signals. The reward feedback may be sparse, unspecific, and delayed. The goal of the agent is to adapt its policy, which may be represented by (or be based on) a NN, such that the expected reward is maximized. The gradient of the performance measure with respect to NN parameters can usually not be computed (but estimated in case of stochastic policies, e.g., see [96, 56]).

Evolutionary algorithms have proved to be powerful and competitive methods for solving RL problems [64, 38, 72]. The recent success of evolved NNs in game playing [12, 23, 59, 94] demonstrates the potential of combining NNs and evolutionary computation for RL. The possible advantages of EAs compared to standard RL methods are that they allow — in contrast to the common temporal difference learning methods — for direct search in the space of (stochastic as well as deterministic) policies. Furthermore, they are often easier to apply and more robust with respect to the tuning of the meta-parameters (learning rates, etc.). They can be applied to non-differentiable function approximators and even optimize their underlying structure.

Closely related is the research area of evolutionary robotics devoted to the evolution of "embodied" neural control systems [66, 57, 70, 101]. Here promising applications of the principle of co-evolution can be found.

7.3.2.3 Evolving network ensembles

Ensembles of NNs that cooperatively solve a given task can be preferable to monolithic systems. For example, they may allow for task decomposition that is necessary for efficiently solving a complex problem and they are often easier to interpret [85]. The population concept in EAs appears to be ideal for designing neural network ensembles, as, for example, demonstrated for classification tasks in [58, 11]. In the framework of decision making and games, Mark et al. [62] developed a combination of NN ensembles and evolutionary computation. Two ensembles are used to predict the opponent's strategy and to optimize the own action. Using an ensemble instead of a single network ensures to be able to maintain different opponent experts and counter-strategies in parallel. The EA is used to determine the optimal input for the two network ensembles. Ensembles of networks have also turned out a superior alternative to single NNs for fitness approximation in evolutionary optimization. In [51] network ensembles have been optimized with evolution strategies and then used as metamodels in an evolutionary computation framework. Beside the increase in approximation quality an ensemble of networks has the advantage that the fidelity of the networks can be estimated based on the variance of the ensemble.

7.3.2.4 Optimizing kernel methods

Adopting the extended definition of structure as that part of the adaptive system that cannot be optimized by the learning algorithm itself, model selection

of kernel-based methods is a structure optimization problem. For example, choosing the right kernel for an SVM [14, 15, 81] is important for its performance. When a parameterized family of kernel functions is considered, kernel adaptation reduces to finding an appropriate parameter vector. These "hyperparameters" are usually determined by grid search, which is only suitable for the adjustment of very few parameters, or by gradient-based approaches. When applicable, the latter methods are highly efficient albeit susceptible to local optima. Still, the gradient of the performance criterion w.r.t. the hyperparameters can often neither be computed nor accurately approximated. This leads to growing interest in applying EAs to model selection of SVMs. In [26, 77, 39, 97], evolution strategies (i.e., EAs tailored for real-valued optimization) were proposed for adapting SVM hyperparameters, in [21, 27] genetic algorithms (EAs that represent candidate solutions as fixed-length strings over a finite alphabet) were used for SVM feature selection.

7.3.2.5 Computational neuroscience and brain-inspired architectures

There are only a few applications of evolutionary computation in brain science [4, 80, 92, 44, 75], although evolutionary "analysis by synthesis" guided by neurobiological knowledge may be a powerful tool in computational neuroscience. The challenge is to force artificial evolution to favor solutions that are reasonable from the biological point of view by incorporating as much neurobiological knowledge as possible in the design process (e.g., by a deliberate choice of the basic system structure and constraints that ensure biological plausibility).

In the field of brain-inspired vision systems [28, 102] EAs have been used to optimize the structure of the system (i.e., feature banks or hierarchical layers) and to determine a wide variety of parameters. Evolutionary algorithms have been successfully applied to the Neocognitron structure [98, 68, 86], which was one of the first hierarchical vision systems based on the structure of its biological counterpart [28]. More recent work employed evolution strategies to optimize the nonlinearities and the structure of a biologically inspired vision network, which is capable of performing a complex 3D real world object classification task [78, 79]. The authors used evolutionary optimization with direct encoding that performed well in an 1800-dimensional search space. In a second experiment evolutionary optimization was successfully combined with local unsupervised learning based on a sparse representation. The resulting architecture outperformed alternative approaches.

7.4 Networks that learn to learn

The ability to learn (online) is one of the most distinguishing features of artificial NNs. The idea behind the "learn to learn" concept discussed in this section

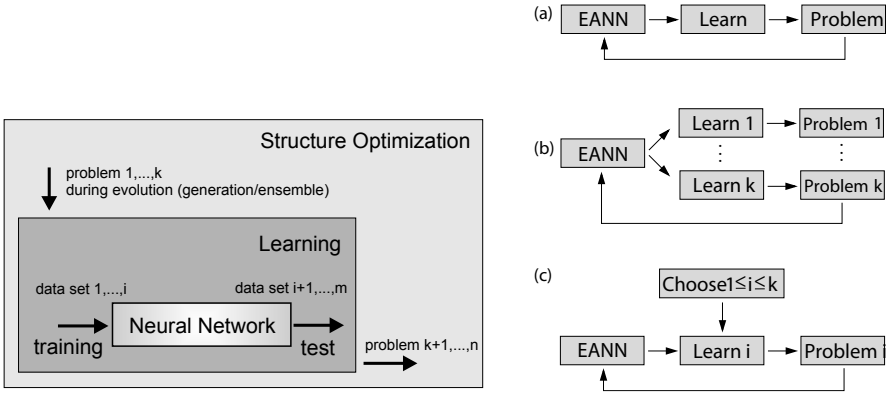


Fig. 7.4. Evolutionary structure optimization for problem classes. Left: Methods to achieve first and second order generalization in neural network learning and evolution (structure optimization). Right: Standard neural network learning (a), parallel switching between problems (b) and sequential switching (c). EANN denotes a neural network optimized by an evolutionary algorithm.

is that the goal of evolutionary NN structure optimization should be the ability to efficiently learn new related problems during operation, see [34, 37]. Here "efficient" means fast and based on incomplete data. The term "new related problems" is more difficult to define. The problems must have some common structure that can be captured by the EA and reflected in the NN architecture. Learning a different problem class goes beyond standard generalization. The latter means generalizing from a finite set of training samples to arbitrary samples drawn from the same distribution as, for example, formally defined at the end of section 7.2.1. Facing a different problem from the same class means that the underlying distribution has changed while belonging to the set of distributions which define the class and which have some common features that can be represented by the structure.

Therefore we speak of "second order generalization" for the ability to efficiently switch between problems, see figure 7.4 (left). In the notation introduced by Thrun and Pratt [99], this ability belongs to the area of representations and functional decompositions. However, in the evolutionary approach, this functional decomposition is self-organized during the evolutionary process. There are basically two different ways in which second order generalization can be achieved and used: the parallel and the sequential way. In figure 7.4 (right, a) the standard approach to learn one problem with an NN is shown. In part (b), the parallel approach is shown. The network is optimized during evolution in order to learn one of a number of possible problems. The actual decision is made after the network's structure has been fixed by evolutionary search. However, during the search the network's structure must be optimized in order to cope with any of the possible problems. Thus, each structure is ap-

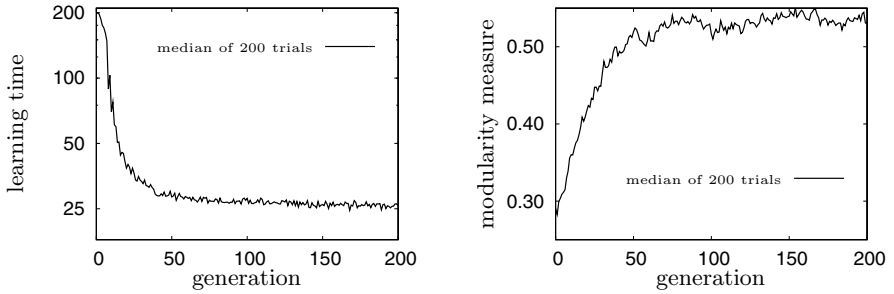


Fig. 7.5. Evolving networks that "learn to learn." During evolution the network structures adapt to a special problem class. This specialization leads to a reduced time for learning a new instance of the problem class, see left plot. Since the class in this example consists of separable problems, the degree of modularity of the network structures increases over time, as shown in the right plot (cf. [35]).

plied to all problems (or a random subset of problems of the respective class). For each problem the weights are newly initialized. The fitness of the network is determined by the mean (or median or weighted sum) of the networks' individual performances. In figure 7.4 (right, c), the network has to learn a number of problems one after the other during operation. The network's structure has been optimized in such a way that switching from problem to problem can be achieved most efficiently in the above sense. The weights are not randomly initialized (like in (b)), but averaged Lamarckian inheritance [36] is used to exploit information on previous problems for the next problem belonging to the same class. Again, the fitness of the network is determined by the mean (or median or weighted sum) of the networks' individual performances.

From the NFL theorems (see section 7.2.3) we conclude that adaptive systems have to be specialized towards a particular problem class to show above average performance. Second order generalization can be viewed as such a specialization.

7.4.1 Modularity

A simple example of how to build NNs that "learn to learn" was given in the study [35], where Hüsken et al. considered feed-forward NNs that had to learn binary mappings $\{0,1\}^6 \rightarrow \{0,1\}^2$ assigning target values $\mathbf{y} = (y_1, y_2)' \in \{0,1\}^2$ to inputs $\mathbf{s} = (s_1, \ldots, s_6)' \in \{0,1\}^6$. The class of mappings was restricted to those which are separable in the strict sense that y_1 only depends on the inputs $s_1, \ldots, s_3$ and y_2 only on $s_4, \ldots, s_6$. The mappings changed over time and a simple EA was employed to create feed-forward network structures that quickly learn a new instance of the problem class. The fitness of an NN structure was determined by the time needed to learn a ran-

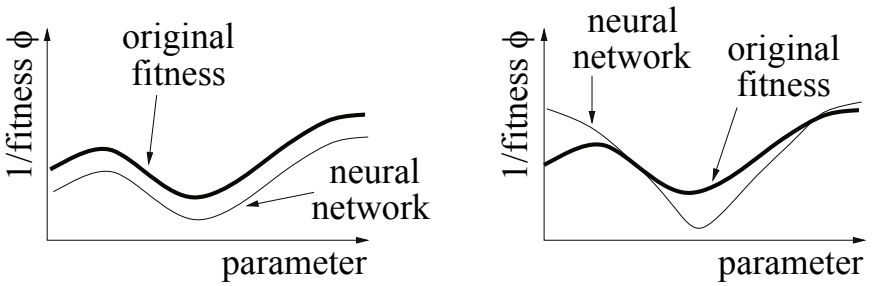


Fig. 7.6. Although the approximation errors of the neural network models are quite high, the optimization based on the approximation models leads to the desired optimum of the fitness under rank-based selection.

domly chosen problem instance, that is, the sequential switching approach depicted in figure 7.4 was used.

After a few generations, the networks adapted to the special, restricted problem class and the learning time decreased drastically, see figure 7.5. In this toy example, it is obvious that NN structures that are modular in the sense that they process the inputs $s_1, \ldots, s_3$ and $s_4, \ldots, s_6$ separately without interference are advantageous. When measuring this special kind of modularity during the course of evolution, it turned out that the modularity indeed increased, see figure 7.5, right plot.

In [53] modularity is analyzed in the context of problem decomposition and a novel modular network architecture is presented. Modularity is related to multi-network systems or ensembles for which a taxonomy is presented. A co-evolutionary framework is used to design modular NNs. The model consists of two populations, one consisting of a pool of modules and the other synthesizing complete systems by drawing elements from the first. In this framework, modules represent parts of the solution which co-operate with each other to form a complete solution. Using two artificial tasks the authors demonstrate that modular neural systems can be co-evolved. At the same time, the usefulness of modularity depends on the learning algorithm and the quality function.

7.4.2 Real-world application

Evolutionary algorithms combined with computational fluid dynamics (CFD) have been applied successfully to a large variety of design optimization problems in engineering (e.g., [90, 24, 67]). The fluid-dynamics simulations necessary to determine the quality of each design are usually computationally expensive, for example the calculation of the three-dimensional flow field around a car takes between 10-30 hours depending on the required accuracy. Therefore, metamodels or surrogates are used during the search to approximate the

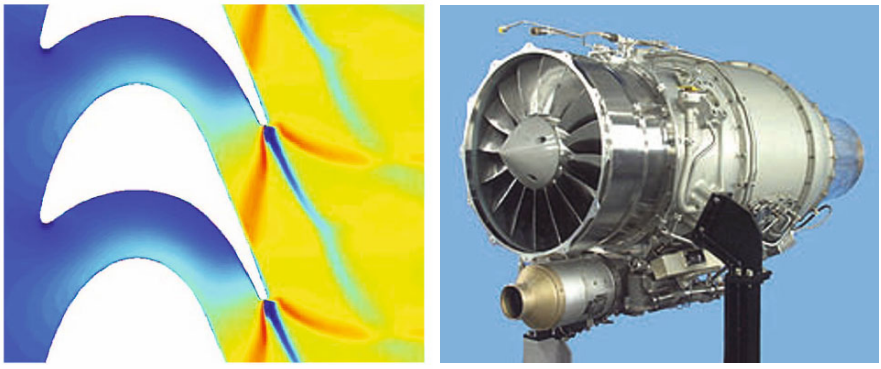


Fig. 7.7. The flow field of a turbine blade cascade for a gas turbine engine shown on the right.

results of the CFD simulations. Although first approaches to combine fitness approximation with EAs are relatively old [31], it is only in the last couple of years that the field has received wider attention, see [47] for a review. It has been revealed that the strategy to keep the update of the metamodel and the optimization process separate is not advisable, since the optimization is easily misled if the modeling quality is limited (which is often the case in practical applications). Jin et al. [50] have suggested to use the metamodel alongside the true objective function to guarantee correct convergence. Furthermore, the use of NNs as models is particularly advantageous because of their online learning ability. Thus, the approximation quality of NNs can be continuously improved during the optimization process when new CFD data is available (e.g, [50, 73, 69, 30]). It is interesting to note that the standard mean squared error measure of NNs is not necessarily the best means to determine the quality of NNs that are employed as surrogates. Figure 7.6 shows why this is the case. During evolutionary search, the absolute error of the NN is of no concern, as long as the model is able to distinguish between "good" and "bad" individuals.

7.4.2.1 Evolution of the metamodel

Neural networks that are used as metamodels during evolutionary search should have the best possible architecture for the approximation task. Therefore, EAs are employed to determine the structure of the networks offline, for example using data from previous optimization tasks. Weight adaptation is conducted during the evolutionary design optimization whenever new data are available.

This framework has been employed in [36] for the optimization of turbine blades of a gas turbine engine. The flow field around a turbine blade and the engine are shown in figure 7.7. Navier-Stokes equations with the (k-ϵ) tur-

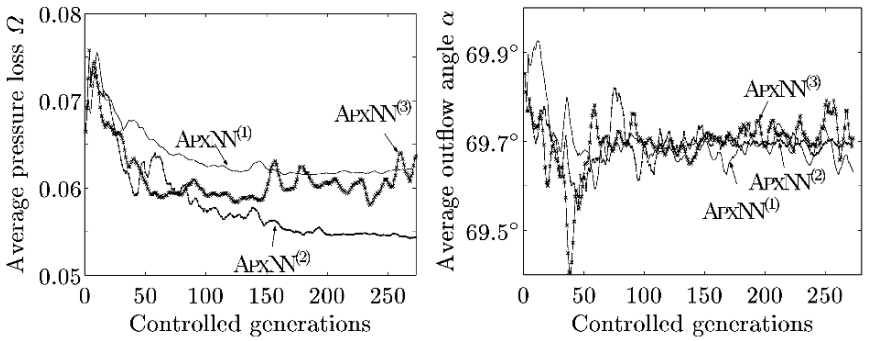


Fig. 7.8. Results normalized to the number of generations where the CFD simulations have been used. ApxNN(1) denotes a fully connected neural network, for ApxNN(2) the network structure has been evolutionarily optimized and for ApxNN(3) the network has been optimized to switch between different design domains (problem classes) most efficiently.

bulence model were used for the two dimensional CFD simulations. During optimization the pressure loss was minimized subject to a number of geometrical and functional constraints, in particular the target outflow angle α was set to 69.70 deg. The turbine blades were represented by 26 control points of non-uniform rational B-splines. The (x, y)-coordinates of the control points were optimized using a (2,11)-evolution strategy, further details can be found in [36].

The results of the optimization are given in figure 7.8. The average pressure loss and outflow angle are shown that have been reached in the evolutionary design optimization of the turbine blade. The three curves represent three different strategies to define the architecture of the NN that has been used as a metamodel during search. The model of the first type (ApxNN(1)) uses a fully connected architecture. The weights are initialized by means of offline learning, using training data collected in a comparable blade optimization trial (e.g., different initialization but the same number of control points of the spline and the same fitness function). The second type of network model (ApxNN(2)) was optimized offline with an EA using data generated in a previous optimization run. The third approach will be discussed in the next section. It is evident that the evolutionarily optimized NN structure clearly outperforms the fully connected model in the practical application.

7.4.2.2 *Learn* surrogates to *learn* CFD

We already discussed the idea to evolve the architecture of NNs not just for one specific problem but instead to optimize the network so that it is able to quickly adapt to problems belonging to one class. We can transfer this idea

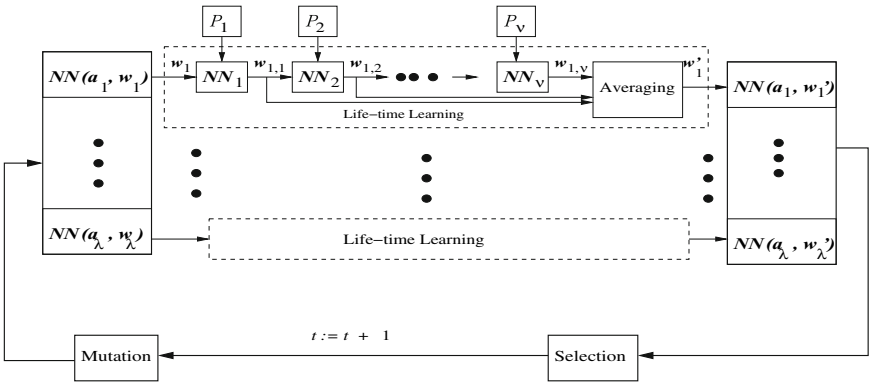


Fig. 7.9. Evolutionary structure optimization for problem classes with averaged Lamarckian inheritance, where the $\mathcal{P}_i$ denote the different problems belonging to one class and NN_i the network after learning $\mathcal{P}_i$. The symbol $w_{i,j}$ denotes the set of weights of the ith network after learning the jth problem, w_i' refers to the weights of the ith network after learning all ν problems. Averaged Lamarckian evolution is used to take the different problem characteristics into account for determining the set w_i', details can be found in [36]. The symbol a_i denotes the architecture or structure of the neural network, which is not changed during the sequential learning of problems $1 \ldots \nu$.

to the problem domain of surrogates for approximation during evolutionary design optimization by sub-dividing the CFD samples into groups (problems) belonging to one and the same class namely the approximation of CFD data for evolutionary search. This is a reasonable approach because we do not expect to evolve a network that approximates the CFD results well for the whole optimization. Instead, since the surrogate and the original CFD simulation are mixed during search, new data samples are available and the network can be adapted online. Thus, the best network is the one that is particularly well suited to continuously and quickly learn new CFD approximations during the evolutionary design optimization. In figure 7.9 the framework for the evolutionary optimization of the NN for problem classes is shown. To avoid confusion, we point out that the evolutionary optimization of the architecture is still decoupled from the evolutionary design optimization, where the best network is used as a surrogate.

The results for the network that has been evolved to quickly adapt to problems from one and the same class are shown as ApxNN(3) in figure 7.8. We observe that during the first generations ApxNN(3) scores much better than ApxNN(1) (the fully connected NN) and similar to ApxNN(2) (the network whose structure was optimized using a standard evolutionary approach to minimize the approximation error for all data offline). However, in later generations, the performance of ApxNN(3) deteriorates and becomes unstable. Although this

behavior is not yet fully understood, we believe that one reason might be the different update frequencies between the offline problem class training and the online design optimization. The update frequency denotes how often the original CFD simulation is called within a certain number of generations. As this frequency is adapted depending on the fidelity of the approximation model, it changes differently during offline structure optimization of the NN and online application of the network as a surrogate for the design optimization. Therefore, the definition of the problem class might change, which is difficult to cope with for the network.

7.5 Conclusion

Organic computing calls for adaptive systems. In order to be efficient and robust, these systems have to be specialized to certain problem classes comprising those scenarios they may face during operation. Nervous systems are perfect examples of such specialized learners and thus are prime candidates for the substrate of organic computing.

Computational models of nervous systems like artificial neural networks (NNs) have to be revisited in the light of new adaptation schemes that focus on the structure of the system and address issues like modularity, second-order generalization and learning efficiency.

At the same time, we promote the combination of evolutionary algorithms (EAs) and NNs not just because of an appealing metaphor, but also and foremost because EAs have proved to be well suited to solve many of the difficult optimization problems occurring when designing NNs, especially when higher order optimization methods cannot be applied. The field of evolutionary neural systems is expanding in many different directions as we have shown in this chapter. We have demonstrated how NNs can be evolved that are specialized to certain problem classes. Although still in its beginnings, this second order learning is not restricted to toy problems but has already found real-world technical applications.

Still, much is left to do to establish the design triangle learning–development–evolution of neural systems in such a way that they can demonstrate their full potential. Results from brain science highlight the importance of architecture and of the way the architecture is constructed during ontogenesis. Although the incorporation of evolution and development into computational neuroscience is still in its beginning, we believe that this will be a promising approach to organic computing.

Acknowledgments

Christian Igel acknowledges support from the German Federal Ministry of Education and Research within the Bernstein group "The grounding of higher brain function in dynamic neural fields".

References

1. H. A. Abbass. An evolutionary artificial neural networks approach for breast cancer diagnosis. *Artificial Intelligence in Medicine*, 25(3):265–281, 2002.
2. H. A. Abbass. Speeding up backpropagation using multiobjective evolutionary algorithms. *Neural Computation*, 15(11):2705–2726, 2003.
3. M. Anthony and P. L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.
4. K. Arai, S. Das, E. L. Keller, and E. Aiyoshi. A distributed model of the saccade system: simulations of temporally perturbed saccades using position and velocity feedback. *Neural Networks*, 12:1359–1375, 1999.
5. M. A. Arbib, editor. *The Handbook of Brain Theory and Neural Networks*. MIT Press, 2 edition, 2002.
6. M. A. Arbib. Towards a neurally-inspired computer architecture. *Natural Computing*, 2(1):1–46, 2003.
7. H.-G. Beyer, H.-P. Schwefel, and I. Wegener. How to analyse evolutionary algorithms. *Theoretical Computer Science*, 287:101–130, 2002.
8. C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.
9. C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.
10. O. Bousquet, S. Boucheron, and G. Lugosi. Introduction to Statistical Learning Theory. In *Advanced Lectures in Machine Learning*, volume 3176 of *LNAI*, pages 169–207. Springer-Verlag, 2004.
11. A. Chandra and X. Yao. Evolving hybrid ensembles of learning machines for better generalisation. *Neurocomputing*, 69(7–9):686–700, 2006.
12. K. Chellapilla and D. B. Fogel. Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE*, 87(9):1471–1496, 1999.
13. C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, 2002.
14. C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
15. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.
16. I. Das and J. E. Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. *Structural Optimization*, 14(1):63–69, 1997.
17. P. Dayan and L. F. Abbott. *Theoretical neuroscience: Computational and mathematical modeling of neural systems*. MIT Press, 2001.
18. K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2001.
19. L. Devroye and L. Gyorfi. *A Probabilistic Theory of Pattern Recognition*. Springer-Verlag, 1997.
20. S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.
21. D. R. Eads, D. Hill, S. Davis, S. J. Perkins, J. Ma, R. B. Porter, and J. P. Theiler. Genetic algorithms and support vector machines for time series classification. In B. Bosacchi et al., editors, *Applications and Science of Neural*

Networks, Fuzzy Systems, and Evolutionary Computation V., volume 4787 of *Proceedings of the SPIE*, pages 74–85, 2002.
22. D. B. Fogel, editor. *Evolutionary Computation: The Fossile Record*. IEEE Press, 1998.
23. D. B. Fogel, T. J. Hays, S. L. Hahn, and J. Quon. A self-learning evolutionary chess program. *Proceedings of the IEEE*, 92(12):1947–1954, 2004.
24. K. Foli, T. Okabe, M. Olhofer, Y. Jin, and B. Sendhoff. Optimization of micro heat exchanger: CFD, analytical approach and multi-objective evolutionary algorithms. *International Journal of Heat and Mass Transfer*, 49(5-6):1090–1099, 2005.
25. C. M. Friedrich and C. Moraga. An evolutionary method to find good building-blocks for architectures of artificial neural networks. In *Sixth International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems (IPMU'96)*, volume 2, pages 951–956, 1996.
26. F. Friedrichs and C. Igel. Evolutionary tuning of multiple SVM parameters. *Neurocomputing*, 64(C):107–117, 2005.
27. H. Fröhlich, O. Chapelle, and B. Schölkopf. Feature selection for support vector machines using genetic algorithms. *International Journal on Artificial Intelligence Tools*, 13(4):791–800, 2004.
28. K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 39:139–202, 1980.
29. A. Gepperth and S. Roth. Applications of multi-objective structure optimization. *Neurocomputing*, 6(7-9):701–713, 2006.
30. L. Graening, Y. Jin, and B. Sendhoff. Efficient evolutionary optimization using individual-based evolution control and neural networks: A comparative study. In M. Verleysen, editor, *13th European Symposium on Artificial Neural Networks (ESANN 2005)*, pages 273–278, 2005.
31. J. J. Grefenstette and J. M. Fitzpatrick. Genetic search with approximate fitness evaluations. In J. J. Grefenstette, editor, *International Conference on Genetic Algorithms and Their Applications*, pages 112–120. Lawrence Erlbaum Associates, 1985.
32. F. Gruau. Automatic definition of modular neural networks. *Adaptive Behavior*, 3(2):151–183, 1995.
33. S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1998.
34. M. Hüsken, J. E. Gayko, and B. Sendhoff. Optimization for problem classes – Neural networks that learn to learn. In X.Yao, editor, *IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*. IEEE Press, 2000. 98-109.
35. M. Hüsken, C. Igel, and M. Toussaint. Task-dependent evolution of modularity in neural networks. *Connection Science*, 14(3):219–229, 2002.
36. M. Hüsken, Y. Jin, and B. Sendhoff. Structure optimization of neural networks for aerodynamic optimization. *Soft Computing*, 9(1):21–28, 2005.
37. M. Hüsken and B. Sendhoff. Evolutionary optimization for problem classes with Lamarckian inheritance. In S.-Y. Lee, editor, *Seventh International Conference on Neural Information Processing – Proceedings*, volume 2, pages 897–902, Taejon, Korea, November 2000.

38. C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In R. Sarker et al., editors, *Congress on Evolutionary Computation (CEC 2003)*, volume 4, pages 2588–2595. IEEE Press, 2003.
39. C. Igel. Multiobjective model selection for support vector machines. In C. A. Coello Coello et al., editors, *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005)*, volume 3410 of *LNAI*, pages 534–546. Springer-Verlag, 2005.
40. C. Igel and M. Kreutz. Operator adaptation in evolutionary computation and its application to structure optimization of neural networks. *Neurocomputing*, 55(1-2):347–361, 2003.
41. C. Igel and P. Stagge. Effects of phenotypic redundancy in structure optimization. *IEEE Transactions on Evolutionary Computation*, 6(1):74–85, 2002.
42. C. Igel and M. Toussaint. On classes of functions for which No Free Lunch results hold. *Information Processing Letters*, 86(6):317–321, 2003.
43. C. Igel and M. Toussaint. A No-Free-Lunch theorem for non-uniform distributions of target functions. *Journal of Mathematical Modelling and Algorithms*, 3(4):313–322, 2004.
44. C. Igel, W. von Seelen, W. Erlhagen, and D. Jancke. Evolving field models for inhibition effects in early vision. *Neurocomputing*, 44-46(C):467–472, 2002.
45. C. Igel, S. Wiegand, and F. Friedrichs. Evolutionary optimization of neural systems: The use of self-adptation. In M. G. de Bruin et al., editors, *Trends and Applications in Constructive Approximation*, number 151 in International Series of Numerical Mathematics, pages 103–123. Birkhäuser Verlag, 2005.
46. J. Jägersküpper. How the (1+1) ES using isotropic mutations minimizes positive definite quadratic forms. *Theoretical Computer Science*, 36(1):38–56, 2006.
47. Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12, 2005.
48. Y. Jin. *Multi-objective Machine Learning*. Springer-Verlag, 2006.
49. Y. Jin, T. Okabe, and B. Sendhoff. Neural network regularization and ensembling using multi-objective evolutionary algorithms. In *Congress on Evolutionary Computation (CEC'04)*, pages 1–8. IEEE Press, 2004.
50. Y. Jin, M. Olhofer, and B. Sendhoff. A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation*, 6(5):481–494, 2002.
51. Y. Jin and B. Sendhoff. Reducing fitness evaluations using clustering techniques and neural network ensembles. In K. Deb et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO*, volume 1 of *LNCS*, pages 688–699. Springer-Verlag, 2004.
52. R. R. Kampfner and M. Conrad. Computational modeling of evolutionary learning processes in the brain. *Bulletin of Mathematical Biology*, 45(6):931–968, 1983.
53. V. R. Khare, X. Yao, and B. Sendhoff. Multi-network evolutionary systems and automatic decomposition of complex problems. *International Journal of General Systems*, 35(3):259–274, 2006.
54. H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4:461–476, 1990.
55. C. Koch and I. Segev. The role of single neurons in information processing. *Nature Neuroscience*, 3:1171–1177, 2000.
56. V. R. Konda and J. N. Tsitsiklis. On actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166, 2003.

57. H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974–978, 2000.
58. Y. Liu, X. Yao, and T. Higuchi. Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, 4(4):380–387, 2000.
59. S. M. Lucas and G. Kendall. Evolutionary computation and games. *Computational Intelligence Magazine, IEEE*, 1(1):10–18, 2006.
60. M. Mahner and M. Kary. What exactly are genomes, genotypes and phenotypes? And what about phenomes? *Journal of Theoretical Biology*, 186(1):55–63, 1997.
61. D. P. Mandic and J. A. Chambers. *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*. John Wiley and Sons Ltd, 2001.
62. A. Mark, H. Wersing, and B. Sendhoff. A decision making framework for game playing using evolutionary optimization and learning. In Y. Shi, editor, *Congress on Evolutionary Computation (CEC)*, volume 1, pages 373–380. IEEE Press, 2004.
63. G. F. Miller and P. M. Todd. Designing neural networks using genetic algorithms. In J. D. Schaffer, editor, *Proceeding of the 3rd International Conference on Genetic Algorithms*, pages 379–384. Morgan Kaufmann, 1989.
64. D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette. Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence Research*, 11:199–229, 1999.
65. S. Nolfi. Evolution and learning in neural networks. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 415–418. MIT Press, 2 edition, 2002.
66. S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. Intelligent Robotics and Autonomous Agents. MIT Press, 2000.
67. S. Obayashi, Y. Yamaguchi, and T. Nakamura. Multiobjective genetic algorithm for multidisciplinary design of transonic wing planform. *Journal of Aircraft*, 34(5):690–693, 1997.
68. Z. Pan, T. Sabisch, R. Adams, and H. Bolouri. Staged training of neocognitron by evolutionary algorithms. In P. J. Angeline et al., editors, *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 1965–1972. IEEE Press, 1999.
69. M. Papadrakakis, N. Lagaros, and Y. Tsompanakis. Optimization of large-scale 3D trusses using evolution strategies and neural networks. *International Journal of Space Structures*, 14(3):211–223, 1999.
70. F. Pasemann, U. Steinmetz, M. Hülse, and B. Lara. Robot control and the evolution of modular neurodynamics. *Theory in Biosciences*, 120(3-4):311–326, 2001.
71. M. Patel, V. Honavar, and K. Balakrishnan, editors. *Advances in the Evolutionary Synthesis of Intelligent Agents*. MIT Press, 2001.
72. A. Pellecchia, C. Igel, J. Edelbrunner, and G. Schöner. Making driver modeling attractive. *IEEE Intelligent Systems*, 20(2):8–12, 2005.
73. S. Pierret. Turbomachinery blade design using a Navier-Stokes solver and artificial neural network. *ASME Journal of Turbomachinery*, 121(3):326–332, 1999.

74. W. B. Powell, A. G. Barto, and J. Si. *Handbook of Learning and Approximate Dynamic Programming*. Wiley-IEEE Press, 2004.
75. E. T. Rolls and S. M. Stringer. On the design of neural networks in the brain by genetic evolution. *Progress in Neurobiology*, 6(61):557–579, 2000.
76. G. Rudolph. *Convergence Properties of Evolutionary Algorithms*. Kovač, Hamburg, 1997.
77. T. P. Runarsson and S. Sigurdsson. Asynchronous parallel evolutionary model selection for support vector machines. *Neural Information Processing – Letters and Reviews*, 3(3):59–68, 2004.
78. G. Schneider, H. Wersing, B. Sendhoff, and E. Körner. Coupling of evolution and learning to optimize a hierarchical object recognition model. In X. Yao et al., editors, *Parallel Problem Solving from Nature (PPSN)*, LNCS, pages 662–671. Springer-Verlag, 2004.
79. G. Schneider, H. Wersing, B. Sendhoff, and E. Körner. Evolutionary optimization of an hierarchical object recognition model. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 35(3):426–437, 2005.
80. S. Schneider, C. Igel, C. Klaes, H. Dinse, and J. Wiemer. Evolutionary adaptation of nonlinear dynamical systems in computational neuroscience. *Journal of Genetic Programming and Evolvable Machines*, 5(2):215–227, 2004.
81. B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.
82. C. Schumacher, M. D. Vose, and L. D. Whitley. The No Free Lunch and description length. In L. Spector et al., editors, *Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 565–570, San Francisco, CA, USA, 2001. Morgan Kaufmann.
83. B. Sendhoff. *Evolution of Structures – Optimization of Artificial Neural Structures for Information Processing*. Shaker Verlag, Aachen, 1998.
84. B. Sendhoff and M. Kreutz. A model for the dynamic interaction between evolution and learning. *Neural Processing Letters*, 10(3):181–193, 1999.
85. A. J. C. Sharkey. On combining artificial neural nets. *Connection Science*, 8(3-4):299–313, 1996.
86. D. Shi and C. L. Tan. GA-based supervised learning of neocognitron. In *International Joint Conference on Neural Network (IJCNN 2000)*. IEEE Press, 2000.
87. H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995.
88. J. Šíma. Training a single sigmoidal neuron is hard. *Neural Computation*, 14:2709–2728, 2002.
89. J. Šíma and P. Orponen. General-purpose computation with neural networks: A survey of complexity theoretix results. *Neural Computation*, 15(12):2727–2778, 2003.
90. T. Sonoda, Y. Yamaguchi, T. Arima, M. Olhofer, B. Sendhoff, and H.-A. Schreiber. Advanced high turning compressor airfoils for low Reynolds number condition, Part 1: Design and optimization. *Journal of Turbomachinery*, 126:350–359, 2004.
91. E. D. Sontag. Recurrent neural networks: Some systems-theoretic aspects. In M. Karny et al., editors, *Dealing with Complexity: A Neural Network Approach*, pages 1–12. Springer-Verlag, 1997.

92. O. Sporns, G. Tononi, and G. M. Edelman. Theoretical neuroanatomy: relating anatomical and functional connectivity in graphs and cortical connection matrices. *Cerebral Cortex*, 10(2):127–141, 2000.
93. P. Stagge and B. Sendhoff. An extended Elman net for modeling time series. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Artificial Neural Networks (ICANN'97)*, volume 1327 of *LNCS*, pages 427–432. Springer-Verlag, 1997.
94. K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Evolving neural network agents in the NERO video game. In *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05)*. IEEE Press, 2005.
95. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
96. R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla et al., editors, *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.
97. T. Suttorp and C. Igel. Multi-objective optimization of support vector machines. In Y. Jin, editor, *Multi-objective Machine Learning*, volume 16 of *Studies in Computational Intelligence*, pages 199–220. Springer-Verlag, 2006.
98. M.-Y. Teo, L.-P. Khoo, and S.-K. Sim. Application of genetic algorithms to optimise neocognitron network parameters. *Neural Network World*, 7(3):293–304, 1997.
99. S. Thrun and L. Pratt, editors. *Learning to Learn*. Kluwer Academic Publishers, 1998.
100. J. Tsitsiklis and D. Bertsekas. *Neurodynamic programming*. Belmont, MA: Athena Scientific, U.S.A., 1996.
101. J. Walker, S. Garrett, and M. Wilson. Evolving controllers for real robots: A survey of the literature. *Adaptive Behavior*, 11:179–203, 2003.
102. H. Wersing and E. Körner. Learning optimized features for hierarchical models of invariant recognition. *Neural Computation*, 15(7):1559–1588, 2003.
103. S. Wiegand, C. Igel, and U. Handmann. Evolutionary multi-objective optimization of neural networks for face detection. *International Journal of Computational Intelligence and Applications*, 4(3):237–253, 2004.
104. D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.
105. D. H. Wolpert and W. G. Macready. No Free Lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.
106. D. H. Wolpert and W. G. Macready. Coevolutionary free lunches. *IEEE Transactions on Evolutionary Computation*, 9, 2005.
107. X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

8

Organically Grown Architectures: Creating Decentralized, Autonomous Systems by Embryomorphic Engineering

René Doursat

Institut des Systèmes Complexes (ISC),
Centre de Recherche en Epistémologie Appliquée (CREA),
CNRS and Ecole Polytechnique, 57-59, rue Lhomond, 75005 Paris, France.
doursat@shs.polytechnique.fr

Summary. Exploding growth in computational systems forces us to gradually replace rigid design and control with decentralization and autonomy. Information technologies will progress, instead, by "meta-designing" mechanisms of system self-assembly, self-regulation and evolution. Nature offers a great variety of efficient complex systems, in which numerous small elements form large-scale, adaptive patterns. The new engineering challenge is to recreate this self-organization and let it freely generate innovative designs under guidance. This article presents an original model of artificial system growth inspired by embryogenesis. A virtual organism is a lattice of cells that proliferate, migrate and self-pattern into differentiated domains. Each cell's fate is controlled by an internal gene regulatory network. *Embryomorphic* engineering emphasizes hyperdistributed architectures, and their development as a prerequisite of evolutionary design.

Key words: complex systems, artificial development, evolutionary computation, systems design, embryomorphic engineering

8.1 Introduction: designing complexity

8.1.1 Toward decentralized, autonomous systems

Today's information and communication systems are characterized by exploding growth in the number of components and complexity of their interactions. Systems engineers are confronted with an insatiable demand for functional innovation, robustness, scalability and security. This upward trend is accelerating at all levels of organization, whether hardware (integrated components), software (program modules) or networks (applications and users). Famously anticipated by Moore's law, the number of transistors on a microprocessor has climbed five orders of magnitude in the past 35 years. Similarly, operating systems and other very large computer programs commonly contain hundreds of

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_8,

millions of source lines of code (SLOC). Over one billion people routinely use the Internet, which connects half a billion hosts. In sum, an increasing number of users with greater mobility are constantly requiring more sophisticated functionality from larger applications running on faster architectures.

Consequently, computer scientists and engineers are gradually led to rethink the traditional perspective on systems design, i.e., the dogma of a totalistic act of creation imposing order and organization *exogenously*. The growth in complexity has already been accompanied by a de facto segmentation and distribution of the traditionally centralized control over systems design. This march toward decentralization is somewhat discernible in the fields of integrated circuit design and software development, where engineers collaborate in large teams around relatively independent components and modules. It has become even more apparent with the advent of leaderless open source communities, and most striking in the spontaneous growth of the Internet and World-Wide Web. To some degree, information architects and engineers are already beginning to lose grip on their creation, which exceeds the capacity of a single human mind. Therefore, rather than insisting on rigidly designing computational systems or system parts in every detail, the trend should be to "step back" even further and concentrate more on establishing the *generic conditions* that will allow and encourage those systems to self-assemble, self-regulate and evolve. In fact, future progress in information and communication technologies could ultimately depend on our ability to foster systems that *endogenously* grow, function, repair themselves and, more importantly, adapt and improve. This need is probably most acute in software development, which is currently less an exact science than a skillful art — the accumulation by trial and error of a corpus of design patterns and numerical recipes. Ironically, machines that are perfectly logical and regular still rely entirely on intuitive and fallible human instructions. The burden is fully on the side of our human cognitive system to instruct artificial systems, but this ability is now reaching its limits with very large architectures, as attested by the extremely high cost (in effort, time and money) devoted to source code development, maintenance and debugging.

A major challenge will thus be for information systems to step beyond their current state of heteronomy, where they are fully subjected to a designer, toward states of increasing organizational and functional autonomy. Biological organisms, which might give the *illusion* of deliberate design, are in fact the product of *undesigned evolution* through random variation and non-random natural selection, excluding the need to invoke any form of intelligent design for them (which would also necessarily be, by recursive reasoning, of a supernatural kind). By contrast, artificial structures will always possess a causal design link originating from their human makers, while at the same time this link promises to become less and less clear or direct. In the design versus evolution spectrum (figure 8.1), classical engineering is currently set on the "intelligent design" (ID) notch, with the opposite end occupied by biology under "undesigned evolution" (UE). The expectation is that the engineering

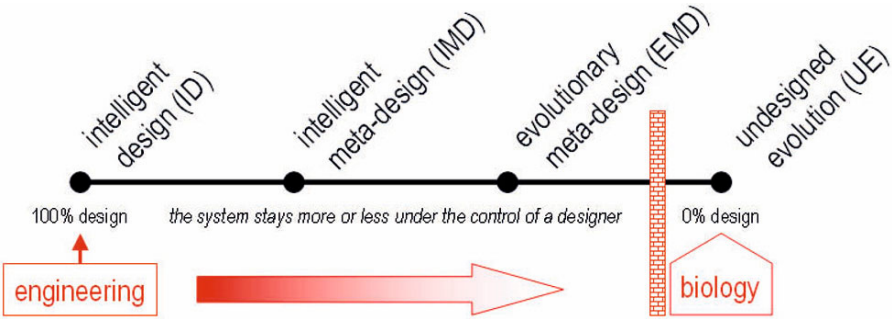


Fig. 8.1. Four stops and one brick wall on the design-evolution line (see text).

paradigm should progressively shift in the direction of biology through intermediate stages, but without fully reaching UE. In a first stage that we could refer to as "intelligent *meta-design*" (IMD), designers will focus on creating generative mechanisms rather than the systems themselves (Table 8.1). If we metaphorically imagined, on the contrary, biology drifting toward more design, ID would be the equivalent of directly assembling an animal's organs and limbs together, whereas IMD would correspond to creating the laws of cellular development, preparing the zygote's DNA and let it grow. Still closer to UE, in a stage we could call "evolutionary meta-design" (EMD), an even more disengaged meta-architect could also create the laws of variation and selection, prepare some primitive ancestor system (in the reverse biological metaphor, a prokaryote, for example) and step back to let evolution invent the rest. Applied to artificial systems, this paradigm shift is the inspiration of the present work. It emphasizes the importance of constituting fundamental laws of *development* and developmental *variations* in the IMD stage, before these variations can be selected upon in the EMD stage. In the framework of genetic algorithms and evolutionary computation, this means an *indirect* or *implicit* mapping (as opposed to direct or explicit) from genotype to phenotype.

8.1.2 Harnessing complex systems

Looking around, we observe an abundance of autonomous, emergent systems in the environment, whether in nature (geological patterns, biological cells, organisms, animal societies, ecosystems) or spontaneously emerging human super-structures (cities, markets, the Internet). Naturally decentralized, unplanned systems are robust and efficient, and constitute the overwhelming majority of system types. It is our artificially centralized and planned systems that are fragile, costly to build and rare, as they require a higher intelligence to arise. Our cognitive viewpoint, accustomed to the illusion of a central consciousness, traditionally refers to such decentralized systems as "complex",

systems design	systems "meta-design"
heteronomous order	autonomous order
centralized control	decentralized control
manual, extensional design	automated, intentional design
the engineer as a micromanager	the engineer as a lawmaker
rigidly placing components	allowing fuzzy self-placement
tightly optimized systems	hyperdistributed and redundant systems
sensitive to part failures	insensitive to part failures
need to control	prepare to adapt and self-regulate
need to redesign	prepare to learn and evolve

Table 8.1. Some contrastive features of systems design and "meta-design".

whereas in fact they might be simpler than our familiar contraptions with their uniquely hierarchical arrangement.

Complex systems are composed of a great number of small, repeated elements that interact locally and produce collective behavior at a macroscopic scale. They are characterized by a high degree of decentralization and self-organization, exhibiting spontaneous pattern formation (self-assembly) and homeostatic autonomy (self-regulation). Most complex systems are also adaptive, in the sense that they are able to learn and/or evolve through feedback from their external fitness to their internal architecture. The elements composing the system are themselves often internally structured as networks of smaller entities at a finer scale. For example, one cell can be modeled as a self-regulatory network of genetic switches, one social agent (insect) as a network of decision rules, or one neural unit as a local assembly of neurons (oscillator system). Conversely, agents can also interact collectively at the level of clusters or subnetworks (organs, assemblies, cliques) to combine in a modular fashion and form larger collectives. Thus, from both perspectives, complex systems can often be described as "networks of networks" on several hierarchical levels. The higher levels connecting elements or clusters of elements are generally spatially extended (cell tissues, cortical areas, ant colonies), whereas the lower levels inside elements are generally nonspatial (gene nets, neurons, rule sets). Elements follow the dynamics dictated by their inner networks and also influence neighboring elements through the emission and reception of signals (chemical, electrical). The attractors of the internal dynamics are fixed-point states or limit cycles, and the behavior of the whole connected system can be rephrased in terms of synchronization among autonomous dynamical subsystems. The work presented in this chapter is an instance of this paradigm based on a 2-D lattice of coupled gene regulatory networks.

Such natural adaptive systems, biological or social, could become a new and powerful source of inspiration for emerging information and communica-

tion technologies in their transition toward autonomous systems. This joins recent trends advocating and announcing the convergence of four scientific disciplines: nanoscience, biotechnology, information technology and cognitive science. Called NBIC in the US [2], these fields of investigation combine all the components of bio-inspired complex systems engineering, i.e., swarms of small components (Nano), biological complexity (Bio), systems design (Info) and artificial intelligence (Cogno). Also described by the Future and Emerging Technologies (FET) program of the European Union [5], this scientific perspective is close to several other initiatives, such as organic computing ([37], and this volume, in particular chapters 1 and 2), amorphous computing [1, 25, 41], natural computation, e.g., [28], complex systems engineering [23], ambient intelligence, and pervasive or ubiquitous computing [40].

As indicated above, however, a major difference with biological systems is that human-made systems will (and hopefully should), by definition, always remain under the guidance of a human designer to some degree, never breaking the barrier to the absolute UE stage that characterizes biology (figure 8.1). While we want to achieve meta-designing artificial systems that can grow (IMD stage) and evolve (EMD stage), it is obviously our intent to keep a partially "visible hand" on these systems, i.e., (a) some meta-control over their execution and (b) certain requirements about their structure or function. The important questions of control and optimization of complexity will be respectively addressed in sections 8.3 and 8.4 below.

8.1.3 Artificial development

The field of Artificial Life (AL) is chiefly concerned with the simulation of life-like or organismal processes through computer programs or robotic devices that generally are of a distributed nature and operate on a multitude of interacting components. Researchers in AL attempt to design and construct systems that have the characteristic of living organisms or societies of organisms out of non-living parts, whether virtual (software agents) or physical (electromechanical components, chemical materials). AL is, therefore, a "bottom up" attempt to recreate or synthesize biological phenomena with the goal of producing adaptive and intelligent systems. In this sense, it can be contrasted with the traditional "top down" analytical approach of Artificial Intelligence based on symbolic systems. Although not all AL systems are "complex", in the sense of a multitude of elements, AL is one of the most important and rapidly developing domains within the federation of complex systems research. In particular, it actively promotes biology-inspired engineering as a new paradigm complementing or replacing classical physics-based engineering.

AL opens entirely new perspectives in software, robotic, electrical, mechanical or even civil engineering. Can a sophisticated device or building architecture construct itself from a large reservoir of small components? Can a robot rearrange its parts and evolve toward better performance without explicit

instructions? Can a swarm of software agents self-organize and collectively innovate in problem-solving tasks? Among the great variety of biological systems that inspire and guide AL research, three broad areas can be identified according to the scale of their elementary components: (a) molecular or cellular systems, (b) anatomical or functional systems, and (c) individual or societal systems. Artificial molecular and cellular models find inspiration in the spontaneous organization of complex chemical and organic structures, such as protein self-assembly or organism development (e.g., [22], and chapter 9 of this volume). Applications are linked to nanotechnologies for biomedical or integrated electronic purposes ("smart materials", MEMS). On the anatomical and functional level, robotic parts (limbs, sensors, actuators) and local behavioral modules are integrated and put in interaction to produce emergent action in a single autonomous device, aiming toward adaptivity and nonsymbolic intelligence. This is the scope of "reactive" or "embodied" robotics, exemplified by insect-like robots and evolving mechanical morphologies (e.g., [20]). Finally, entire colonies of virtual or robotic creatures also constitute important objects of interest because of their unique properties of collective self-organization and diversity-inducing evolution (e.g., [41]). Generically termed "swarm intelligence", new methodologies such as ant colony optimization or particle swarm optimization are derived from the observation of animal societies and applied to problem-solving tasks.

The preferred computational tools of AL are cellular automata, multiagent networks and genetic algorithms. Complex networks form the natural structural backbone of AL models. Their topology can vary from regular lattices with nearest neighbor connectivity (cellular automata) to irregular graphs (random, small-world) containing long-range interactions. The first kind is spatially extended, in 2-D or 3-D, while the second generally does not rely on a background notion of space or Euclidean distance. This chapter presents an original AL study of the spatially explicit kind. With respect to the above classification, it addresses level (a) of system organization, specifically the computational modeling and simulation of the fundamental principles of self-patterning and self-assembly during embryogenesis. The development of an entire organism from a single cell is the most striking example of self-organization *guided* by information — in this case, genetic. In the present model, a virtual organism is represented by a mass of cells that proliferate, migrate and self-pattern into differentiated domains. Each cell contains an internal gene regulatory network and acquires a specific expression identity by interaction with positional information transmitted through neighboring cells. Different identities trigger different cell behaviors, which in turn induce new identities. In sum, development is driven by only a few fundamental laws of cell division and movement, propagation of genetic expression and positional information. The final architecture of the organism depends on the detailed interplay between the various rates of these processes.

Ultimately, on this score of "theme and variations" (laws and parameters), evolution is the player. Most importantly, the link between the genetic

parameters and the morphological features of the system is not arbitrary, as is generally the case in many evolutionary algorithm techniques, but expresses itself through a genuine developmental approach at the microscopic cellular level. The phenotype is a macroscopic *emergence* of the unfolding genotype, not an ad hoc one-to-one mapping. Possible future hardware applications of this model include systems in which nano-units containing the same instructions are mass-produced at low cost and mixed in a homogeneous material, where they self-organize without the need for reliability or precise arrangement as in traditional VLSI [1, 25]. Software or network applications (servers, security) could involve a swarm of small-footprint software agents that diversify and self-deploy to achieve a desired level of functionality. In all cases, embryo-inspired architectures suggest a "fine-grain" approach to systems design, i.e., one based on hyperdistributed collectives of a great number of very simple and relatively ignorant, cloned elements. This approach is called here *embryomorphic* engineering.

The remainder of the chapter is organized as follows. After preliminary remarks on the genetic causality of biological development in section 8.2, a virtual embryogenesis model, "the organic canvas", is described in section 8.3 in four incremental steps. Section 8.3.1, "the self-painting canvas", introduces the concept of genetically guided *self-patterning*. Section 8.3.2, "the growing canvas", adds a *multiscale* and *modular* dimension to this pattern formation process. Section 8.3.3, "the deformable canvas", brings in *self-assembly* through three critical mechanisms of cell movement: adhesion, division and migration. Finally, section 8.3.4, "the excitable canvas", briefly explores the possible *computing* capabilities of a fully developed organism. The purpose of section 8.3 is, thus, to lay out the IMD foundations of the model, by showing an example of *lawmaking* of artificial system development with inspiration from biology. Section 8.4 then discusses the transition to the EMD stage, i.e., the role that *evolution* could play in shaping the genome at the basis of the developmental process, and inventing new architectures. Specifically, it addresses the paradox of "planning the autonomy" at the center of the complex systems engineering enterprise.

8.2 The genetic causality of biological development

8.2.1 Free versus guided morphogenesis

Complex patterns produced by nature have always been a source of great fascination for philosophers and scientists. Ripples in sand dunes, spots in animal coats, geometric figures in plants, and a multitude of meanders, spirals, branches, lattices, and others, can be observed everywhere. Whether inanimate structures or living organisms, all these processes are instances of decentralized morphological self-organization and, as such, were not easily amenable to analysis and explanation. For a long time, in fact, the old cross-disciplinary and abstract problem of the "form" was deemed non-objective

by mainstream physics and relegated to the phenomenological realm of experience. Only relatively recently was it revived as a valid object of scientific inquiry, under rising needs for a better understanding, prediction and control of geophysical and biological systems. New computer technologies and numerical simulations created dramatic new advances in the understanding of objectively measurable complex forms and their emergent properties of order and chaos.

A taxonomy of all emergent patterns would contain many dimensions: inert vs. living, natural vs. human-induced, small-scale vs. large-scale, and so on. The present study focuses on a major distinction between what will be referred to as "*free* forms" and "*guided* forms" (figure 8.2). Free forms essentially result from the amplification of unstable fluctuations, as proposed by Turing in his now classical reaction-diffusion model of morphogenesis [35], which was further developed and popularized by Gierer and Meinhardt [14]. A pigmented medium, such as an animal coat, undergoes a symmetry-breaking due to positive feedback based on the short-range diffusion of an activator substance, reacting with negative feedback based on the long-range diffusion of an inhibitor substance [44]. In 2-D domains, this typically generates spots or stripes of alternating color (figure 8.2a-b). Setting aside questions about the actual existence of activator and inhibitor "morphogen" agents, it remains that the pattern formation phenomena covered by this model are fundamentally random and unpredictable. Are there going to be four, five, or six spots? Although the patterns are often statistically homogeneous and can be described by a characteristic scale or order parameter (diameter of the spots, width of the stripes), morphological details such as position, orientation and number are *not* invariants of the system. Another example of free patterning is given by convection cells in a heated fluid, such as the ones observed in the well-known Rayleigh-Bénard instability. Given the temperature gradient and other parameters of the fluid, it is possible to calculate the typical size of the polygonal convection domains but, again, *not* their precise shape and spatial arrangement.

Turing-like reaction-diffusion principles might be able to account for some pattern formation effects in biological development, such as mammal coat [44], butterfly wing spots [26], angelfish stripes [18], or seashell motifs [21], yet these effects seem only secondary — literally "superficial" — compared to the overall form of an organism. The precisely arranged body shape of animals, made of articulated segments and subparts (figure 8.2c-d), is not the result of free-forming random instabilities. It is a fundamentally *guided* morphogenesis process that plays out under deterministic control from the genome. Except for very rare cases of malformation, all members of a pentadactyl mammalian species reliably develop five digits, not sometimes four or sometimes six. All healthy embryos of *Drosophila* exhibit exactly seven bands of differentiated gene expression along the anteroposterior axis, which then give rise to 14 segments. Each one of these mammal digits or insect segments is independently controlled by a specific combination of genes. At every time step in the de-

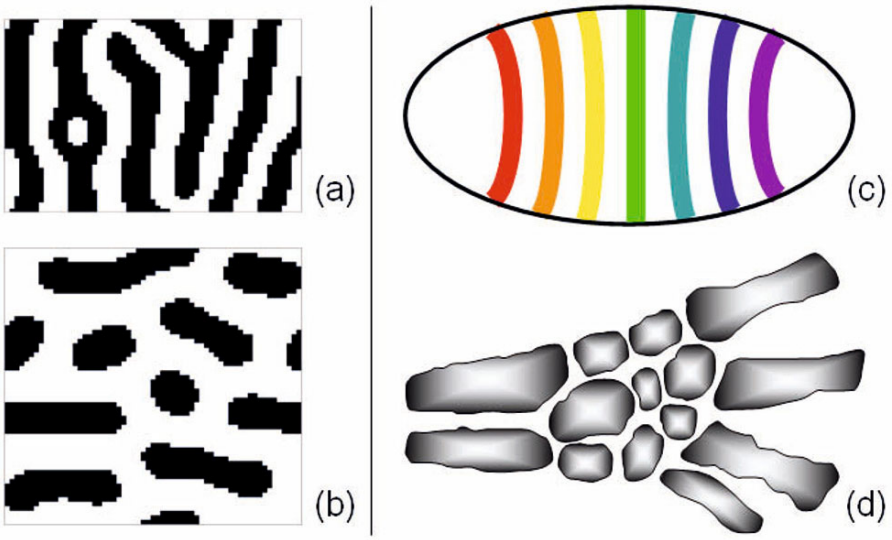


Fig. 8.2. Free vs. guided morphogenesis. A simple activator-inhibitor cellular automata model, such as Young's [44], creates stripes (a) and spots (b) in variable positions and unpredictable numbers. By contrast, the stripes (c) and spots (d) of developing animal segments are tightly controlled by multiple sets of genes, leaving very little room for chance arrangements.

velopment of an embryo, a homogeneous region of the overall embryo pattern is defined as a local group of cells that have the same gene expression profile, i.e., the same dynamic regime of RNA and protein concentrations.

In summary, biological forms are not statistically uniform. They are rich in morphological information and cannot be reduced to one characteristic scale like reaction-diffusion patterns. Some free pattern motifs (spots, stripes) can be embedded in a guided form (leopard, angelfish). Conversely, a guided form can be duplicated and distributed in free patterns (e.g., hundreds of copies of the same flower shape on the branches of one tree). Biological forms can thus combine a little free patterning with a lot of guided morphogenesis. It is the latter kind that the present work aims at modeling and reproducing as a possible paradigm of information-driven systems growth.

8.2.2 Development: the missing link of the modern synthesis

Darwin discovered the evolution of species, based on random variation and nonrandom natural selection, and established it as a central fact of biology. During the same period, Mendel brought to light the laws of inheritance of traits. In the twentieth century, his work was rediscovered and became the foundation of the science of genetics, which culminated with the revelation of DNA's role in heredity by Avery and its double-helix structure by Watson and

Crick. By integrating evolution and genetics together, the "modern synthesis" of biology has demonstrated the existence of a fundamental correlation between genotype and phenotype. Mutation in the first is causally related to variation in the second. Yet, 150 years after Darwin's and Mendel's era, the nature of the link from genes to organismal forms, i.e., the actual molecular and cellular basis of the *mechanisms* of development, are still unclear. To quote Kirschner and Gerhart [16, page ix]:

> "When Charles Darwin proposed his theory of evolution by variation and selection, explaining selection was his great achievement. He could not explain variation. That was Darwin's dilemma [...] To understand novelty in evolution, we need to understand organisms down to their individual building blocks, down to their deepest components, for these are what undergo change."

Understanding variation by comparing the actual *developmental* processes of different species is the primary concern of the field of evolutionary development biology, or "evo-devo". The genotype-phenotype link cannot remain an abstraction if we want to unravel the *generative* laws of development and evolution. The goal is to unify what Darwin called the "endless forms most beautiful" of nature [7], and reduce them to *variants* around a common theme [39]. The variants are the specifics of genetic information; the common theme is the developmental dynamics that this information guides. Modern synthesis postulates this reduction in principle but has never truly explained it physically.

How does a static, nonspatial genome dynamically unfold in time and 3-D space [13]? *How* are morphological changes correlated with genetic changes? Looking at the full evolutionary *and* developmental picture should also be a primary concern of systems engineering and computer science when venturing in the new arena of autonomous architectures. Optimization techniques inspired by biology in its traditional modern synthesis form have principally focused on evolution, giving rise to evolutionary computation and genetic algorithms based on metaphorical "genes", "reproduction", "mutation" and "selection". However, the great majority of these approaches rely on a direct mapping from artificial genomes to artificial phenotypes, which includes very few or no elements of morphodynamics. The present work's ambition is to contribute to restore the balance between evo and devo by shifting the emphasis on developmental meta-design as a *prerequisite* of evolutionary meta-design.

8.3 Description of the model: the organic canvas

This part describes a model of embryomorphic system development partly introduced in [11].

8.3.1 The self-painting canvas: gene-guided patterning

8.3.1.1 Gene regulatory networks

The central dogma of molecular biology states that a segment of DNA representing a gene is transcribed into messenger RNA, and then mRNA is translated into proteins by ribosomes and transfer RNA. The present model adopts a highly simplified one-to-one view of the gene-protein processing chain, ignoring additional effects such as post-transcriptional RNA splicing or post-translational protein modifications. However, it still retains and places at its core the concept of *gene regulation*. DNA contains non-coding sequences that play a critical regulatory role in the expression of genes. Various proteins can selectively bind to regions of the DNA strand upstream of a gene domain and interfere, positively or negatively, with the RNA polymerase responsible for gene transcription. The two main classes of transcription factors are "activators" and "inhibitors" that respectively encourage and hinder gene expression. In a binary view, the regulatory sites are "switches" that literally turn genes on and off. Regulatory proteins bind to regulatory sites as keys fit into locks, which can cluster and combine to form complex regulatory functions. Lock-key pairs are reused for different genes or even the same gene at different times and places of the developing organism. Since regulatory proteins are themselves the product of gene expression, the cell's total biosynthetic activity can be approximately represented by a *gene regulatory network* (GRN), where proteins are considered hidden variables (figure 8.3a,b,e). In sum, gene expression is controlled by regulatory switches, which are themselves controlled by gene expression.

8.3.1.2 Patterning from gene regulation in embryo space

How does this complex web of many-to-many regulatory interactions unfold in 3-D space and time to create pattern formation? The pattern domains of embryogenesis are differentiated regions of gene expression, or *identity domains*. They represent the "hidden geography" of the embryo [9]; at any period of its development, the organism is segmented into multiple compartments of approximately homogeneous gene expression levels (see figure 8.4 for a preview). These compartments can be visualized, for example, by *in situ* hybridization methods (i.e., complementary RNA strands that recognize specific mRNA and are labeled with fluorescent or radioactive compounds). Based on these facts, simple recursive reasoning yields the following patterning rule. First, it is assumed that gene expression levels in each cell (or, equivalently, mRNA or protein concentration levels) can be represented by quasi-static variables, because their reaction kinetics quickly converges to constant attractor values. Then, the combined regulatory action of three genes A, B, and C upon one gene I can be denoted by $I = f(A, B, C)$, where I, A, B, and C represent the stable expression levels of those four genes within a given time

interval (figure 8.3b). Denoting by $\mathbf{r} = (x, y, z)$ the coordinates of a cell, $I(\mathbf{r})$ represents the spatial landscape of gene I's expression level across the embryonic cell population. Therefore, through the dependency in f, the basic patterning rule states that a gene landscape $I(\mathbf{r})$ results from a geometric interaction between several earlier gene landscapes $A(\mathbf{r})$, $B(\mathbf{r})$, and $C(\mathbf{r})$ (figure 8.3c,d,g). Typically, in a simplified binary format, gene levels are coded by two values, 1 for "high" and 0 for "low", and $I(\mathbf{r})$ defines a geometrical domain D_I such that $\mathbf{r} \in D_I \Leftrightarrow I(\mathbf{r}) = 1$. In this case, function f has a logical type, e.g., $I = (\neg A \wedge B \wedge C) = (1 - A)BC$, and the domain of high I expression is simply the intersection of high B, high C and low A expression, i.e., $D_I = (D - D_A) \cap D_B \cap D_C$, where D denotes the entire domain of the organism.

Thus, combinations of switches can create new patterns by union and intersection of precursor patterns. This principle was demonstrated in the periodic striping of the *Drosophila* embryo along its anteroposterior (A/P) axis. The dorsoventral (D/V) and proximodistal (P/D) axes are also segmented into distinct bands or layers and, by intersection with the A/P stripes, give rise to smaller domains such as the organ primordia and "imaginal discs". These groups of cells mark the location and identity of the fly's future appendages (legs, wings, antennae). Going back in time, the whole process starts with the establishment of concentration gradients due to the diffusion of various maternal proteins across the initial cluster of cell nuclei, the syncytium. These gradients are the functional equivalent of a coordinate system. The particular combination of protein concentration in each point becomes the first regulatory trigger in a cascade of gene expression. Let X, Y and Z represent the concentration levels of three hypothetical proteins that vary anisotropically along the three dimensions of an abstract embryo. For example, assuming uniform gradients $\nabla X = (\alpha, 0, 0)$, $\nabla Y = (0, \beta, 0)$ and $\nabla Z = (0, 0, \gamma)$, we obtain three linear concentration landscapes $X(\mathbf{r}) = \alpha(x - x_0)$, $Y(\mathbf{r}) = \beta(y - y_0)$ and $Z(\mathbf{r}) = \gamma(z - z_0)$. The first set of genes will be expressed in domains defined by regulatory functions of the type $A(\mathbf{r}) = g(X(\mathbf{r}), Y(\mathbf{r}), Z(\mathbf{r}))$, and $B(\mathbf{r}) = h(X(\mathbf{r}), Y(\mathbf{r}), Z(\mathbf{r}))$, etc. Then, these primary domains will intersect to give rise to secondary domains such as $I(\mathbf{r}) = f(A(\mathbf{r}), B(\mathbf{r}), C(\mathbf{r}))$, etc., as explained above. In summary, embryogenesis consists of a cascade of morphological refinements supported by a cascade of gene regulation reactions. Molecular gradients provide *positional information* [43] that is integrated along several spatial dimensions, in each cell nucleus, through a chain reaction of keys and locks.

8.3.1.3 The positional-boundary-identity gene network model

The principle of recursive morphological refinement suggests that, despite numerous feedback loops and an overall complex topology, developmental GRNs seem to be broadly organized in successive gene groups that correspond to successive growth stages and anatomical modules of the embryo. The early

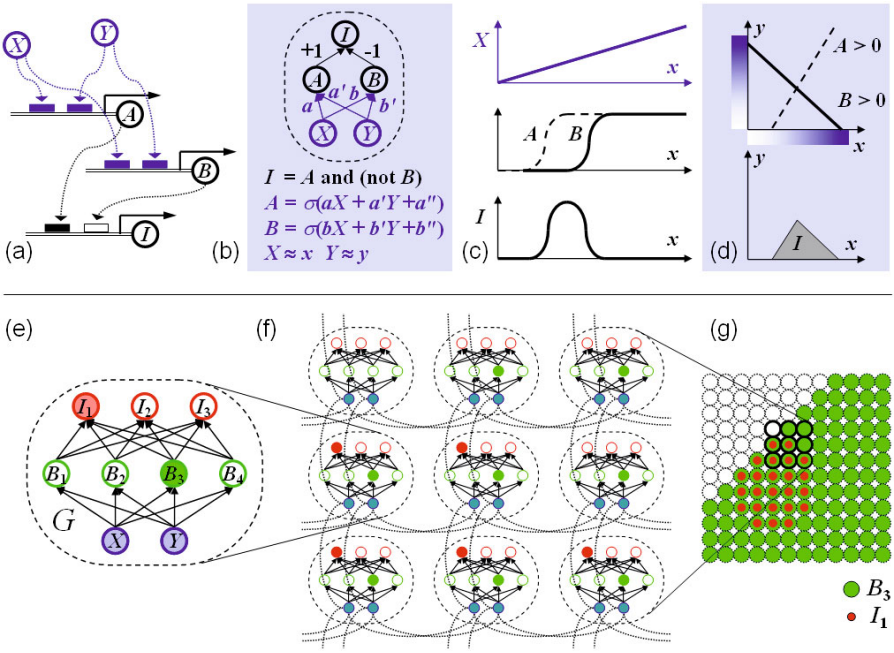


Fig. 8.3. Principles of spatial patterning from a lattice of positional-boundary-identity (PBI) gene networks. (a) Schematic top-down view of gene regulatory interactions on DNA strands. Proteins X and Y combine to promote the transcription of genes A and B by binding to their upstream regulatory sites, which produces proteins A and B (assuming a simple one gene-one protein relationship). Thereafter, A promotes, but B represses, the synthesis of I. (b) Formal bottom-up view of the same GRN. (c) Variation of expression levels on one spatial axis, construed as a chain of GRNs. The concentration of X follows a gradient created by diffusion. This gradient triggers a gain response in A and B at two different thresholds, thus creates boundaries at two different x coordinates (for a given Y level). These domains in turn define the domain of identity gene I, where A levels are high but B levels are low. (d) Same spatial view in 2-D. The domain of I covers the intersection between high A and low B. (e) Same type of PBI gene regulatory network as (a-b) with more nodes, denoted by G. (f) Detailed view of the architecture underlying the 2-D patterning of (d). Network G is repeated inside every cell of a lattice. (g) Local coupling of positional nodes creates gradients that create patterning. While G's structure and weights are cloned, node *activities* vary from cell to cell.

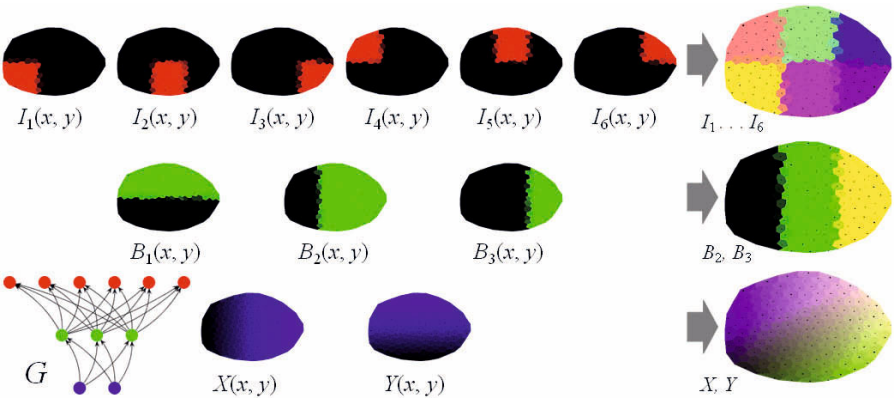


Fig. 8.4. Checkered self-patterning (top right) created by a simple 2P-3B-6I gene regulatory network G (as in figure 8.3b) in a 200-cell oval-shaped embryo. Each embryo view is selectively "dyed" for the expression map of one of the 11 genes, or a partial combination of these genes. With $X = x/x_{max}$, $Y = y/y_{max}$, weights are such that: $B_1 = \sigma(Y - 1/2)$, $B_2 = \sigma(X - 1/3)$, $B_3 = \sigma(X - 2/3)$; $I_5 = B_1 B_2 (1 - B_3)$, $I_6 = B_1 B_3$, etc.

striping process of *Drosophila* is controlled by such a regulatory hierarchy containing five main tiers of genes [8]. The present model relies on a three-tier caricature of the same idea, the *positional-boundary-identity* (PBI) network (figure 8.3b,e). In a 2-D virtual embryo, the bottom layer contains two "positional" nodes, X and Y; the middle layer, n "boundary" nodes $\{B_i\}_{i=1\ldots n}$; and the top layer, m identity nodes $\{I_k\}_{k=1\ldots m}$. Variables X, Y, B_i and I_k denote the gene expression levels of each of the $2 + n + m$ nodes. First, the positional activities follow gradients across the embryo, e.g., $X(\mathbf{r}) = \alpha(x - x_0)$ and $Y(\mathbf{r}) = \beta(y - y_0)$ as above. Then, in each cell, the boundary nodes compute linear discriminant functions of these positional nodes through the equation $B_i = \sigma(L_i(X,Y)) = \sigma(w_{ix}X + w_{iy}Y - \theta_i)$, where $\{w_{ix}, w_{iy}\}_{i=1\ldots n}$ are the regulatory weights from X and Y to B_i, parameter θ_i is B_i's threshold value, and sigmoid function σ is defined by $\sigma(u) = 1/(1 + e^{-\lambda u})$. The effect of a boundary node is, thus, to diagonally segment the embryo's plane into two half-planes of strong and weak expression levels (1 and 0). Finally, the identity gene levels are given by logical combinations of the near-binary expression levels of the boundary genes, for example, by calculating the products $I_k = \prod_i |w'_{ki}|(w'_{ki}B_i + (1 - w'_{ki})/2)$, where $w'_{ki} \in \{-1, 0, +1\}$ represent ternary weights from B_i to I_k. Thus, the contributing factor coming from B_i can take three possible values: $(1 - B_i)$, 0, or B_i. In the PBI model, the "identity domains", i.e., the regions of high I expression, are made of polygons at the intersection of multiple straight boundary lines (figures 8.3g and 8.4).

8.3.1.4 A lattice of gene regulatory networks

The full architecture of the virtual embryo is a network of networks. It consists of a lattice of cells, where each cell contains a gene regulatory network (GRN) [24, 30, 36] that can be, for example, of the PBI type just described (figure 8.3f). This lattice, however, is not necessarily rectangular or even regular. In the most general case, it is a swarm of nodes $c = 1 \dots N$ representing the cells' nuclei, with arbitrary coordinates $\{\mathbf{r}^c\}_{c=1\dots N}$, where $\mathbf{r}^c = (x^c, y^c)$ for an embryo in 2-D space or $\mathbf{r}^c = (x^c, y^c, z^c)$ for one in 3-D space. The nuclei are connected by edges that represent neighborhood relationships and dynamic coupling between GRNs (figure 8.8). The existence of an edge between two nodes c and d is established with respect to their Euclidean distance $\|\mathbf{r}^c - \mathbf{r}^d\|$, typically using a nearest-neighbor rule or the Delaunay triangulation that avoids gaps and crossings. The cell membranes can be either round or defined by their Voronoi region; in this model, no difference is made between a full cell volume and its nucleus. Denoting by G^c the GRN of cell c, a macroscopic edge $c \leftrightarrow d$ generally represents a complex coupling link between multiple gene nodes of G^c and G^d. In the experiments presented here, the embryo is a 2-D quasi-hexagonal Delaunay lattice and G^c is a PBI network. Intercellular coupling is restricted to X and Y positional nodes, where coupling strength between concentration levels X^c and X^d is symmetrical and depends on only $|x^c - x^d|$, and similarly for Y and y. As described above, this causes protein X to anisotropically diffuse on the x axis, following a gradient $X(\mathbf{r}^c) \cong \alpha(x^c - x_0)$, which is interpreted by the B and I layer inside each cell c and creates a pattern of gene expression $\{I_k(\mathbf{r}^c)_{c=1\dots N}\}_{k=1\dots m}$ on the lattice (figure 8.3g). In the binary approximation, where I_k is a product of near-binary B_i activities, the pattern consists of a patchwork of polygonal domains $\{D_{Ik}\}_{k=1\dots m}$ that can be partially overlapping. The embryo's partitioning into D_{Ik} territories is similar to the colorful compartments between lead cames in stained-glass works.

8.3.1.5 The feed-forward dynamics of gene network topology

In summary, under the simple feed-forward hypothesis, developmental genes are roughly organized in tiers, or "generations". Earlier genes map the way for later genes, and gene expression propagates in a directed fashion. First, positional morphogens create half-plane domains, and then domains intersect. Naturally, this is a crude caricature of real developmental GRNs. Although biological research has not fully unraveled the complex webs of regulatory gene-protein-gene interactions in any species, two important topological features of these webs have long been recognized, namely: (1) the existence of recurrent loops [15] and (2) gene multivalence. These features are not taken into account in the feed-forward network model, but small improvements should suffice to accommodate them. Concerning feature (1), recurrent loops and "feedback"

interactions can be added *within* each layer, while keeping the general multilayered architecture and prohibiting feedback from a higher layer to a lower layer. Feature (2) means that the same developmental genes can be reused at different periods and locations in the organism. Such genes, also called "toolkit genes", are in fact triggered by different switch combinations (multiple clusters of upstream regulatory sites on the DNA). Therefore, they can be formally represented by *duplicate* nodes placed in different tiers of the feed-forward network. In graph topological formalism, toolkit genes look like "hub" nodes that receive and send out numerous links. From the *dynamical* and activity propagation viewpoint, however, this resemblance disappears by segregating specific functional combinations of incoming and outgoing links from each other and duplicating the node. This suggests that developmental GRNs might not be so "complex" after all, in the scale-free [4] or small-world sense [38], but might essentially retain a directed acyclic graph (DAG) topology at the macroscopic level, filled with smaller cliques of recurrently connected nodes at a finer level. A longer discussion about the realism of GRN topology will not be pursued here. The main thrust of the present study is to motivate new ways of designing artificial systems by drawing inspiration from biological development, not to give a faithful account of biological reality.

8.3.2 The growing canvas: multiscale, recursive patterning

8.3.2.1 Hierarchical subpatterning

The primitive PBI network architecture used here is similar to the multilayered "perceptron" model of artificial neural networks. Its *generalization* power, i.e., ability to generate a wide variety of patterns, is problematic. A three-tier perceptron is theoretically universal, as it can produce any desired segmentation of the input space — here, the 2-D space of the input nodes X and Y, respectively, equivalent to coordinates x and y. This is in contrast to a two-tier PB perceptron without hidden layer, which can accomplish only linear partitions. In 2-D, again by analogy with stained-glass techniques, it means that any scene motif or embryo map in principle can be completely delineated by boundary lines (the "hidden units"), however fine its details may be. The homogeneous identity domains I (the "classes") then appear at the intersection of the half-planes defined by the B lines. Additionally, other types of B contours than straight lines can be employed; instead of the linear kernel L_i in $B_i = \sigma(L_i(X, Y))$, the boundary discriminant functions can be polynomial kernels $B_i = \sigma(P_i(X, Y))$ or other kernel types. However, an important question remains about the cost of this versatility. How many boundary nodes are necessary and sufficient to cover all the components of a segmentation pattern? As long as the different identity domains are not too numerous and remain reasonably connected, only a few B nodes are needed. On the other hand, as the identity domains become smaller and more fragmented, the number of B nodes increases rapidly, eventually tending to infinity in the limit of discontinuous points.

Thus, although theoretically versatile, the PBI network is in practice limited by scaling. As discussed earlier, biological embryo patterns develop in numerous incremental stages. An adult organism is produced through gradual morphological refinement, following a cascade of gene expression regulation from precursor gene tiers to secondary gene tiers, from secondary gene tiers to tertiary gene tiers, and so on. To account for this effect, the present model GRN is extended to include a pyramidal *hierarchy* of PBI networks, referred to as H-PBI (figure 8.5) and denoted by Γ. A pattern is now generated in a recursive fashion. First, the base PBI network G_0 at the root of Γ establishes the largest pre-identity domains (figure 8.5b). Then, in the next stage, another set of PBI subnetworks G_1, G_2, etc., partition these pre-identity domains into smaller identity compartments at a finer scale (figure 8.5c), and so on. The onset of a later PBI subnetwork $G_{\mu'} = \{X', Y', B'_{i'}, I'_{k'}\}$ is always controlled by one or several of the I nodes of an earlier PBI subnetwork $G_\mu = \{X, Y, B_i, I_k\}$. Formally, this can be written: $X'(\mathbf{r}) = \alpha'(x - x'_0)I_k(\mathbf{r})$, and same for Y'. It means that X' and Y' follow local gradients only inside, precisely, the domain D_{Ik} delimited by one of the identity nodes $I_k = 1$, and that they are zero everywhere else. This causal relationship is similar to the imaginal discs of *Drosophila*; once a territory D_{Ik} has been marked to be the future site of a leg or a wing (high I_k activity), a local coordinate system arises inside D_{Ik} in the form of gradients (such as X' and Y'), which then trigger the formation of a new subpattern $\{I'_{k'}\}_{k'=1\ldots m'}$ inside this territory, and so on [8]. Form details are added in a hierarchical or "fractal" fashion, analogous to the local inclusion of small stained glass motifs into bigger ones. Fractal patterning has also been explored in "map L-systems" [32], but using symbolic rules in an explicit geometrical representation.

8.3.2.2 Expansion

Simultaneously, the embryo grows as cells continue to divide and proliferate (figure 8.5a'-c'). Hence, multiscale patterning actually consists of two fundamental processes playing out in parallel: (1) the partitioning of identity domains into smaller identity domains, and (2) the continuous expansion of identity domains. During cell division or mitosis, the two daughter cells inherit the current expression state of the mother cell. Biologically, this state corresponds to mRNA, protein and metabolic concentrations; in the present model, it is represented by the set of values of the GRN nodes. Domains thus preserve their identity during expansion (the I nodes of G_0), while they are also occupied by new local gradients of positional information, i.e., new regional coordinate systems (X', Y') that activate the next PBI module in the hierarchy (G_1 or G_2) Biologically, these new local gradients might emerge from a diffusion process similar to the original diffusion of the global coordinates X, Y. For example, during proliferation a small number of daughter cells asymmetrically inherit more signaling proteins than their siblings, and then these proteins start diffusing from one border of the domain where the cells

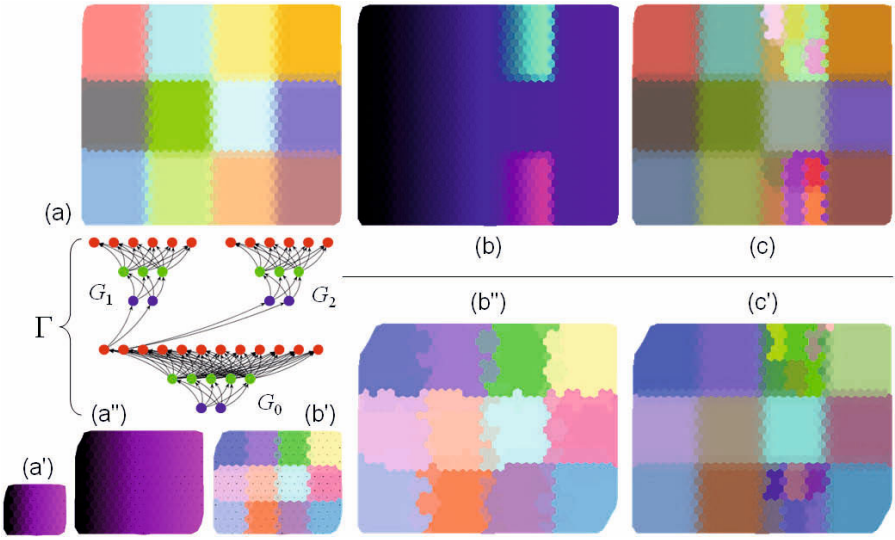


Fig. 8.5. Static and growing multiscale canvas. On a 32×32 hexagonal lattice of cells, an H-PBI gene network Γ gives rise to a "fractal" pattern in two steps: first, the base subnet G_0 (5B-12I) marks 12 rectangular segments (a) as in figure 8.4; then, two secondary subnets G_1 and G_2 (3B-6I) triggered by I_1 and I_2 create local gradients in two of those segments (b), subsegmenting them into six smaller domains (c). An equivalent pattern is obtained by a cell mass uniformly expanding from 8×8 (a') to 16×16 (a"-b') to 32×32 cells (b"-c'), while patterns continue to form and gradients continue to diffuse, as above.

gathered. However, in the present version, the gradients of P node activity are not modeled as diffusion processes but directly calculated from the geometrical shape of the identity domain boundaries according to $X'(\mathbf{r}) = \alpha'(x - x'_0)$ and $Y'(\mathbf{r}) = \beta'(y - y'_0)$. This shortcut is a slight violation of the localized dynamics, and is only aimed at replacing the lengthy convergence process of heat-like diffusion with its already-known final state (an approximately linear gradient, depending on the boundaries).

In this section, it is also assumed that all cells divide equally and simultaneously, i.e., the medium expands uniformly according to a geometrical law. For example, in a regular planar hexagonal lattice containing N cells, there is an average of $3N$ intercellular edges. At each expansion step, one new cell is added in the center of every $c \leftrightarrow d$ edge, and edge lengths are doubled to restore the original intercellular distance. The result is a new regular hexagonal lattice that has $4N$ cells and a surface area four times larger, but still has the same shape as the previous lattice (figure 8.5a", b"). Note that, since the medium is expanding, at the same time as new gradients emerge and finer details are added, the typical scale of patterning is not diminishing but in fact remains approximately constant, being on the order of magnitude of an aver-

age cell size. Following an artistic metaphor, the "growing canvas" continues to paint itself using the same "brush size" [9].

8.3.2.3 Modularity

Ordering genes in a multiscale hierarchy of the H-PBI type is a convenient way to guide self-patterning. Instead of trying to render all morphological details at once, these details arise in successive waves of expression from the broadest territories down to the finest patches. However, an even greater benefit intrinsic to a hierarchical network is *modularity*. As soon as layers and subnetworks have been defined, they can be reused as units of local computation. Modularity is a fundamental principle of genotype-phenotype economics in development and evolution [31]. Biological organisms often contain numerous repeated or "serially homologous" parts in their body plan [8]. This is most striking in the segments of arthropods (several hundreds in millipedes) or the vertebrae, teeth and digits of vertebrates. After duplication, these parts tend to diversify and evolve more specialized structures (lumbar vs. cervical vertebrae, canines vs. molars). Homology exists not only within individuals but also between different species, as classically shown by comparing the forelimbs of tetrapods from the bat to the whale. Recently, genetic sequencing has revealed that many stretches of DNA are in fact identical or highly similar. This came to support the idea that homology is the evolutionary result of *duplication* followed by *divergence* through mutation (and, sometimes, loss again).

Beyond biology, modularity is also a pervasive trait of many other natural complex systems [6]. In systems engineering, it means not only copying and reusing a partial design in different locations of an architecture, but also being able to independently modify these copies. In the present H-PBI model, this corresponds to connecting several identity genes $I_1 \dots I_k$ of a base network G_0 to a *unique* subnetwork G_1 (figure 8.6a) or, alternatively, *multiple copies* of the same subnetwork G_1, G_2, etc. (figure 8.6d). In the first case, the local pattern generated by G_1 is always identical in all primary domains $D_{I1} \dots D_{Ik}$, whether appearing in its original form (the eight +-shaped subdivisions in figure 8.6b) or in a mutated form (X-shaped subdivisions in figure 8.6c). In the second case, the local pattern can be initially identical (as in figure 8.6b-c), but then it has the possibility of evolving independently in each location and produce dissimilar variants of itself (the miscellaneous inclination angles θ of figure 8.6e). Additional mutations in the base network G_0 can change the whole body map (thinner center row d and thicker borders of figure 8.6f) without affecting the individual motifs G_1, G_2, etc. Modularity therefore plays at all levels of the GRN. The demonstration of figure 8.6 is similar to the "arthromorph" program [10] that generates chains of limbed segments representing artificial arthropods. In that simulation, "genes" control mutations at three levels: globally (same variation in all segments), in groups (same variation in a few adjacent segments only) or individually (distinct variation in every

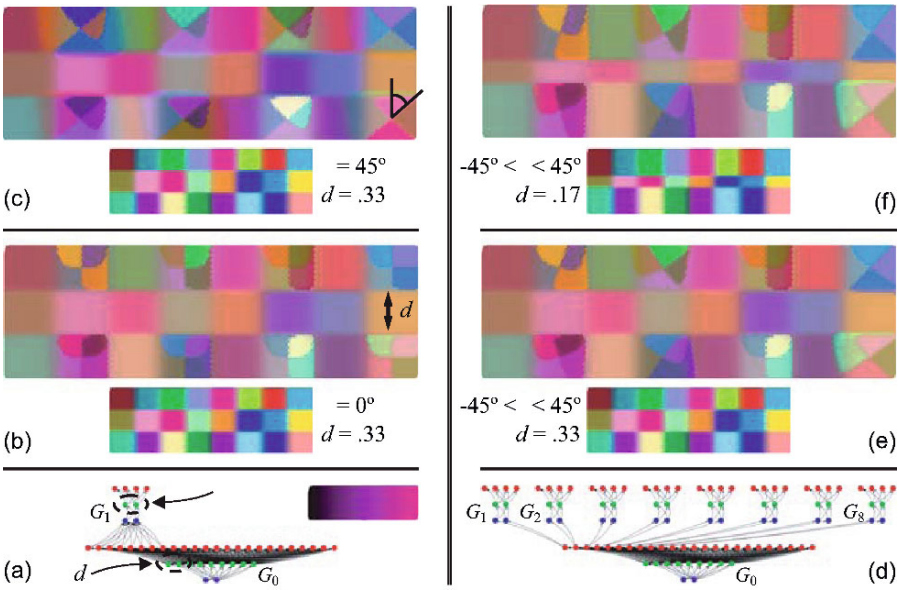


Fig. 8.6. Modularity in a segmented embryo (see text). A cell mass uniformly grows from 12×32 (inset) to 24×64 to 48×128. (a)-(c) A mutation in the unique subnetwork G_1 generates the same angle in all eight "limb" patterns. (d)-(e) Independent mutations in duplicated subnetworks $G_1 \ldots G_8$ create different angles. (f) A mutation in the base network G_0 modifies the limbs' height, without affecting their internal patterns.

segment). However, the virtual genes of arthromorphs code directly and arbitrarily for their macroscopic geometrical features. The example of figure 8.6 achieves a similar effect through the decentralized emergence of a myriad of microscopic states in a multicellular developmental model.

8.3.3 The deformable canvas: cell adhesion, division and migration

The growing canvas model based on the hierarchical gene regulation network H-PBI (section 8.3.2) is more powerful in generating a wide range of patterned images than the fixed canvas using a single three-tier PBI (section 8.3.1). With a growing canvas, it should be possible to meta-design a generative algorithm that could reconstruct any given image in a multiscale, fractal-like fashion. Such an algorithm would automatically "reverse compile" an image to produce the correct pyramidal GRN to be placed in every cell of the expanding lattice (figure 8.7). However, this is not the primary object of this work. What was achieved so far is only a model of genetically guided *patterning*, not morphogenesis in the sense of *shape* formation. The canvas' growth presented in the previous section is geometrically homothetic, i.e.,

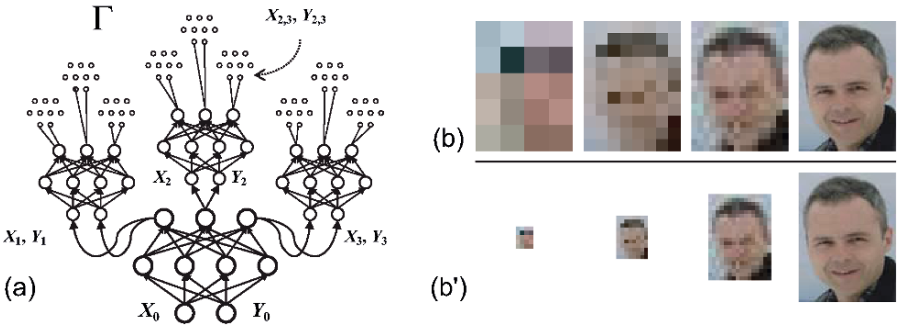


Fig. 8.7. The universal growing canvas (conceptual illustration, not actual simulation). As in figure 8.5, a generalized hierarchical GRN (a) could in principle reconstruct any image in a multiscale iterative fashion, in a fixed (b) or uniformly expanding (b') mass of cells (self-portrait metaphor after [9]).

an initial rectangular sheet of cells in 2-D remains roughly rectangular, even though its internal patterning can become very intricate.

Continuing to explore the principles of multicellular development as an inspiration for the self-organization of artificial systems, the model will be further improved to incorporate elements of cellular *biomechanics*. What is missing from the previous homothetic canvas is a topological *deformation* dynamics, or "morphodynamics", that can confer a nontrivial shape to the organic system. To this purpose, three principles are added in schematic formulations: (1) *elastic* cell rearrangement under differential *adhesion*, (2) inhomogeneous cell *division*, and (3) tropic cell *migration*. Practically for the model, all these mechanistic principles have the effect of varying the cells' coordinates in 2-D or 3-D space. Lastly and more importantly, we need to add a rule that relates those principles to the original self-patterning process. In this new "deformable canvas" model, a critical part of our meta-design effort is the establishment of a *functional dependency between cell identities and mechanical cell behaviors*. Just as the identity nodes I_k can propagate gene expression activity into subordinate PBI modules to create local segmentation patterns, the same I_k nodes now also trigger behavioral changes of the above (1), (2), (3) types in the cells where they are highly active.

8.3.3.1 Differential cell adhesion and elasticity

Lattice edges connecting cell centers are modeled as springs with force constant k and length r_0. Viscous resistance is also included with coefficient η. Thus, the equation of movement reads

$$m \cdot \ddot{\mathbf{r}}^{cd} = -k \left(1 - \frac{r_0}{\|\mathbf{r}^{cd}\|}\right) \mathbf{r}^{cd} - \eta \dot{\mathbf{r}}^{cd} \,. \tag{8.1}$$

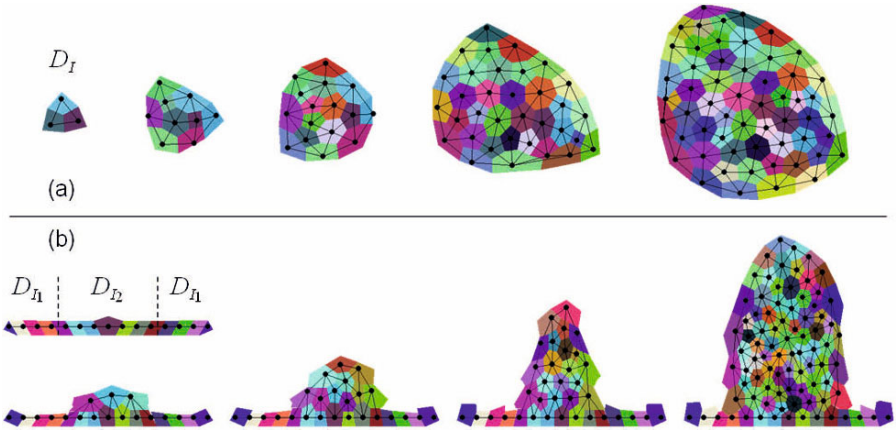


Fig. 8.8. Cell adhesion and elasticity. A simple mesh model illustrates the biomechanical behavior of a growing cell mass. No genetic network is used here; cells have arbitrary colors. Lattice edges and polygons result from a Delaunay-Voronoi tessellation (corrected on the periphery). (a) Isotropic "blob" of identical type-I cells dividing at 1% rate, in which nearby daughter cells rearrange under elastic forces (see text). (b) Anisotropic "limb" growth: from the initial 2-type cell sheet, only the center domain D_{I2} and its offspring divide (upward stretch modeled by 2x:y anisotropic rescaling). The eight lateral cells have different identity I_1 and no adhesion to the I_2 lineage.

Neglecting the effect of inertia, the coordinate update rule at each time step $\Delta t = 1$ becomes:

$$\Delta \mathbf{r}^c = -\Delta \mathbf{r}^d = \frac{\Delta \mathbf{r}^{cd}}{2} = -\frac{k}{2\eta}\left(1 - \frac{r_0}{\|\mathbf{r}^{cd}\|}\right)\mathbf{r}^{cd}\,. \tag{8.2}$$

Under this simple model of elastic rearrangement, each cell tends to optimize the distance $\|\mathbf{r}^{cd}\|$ with its neighbors' nuclei to reach r_0, i.e., occupy a convex volume of typical diameter r_0 (figure 8.8). Biological cells also stick to each other by means of adhesion proteins that cover their membrane. The great diversity of adhesion proteins gives them the ability to selectively recognize each other, thereby modulating the intercellular adhesion force or "stickiness". Some cells slide along one another without attaching, while other form tight, dense clumps. In the elastic force model, differential adhesion can be modeled by allowing the spring constant to vary from edge to edge, which means replacing k with k^{cd} in the above equation. For the limb-like growth illustrated in figure 8.8b), there is no adhesion between domains D_{I1} and D_{I2}, so $k^{cd} = 0$ between any cell c of identity I_1 and any cell d of identity I_2. In figure 8.9c'-d', adhesion is zero between either D_{I1} or D_{I2} and the rest of the embryo.

8.3.3.2 Inhomogeneous cell division

This mechanism is similar to cell proliferation at the basis of the homothetic expansion seen in section 8.3.2.2. The new aspect here is that cells divide according to a *non-uniform* probability that essentially depends on their *genetic identity*, i.e., the D_I domains to which they belong (figure 8.9). This means that the probability of division of cell c located in $\mathbf{r}^c$, denoted by $p(\mathbf{r}^c)$, depends at any time only on the current state of activity of the I nodes in the cell's H-PBI gene network: $p(\mathbf{r}^c) = p(I_1(\mathbf{r}^c), I_2(\mathbf{r}^c), \ldots)$.

Two daughter cells c and d resulting from the division of cell c under this probability are initially positioned next to each other at a small distance δ and a cleavage angle θ, also drawn at random (possibly from a non-uniform distribution in the case of anisotropic proliferation). By denoting $\mathbf{r}^{cd} = \mathbf{r}^c - \mathbf{r}^d$ the vector on directed edge $c \leftarrow d$, this means $\mathbf{r}^{cd} = (\delta \cos\theta, \delta \sin\theta)$. Then, the positions of c, d and the cells in their neighborhood are rearranged under elastic constraints implemented by the edges (explained in the previous section). Differential proliferation rates based on genetic identities produce bulges and deformations in the embryo shape, as some compartments expand faster than others (figure 8.9a-d), resembling organogenesis. Using anisotropic cleavage planes and anisotropic rescaling transformation $x : y \rightarrow ax : by$, this model can also generate directional offshoot akin to limb development (figures 8.8b and 8.9a'-d').

8.3.3.3 Tropic cell migration

A specificity of animal development, largely absent from plant development, is cell migration. Individual cells or groups of cells burrow their way through the cellular mass and extracellular matrix to colonize remote locations of the developing embryo. Depending on the adhesion properties of the migrating cells, they can either globally preserve their neighborhood relationships by "flocking" together or, on the contrary, detach from their immediate surroundings to create new intercellular bonds elsewhere. In the first case, migration happens *en masse* and takes the aspect of elastic sheet deformation. The most striking example of collective crowd movement is gastrulation, a complex folding event that forms the fundamental germ layers of the embryo in its earliest stages. The second case is best illustrated by neural crest cells that leave the dorsal neural tube to form other structures far from their source. However, it is often unclear which type of migration is predominant and most biomechanical deformation processes involve a mix of collective and individual dynamics. Often, an "active" group of cells entrains a more "passive" mass in its trajectory, like a locomotive. The existence of cells that act like singularities makes especially difficult a description purely based on continuous surfaces and differential geometry and requires discrete multi-agent simulations. The locomotion mechanisms responsible for cell migration are not fully understood. Generally, a cell is motivated to migrate by attraction toward specific

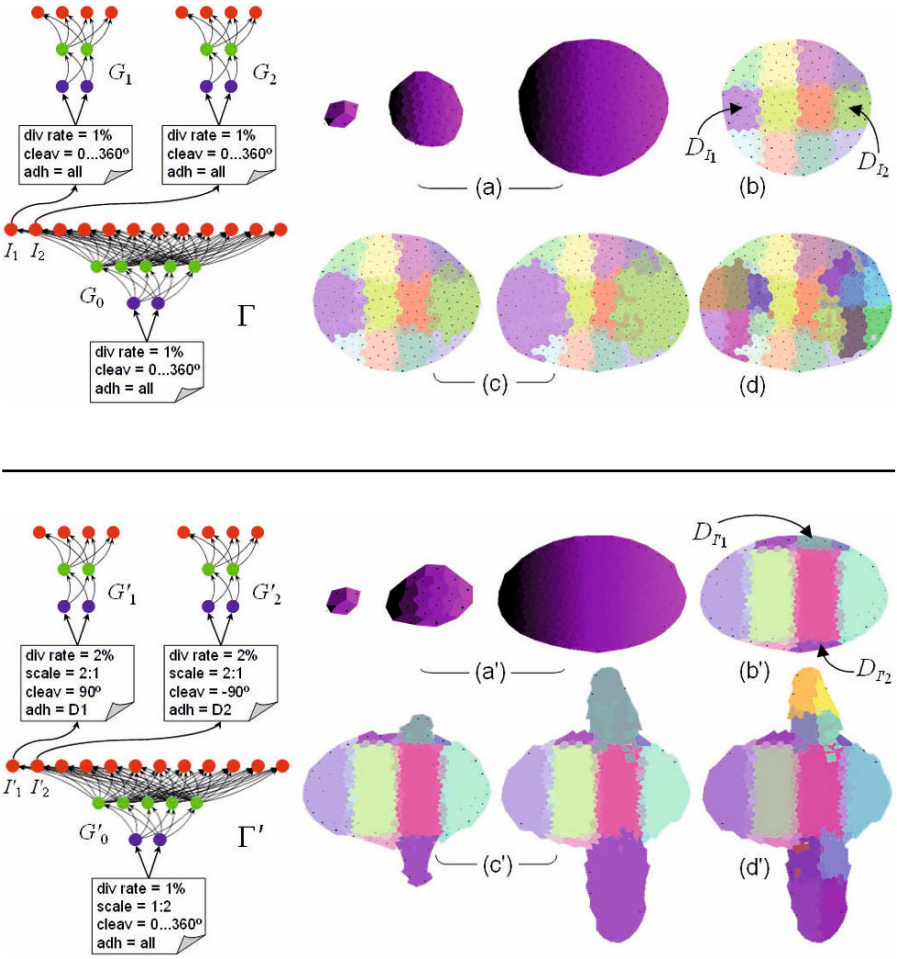


Fig. 8.9. "Organogenesis" by non-uniform cell proliferation. As in figure 8.5, a checkered embryo emerges from an H-PBI gene regulatory network Γ. Here, new cellular behavioral rules are added. Cells with high levels of identity genes I_1 and I_2 are prompted to further divide at the rate of 1% (c) (while others have stopped), before expressing subpatterns G_1 and G_2 in their newly formed anterior and posterior territories (d). Different weights in base module G'_0 of Γ' make a thicker central row and place $D_{I'1}$ and $D_{I'2}$ on the dorsal and ventral sides (b'). Moreover, different values of cleavage angles, anisotropic rescaling and adhesion coefficients ($k^{cd} = 0$ between $D_{I'1,2}$ and the rest) provoke I'_1 and I'_2 cells to grow "limbs" (c'), which are also subpatterned by G'_1 and G'_2 (d').

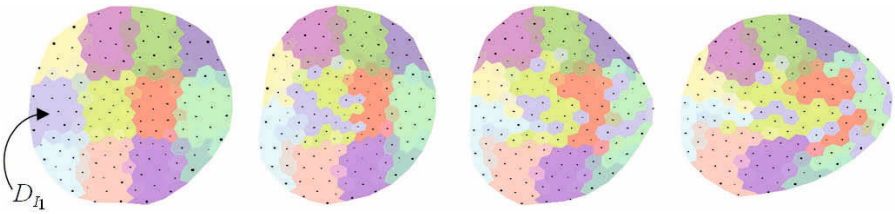


Fig. 8.10. Cell migration. Using the same gene regulatory network Γ as figure 8.9, the behavioral parameters of cells I_1 are replaced with a migration rule. Before proliferating, these cells push their way across the embryo toward increasing X concentration (here, eastward). This is modeled by adding to $\Delta\mathbf{r}$ a bias $\mathbf{u}$ that depends on genetic identity.

chemical signals that it recognizes ("chemotaxis"). These signals trigger its motion and guide it on its route toward its target. In the present model, this behavior can be simply implemented by adding an identity-dependent bias vector $\mathbf{u}^c = \mathbf{u}^c(I_1(\mathbf{r}^c), I_2(\mathbf{r}^c), \ldots)$ to the $\Delta\mathbf{r}^c$ equation of section 8.3.3.1 (figure 8.10).

8.3.4 The excitable canvas: organic computing

Meta-designing laws of artificial development with inspiration from biology, as the present model attempts to demonstrate, is a challenging engineering task. It combines schematic modeling of natural complex systems, such as embryogenesis, and departure from the natural model toward free invention. Yet, developmental modeling is only the first half of the IMD effort. Another important problem is *functional* meta-design. Once an "organic system" is mature, what could be its computing capabilities? What do the artificial cells and organs (identity domains) of an embryomorphic system *represent* in practice?

In biology, it is difficult to establish a distinction between a purely developmental and a purely physiological regime. Real cells are already "functional" as soon as they are produced by mitosis and this function partakes in development. For example, the bioelectrical signals endogenously generated by neurons play a critical role in establishing synaptic contacts in the brain. While connectivity obviously supports the exchange of activity, there also exists an important feedback from activity to connectivity dynamics. Nodes start communicating before edges are fully built, leading to self-structuring of the network [12].

In artificial embryomorphic computing systems, it is conceivable to keep these two phases relatively separate by distinguishing between "activity for development" and "activity for function". The intercellular transmission of positional information by regulatory coupling between genes (figure 8.3b) represents activity-for-development. Beyond a certain degree of maturity of the growing system, this type of activity would gradually or abruptly diminish

and give way to the other type of activity, serving a different purpose toward functional computing. It does not mean that the system would entirely stop developing; the morphodynamics would still be active, mostly to fulfill important self-repair tasks and provide robustness to perturbations. If one or several cellular components fail, they could be quickly replaced by the still-active growth potential of their neighbors. (Self-repair properties have not been verified yet in the current embryomorphic model.) Yet developmental activity would be mostly dormant during functional maturity.

Now, after finishing the self-assembling stage and while constantly under self-repairing mode, what *type* of computation could be carried out by the embryomorphic system? As a speculative proposal, by its very 2D or 3D spatial nature, the organism could become the substrate of *excitable media* dynamics. After creating slow, quasi-static developmental patterns, cells could form fast and transient *dynamical* patterns. Depending on their identity domain, local groups of cells would synchronize in different ways and enter various regimes of collective spatiotemporal order (figure 8.11). Computation in the organic canvas would consist of emerging patches of moving and shimmering spots, stripes, target or spiral waves. We find again the types of reaction-diffusion patterns of figure 8.2a, which played only a limited role in development, now appearing and disappearing on the short time scale, and on top of genetically guided development. These phenomena are common in nonlinear chemical reactions or multicellular structures [42, 34], such as slime mold aggregation, heart tissue activity, neural networks, etc. These systems all have in common the ability to position themselves in a critical state, from which they can rapidly bifurcate between chaos and order. In this perspective, the "self-made tapestry" [3] would become a self-made screen or "sensitive plate" of coupled oscillatory units in the sense that certain external patterns of initial conditions can quickly trigger internal patterns of collective response from the units.

This is already the case in neural computation. New spiking network models that take into account the fine temporal structure of neural signals have revealed a great diversity of collective spatiotemporal regimes: synchronization and phase locking, delayed correlations and traveling waves, rhythms and chaos. Through recurrent (and plastic) synaptic connections, neural cells transiently interact as dynamical subnetworks that promise an immense richness of coding expression and computational power, combining the discrete and the continuous. What could still be missing from the current embryomorphic model are long-range connections. The inspiration from embryogenesis is currently limited to geometrical 2-D or 3-D lattices but might be complemented with complex "N-D" networks arising in ways similar to neurogenesis and synaptogenesis.

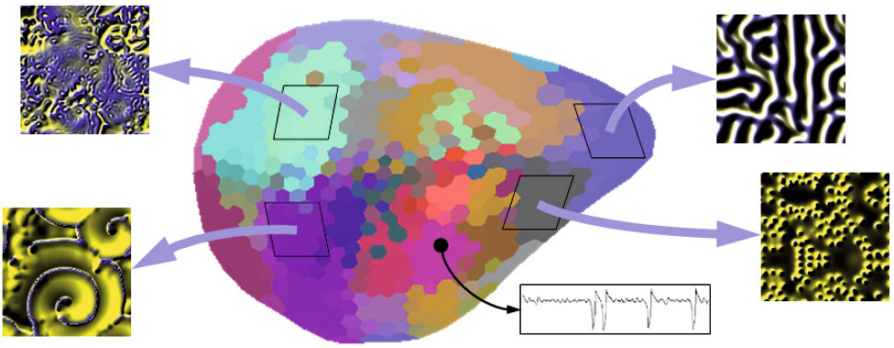


Fig. 8.11. The excitable canvas. A hypothetical view of an embryomorphic system in which genetic identity domains (colored patches) could support excitable media (zoom-in square insets), through coupling between quasi-oscillatory units (example of one temporal signal in the rectangle inset). Various regimes of dynamical activity could emerge on these substrates. Such spatiotemporal patterns might hold a great potential for representational and computing properties. (The square insets are snapshots of simulations run on Tim Tyler's demo applet http://texturegarden.com/java/rd, which implements the Gray-Scott model described in [29] under various sets of parameters.)

8.4 Discussion: planning the autonomy

At the core of the complex systems engineering enterprise lie paradoxical objectives: How can decentralization be controlled? How can autonomy be planned? How can we expect specific characteristics from systems that are otherwise free to invent themselves? The challenge is not only to *allow* self-organization and emergence but, more importantly, to *guide* them. First, it consists of preparing the conditions and mechanisms favorable to a robust and *reproducible* — as opposed to random — pattern formation process, under genetic control (IMD). Second, it consists of steering this process toward desired *goals*, while simultaneously leaving the door open to the spontaneous generation of innovative designs by evolution of the genes (EMD).

8.4.1 Growth, function, evolution

When meta-designing an embryomorphic artificial system, the engineer faces three main tasks: IMD1: How does the system *grow*? IMD2: How does the system *function*? EMD: How does the system *evolve*? The goal of the IMD phases is to define the developmental and computing mechanisms. The goal of the EMD phase is to define the rules of their evolution by variation and selection (section 8.4.2).

8.4.1.1 Developmental mechanisms

Growth results from a combination of elementary mechanisms, described and elaborated in sections 8.2 and 8.3. At the microscopic level of the embryomorphic model, it is grounded in a repertoire of basic cellular behaviors: cells change state (genetic expression), communicate (positional signals), cluster or detach (differential adhesion), travel (migration), create offspring (mitosis) or die (apoptosis). At the mesoscopic level of cell populations, several morphogenetic processes emerge: *guided patterning* through GRN-controlled expression maps, *organogenesis* through differential adhesion and domain-specific division rates, folding and *deformation* through elastic constraints and *sculpting* through cell removal. Additionally, superficial *free patterning* (spots, stripes) can also arise by reaction-diffusion (figure 8.2a). Starting from a single cell, a complex and organized architecture develops through the repeated application of a series of these principles, identically programmed inside each cell. Task IMD1 consists of choosing among these principles and designing their dynamics and parameters.

8.4.1.2 Functional mechanisms

Function roughly starts after growth (see section 8.3.4). Task IMD2 is about defining the *nature* of the cells and the type of computation that they carry out at the microscopic level. It also involves defining the range of macroscopic abilities of the system and its input/output interfaces with the environment. Do cells represent some kind of hardware components on a board, taking part in global digital-analog electric or optical activity patterns? Are they small modules of software logic that execute symbolic instructions on more conventional architectures in a "virtual machine" fashion? Can they actually be physical parts and blocks, joined together to support sensing, planning and acting in a robot? Or are they even full-fledged robots that coordinate in swarm formations for collective performance?

8.4.1.3 Evolutionary mechanisms

Evolution of both growth and function is the responsibility of the evolutionary meta-designer (EMD), who must define how the system *varies* (randomly) and how it is *selected* (nonrandomly). These points are discussed in the next section.

8.4.2 Selection without expectations

Three degrees of constraints that drive the fitness criteria and the artificial selection process can be identified, in decreasing intensity: (1) selecting for a specific system *architecture*, (2) selecting for a specific system *function*, and (3) selecting the *unexpected*.

8.4.2.1 Selecting for architecture

On the first level, the EMD engineer imposes tight requirements to obtain particular organismal patterns and shapes from the development process. Here, a reverse engineering problem must be addressed: Given a desired phenotype, what should be the genotype that will reliably reproduce this phenotype? One solution, if available, is the deterministic reverse compilation of the genotype from the phenotype. This is what Nagpal [25] has achieved in her virtual origami model. Given a macroscopic folding recipe (based on an ordered set of lines), she can automatically generate the exact microscopic developmental rules that each identical point of the medium must follow to reproduce the shape (based on wave propagation and state change). It is possible that such a method is also within reach in the present embryomorphic model, but this has not been investigated to-date.

In most cases, however, it is safe to assume that no algorithm for the reverse compilation of the genotype from the phenotype is available. Depending on the number of dimensions along which the system may vary, the search in parameter space can then appear nearly impossible. In the embryomorphic model presented in section 8.3, these parameters are the set of gene-to-gene regulatory weights together with the local behavioral rates of division, migration, gradient diffusion, and so on. (typically, the Γ network of figure 8.9, branching out like figure 8.7a). A naive fitness function rewarding only the final shape would create one (or possibly several) "narrow" peaks limited to local neighborhoods of those parameters, which would be unreachable for all practical purposes. However, this is the fallacy of "Mount Improbable" explained by Dawkins [10]. Biological evolution does not create complex organisms in one shot, but through a multitude of successive steps, randomly generated and nonrandomly selected. Each favored step brings a small "improvement", adding to the body plan a little more complexity, which gets rewarded by successful interaction with the physical environment. A famous example is how the eye has evolved multiple times from mere photosensitive cells, through gradual inward curvature of the epithelial tissue, condensation of the lens, etc. These changes were probably encouraged by an ever-increasing quality of the projected image (hence, an increasing survival probability under co-evolution pressure), as modeled by Nilsson and Pelger [27]. Behind the daunting cliff of a high fitness peak, hides a long and gentle upward slope.

Similarly, in the artificial systems challenge, an evolutionary meta-designer should also replace a jagged final-shape fitness landscape with a smooth transitional-shape fitness landscape. The best method to accomplish this is to define a score value or "distance" to the desired shape. This value must be an increasing function of favorable mutations, i.e., mutations that bring the developed system closer to the ideal template. It is conjectured here that this well-known principle of gradual search might actually *benefit*, not suffer, from the high parameter dimensionality offered by a true underlying *embryomorphic* model, such as the one presented in this chapter. A hierarchical gene

regulatory network of the H-PBI type hyperdistributed in a large cell mass might be in a better position to provide the necessary fine-grain mutations required by the gentle slope approach than the more direct genotype-phenotype mapping of traditional genetic algorithms.

8.4.2.2 Selecting for functionality

Why make the effort to devise a sophisticated self-organizing system only to force this system to produce a specific shape? Why not directly build the final shape in the first place? One important benefit would be robustness through homeostasis. Even though the final plan is known in advance, a genuine developmental dynamics is also expected to be intrinsically "self-repairing" (as mentioned in section 8.3.4). Nonetheless, requiring a specific architecture somehow defeats the idea of "stepping back" and meta-designing. Intervening in the microscopic placement of cells, whether by reverse compilation or fine-tuning, literally reintroduces the micromanagement attitude of classical systems design.

On the contrary, EMD engineers should abstract themselves even further from structural details and concentrate on selecting for the *functionality* of their system, otherwise leaving it complete freedom of morphology. Here, the same gradual optimization strategy as described above can be employed, except that the continuous distance quantity would not measure morphological resemblance but rather the closeness of *performance* to predefined goals. Given a task or repertoire of tasks to accomplish, it means ranking candidate systems according to their *partial* success in fulfilling these tasks, then allowing the most successful ones to reproduce faster and mutate, and so on. Afterwards, it is always instructive for the meta-designer to open the "black box" of the winning architectures and try to understand how they have come about and which specific subsystems or modules related them to success. The solutions "invented" by spontaneous evolution are often surprising and convoluted, in other words, remote from what a human designer would have conventionally designed.

Functional selection under free organization is the strategy adopted by most evolutionary computation works that also contain elements of distributed architectures or (small-size) complex systems. For example, this is the case of the logical functions computed by randomly composed multi-instruction programs in Avida [19], the locomotion abilities created by randomly articulated multi-segment robots in Golem [20] or Framsticks [17], or the shooting skills of intelligent video game agents emerging from randomly assembled multi-neuron networks in Nero [33]. Again, it is argued here, although not proven, that an even larger number of agents, such as in multicellular embryogenesis, would be even more favorable to a successful evolutionary search.

8.4.2.3 Selecting the unexpected

The increase in system size, however, might also require the EMD engineer to "let go" one step further and give up on specific selection requirements altogether. In summary, it is likely that the ultimate reconciliation between the antagonistic poles of planning and autonomy would be based on two complementary aspects. In order for an evolutionary process to successfully find "good" regions in systems state space, (a) the variation-by-mutation mechanisms must be fine-grained and rich enough to offer a large number of search paths (system size) and, at the same time, (b) the selection criteria must be loose enough to allow a large number of fitness maxima ("letting go"). With more search paths covering more fit regions, evolution is more likely to find good matches. Point (a) concerns the intrinsic ability of complex systems to create a *solution-rich* space [23] by combinatorial tinkering on highly redundant parts. Variational power is the most critical aspect of evolutionary processes; developmental systems made of a great number of small self-assembling components have the unique ability among all systems to generate behavior-rich variations. Point (b) concerns the ability of the meta-designers to relax their specifications, within reasonable limits of what is permitted by the system's environment and the problem at hand, and accept to be surprised — hopefully, in a pleasant way — by the outcome. Organic systems engineers will probably need to learn how to greatly diversify their demands and rather stand on the side, ready to harvest possibly "interesting" or "useful" organisms from a free-range menagerie.

Acknowledgments

This work was made possible in part by the EU projects *Embryomics* (FP6-NEST-2003-1-12916) and *BioEmergences* (FP6-NEST-2004-Path-IMP-28892). I thank Paul Bourgine for his warm support and encouragements. I am also grateful to Francisco Vico, Daniel Lobo, Jan Gecsei, Gregory Horman, and Rolf Würtz for their useful comments on the manuscript and editing help.

References

1. H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight Jr., R. Nagpal, E. Rauch, G. Sussman, and R. Weiss. *Amorphous Computing.* MIT Artificial Intelligence Laboratory memo no. 1665, Aug. 1999.
2. W. S. Bainbridge, and M. C. Roco, eds. *Managing Nano-Bio-Info-Cogno Innovations: Converging Technologies in Society.* Berlin: Springer Science and Business Media, 2006.
3. P. Ball. *The Self-Made Tapestry: Pattern Formation in Nature.* Oxford University Press, 1999.

4. A.-L. Barabási, and R. Albert. Emergence of scaling in random networks. *Science* **286**: 509-512, 1999.
5. *Beyond the Horizon: Anticipating Future and Emerging Information Society Technologies* Final Report, ERCIM, FET, IST, EU, 2006.
6. W. Callebaut, and D. Rasskin-Gutman, eds. *Modularity*. The MIT Press, 2005.
7. S. B. Carroll. *Endless Forms Most Beautiful: The New Science of Evo Devo and the Making of the Animal Kingdom*. W. W. Norton & Company, 2005.
8. S. B. Carroll, J. K. Grenier, and S. D. Weatherbee. *From DNA to Diversity*, Blackwell Scientific (Malden, MA), 2001.
9. E. Coen. *The Art of Genes*. Oxford University Press, 2000.
10. R. Dawkins. *Climbing Mount Improbable*. W. W. Norton & Company, 1996.
11. R. Doursat. The growing canvas of biological development: Multiscale pattern generation on an expanding lattice of gene regulatory networks. *InterJournal: Complex Systems* **1809**, 2006.
12. R. Doursat, and E. Bienenstock. Neocortical self-structuration as a basis for learning. *5th International Conference on Development and Learning (ICDL)*, Indiana University, Bloomington, Indiana, May 31-June 3, 2006.
13. G. M. Edelman. *Topobiology: An Introduction to Molecular Emrbyology*. Basic Books, 1988.
14. A. Gierer, and H. Meinhardt. A theory of biological pattern formation, *Kybernetik* **12**: 30-39, 1972.
15. S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology* **22**: 437–467, 1969.
16. M. W. Kirschner, and J. C. Gerhart. *The Plausibility of Life: Resolving Darwin's Dilemma*. New Haven and London: Yale University Press, 2005.
17. M. Komosinski, and S. Ulatowski. Framsticks: Towards a simulation of a nature-like world, creatures and evolution. In D. Floreano, J.-D. Nicoud, and F. Mondada, eds., *5th European Conference on Advances in Artificial Life (ECAL-99)*, pp261-265, Lausanne, Sept. 13-17, 1999.
18. S. Kondo, and R. Asai. A reaction-diffusion wave on the skin of the marine angelfish *Pomacanthus*. *Nature* **376**: 765-768, 1995.
19. R. E. Lenski, C. Ofria, R. T. Pennock, and C. Adami. The evolutionary origin of complex features. *Nature* **423**: 139-144, 2003.
20. H. Lipson, and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature* **406**: 974-978, 2000.
21. H. Meinhardt. *The Algorithmic Beauty of Sea Shells*. Springer-Verlag, 1998.
22. J. F. Miller, and W. Banzhaf. Evolving the program for a cell: from French flags to Boolean circuits. In S. Kumar and P. J. Bentley, eds., *On Growth, Form and Computers*. Academic Press, 2003.
23. A. A. Minai, D. Braha, and Y. Bar-Yam. Complex engineered systems. In D. Braha, Y. Bar-Yam and A. A. Minai, eds., *Complex Engineered Systems: Science Meets Technology*. Springer Verlag, 2006.
24. E. Mjolsness, D. H. Sharp, and J. Reinitz. A connectionist model of development. *Journal of Theoretical Biology*, 152: 429–453, 1991.
25. R. Nagpal. Programmable self-assembly using biologically-inspired multi-agent control. *1st Int Conf on Autonomous Agents*, Bologna, Italy, July 15-19, 2002.
26. H. F. Nijhout. A comprehensive model for colour pattern formation in butterflies. *Proc. R. Soc. Lond. B* **239**: 81-113, 1990.
27. D. E. Nilsson, and S. Pelger. A pessimistic estimate of the time required for an eye to evolve. *Proceedings of the Royal Society of London B* **256**: 53-58, 1994.

28. L. Nunes de Castro. *Fundamentals of Natural Computing: Basic Concepts, Algorithms, and Applications*. Chapman & Hall/Crc Computer and Information Sciences, 2006.
29. J. E. Pearson. Complex patterns in a simple system. *Science* **261**: 189-192, 1993.
30. I. Salazar-Ciudad, J. Garcia-Fernández, and R. Solé. Gene networks capable of pattern formation. *Journal of Theoretical Biology* **205**: 587–603, 2000.
31. G. Schlosser, and G. P. Wagner, eds. *Modularity in Development and Evolution*. The University of Chicago Press, 2004.
32. P. Siero, G. Rozenberg, and A. Lindenmayer. Cell division patterns: syntactical description and implementation. *Computer Graphics and Image Processing* **18**: 329-346, 1982.
33. K. Stanley, B. Bryant, and R. Miikkulainen. Real-time evolution in the NERO video game. *IEEE Symposium on Computational Intelligence and Games*, pp182-189, Essex University, Colchester, UK, April 4-6, 2005.
34. H. L. Swinney, and V. I. Krinsky, eds. *Waves and patterns in chemical and biological media*. The MIT Press, 1991.
35. A. M. Turing. The chemical basis of morphogenesis. *Phil. Trans. R. Soc. London B* **237**: 37-72, 1952.
36. G. von Dassow, E. Meir, E. M. Munro, and G. M. Odell. The segment polarity network is a robust developmental module. *Nature* **406**: 188–192, 2000.
37. C. von der Malsburg, R. P. Würtz, and A. Schäfer. *The Organic Computing Group*, http://www.organic-computing.org, 2006.
38. D. J. Watts, and S. H. Strogatz. Collective dynamics of "small-world" networks. *Nature* **393**: 440-442, 1998.
39. G. Webster, and B. Goodwin. *Form and Transformation: Generative and Relational Principles in Biology*. Cambridge University Press, 1996.
40. M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM* **36**: 75-84, 1993.
41. J. Werfel, and R. Nagpal. Extended stigmergy in collective construction. *IEEE Intelligent Systems* **21**(2): 20-28, 2006.
42. A. Winfree. *The geometry of biological time*. Springer-Verlag, 1980, 2001.
43. L. Wolpert. Positional information and the spatial pattern of cellular differentiation development. *Journal of Theoretical Biology* **25**: 1–47, 1969.
44. D. Young. A local activator-inhibitor model of vertebrate skin patterns. *Mathematical Biosciences* **72**: 51-58, 1984.

9

Artificial Development

Simon Harding, Wolfgang Banzhaf

Computer Science, Memorial University of Newfoundland, St. John's, NL, A1B 3X5, Canada.
simonh@cs.mun.ca, banzhaf@cs.mun.ca

Summary. There is growing interest in the use of analogies of biological development for problem solving in computer science. Nature is extremely intricate when compared to human designs, and incorporates features such as the ability to scale, adapt and self-repair that could be usefully incorporated into human-designed artifacts. In this chapter, we discuss how the metaphor of biological development can be used in artificial systems and highlight some of the challenges of this emerging field.

Key words: Developmental systems, genetic algorithms, pattern formation

9.1 Introduction

Organic processes can serve as inspiration for computational systems. The design and functionality of organic systems is both a challenge and an existence proof of achievements of Nature posed to the human designer. In this context it has turned out that the level of complexity commanded in the design of natural systems is still unparalleled. To put the scalability of biological systems into perspective: Microsoft's latest version of Windows has 10^6 lines of code, some Linux distributions have 10^7 lines of computer code, but there are 10^{14} cells in the human body each much more complex than a line of code. Such complexity would be out of reach of human designers, who even struggle to keep the simplest designs and manufacturing processes defect-free. With the increasing complexity of both hardware and software, there is growing need for new design techniques that allow us to work with such complexity.

Evolutionary algorithms that mimic Darwinian evolution have provided part of the answer to this problem. Algorithms such as genetic algorithms and genetic programming [2, 7, 22] allow us to quickly search through vast search spaces, finding novel solutions to our problems. However, there are limits to how evolvable certain applications are, especially those that – with a naïve implementation – would require long genotypes. Under such circumstances

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_9,

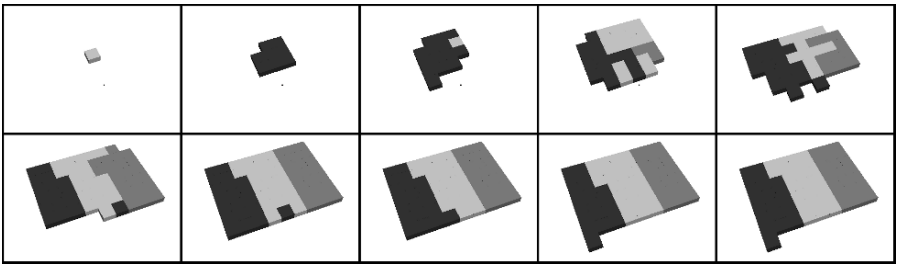


Fig. 9.1. Miller's French Flag Problem

evolution can find it difficult to fine-tune the genotype — mutations are more likely to be disruptive than beneficial and the combinatorics of search spaces is prohibitive. It would be impossible to evolve a functioning computer program with millions of lines of code, or a circuit with a hundred million components with our present-day methods. So how does nature correctly assemble a system containing trillions of cells?

Nature uses the process of development, by which a fertilized egg grows into a fully mature individual. Here we shall discuss how one can grow complicated programs and other structures by starting from an artificial zygote. For example, figure 9.1 shows a single artificial cell developing into a predefined pattern [21]. Each cell contains a computer program, discovered by evolution, that when executed contains instructions for growing the virtual organism. In this chapter, we shall provide a brief discussion of the biology of development, and shall link it to artificial development. We will also present some ideas for improving the current state of developmental models, in particular the use of physical analogies.

9.2 Embryogenesis

Embryogenesis is the first stage of the developmental process by which the embryo is formed. The development starts with the zygote, which is the fertilized ovum. The zygote undergoes a process called cleavage, where mitosis splits the cell into two (and later more) identical cells. Even at this stage, spatial patterns are being produced. Egg cells are asymmetric, and the first cleavage occurs along one axis, the second cleavage on an axis perpendicular to it. The developing embryo receives a substantial amount of information in the starting configuration from its zygote — this is called molecular prepatterning [23]. In insect ovaries, in addition to the egg, there are other cells, such as nurse cells, which produce proteins required for growth, and pass these into the egg cell. They contain maternal DNA that has an important role in the embryo's growth.

Mitosis repeats until 128 cells have been produced, at which point the next phase of development, called blastulation, occurs. At this stage, the cells are

still stem cells — they have yet to differentiate into a specialized cell type. In mammals, the cells organize into a spherical structure called a blastocoele, which subsequently develops into a blastocyst. The outer cells (called the trophoblast) will go on to form the placenta, and the inner cells (called the embryoblast) will form the embryo. Subsequently, the blastocoele forms three layers, the ectoderm, endoderm, and mesoderm, that develop into the internal organs of the animal.

The cells differentiate into their organ types, and groups form certain physical shapes. Shapes themselves are the result of a small number of different processes, governed by differences in cell adhesion and rates of differentiation. The cells can form tubes or sheets, or condense into clusters. We further discuss the role of physics in section 9.5.

Positional information, used by the genes of the embryo comes from chemical gradients that are set up by proteins from the maternal RNA. These divide the space inside the egg, and form the axes along which different developmental processes occur. In Drosophila, for example, nurse cells anchored at one end of the egg contain a gene that produces a protein (Bicoid) that diffuses across the egg, setting up a gradient. As a morphogen, the concentration of this protein determines whether certain other genes are expressed and hence control the fate of cells: A high concentration results in the formation of head and thorax. The embryo develops, however, without these features if the gradient is artificially suppressed.

The genes in the embryo are responsible for the majority of the developmental processes. They produce proteins which not only build the cellular structures, but also form part of the control process. The program stored in genes is the aspect typically replicated by evolved programs in artificial developmental processes. The genes form networks of interactions, where the behavior of one gene represses or triggers the behavior of others. However it is important to note that in addition there are maternal genes that sit at the top of this complex hierarchy of gene interactions. Maternal genes influence the zygotic genes, which in this instance are labeled gap genes, since in Drosophila a mutation in one of these genes results in a gap in the developing body plan. These genes in turn influence other layers in the gene hierarchy — at the bottom of this diagram are genes that influence the development of the wings. Effectively, each layer in the hierarchy occurs at different stages in the developmental cycle, with the lowest level occurring last, and interacting the least with the maternal RNA.

Development is sensitive to its environmental conditions. Amounts of food, water, sun and other resources play an important role in defining the growth of an animal as well as do external signals. If any of these resources is deficient, the body develops in a different way. For example, fish release hormones into the water that prevent further growth if their concentration gets too high. This prevents overuse of food resources in the wild, but also explains why the size of fish held in captivity is dependent on their tank size. Plants grow to

fit their containers — or their roots flow around rocks and other obstacles underground.

In recent years, some researchers have started to look at embryogenesis as an inspiration for the development of electronic circuits. This field of inquiry has been named *Embryonics* [24].

9.3 Scalability, Plasticity and Robustness

Multicellular organisms start from a single cell, which develops into a complete organism – potentially containing trillions of cells. Each cell contains the genetic information, encoded in DNA, that controls its properties. However, DNA does not directly specify these properties. Instead, it instructs the construction and development of a cell, using as much environmental information and material as possible. The human genome consists of about $3.2 \cdot 10^9$ base pairs, coding for approximately 25,000 genes which in turn produce 10^{14} cells. A large portion of the genome contains regulatory information.

9.3.1 Scalability

A good example of scalability in artificial development comes from neural networks. Scalability in this context means that it is possible to grow more neurons if they are needed in the neural net to solve harder problems. Specifying the properties of each cell would be computationally infeasible due to combinatorics and training algorithms like back-propagation would be unsuitable. As a result, direct evolution of weights and topology of the net sooner or later will encounter a size barrier, where training is no longer computationally practicable.

Can we write a computer program that will grow an arbitrarily sized neural net with the right properties? Is this easier than evolving the network directly? Here we are combining evolution, development and learning. Could we make the neural network perform difficult tasks? For example, say we want to write a voice recognition program. The scalable approach would require that we make it work satisfactorily on a cell phone processor, good on a desktop and amazingly well on a supercomputer. In this example, growing would allow the program to fit the processor and memory availability. Would we find that evolution/development will discover unusual topologies that one would not have considered otherwise?

Kitano was one of the first to use artificial development to produce neural networks [17]. Kitano evolved L-system rules that produced the connection matrix of neurons, specifying the network's topology and their weights. He found that, unlike conventional direct encoding, the developmental encoding scaled well. Kitano developed a method for evolving the architecture of an artificial neural network using a matrix re-writing system that manipulated adjacency matrices [17]. According to his results, this method produced results

superior to direct methods (i.e. a fixed architecture, directly encoded and evolved). It was later claimed, however, that the two approaches were of equal quality [28].

Gruau devised a graph re-writing method called cellular encoding [12]. Cellular encoding is a language for local graph transformations that controls the division of cells growing into artificial neural networks. The cells, which we can identify as nodes in the ANN, store connection strengths (weights) and a threshold value. The cells also store a grammar tree that defines the graph re-writing rules and a register that defines the start position in the grammar tree. The grammar tree was evolved using an evolutionary algorithm and the method was shown to be effective at optimizing both the architecture and weights at the same time. It scaled better than direct encoding, where all the weights had to be evolved independently [13].

Federici has successfully evolved spiking neural networks that are constructed with a developmental system [9]. Here the developmental system outperformed direct encoding by a considerable margin. However, as the parameters of neither experiment were optimized, it may not be a fair test of the algorithms' ability.

Human designs are often limited by their ability to scale and adapt to changing needs. Our rigid design processes often constrain the design to solving the immediate problem, with only limited scope for change. Organisms, on the other hand, appear to be able to maintain functionality through all stages of development, despite a vast change in the number of cells from the embryo to a mature individual. It would be advantageous to empower human designs with this on-line adaptability through scaling, whereby a system can change complexity depending on conditions. We should expect organic computing to solve a related problem: Adaptation of the complexity of algorithmic approaches to problems of variable difficulty.

9.3.2 Plasticity

In nature we find that designs scale, as is evident from the growth of animals after gestation: The various functions of the organism still work all the way from infancy to adulthood. This gives organisms the plasticity to grow to sizes that fit the environment, without sacrificing reproductive capability. The hormones in the water already mentioned limit the growth of fish, and this prevents fish from becoming too large if there is not enough room, and hence allow the fish to survive with less competition for resources. Similar effects can be found in artificial developmental systems. For example, in L-systems the models of plants can be run to any size, and still retain the same morphology or shape. L-systems were originally used to model the development of plants, however they have also been used for producing neural networks [5], protein structure prediction [8] and object design [16].

These results, both in natural and artificial contexts, give us some confidence that if we wish to produce designs with a large, and potentially chang-

ing, number of components, we should be able to utilize the developmental approach. For example, if we wish to build a control system for a large plant, such as a power station, the design should yield a stable, fail-safe control system. As with all mechanical and electrical systems, faults develop over time. It would be beneficial for the system to cope with component failure, and the system should be able to maintain its function in the event that a sensor fails. As the needs of the plant change, the system may be required to grow to accommodate new features, perhaps a boiler would be added with new sensors, actuators and terminals; could we use the principles of developmental systems to allow the control system to automatically "grow" and adapt to accommodate this? We may also find we have to deal with things that are very biological – lag in signals, different rates of signal firing, different sensor types. We therefore have to accommodate two types of change, one in the physical structure and another in the amount and quality of information in the system. By taking inspiration from the plasticity of biological systems, we may be able to produce systems that can handle such changes effectively – a task that is difficult for traditional approaches.

9.3.3 Robustness

Biological systems are remarkably tolerant to failure in individual components, and this is clearly a desirable attribute for engineered systems. The ability to regenerate lost or damaged limbs, tissues or organs is common in animals — although the abilities vary. Some animals, such as newts have the ability to re-grow entire limbs. Humans cannot regenerate limbs, however they can re-grow ribs and fingertips. The liver is also able to regenerate, and the skin is constantly being replaced.

The processes involved in development and regeneration are related. For example, during the early development of a fetus it is possible for it to fully recover from a deep cut. However, later during development the regrowth is not as effective and the fetus becomes scarred. The self-repair of the newt involves a layer of cells growing over the injured stump, which revert to stem cells. These stem cells, like those in the developing embryo, can become any cell type and allow the missing limb to re-develop.

Gerhart and Kirschner [11] describe four properties of cells that lead to developmental flexibility: (i) The destruction of a small number of cells can be tolerated, as there is enough redundancy that others in the group can replace it. (ii) As all the cells in a group perform the same action, their arrangement does not matter. (iii) Moving cells from one group to another equivalent group is possible, as the cells can adapt to the local stimuli. (iv) Finally, if an organizer, such as the bicoid gene described in section 9.2, is moved then the cells respond to their new distance from the organizer.

Which of these features are relevant and appropriate to implement in artificial development? In conventional engineering, redundancy is the normal

approach to implement fault tolerance and robustness. Although natural systems do have redundancy, for example the duplication of entire organs such as eyes and lungs, they also have the ability re-grow anything from missing cells to entire limbs. Current hardware technology does not allow a similar feat.

However, perhaps we can look forward to the time when nanobots will give hardware this ability. The application of developmental systems in pattern formation for such systems is obvious — nanobots will have limited computational and sensing abilities — much like real cells. Hence, this type of approach could be applicable in these scenarios.

Bentley shows that inherently fault tolerant computer programs can be evolved, i.e. one can damage bits of the code, and its behavior will degrade gracefully [3]. As the author notes, it is difficult to test if development gives an advantage in this case. The developmental program is longer, and hence may be more susceptible to faults (for example from faulty memory) — however, when the developmental program is corrupted, behavior degrades gracefully. The developmental program also required more computation to execute, in terms of the simulation of growing the artificial organism. Is this trade-off ultimately worth it? Bentley believes that the fragility of the solutions may be caused by the "conventional (brittle) nature of the programming language, compiler and hardware", and hence we may have to think differently about the methods of implementing these computational systems. For example, if we were to apply the cellular computing metaphor to hardware then so far all attempts have required vast amounts of hardware relative to the size of the problem being solved – and far in excess of what would be required from traditional n-module redundancy. Perhaps one could get evolution to find solutions that are more than the sum of their parts, which would gives us back the advantage? Such a system may allow us to evolve circuits with high component counts, that also have intrinsic fault tolerance – a task not yet achieved with a purely evolutionary approach.

9.4 Evolvability and search spaces

Artificial developmental systems are an example of indirect genotype-to-phenotype mapping. In development, genotypes are typically shorter than the phenotypes they represent, which means that development can be viewed as a decompression algorithm. This changes the way in which evolution tackles the search space.

For some encodings, the genotype may only be able to map to a limited part of the search space, as it may be the case that the number of states potentially represented by a short genotype is lower than the number of states in the phenotype space. For such systems, parts of the phenotype space are ignored, which can potentially benefit evolution, although care needs to be taken to ensure that potential solutions can be accessed.

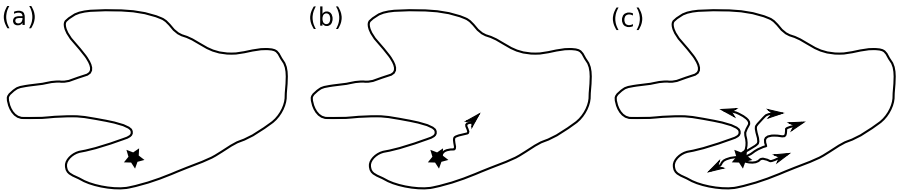


Fig. 9.2. Direct encoding, (a), only tests the search space at a single point specified by the genotype (shown as a star within the search space). A developmental encoding (b) can travel through the search space when developing. A stochastic developmental encoding (c) could take this further, and search a large area of the space, but still be the result of a single genotype.

A stochastic genotype-phenotype mapping could be viewed as a lossy decompression algorithm. The benefit here is that it enables sampling from a greater region of the phenotype space than a deterministic mapping. Natural systems are often subject to high levels of noise coming from a variety of sources, from thermal noise at a small scale to unpredictable external factors. Natural evolution has found mechanisms to cope with such noise, however in artificial systems we tend to avoid this form of stochastic behavior. Artificial evolution [30, 20, 14] shows that algorithms are not only capable of operating in such situations, but actually benefit from the presence of noise. Artificial chemistries [1] and esoteric programming languages such as "Whenever", execute program instructions in random orders - yet are still able to implement a desired computation.

One potentially undesirable feature of stochastic mapping is that for fitness evaluation each individual will need to be tested a number of times to ensure that the fitness is an accurate sample of the phenotype space that the genotype maps to. Another consequence is that it can no longer be guaranteed that a particular genotype will produce a particular phenotype, which suggests that such approaches would be most useful where a phenotype still adequately performs even if it is imperfect. On the other hand, the degree of accuracy can be graded by the amount of computational power invested into the mapping.

In a typical setting artificial development mimics the cellular structure used by nature, where cellular modules cooperate to perform a particular task. This approach has implications for the types of problems that we can attempt to solve, as it may be that some problems are not easily mapped onto this format. Indeed, human designers have difficulties in implementing such systems, and this is particularly evident in programming parallel systems, or indeed in defining rules for cellular automata. Miller reports that evolving the developmental French Flags is hard - with very few runs being successful [21]. It is still unclear whether developmental systems are easier to evolve than non-developmental systems. Although the genotype may be shorter - and hence fewer variables have to be manipulated, development can be expected to distort the fitness landscape. For example, in [15], the evolvability of a simple

developmental encoding was investigated, and it was found that evolution was less effective at finding solutions using development than with direct encoding.

Roggen and Federici compared evolving direct and developmental mappings for the task of producing specific two dimensional patterns of various sizes (the Norwegian Flag and a pattern produced by Wolfram's 1D CA rule 90) [27]. They showed in both cases that, as the pixel dimensions of the patterns increased, the developmental methods outperformed the direct methods. It is noteworthy that performance disparity was much more marked for the relatively regular Norwegian flag pattern than for pattern generated by a 1D cellular automata. Hornby and Pollack evolved context free L-systems to define three dimensional objects (table designs) [16]. They found that their generative system could produce designs with higher fitness faster than direct methods. They point out that generative or developmental systems will scale better than direct methods when modularity is present. In furniture design, for instance, if there is a module that is responsible for producing a table leg, evolution only needs to alter and perfect one module rather than having to independently adjust an arbitrary number of independent table leg producing coding regions. A number of genotype-phenotype mappings on a problem of creating a tessellating tile pattern were examined in [4]. The authors found that an indirect developmental mapping (that they referred to as an implicit embryogeny) could evolve tiling patterns much more quickly than a variety of other representations (including direct) and further, that they could be subsequently grown into much larger sized patterns.

One drawback that they reported was that implicit embryogeny tended to produce the same types of pattern (i.e. of relatively low complexity). As we will see later our results support this finding. In these applications, it appears that development is satisfactory for low complexity problems, where there are many regularities - possibly regardless of scale. Like direct encodings, their behavior does deteriorate as the phenotype scales. One can speculate that this is due to the decrease in the genotype-phenotype correlation with an increase in complexity of the phenotype, which in turn reduces evolvability, as Lehre and Haddow found [18].

Heritable information can also be passed on through mechanisms other than the DNA, and this will affect evolvability and the developmental processes. Such information is the subject of the field of Epigenetics [25]. It includes the maternal influences described earlier and modifications to the genome caused by the mother's interaction with the environment. It appears that in addition to the standard base pair encoding in DNA, the genome also carries another code, the epigenetic code, attached to DNA, and that information provided by this code affects the expression of certain genes. Heritable epigenetic information alters the packing density of the DNA, changing the likelihood of genes being expressed. For example, research comparing the development of twins, shows that epigenetic codes may be more sensitive to environmental influences than DNA [25], where it is reported that just by making changes to the diet of a pregnant mouse, the coat color of pups can

be changed. The role of epigenetics in nature is still controversial, but there are a growing number of examples demonstrating how the environment alters gene expression.

In addition to the effects on development, epigenetics also may affect the evolvability of a species, as epigenetic information is somewhat heritable. Roemer et al. showed that manipulations of epigenetic information in mice were passed down to offspring [26], and comment that "If epigenetic inheritance indeed exists, what is its evolutionary significance? The extent of its effects will depend on the number of genetic loci in the genome that can be modified epigenetically, and on the stability of the modifications. Whether 'epimutations' have any adaptive significance also remains to be established. It should be emphasized that this type of inheritance is rooted firmly in Darwinian selection, with selection possibly both for the modified locus and for the genes that control epigenetic modifications." The use of maternal effects has been demonstrated in developmental neural networks. Matos et al. [19] found that the use of the maternal genotype decreased evolvability. They speculate that this may be due to lag from evolutionary momentum. In a second experiment, they looked at how placental interaction with the mother affected the evolvability of the neural networks. Here they found improvements over a standard developmental approach. It is clear that maternal influences shape developmental behavior, and developing suitable analogies may help artificial development.

9.5 The role of physics

Development in the real world is not just the product of genes. There are interactions with the environment, and in particular the limitations of biological chemistry and physics constrain what biological processes can do. It is likely that primitive life forms relied more on the properties of matter, such as viscoelasticity and chemical/mechanical excitation, rather than on gene expression. Forgacs and Newman argue that such physical properties are a "rough sort development", and that we should not expect either genes or physics to be sufficient on their own [10]. Earlier we discussed that the shapes of forming organs are the result of a small number of processes. These basic processes are determined by the physical properties of the cell namely adhesion, diffusion and viscoelasticity — which, incidentally, are also found in non-living systems and are not under any form of genetic control.

Early multicellular life consisted of cell aggregates. These cell aggregates would have the ability to self-organize into patterns, based on the chemical activity of each cell. Essentially, a cell aggregate would be an excitable medium. The primary role for cell adhesion would be for tissue formation. Due to their chemistries, biological cells have different rates of cell adhesion, which leads to an interesting property during early stages of development. Mature tissues have strong, long-lived links between the cells. Cell adhesion also allows for

ions and small molecules to pass between neighboring cells – without allowing other ions or molecules from the outside to get in.

During early development, however, cells are not joined in this way, but move easily as if in a liquid. This, combined with differential adhesion, forces the cells to become sorted as they move. Furthermore, some cells have adhesive polarity, which causes certain patterns to be formed, since the cells wish to reorganize themselves into a stable, low energy state. A striking example is when a mixture of cells (in this instance from the endodermal and ectodermal germ layers) from an organism called a hydra, are mixed together (producing a random pattern): They will sort themselves into the precise arrangement found in the original organism [29].

When cells evolved to have a variety of types, each with different adhesive properties, these effects of cell sorting occurred and new spatial patterns were constructed. As those properties were coupled with the evolution of polarized cells, the cells could form lumens or elastic sheets. In artificial development, we can use the properties of cell adhesion to generate some target patterns without having to evolve a gene regulatory network (or equivalent). To illustrate this, we present in the following section a method for evolving patterns, including the familiar French flag, using differential cell adhesion.

We would expect that using physical effects such as cell sorting would have limited utility on its own. However, just like in nature, a combination of inherent physical effects and control by a genotype might yield a high degree of sophistication. One advantage of a strong bias toward the physical control of development, compared to the genetic control, is that new cell formations can be achieved through minute genetic change. This may be very important for search algorithms, as one can explore the search space in unusual ways. In effect, a combination of both physics and genetics, and different ratios of the influence between these two factors may give a search heuristic that contains two very different algorithmic aspects. Perhaps as in nature, the balance between the responsibilities of each will be automatically optimized by evolution.

In artificial development the constraints of reality provided by physics do not exist. That means that their benefits, namely to guide and constrain search, are lacking. Because it appears that physics is useful in natural systems, one should perhaps find an analogous artificial physics for artificial development. At present, it is unclear what the artificial equivalents of cell adhesion, surface tension, gravity and diffusion are. It is also unclear what the relationship would be between developmental physics and the physics of the hardware on which the artificial developmental system is implemented. Work in evolvable hardware has shown that evolution is able to make use of the physical properties of its environment, and perhaps we can expect the same from development.

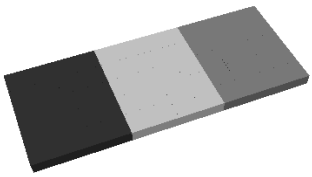


Fig. 9.3. Target French flag pattern.

9.6 Results from a Cell Sorting Experiment

Our model for cell growth consists of a grid whose points can either hold a cell or be empty. There are three different cell types (to map to the red, white and blue fields of the flags) and each of them has its own adhesive properties – to be determined by evolution. To simulate the flow of cells, we employ a simple mechanism whereby cells that wish to move can jump into a neighboring empty cell, or swap places with an existing neighboring cell. When the simulation is run, a cell is picked and a calculation is performed to see whether the entropy of the cell would drop if it were to swap with any of its neighboring cells, as described in [10, chapter 4]. The energy of a particular cell is calculated as the sum of the differences between the adhesion coefficients of the center cell and its neighbors. If a suitable swap is found, then cells swap position. This is repeated a number of times; the number of cell swaps allowed is determined by evolution. We also allow the cells to split, in order for the artificial embryo to grow. If there is an empty neighboring cell, then a cell can divide into this gap and take on the same cell type as its parent. The number of times cells are allowed to split is also determined by evolution.

For these experiments we tried two different approaches to the evolutionary system. In the first, we use two different chromosomes. In one, the chromosome specifies the cell adhesion properties of all cell types, the maximum number of swaps allowed and the maximum number of times cells are allowed to split. The other chromosome type contains a list of cells and their positions, in addition to the above properties. This list is used to define the starting configuration of the developing embryo. These initial cell positions may be analogous to the maternal influences described in section 9.2.

We also investigate the behavior without the evolved starting positions but with a scaled down version of the target, defined by hand. Here only the adhesion coefficients need to be evolved.

Adhesion coefficients are represented by floating point numbers. The cell positions are stored as a variable-length list of coordinates and cell types. In this model we ignore the possibility of cell polarization. Integers are used to store the number of iterations the simulation runs, and how many cell divisions are possible.

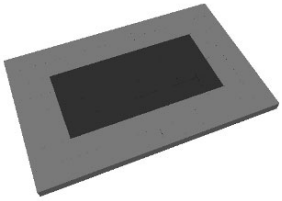


Fig. 9.4. Target cell cluster

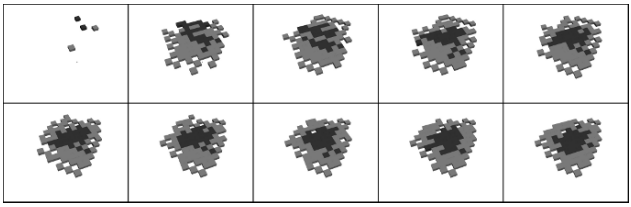


Fig. 9.5. Example of developing cell cluster, where the cell sorting moves the darker cells to the middle of the cell mass.

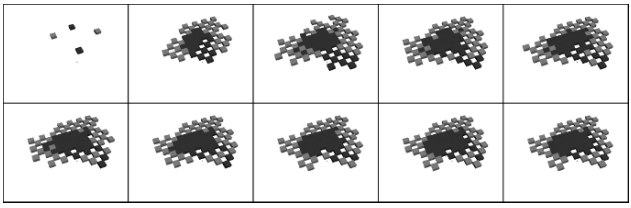


Fig. 9.6. A second example of a developing cell cluster. Again a central dark mass is formed, but here the outer cells produce a spaced pattern.

The fitness function determining the success of a solution counts the number of cells in resultant arrangements that were the same as in the target.

A basic evolutionary algorithm was used, consisting of a population of 50 individuals with tournament selection and elitism. In addition to mutation, we employ a basic two-point crossover on the genotype. Evolution was allowed to run for 5000 generations.

For these experiments, the target pattern was a cell cluster surrounded by an outer cell layer. Figure 9.4 shows the target image, which has similar form to some biological formations such as retinas ([10, page 93]).

Figures 9.5 and 9.6 shows two examples of evolved cell clusters, here evolution was allowed to determine the starting configuration for the cells (top left frame of each sequence). The behavior is similar to that created when the initial target pattern was not specified by the chromosome, such as in runs illustrated in the first ten frames of figure 9.7.

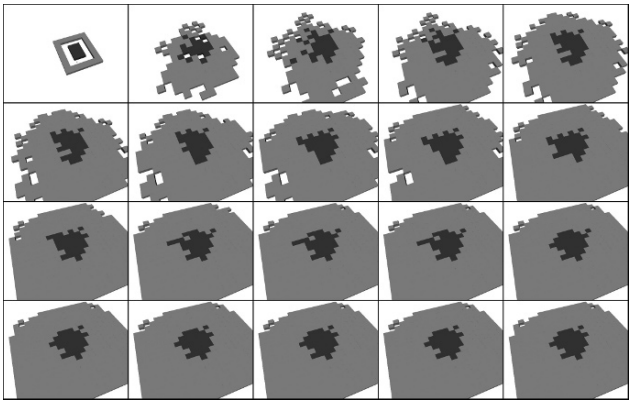


Fig. 9.7. Running the cell sorting beyond the period specified by evolution results in stable patterns. Here the first ten frames are from the period found during evolution to produce maturity. The following frames show the effect of running the simulation for additional time.

Figure 9.7 shows the effect of running a simulation for twice the period of time specified by evolution. The first 10 frames show the sequence that was used in fitness evaluation, and the following frames show what happens after the embryo has reached "maturity". We see that the general shape remains consistent, and that the center of the cell continues to adjust until it finds a point of minimum energy and stabilizes.

As in nature, embryos undergoing development are able to repair damage to some extent. Figure 9.8 shows the same developing embryo as figure 9.7, but this time the embryo is damaged by removing a band of cells. The embryo remains disrupted, however it starts to reform into the target pattern. This ability was not selected for during evolution, and is the result of the physics of development being used for this secondary purpose. Differential cell adhesion is not only responsible for sorting the cells into groups but also for bringing different cell clusters together. This is likely to be one of the mechanisms used for repair in the developing organism.

Other shapes can be produced by evolving the starting configurations, and allowing cell growth and cell sorting to finalize the pattern. For example, figure 9.9 shows an evolved French flag. In contrast to the previous French flag patterns discussed here, this one did not require the evolution of a program to control the behavior of the cells.

The final example is a complicated pattern, based on the types of behavior seen during this artificial development. Figure 9.10 shows a checkerboard in the shape of a triangle, attached to a basic two-color "flag". The purpose of these shapes is to demonstrate that we can evolve a variety of target patterns that have rounded shapes, solid layered masses, shapes containing patterns of empty space and regular structures with sharp edges. In nature there are a limited number of forms that cell groups can form, and these basic patterns

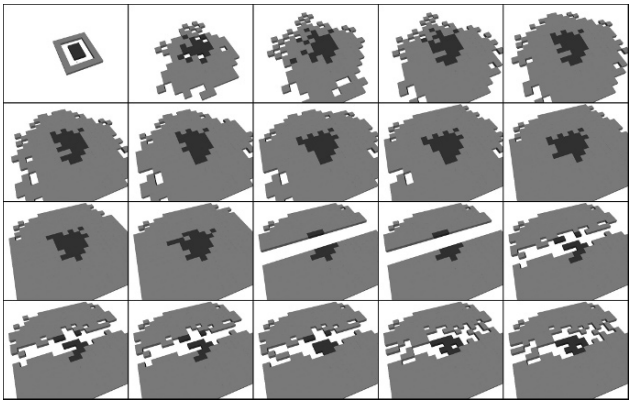


Fig. 9.8. An example of the regrowth of a damaged artificial embryo

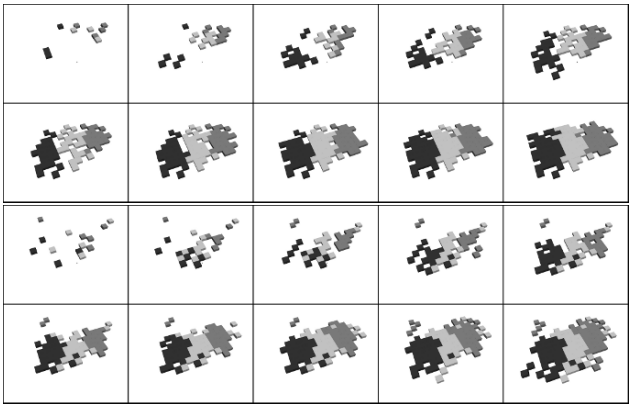


Fig. 9.9. Two examples of a French Flag produced by cell sorting.

are used as building blocks for organs. Without differential cell adhesion we found it impossible to get checkerboard patterns to evolve. The nearest that could be obtained were three groups of cells with a large amount of mixing. This demonstrates that cell movement can be a useful and important part of developmental systems.

Figure 9.11 shows an interesting result observed during testing of the simulation software. Here, each cell group has slightly different cell adhesion properties and the initial state is a randomized cell cluster. However, without any guidance from evolution, a rough French flag pattern is produced. As in nature, certain patterns are perhaps an inevitable consequence of the physical properties of the cells that make them. If this is true, it is important to understand to what extent the patterns are restricted by biological development and to draw conclusions for artificial development.

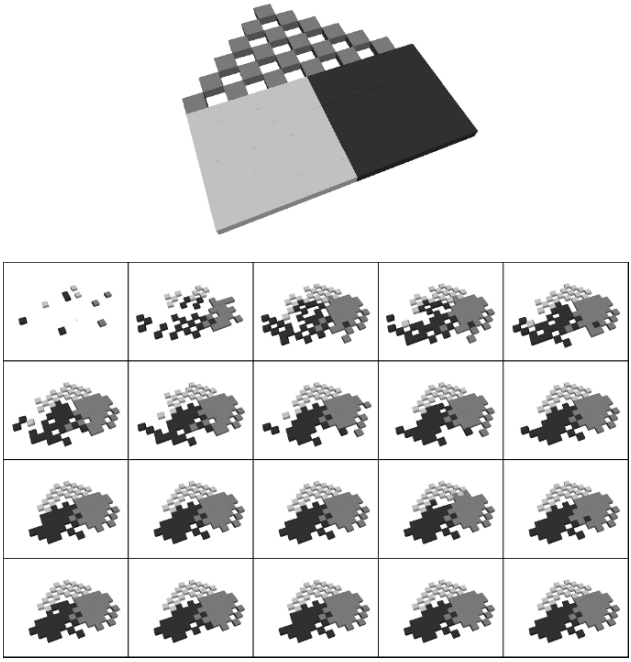


Fig. 9.10. Target checker board and flag, and an example of a developing embryo that forms this pattern.

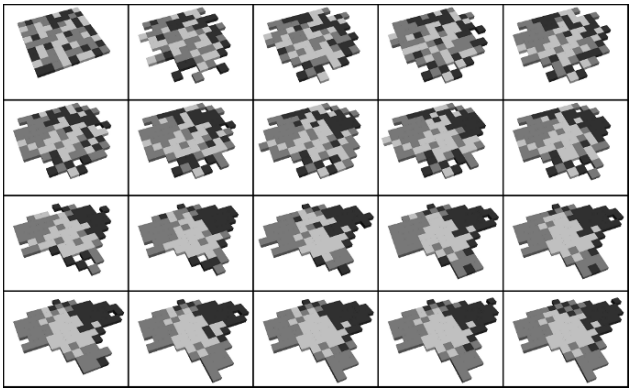


Fig. 9.11. French flag emerging because of differential cell adhesion

9.7 Conclusions

Developmental systems may prove to be a very useful technique in computer science. The field is, however, still in its infancy, and it is difficult to see how the developmental analogy can be applied to many of the typical problems in computer science. Specifically, the challenge is to map development into a computational domain. The applications described here have demonstrated its utility in producing patterns, whether as abstract images or as topographies for neural networks, but transforming these preliminary ideas into a more generalized and practical computational system remains to be done. Downing argues that this is "largely because embryogenesis evolved for the purpose of synthesizing 3-dimensional structure from a linear code, not for growing Universal Turing Machines" [6], and that while we can map problems onto a developmental framework, it is unclear whether this is an inherently suitable approach. Despite these issues, development has many features that are attractive in artificial systems — and if we can get these ideas to work, we will have another powerful, bio-inspired technique to apply.

References

1. W. Banzhaf and C. Lasarczyk. Genetic programming of an algorithmic chemistry. *Genetic Programming Theory and Practice II*, 8:175–190, 2004.
2. W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming: An Introduction.* Morgan Kaufmann, San Francisco, 1998.
3. P. Bentley. Investigations into graceful degradation of evolutionary developmental software. *Natural Computing*, 4(4):417–437, 2005.
4. P. Bentley and S. Kumar. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 35–43, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann.
5. A. Carrascal, D. Manrique, D. Pérez, J. Ríos, and C. Rossi. Growing axons evolving l-systems. In M. Hamza, editor, *proceedings Artificial Intelligence and Applications*, volume 403. Acta Press, 2003.
6. K. L. Downing. Developmental models for emergent computation. In A. M. Tyrrell, P. C. Haddow, and J. Torresen, editors, *International Conference on Evolvable Systems(ICES)*, volume 2606 of *Lecture Notes in Computer Science*, pages 105–116. Springer, 2003.
7. A. Eiben and J. Smith. *Introduction to Evolutionary Computing.* Springer, Berlin, 2003.
8. G. Escuela, G. Ochoa, and N. Krasnogor. Evolving L-systems to capture protein structure native conformations. In M. Keijzer, A. Tettamanzi, P. Collet, J. I. van Hemert, and M. Tomassini, editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 74–84, Lausanne, Switzerland, 30 Mar. - 1 Apr. 2005. Springer.
9. D. Federici. Evolving developing spiking neural networks. In *proceedings of CEC 2005 IEEE Congress on Evolutionary Computation*, pages 543– 550, 2005.

10. G. Forgacs and S. A. Newman. *Biological Physics Of The Developing Embryo.* Cambridge University Press, 2005.
11. J. Gerhart and M. Kirschner. *Cells, embryos, and evolution : toward a cellular and developmental understanding of phenotypic variation and evolutionary adaptability.* Malden, Mass. : Blackwell Science, 1997.
12. F. Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm.* PhD thesis, Laboratoire de l'Informatique du Parallilisme, Ecole Normale Supirieure de Lyon, France, 1994.
13. F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89, Stanford University, CA, USA, 28–31 1996. MIT Press.
14. S. Harding. *Evolution In Materio.* PhD thesis, University of York, 2005.
15. S. Harding and J. F. Miller. The dead state: A comparison between developmental and direct encodings. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, 2006.
16. G. S. Hornby and J. B. Pollack. The advantages of generative grammatical encodings for physical design. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 600–607, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 2001. IEEE Press.
17. H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4(4):461–467, 1990.
18. P. Lehre and P. Haddow. Developmental mappings and phenotypic complexity. In *proceedings of IEEE Congress on Evolutionary Computation(CEC) 2003*, pages 62–68, 2003.
19. A. Matos, R. Suzuki, and T. Arita. Evolutionary models for maternal effects in simulated developmental systems. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 149–150, New York, NY, USA, 2005. ACM Press.
20. J. Miller and M. Hartmann. Untidy evolution: Evolving messy gates for fault tolerance. In *Proceedings of The 4th International Conference on Evolvable Systems: From Biology to Hardware, ICES2001*, volume 2210 of *Lecture notes in computer science*, pages 14–25, Tokyo, Japan, 2001. Springer-Verlag.
21. J. F. Miller. Evolving a self-repairing, self-regulating, french flag organism. In K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. K. Burke, P. J. Darwen, D. Dasgupta, D. Floreano, J. A. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. M. Tyrrell, editors, *GECCO (1)*, volume 3102 of *Lecture Notes in Computer Science*, pages 129–139. Springer, 2004.
22. M. Mitchell. *An introduction to genetic algorithms.* MIT Press, Cambridge, MA, USA, 1996.
23. C. Nüsslein-Volhard. *Coming To Life.* W. W. Norton and Company, 2006.
24. C. Ortega-Sanchez, D. Mange, S. Smith, and A. Tyrrell. Embryonics: A bio-inspired cellular architecture with fault-tolerant properties. *Genetic Programming and Evolvable Machines*, 1(3):187–215, 2000.
25. J. Qiu. Epigenetics: unfinished symphony. *Nature*, 441:143–145, May 2006.
26. I. Roemer, W. Reik, W. Dean, and J. Klose. Epigenetic inheritance in the mouse. *Current Biology*, 7:277–280, 1997.

27. D. Roggen and D. Federici. Multi-cellular development: is there scalability and robustness to gain? In X. Yao, E. Burke, and J. L. et al., editors, *proceedings of Parallel Problem Solving from Nature 8, Parallel Problem Solving from Nature (PPSN) 2004*, pages 391–400, 2004.
28. A. Siddiqi and S. Lucas. A comparison of matrix rewriting versus direct encoding for evolving neural networks. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, (Piscataway, NJ, USA)*, pages 392–397. IEEE Press, 1998.
29. U. Technau and T. Holstein. Cell sorting during the regeneration of hydra from reaggregated cells. *Developmental biology*, 151(1):117–27, 1992.
30. A. Thompson and C. Wasshuber. Design of single electron systems through artificial evolution. In *Int. J. Circuit Theory and Applications*, volume 28 (6), pages 585–599, 2000.

10

Self-adaptive Worker-Helper Systems with Self-Organized Task Allocation

Daniel Merkle, Martin Middendorf, and Alexander Scheidler

Department of Computer Science, University of Leipzig, Johannisgasse 26, 04103 Leipzig, Germany
merkle@informatik.uni-leipzig.de, middendorf@informatik.uni-leipzig.de, scheidler@informatik.uni-leipzig.de

Summary. In this chapter a self-organized worker helper system is described, which is part of an abstract Organic Computing system (OC system). It consists of normal worker components and helper components, and the workers need some service from time to time in order to continue with their normal work. The service is done by the helpers, which have reconfigurable hardware to perform the different service tasks. The speed of service for a certain task depends on the amount of resources configured for this task. Strategies are presented that can be used by the helpers to decide whether to accept a service task and how to reconfigure themselves. It is also described how the worker helper system can be organized without global knowledge about the type of service requests and the set of available helper components. In order to obtain a decentralized mechanism and to make it suitable for the paradigm of OC a fully decentralized and dynamic clustering algorithm has been combined with a self-organized task allocation system. Empirical results show that the described worker helper system can adapt to dynamic situations with changing probabilities for service, and that decentralized clustering is able to reduce the reconfiguration cost significantly.

Key words: Task allocation, clustering, decentralized algorithms, reconfigurable hardware, service tasks

10.1 Introduction

Intelligent components form an essential part of nearly all modern technical systems, no matter whether they are used in fabrication sites, for transportation, for communication or in the household. The development of intelligent components that can communicate with each other to exchange data, to coordinate their actions, or to make common control decisions has led to systems that become increasingly autonomous. Ideally, these systems can decide themselves about which tasks have to be executed next and what management tasks are necessary.

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_10,

Organic computing (e.g., [3, 10, 11]) is a new field of computer science with the aim to design and understand computing systems with so-called self-x properties, e.g., self-configurable, self-optimizing, self-healing, and self-servicing systems. One aim of Organic Computing is to use principles of self-organization in order to obtain such systems. An important source of inspiration for OC are the principles of self-organization found in natural systems. Examples for such principles are the task selection principle of social insects or the principles used by the immune system to protect an organism from infection by pathogens. Computing systems with self-x properties following such design principles are called Organic Computing systems (OC systems).

Since am OC system should be highly self-manageable and will typically consist of many autonomous components it is necessary that the system itself provides various types of service tasks for its components. In this chapter we describe how such a system can be designed, not by describing a system for a particular application but by focusing on an abstract model of an OC system to explain general principles of self-organization that can be used to execute the service tasks efficiently. The contents of this chapter are based on work (see [8, 9]) done in the project "Organization and Control of Self-Organizing Systems in Technical Compounds", which is part of the DFG priority program on Organic Computing.

In the model OC system it is assumed that the service tasks are executed by special helper components. The other components of the OC system that do the "normal" work are called worker components. To provide enough flexibility for an OC system that can adapt to the needs of the user or the environment it is further assumed that the service tasks are executed on reconfigurable hardware. Reconfigurable hardware (for example Field Programmable Gate Arrays (FPGAs)) can adapt to a different tasks so that they can be executed more efficiently.

In the first part of this paper we consider strategies for reconfiguring the helper components and for distributing the service requests. To fit the paradigm of OC central control mechanisms have to be avoided in the design of the system. Instead each helper decides autonomously whether to execute a service request from a worker and how to reconfigure itself. The considered strategies will be evaluated under different assumptions about the time needed for the execution of a service task. This time depends on the configuration of the corresponding helper, the time necessary for reconfiguration of a helper and the time needed for communication between a worker and a helper. Different strategies for reconfiguration can lead to different overall efficiencies of the system. The reason is that spending more time for reconfiguration can lead to helpers which are better adapted for the service request and therefore can execute the corresponding service tasks faster. Hence, there is a trade-off between the amounts of time spent for reconfiguration and for execution of the service tasks.

Since the focus of the first part of this chapter is on the reconfiguration strategies and the request acceptance strategies it is assumed that the service

requests are already classified into different types. The type of a service request determines how the different parts of the helper hardware have to be reconfigured so that the helper is able to execute the corresponding service task. It is also assumed that a service request is always sent to a randomly chosen helper. Both assumptions are dismissed in the second part of the chapter, where we study how to organize the worker helper system without global knowledge about the relative number of service requests with different types of hardware requirements for the helpers. In particular, we describe a decentralized system that classifies the service tasks. We also present strategies for the helpers how to specialize to certain types of service tasks. A helper should not satisfy just any service request because it might need much reconfiguration before it can start performing a particular service task. When reconfiguration costs are high this would lead to inefficient execution of service tasks. On the other hand, due to the lack of a global view of the system, the helper can not just chose only service tasks that are perfectly suitable for it and can be done with a small amount of reconfiguration. When the helpers are too selective some service tasks might never been taken by any helper, when they are not selective enough their work becomes inefficient.

Some strategies for self-organization described in this chapter are inspired by the principles of social insects to organize the work within a colony. The service request acceptance strategy of the helpers is inspired by stimulus-threshold models that are used by biologists to explain the self-organized labor division in social insect colonies (see, e.g., [1, 2, 12]). In these models each individual has a personal threshold value for each task, which determines the its preference to work on that task. In addition, each task has a stimulus value, which determines how necessary it is that individuals work on it. The probability of an individual working on a task depends on the ratio of its threshold value for the task and the stimulus value of the task. The lower the threshold value and the higher the stimulus value the more likely the individual will work on the task. In this study we consider a basic scenario where the workers might need only two different types of service. Different scenarios with static or dynamically changing composition of the service tasks are considered. We compare the self-organized task allocation scheme with a fixed allocation scheme where helpers accept any request and are reconfigured to perform all service tasks equally fast. The method used to cluster the service requests is based on decentralized algorithms proposed in [6, 7]. These algorithms are inspired by an approach for ant clustering based on the chemical recognition system of ants proposed in [4, 5]. In this approach the objects to be clustered are considered artificial ants. Then an artificial odor is used that is learned by the artificial ants and represents the odor of their nest (i.e., their cluster).

The chapter is organized as follows. In the next section 10.2 we introduce our model for the OC worker-helper system. The reconfiguration strategies and the task allocation scheme are presented in section 10.3. Empirical results are presented in section 10.4. The decentralized service task clustering and

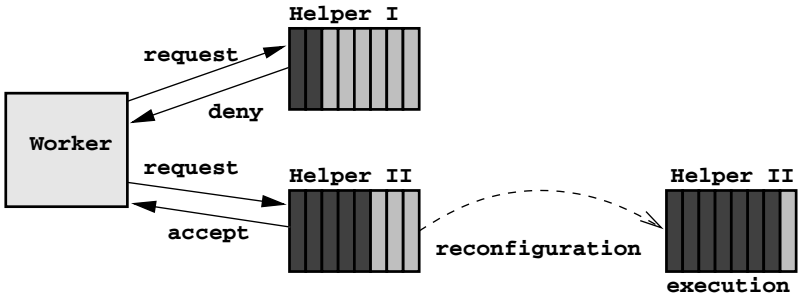


Fig. 10.1. A typical scenario in the worker-helper system: a worker has a service request; the first helper contacted denies the request, because too few resources are configured for the task type; the second helper contacted accepts the task, increases the number of resources for the task type by reconfiguration, and executes the service task.

allocation scheme is described in section 10.5. Corresponding empirical results are described in section 10.6, and conclusions are given in section 10.7.

10.2 Model of the computing system

We now introduce an abstract model of an OC system, which is the basis of the investigations of the first part of this chapter. The computing system has two types of components or members, ordinary members called *workers* and supporting members called *helpers*. Let m be the number of workers and $H_1, \ldots, H_n$ be the n helpers. Each helper has reconfigurable hardware (e.g., a Field Programmable Gate Array (FPGA)), on which it performs the service tasks. The hardware consists of q slices, which can be reconfigured independently from each other (see figure 10.1 for an example). Each slice is always configured to work for only one type of tasks (a total of two types of tasks is assumed here). Different slices can be configured for different types of tasks. A reconfiguration operation changes the type of task that can be executed on the slice. During reconfiguration a helper can reconfigure any number $\leq q$ of slices. The time to reconfigure all slices of a helper is $t_r > 0$ and the time to reconfigure k slices is $(t_r/q) \cdot k$.

Each helper can work on at most one task per time step. The time it takes to finish a task depends on the number of slices configured for the corresponding task type. A helper H with a proportion of $\geq 1/2$ of its slices configured for tasks of type i is called *specialized* for tasks of type i. Let $s(H) = i$ if H is specialized to tasks of type i, $i \in \{1, 2\}$. H is *fully specialized* for tasks of type i when all slices are reconfigured for this type of tasks. The *degree of specialization* of a helper is the relative number of slices that are configured for the type of tasks the helper is specialized to. Let s_j, $j \in \{1, \ldots, n\}$ denote the degree of specialization of helper H_j. Let s_{ij}, $i \in \{1, 2\}$, $j \in \{1, \ldots, n\}$ denote

the relative number of slices that helper H_j has configured for task type i. It is assumed that the execution time of a task is $t_e \cdot q/k$, where $t_e > 0$ is the execution time of a task on a fully specialized helper and k is the number of slices working for the task.

At each time step a worker needs servicing (of either type 1 or type 2) with some probability (called *request rate*) before it can continue its normal work. The *relative request rate* $p_i \geq 0$ for task type i is the probability that a service request is of type i. A worker that needs servicing of type i contacts a randomly chosen helper and requests a service task of type $i \in \{1, 2\}$. If the request is granted the service task is executed by the helper and the worker can continue its normal work. If the request is not granted the worker contacts another (randomly chosen) helper. The first request that a worker does when it needs service is called *initial request* the other requests are called *repeated requests*. A contact (communication) between a helper and a worker takes the time $t_c \geq 0$.

10.3 Task allocation and reconfiguration strategy

The task acceptance strategies and the reconfiguration strategies for the helpers are presented in this section.

10.3.1 Task allocation

A helper always accepts a servicing request when it has at least $q/2$ of its slices configured for the corresponding type of task. Otherwise, the probability that it accepts the request depends on the personal threshold value for this type of task, a stimulus value for the task, the degree of specialization for this type of task, and the relative request rate. The *stimulus value* for a type of task is the number of tasks of this type minus the number of tasks of the other type counting all tasks that are actually requested by the workers. Let T_{ij}, $i \in \{1, 2\}$, $j \in \{1, \ldots, n\}$ denote the threshold of helper H_j for task type i and S_i the stimulus of task $i \in \{1, 2\}$. The probability that helper H_j accepts a request for task i is defined as

$$p_{(p_i, s_j, S_i, T_{ij})} := \begin{cases} \min\{1, f(p_i, s_j) + \dfrac{S_i^2}{S_i^2 + T_{ij}^2}\} & \text{if } s(H_j) \neq i \\ 1 & \text{else} \end{cases} \tag{10.1}$$

where the function $f(p_i, s_j)$ is defined in the following.

As described in the last section a helper always accepts a request when it has at least half of its slices configured for the corresponding type of task. Otherwise it accepts with a probability determined by (10.1). In the following we define the function f used in that formula. We first consider the case of a single helper H and a static situation, assuming that the helper can not be

reconfigured and the service requests arrive at constant rates for both types of task. The motivation for the definition of f is that H should reject so many tasks of the type is it not specialized for that it has an optimal configuration for the resulting relative number of both task that it executes. Therefore the first aim is to determine the optimal percentage of slices that should be configured for type 1 tasks (the rest of the slices is configured for tasks of type 2 for given fixed relative request rates p_i, $i \in \{1, 2\}$. Let us assume that the (normalized) run time for a task on a fully specialized helper is $t_e = 1$). Then the run time for a task of type i on a helper H_j with a fraction of s_{ij} of its slices configured for i is $1/s_{ij}$. The expected run time for a task on a helper H with specialization level s and $s(H) = i$ is $p_i/s + (1-p_i)/(1-s)$. For this case it can easily be shown that the optimal percentage $g(p_i)$ of slices configured for task type i is $g(p_i) = (p_i - \sqrt{p_i - p_i^2})/(2p_i - 1)$.

To define function f in (10.1) we consider a situation where the stimulus value for both tasks is zero. A helper H is considered which rejects tasks for which it is not specialized according to (10.1). Without loss of generality we assume that these tasks are of type 2 and $s(H) = 1$. Since $S_2 = 0$ it follows from (10.1) that tasks of type 2 are rejected with probability $f(p_1, s)$. The function f is now determined such that the relative rates of tasks of different types that are accepted by H are optimal for its current degree of specialization s if this is possible. Otherwise, the relative number of requests for task 2 is too low for an optimal degree of specialization. In this case all tasks of type 2 are accepted. Observe, that $p_1/(p_1 + (1 - p_1)f(p_1, s))$ is the relative rate of tasks of type 1 that are accepted by H. Then $f(p_i, s)$ can be determined as follows. Set $g(p_i/(p_i+(1-p_i)f(p_i, s)) = s$ Then it can be shown that $f(p_i, s) = \min\{1, (p_i/(1 - p_i)) \cdot (s - 1)^2/s^2\}$. Clearly, our definition of f is heuristic and not necessarily optimal for a computing system with several helpers.

10.3.2 Reconfiguration strategies

The reconfiguration strategy of a helper determines how many slices are reconfigured for an accepted task. Two basic reconfiguration strategies are compared here. One of them, called *1-slice reconfiguration strategy*,is that a helper performing a service task always performs a reconfiguration operation so that the number of slices that can execute the corresponding type of service tasks is increased by one (unless all slices have already been configured for the corresponding type of task). A possible problem of the 1-slice strategy is that the execution time of a service task is very long when only a few slices can execute it. Therefore, a variant called *1+half-slice reconfiguration strategy* is considered, which differs in the case that a request for service is accepted when less than half of the slices are configured for the corresponding type of task. In this case the helper reconfigures itself so that half of the slices are configured for the accepted type before execution starts.

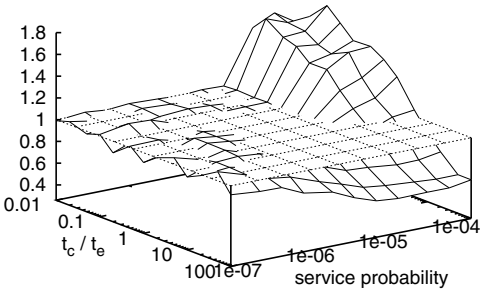


Fig. 10.2. Relative throughput of the self-organized computing system. compared to the static $\mathcal{S}$-system

10.4 Empirical results I

Empirical results for the worker-helper system are presented in this section.

10.4.1 Notation and parameters

Unless stated otherwise the following parameter values have been used for the simulation s. Two types of tasks have been used with an execution time of$t_e = 100$ and a communication cost of $t_c = 10$. The number of workers was set to $m = 100$ and the number of helpers is $n = 10$. Each helper has $q = 10$ slices and the time to reconfigure all slices of a helper is $t_r = 1000$. For all tasks and helpers the same threshold value $T := T_{ij} = 100$, $i \in 1, 2$, $j \in \{1, \ldots, 10\}$ was used. All results are averaged over 20 runs. The standard reconfiguration method is the 1-slice strategy.

We call $\mathcal{S}$-system a simple system where the degree of specialization is fixed to $s_j = 0.5$ for each helper H_j and $f(x) := 1$ (where f is the function in (10.1)). Note, that the $\mathcal{S}$-system, which has a fixed parameter s_j for each helper H_j, performs no reconfiguration operations and therefore has no reconfiguration cost.

10.4.2 Static environment

In order to compare the influence of different reconfiguration heuristics simulations have been made with systems where the helpers use the 1-slice strategy and also with systems where helpers use the 1+half-slice strategy for reconfiguration.

First we show that the communication costs t_c are an important parameter for the behavior of the system. Their influence on the throughput is shown for

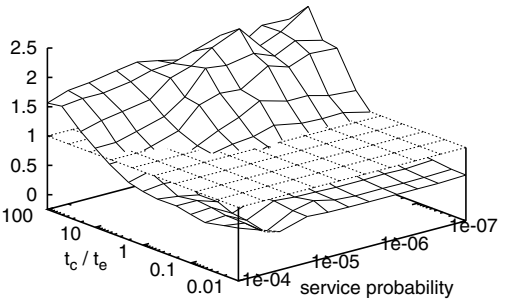


Fig. 10.3. Relative sojourn times of the self-organized computing system compared to the static $\mathcal{S}$-system.

the case that the relative request rates are the same for both types of tasks. Figure 10.2 compares the throughput of a system with self-organized task allocation (using the 1-slice reconfiguration strategy) with the $\mathcal{S}$-system. Shown is the relative performance difference (i.e., throughput of the self-organizing system divided by the throughput of the $\mathcal{S}$-system) of both reconfiguration strategies and service probabilities in $\{0.001, 0.0005, 0.0002, \ldots, 0.000001\}$ and relative communication costs $t_c/t_e \in \{0.01, 0.02, \ldots, 20\}$. It can be seen that for high service probabilities and small values of t_c/t_e (< 1), the throughput of the self-organized system is almost twice as high as for the $\mathcal{S}$-system. Only in the case of small service probabilities and very high relative communication costs ($t_c/t_e \geq 2$) the throughput of the $\mathcal{S}$-system is better. The reason is that in the self-organized system the requests are rejected with some probability, which implies additional communication costs. For small service probabilities and relative communication costs of $t_c/t_e \leq 10$ the performance of both systems is similar.

Figure 10.3 compares the sojourn times of the self-organized system and the $\mathcal{S}$-system. The performance of both systems differs significantly for most parameter combinations. For small values of t_c/t_e (< 1) the sojourn times of the self-organized computing system are smaller for all tested probabilities of service. Note that for a small service probability (e.g. $p = 10^{-7}$) both systems have the same sojourn times for $t_c/t_e = 1$. The reason is as follows: in the self-organizing system a worker needs approximately 2 requests to find a fully specialized helper, in the $\mathcal{S}$-system all requests are accepted, but need twice the time to execute the task. For $t_c/t_e \approx 1$ this behavior. Their influence on the throughput is shown for the case that the relative request rates are the same for both types of tasks. Figure 10.2 compares the throughput of a system with self-organized task allocation (using the 1-slice reconfiguration strategy) with the $\mathcal{S}$-system. Shown is the relative per-

formance difference (i.e., throughput of the self-organizing system divided by the throughput of the $\mathcal{S}$-system) of both reconfiguration strategies and service probabilities in $\{0.001, 0.0005, 0.0002, \ldots, 0.000001\}$ and relative communication costs $t_c/t_e \in \{0.01, 0.02, \ldots, 20\}$. It can be seen that for high service probabilities and small values of t_c/t_e (< 1), the throughput of the self-organized system is almost twice as high as for the $\mathcal{S}$-system. Only in the case of small service probabilities and very high relative communication costs ($t_c/t_e \geq 2$) the throughput of the $\mathcal{S}$-system is better. The reason is that in the self-organized system the requests are rejected with some probability, which implies additional communication costs. For small service probabilities and relative communication costs of $t_c/t_e \leq 10$ the performance of both systems is similar.

Figure 10.3 compares the sojourn times of the self-organized system and the $\mathcal{S}$-system. The performance of both systems differs significantly for most parameter combinations. For small values of t_c/t_e (< 1) the sojourn times of the self-organized computing system are smaller for all tested probabilities of service. Note that for a small service probability (e.g. $p = 10^{-7}$) both systems have the same sojourn times for $t_c/t_e = 1$. The reason is as follows: in the self-organizing system a worker needs approximately 2 requests to find a fully specialized helper, in the $\mathcal{S}$-system all requests are accepted, but need twice the time to execute the task. For $t_c/t_e \approx 1$ this behavior leads to the same sojourn times ($\approx 2 \cdot t_c + t_e$ in the self-organized system and $\approx t_c + 2 \cdot t_e$ for the $\mathcal{S}$-system). For large service probabilities the self-organizing system has smaller sojourn times even for larger relative communication costs. The reason is that in an overloaded system the sojourn times can be approximated by the throughput. If the communication costs are not too large and most of the time all helpers are executing tasks, the throughput is twice as high for the self-organized system. For $t_c = 10$ and 100 workers the expected value of workers requesting service is $100/t_c = 10$ in each iteration. In the self-organized system the chance of being rejected is approximately the same as the probability of being rejected in the $\mathcal{S}$-system since the throughput is only half as high. For even larger communication costs most of the helpers are not executing tasks. As the average number of requests is ≈ 2 in the self organizing system and ≥ 1 in the $\mathcal{S}$-system, the sojourn times are twice as high in the self-organizing system (the execution time can be neglected in that case). Only for high service probabilities and very high relative communication costs ($t_c/t_e \geq 2$) the sojourn of the $\mathcal{S}$-system are better.

In the following an oscillation effect is demonstrated that is typical for many self-organized systems. This effect occurs even in a very simple system with only one helper that has only one slice. For the simulation the probability that a worker needs service is 0.0001, reconfiguration costs are $t_r = 1000$ and each threshold parameter T_{ij} was set to 1000. Figure 10.4 shows the number k_1 and k_2 of actual requests for tasks of type 1 and 2, respectively, over the number of iterations. It can be seen, that the values are oscillating. The reason for this is that a helper specializes for one type of tasks – say type 1 – and then

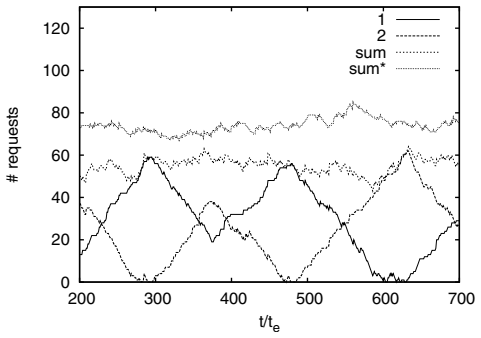


Fig. 10.4. Number of actual requests in a self-organized one-helper system compared to a one-helper $\mathcal{S}$-system; i: actual number of request of type i, sum (sum*): total number of actual requests for the self-organized system (respectively for the $\mathcal{S}$-system).

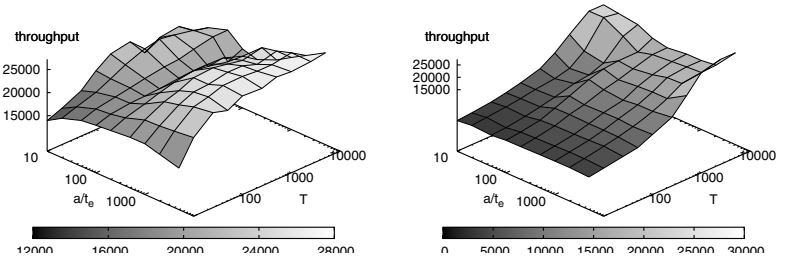


Fig. 10.5. Self-organized computing system: throughput for different thresholds T and different degrees of dynamics a; $t_e = 100$, $t_r = 1000$ and $t_c = 10$, number of slices: 10 (left), 100 (right).

rejects requests for the other type. When the number of (waiting) requests for type 2 increases, the corresponding stimulus value increases as well. This leads to an increased probability that the helper executes tasks of type 2. The reconfiguration operations in this case have the effect that the helper becomes specialized for tasks of type 2. Compared to a simple $\mathcal{S}$-system with one helper the figure shows that the actual number $k_1 + k_2$ of requests that are waiting is smaller in the self-organized system. It should be noted that an oscillating behavior can also occur for more complex systems that have more than one helper.

10.4.3 Environment with changing service probabilities

To investigate the behavior of the self-organized task allocation scheme in dynamic situations the case of changing request rates has been investigated. In

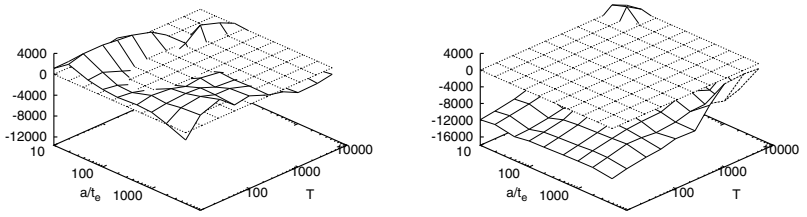


Fig. 10.6. Self-organized service system: throughput difference between the standard system (using the 1-slice reconfiguration strategy) and a system using the 1+half-slice reconfiguration strategy for different thresholds T and different degrees of dynamics a; $t_e = 100$, $t_r = 1000$ and $t_c = 10$, number of slices: 10 (left), 100 (right).

the experiments the two request rates of 0.0004 and 0.0016 were exchanged every $a/t_e \in \{10, 20, 50, 100, 200, 500, 1000, 2000, 5000\}$ time steps. Obviously the threshold parameter $T = T_{ij}$ has a strong influence on the adaptiveness of the computing systems. For a high value for T it is unlikely that the helpers reconfigure themselves (comp. (10.1)). In the experiments threshold values $T \in \{10, 20, 50, 100, 200, 500, 1000, 2000, 10000\}$ were used. For each combination of a/t_e and T the throughput within $3000 \cdot t_e$ time steps was measured. Note, that $a/t_e = 2000$ is a situation where the request rate is changed only once over the considered time interval. The reconfiguration time was set to $t_r = 1000$ and a communication time of $t_c = 10$ was used in the experiments.

The experimental results for the self-organized service system are depicted in figure 10.5 for reconfiguration times of $t_r/t_e = 10$, $t_c/t_e = 0.1$ and number of slices of $q = 10$ and $q = 100$. The throughput of the self-organized system has to be compared to the average throughput of 14109 that is achieved by the $\mathcal{S}$-system. For very large values of T the self-organized service system is not very adaptive. In all tested cases the best performance is achieved for the extreme values of $a/t_e = 10$ or $a/t_e = 2000$. The reason is that using $a/t_e = 2000$ does not require adaptiveness. When $a/t_e = 10$ is used the arrival rates change so often that the situation becomes similar to one where the probabilities for both types of service are identical and fixed. Such a situation does not require adaptiveness and hence a high value of T leads to good performance. The worst throughput is achieved in situations with many slices and a small value for T. In such situations it is likely that a helper excepts a request, even when it has a small degree of specialization for the corresponding type of task. This can lead to very long execution times for the tasks (recall that using only 1 out of k slices for a task increases the execution time by a factor of k compared to execution with full specialization).

Figure 10.6 compares the results of the standard self-organized system (using the 1-slice reconfiguration strategy) with one using the 1+half-slice re-

configuration strategy. The idea behind the 1+half-slice reconfiguration strategy is to make the system more easily adaptive for changes of the relative request rates for different types of tasks. It can be seen in the figure that for 10 slices the 1+half-slice reconfiguration strategy obtains a higher throughput for higher threshold values (and not too small values of a, recall that $a/t_e \leq 30$ leads to a situation similar to the one with constant service probabilities). For 100 slices (where higher adaptivity is even more important) the 1+half-slice reconfiguration strategy is better than the 1-slice strategy. Only for situations with very high threshold values $T > 5000$ and very small values of $a/t_e \leq 30$ the standard reconfiguration strategy is better.

10.5 Decentralized service task clustering and allocation

So far in this chapter it was assumed that the workers choose a helper to ask for service randomly. But in systems with decentralized organization the problem occurs that the workers and helpers might be connected via a network without directly knowing from each other. Another problem in OC systems is that clustering of the service requests according to their type might not be available beforehand, e.g., when there is no global knowledge about the possible service tasks. Both problems are addressed in this second part of the chapter. We show how to organize the worker-helper system in a decentralized way so that the service requests of the workers that are executed by a helper are suitable for it.

Clearly, helpers should not satisfy just any service request because they need much reconfiguration before they can start performing a service task not fitting to their current configuration. On the other hand, they do not have a global view of the system and should not chose only service tasks that are suitable for them and can be executed with a very small amount of reconfiguration. When the helpers are too selective some service tasks might never been taken by any helper.

It is assumed here that each worker includes information about the required helper configuration into its request for service. The approach taken in this chapter is to use a decentralized clustering scheme to cluster the service request packets sent through a network with respect to their hardware requirements for the helper. Each helper might then specialize to service requests of one cluster. Since the hardware resources needed for the service tasks within one cluster are similar this will lead to small reconfiguration costs. Several problems and questions that emerge for such a system are addressed in the following.

10.5.1 Task allocation

A combination of a decentralized clustering algorithm and a self-organized task allocation system is used to organize a worker-helper mechanism via the

network of an OC computing system. The network consists of routers and helper nodes. To each service request corresponds a service packet that is sent into the network. The resource demands for the corresponding service tasks are specified in the packet. The number of service packets created per time step is called the *(packet) arrival rate*.

The problem is to execute as many service tasks as possible and also to have small reconfiguration costs for the helpers. It is assumed now that for each service request packet the number of helpers visited by the packet (number of hops) is counted. If a packet is rejected by a helper this counter is increased by one. If the counter of a packet exceeds a threshold value TTL (time to live) the service request packet is dropped. The fraction of dropped packets (in relation to all packets) is called the *(packet) drop rate*. The main idea is to group the service tasks into clusters which have similar hardware resource demands and to let the helpers specialize by allowing them to execute only the service tasks from one of the clusters. Since all tasks in one cluster are similar with respect to their resource demands, the reconfiguration costs for the helpers of switching between these tasks are relatively small.

To group the service requests a decentralized clustering algorithm is used. Each service request packet is characterized by a vector (v_i, c_i) where $c_i \in \{1, \ldots, n_c\}$ is the cluster number and v_i is a vector that describes the hardware resources needed for the corresponding service task, i.e., the helper configuration required to execute it.

The clustering algorithm is similar to the one from [6, 7]. In these papers decentralized packet clustering algorithms in networks have been investigated for static and dynamic situations (i.e., the data vectors used for the clustering may change over time). The corresponding clustering problem – called Decentralized Packet Clustering Problem (DPC) – is to find for a set of packets $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ sent through the network a good clustering, i.e., a partitioning $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$ of $\mathcal{P}$. Each packet $P_i \in \mathcal{P}$ contains a data vector v_i that is used for the clustering. In the dynamic version called d-DPC the data vector of a packet can change at every time step.

A successful yet simple algorithm to solve the DPC problem is d-DPClust$_{zc}$ – simply called d-DPClust in the following. For this algorithm each packet $P_i = (v_i, c_i)$ has a data vector v_i and a cluster number c_i. Each router r stores a vector of estimated centroids $\mathcal{Z}_r = (z_r^1, \ldots, z_r^{|\mathcal{C}|})$. When a packet P_i arrives in a router the centroid estimation for cluster c_i is modified according to $z_r^{c_i} = (1-\beta) \cdot z_r^{c_i} + \beta \cdot v_i$, where β, $0 < \beta < 1$ is a parameter of the algorithm. Thus, β determines the share that the data vector v_i has of the new centroid estimation. Then, the new cluster number for packet P_i is determined by using the distances of its data vector v_i to the estimated centroids z_r^j, $j = 1, \ldots, |\mathcal{C}|$ that are stored in the router. The packet is assigned to the cluster for which this distance is minimal, i.e. $c_i = \text{argmin}_j ||v_i - z_r^j||$.

Each helper node in the network has an associated cluster number. If the cluster number of a service request packet received by a helper is identical to its cluster number, then the helper is reconfigured to satisfy the resource

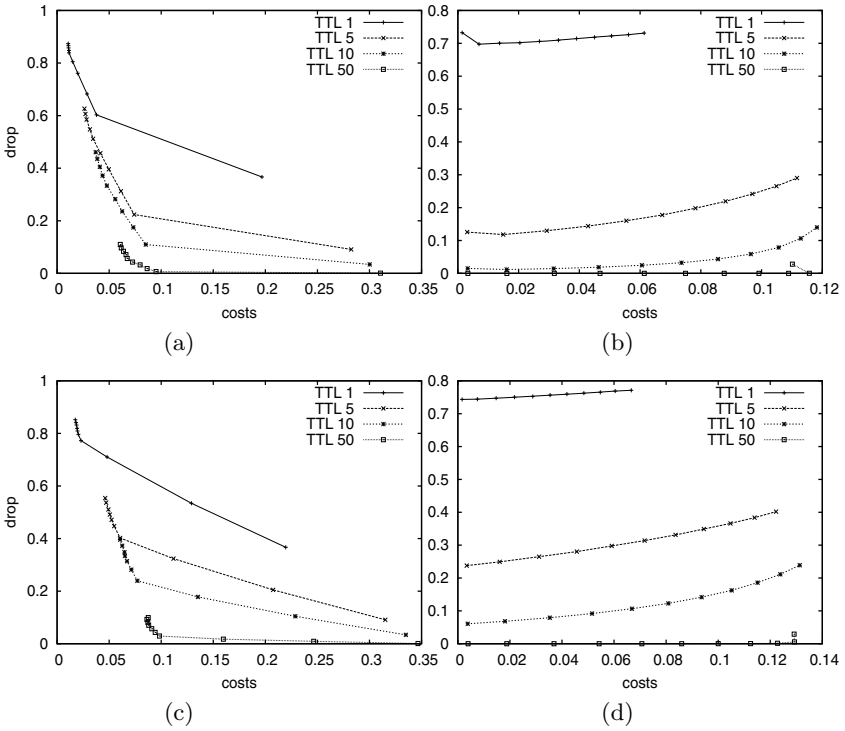


Fig. 10.7. Drop rate/reconfiguration cost trade-off for different scenarios; left column: dots on lines correspond to number of clusters $n_c \in \{1, \ldots, 10\}$ (from left to right); numbers at dots indicate number of clusters used; right column: dots on lines correspond to arrival rates of $\{5, 10, \ldots, 50\}$ (from left to right) packets per simulation step; numbers at dots indicate arrival rate; number of service request classes: 2 (top) and 4 (bottom).

demands specified in the packet. The service task is executed by the helper and the packet is deleted (this is done within one simulation time step). If the helper's cluster number is different from the packet's cluster number, the helper might change its cluster number to the cluster number of the packet. The parameter p determines the probability for this to happen. If the helper does not change its cluster number to the cluster number of the arriving service request, then the service packet is rejected and sent to another node in the network. Note, that a service request is also rejected if the helper is already executing another service request at the same time step.

10.6 Empirical results II

For the test runs it is assumed that v_i is a three dimensional vector, i.e., $v_i = (v^1, v^2, 1 - v^1 - v^2) \in [0,1]^3, v^1 + v^2 \leq 1$. If a service task with data vector $(v^1, v^2, 1 - v^1 - v^2)$ is executed by a helper, then this helper node has to be reconfigured so that the fraction v^1 (respectively v^2 and $1 - v^1 - v^2$) of its slices is configured in mode 1 (respectively mode 2 and mode 3). If a helper is configured according to $(v^1, v^2, 1-v^1-v^2)$ and has to be reconfigured according to $(w^1, w^2, 1 - w^1 - w^2)$, then the costs of this reconfiguration are $\max(|w^1 - v^1|, |w^2 - v^2|, |(1 - w^2 - w^1) - (1 - v^2 - v^1)|)$.

All service requests in the test runs are from up to four different classes. Note, that a partitioning of the service requests that leads to small costs is not given in advance, as packets have random cluster identity when they are created. Within each class of service requests the individual requests are chosen as follows. Let $(c_j^1, c_j^2, 1 - c_j^1 - c_j^2)$ be the center of request class $j, j \in \{1, 2, 3, 4\}$. Then for a new service request with configuration

request vector $(v_i^1, v_i^2, 1 - v_i^1 - v_i^2)$ the value v_i^1 (respectively v_i^2) is chosen randomly from the interval $[c_j^1 - 0.1, c_j^1 + 0.1]$ (respectively $[c_j^2 - 0.1, c_j^2 + 0.1]$). Requests for which $1 - v_i^1 - v_i^2$ is not in the interval $[0.9 - c_j^1 - c_j^2, 1.1 - c_j^1 - c_j^2]$ are discarded. The center of request class 1 is $(1/3, 1/3, 1/3)$, the center of classes 2 (resp. 3 and 4) are $(2/3, 1/6, 1/6)$ (respectively $(1/6, 2/3, 1/6)$ and $(1/6, 1/6, 2/3)$). Unless stated otherwise in each simulation time step 50 packets with service requests were sent into the network. 50 helpers and 50 routers were used. The probability p that a helper changes its cluster was set to 0.01 (unless stated otherwise). Parameter β, which influences the update of the centroid estimation in a router was set to 0.1. If a centroid estimation has not changed by the last 100 packets that arrived at a router, the new centroid estimation is set to the corresponding configuration of the next arriving packet. Each result given in the following is averaged over 10 simulation runs, i.e., each pair of cost/drop values is an average calculated from 10 independent simulations. Each simulation was performed over 10000 steps. The reconfiguration costs given in the figures are the overall reconfiguration costs spent by the helpers when executing the service requests divided by the number of all service requests created during a simulation run.

10.6.1 Number of clusters

The drop rate of the service request packets and the reconfiguration costs are investigated for test runs with different number of service request classes. Using 2 request classes (classes 3 and 4), or 4 request classes (all classes) the number n_c of clusters used by the decentralized clustering algorithm has been varied, with $n_c \in \{1, 2, \ldots, 10\}$. The results are depicted in the left column of figure 10.7 when using TTL$\in \{1, 5, 10, 50\}$. There is a clear trade-off between the drop rate and the reconfiguration costs. When using a larger TTL value, the drop rate is reduced significantly. The reduction of the reconfiguration

costs for increasing number of clusters n_c depends strongly on the number of request classes. A sharp bend in the curves can be seen, as the algorithm utilizes its adaptability. When n_c is smaller than the number of request classes, then some helpers have to execute service requests of more than one class. This leads to relatively high reconfiguration costs as can be seen in figures 7(a) and 7(c), where packets from 2 or 4 service request classes were put into the network. For example, when 2 request classes and a TTL of 5 are used, the costs are reduced from 0.28 when using $n_c = 1$ to 0.07 when $n_c = 2$ is used. A further increase of n_c (larger than the number of service request classes) reduces the costs only slightly. The small reduction results from the fact that the service requests within one class vary slightly with respect to their resource requirements. Therefore, the reconfiguration costs of the helpers can be reduced slightly when the service requests of one class are split into several clusters. The disadvantage is that the packet drop rate increases with a higher number of clusters.

10.6.2 Work load

To compare simulations of the computing system with different work loads different arrival rates of $\{1, 5, 10, 15, \ldots, 50\}$ have been used. The number of clusters for the decentralized clustering algorithm was set to $n_c = 4$ and similar to the preceding subsection the number of service request classes was 2 and 4. The results are depicted in in the right column of figure 10.7.

Obviously, when using a very small (and unrealistic) value of TTL= 1 the drop rate of the packets is very high (always larger than 0.69). But this value is interesting because it shows the average fraction of packets that are not executed by a single helper. The small number of service requests that are executed produce only small reconfiguration costs. When using a higher TTL the drop rate goes down significantly, e.g., for TTL= 5 it is less than 0.3 in all cases. The increase in reconfiguration costs is relatively small in this case (less than 0.13 when using 4 service request classes and an arrival rate of 10). When the value of TTL is 50 nearly no packets are dropped in all the investigated scenarios. Also the reconfiguration costs are small in this case (always < 0.13).

10.6.3 Dynamically Adding and Removing Request Classes

To show the adaptability of the worker-helper system a dynamic scenario was investigated where the set of service request classes for which there are packets in the network changes. Service classes represented by packets in the network are added and deleted during a simulation run. Starting with only one request class (class 1) we successively add classes 2, 3, and 4 every 1000 simulation steps (i.e., packets of the corresponding classes are sent in the network). After that classes 2,3, and 4 were deleted successively every 1000 steps. In figure 10.8 the results are depicted for $n_c \in \{1, 2, 4\}$ clusters. When

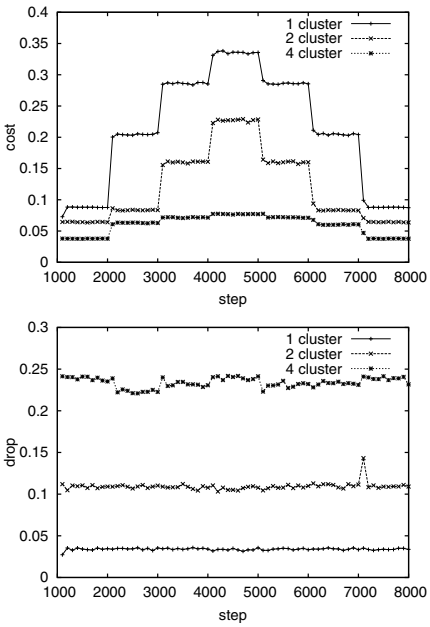


Fig. 10.8. Reconfiguration costs (top) and drop rate of packets (bottom) shown over a simulation run where service request classes are added successively (simulation steps 2000, 3000, and 4000) and then removed (simulation steps 5000, 6000, and 7000); initially (steps 0-999) only one service request class is used; results are given for $n_c \in \{1, 2, 4\}$.

only one cluster is used each additional request class increases the cost significantly, as the helper units have to be reconfigured for different service request classes very often. When using $n_c = 4$ the average reconfiguration costs are much smaller. The additional reconfiguration costs that occur after a new class has been added are due to the fact, that request classes have to be partitioned with fewer clusters (or are not partitioned at all). This leads to higher intra-class reconfiguration costs. These reconfiguration costs are much smaller than the inter-class reconfiguration costs. Also the drop rate is always very small for $n_c = 4$. This result clearly shows the fast adaptive behavior of the decentralized clustering component based on DPClust.

10.7 Conclusion

A self-organized worker-helper system as part of an abstract Organic Computing system (OC system) has been described. The OC computing system consists of normal worker components and helper components where the workers need some service from time to time in order to continue with their normal

work. The service is done by the helpers,, which have reconfigurable hardware to perform different service tasks. The speed of service for a certain task depends on the amount of resources configured for this task.

In the first part of this chapter different strategies have been studied that can be used by the helpers to decide whether they should accept a service task and how they should reconfigure themselves. These strategies are inspired by stimulus-threshold systems used in the literature to explain task allocation in social insects. Empirical results show, for example, that the system can adapt to dynamic situations with changing probabilities of service. A potential problem is that these system can show oscillating behavior.

In the second part of this chapter the problem of organizing the worker-helper system without global knowledge about the type of service requests and the set of available helper components is addressed. In order to obtain a decentralized mechanism and to make it suitable for the paradigm of OC a fully decentralized and dynamic clustering algorithm was combined with a self-organized task allocation system. The clustering algorithm is performed by the routers in the network and classifies the service requests packets that are sent through the network with respect to their hardware resource requirements (which determine the ideal configuration of the helpers when they execute the service task). Empirical results of simulations have shown that the proposed system has a strong adaptive behavior in static and dynamic scenarios and that decentralized clustering is able to reduce the reconfiguration costs significantly.

Acknowledgment

This work was supported by the German Research Foundation (DFG) through the project Organization and Control of Self-Organizing Systems in Technical Compounds within SPP 1183.

References

1. E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg: Fixed response thresholds and the regulation of division of labor in insect societies. Bulletin of Mathematical Biology, 60:753–807, 1998.
2. E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg: Quantitative study of the fixed threshold model for the regulation of division of labor in insect societies. Proc. Roy. Soc. London B 263, 1565-1569. 1996.
3. GI: Organic Computing / VDE, ITG, GI - Positionspapier. 2003, `http://www.gi-ev.de/fileadmin/redaktion/Presse/VDE-ITG-GI-Positionspapier_20Organic_20Computing.pdf`
4. N. Labroche, N. Monmarché, G. Venturini . A new clustering algorithm based on the chemical recognition system of ants. Proc. European Conf. on AI, IOS Press, 345–349 (2002).

5. N. Labroche, N. Monmarché, G. Venturini: AntClust: Ant Clustering and Web Usage Mining. Proc. of GECCO-2003, Springer, LNCS 2723, 25–36 (2003).
6. D. Merkle, M. Middendorf and A. Scheidler. Dynamic Decentralized Packet Clustering in Networks. In: Rothlauf, F. et al. (editor): Proc. of the 2nd European Workshop on Evolutionary Algorithms in Stochastic and Dynamic Environments, LNCS 3449, 574–583, 2005.
7. D. Merkle, M. Middendorf and A. Scheidler. Decentralized Packet Clustering in Router-based Networks. International Journal of Foundations of Computer Science, 16(2): 321-341, 2005.
8. D. Merkle, M. Middendorf and A. Scheidler. Self-Organized Task Allocation for Computing Systems with Reconfigurable Components. Proc. of the 9th International Workshop on Nature Inspired Distributed Computing (NIDISC'06), IEEE, 8 pp., 2006.
9. D. Merkle, M. Middendorf,A. Scheidler: Using Decentralized Clustering for Task Allocation in Networks with Reconfigurable Helper Units Proc. International Workshop on Self-Organizing Systems 2006 (IWSOS 2006), LNCS 4124, 137–147, 2006.
10. C. Müller-Schloer, C. von der Malsburg, R. P. Würtz: Organic Computing. Informatik Spektrum, 27(4):332-336, 2004.
11. H. Schmeck: Organic Computing – A New Vision for Distributed Embedded Systems. Proc. of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005), 201-203, 2005.
12. G. Theraulaz, E. Bonabeau, and J. Deneubourg: Response threshold reinforcement and division of labor in insect societies, Proc. Roy. Soc. London B 265, 327-332, 1998.

11

Concepts for Self-Adaptive and Self-Healing Networked Embedded Systems

Thilo Streichert, Christian Haubelt, Dirk Koch, Jürgen Teich

University of Erlangen-Nuremberg, Hardware-Software-Co-Design Chair, Am Weichselgarten 3, 91058 Erlangen, Germany.
streichert@cs.fau.de, haubelt@cs.fau.de, dirk.koch@cs.fau.de, teich@cs.fau.de

Summary. Networked embedded systems which operate in unattended areas with rare maintenance often make use of redundant resources for guaranteeing reliable service. In this paper, we will present novel concepts for dynamically partitioning and assigning functionality to software as well as hardware reconfigurable resources in a network. As a result, self-adaptive and self-healing systems emerge with a good tradeoff between redundancy and reliability. The proposed concepts are embedded in a three step approach, which 1.) reestablishes the functionality after a resource defect, 2.) optimizes the binding of the running tasks and 3.) creates new replicas of the tasks in the network. In this contribution, we will give an overview over all three parts, but focus on the second step. For this second step, called *dynamic hardware/software partitioning*, we will present algorithms, theoretical optimality bounds for workload distributions as well as experimental results.

Key words: self-adaptive, self-healing, fault tolerance, reliability, networks, embedded systems

11.1 Introduction

Networked embedded systems, such as *automotive networks*, *sensor networks* or networks in the field of *industrial control automation* are organized in a decentralized manner due to several aspects. In particular, they are *distributed* in order to monitor, analyze and control complex environments or installations over a large area and for a long period. They have to fulfill different criteria concerning *fault tolerance* and *reliability* as well as *flexibility* for application areas which demand functionality to vary over runtime. Moreover, networked embedded systems, such as sensor or automotive networks, are usually deployed in unattended areas where administration tasks should be kept at a minimum. For this reason, we will present novel concepts for *self-healing* and *self-adaptive systems* which have to fulfill the following constraints:

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_11,

Distributed computation: Due to fault tolerance requirements, the algorithms should run in a distributed manner at the computational nodes.

Local knowledge: Gathering of data to obtain global knowledge produces communication overhead and results in additional power consumption. Thus, it is preferable to apply algorithms which can take decisions only with local or restricted knowledge.

Up to now, several approaches have been presented for *self-healing* systems which handle *transient*, *permanent* or both kinds of faults simultaneously. In order to handle transient faults, concepts with *temporal redundancy* or *spatial* [8] have been proposed. By (re-)executing tasks on the same or different *computational network nodes*, faults can be detected and repaired. In the case of intermittent faults, N-modular redundancy [10] is able to detect and handle such faults by voting mechanisms. Also hybrid approaches of spatial and temporal redundancy exist and have been investigated throughout the last years. The other category of faults, called *permanent faults*, can be easily handled with three types of redundancy: a) N-modular redundancy, b) dynamic redundancy with hot and cold spares, and c) a hybrid approach of the previous two redundancy types.

While self-healing networked systems can handle defects and reestablish functionality if a link or node defect occurs, *self-adaptiveness* aims at optimizing the system's state to changing execution conditions or varying demands. This property ranges from integrating new tasks in a distributed system to *algorithmic self-adaptation* if, e.g., the quality of sensor values degrades. Especially this algorithmic integration of new tasks will be a matter of this chapter, while the algorithmic self-adaptation will be future work. Due to the separation of functionality and structure or network topology, respectively, we a) allow for exchanging tasks with functional alternatives, b) enable the replication of tasks in the case of node defects, and c) dynamically bind tasks to resources in the network.

Binding tasks to computational nodes has been investigated in many research fields. While the offline approach for hardware/software partitioning has been considered, e.g., in [9, 12], an interesting online approach has been presented by Vahid et al. Based on a platform consisting of reconfigurable hardware and a CPU [13], a profiler extracts critical code regions, decompiles them and synthesizes them to hardware. Achieving an average speedup of 2.6 [17] for different benchmarks, this approach to dynamic hardware/software partitioning shows the potential of dynamically assigning tasks to software or hardware resources. But unfortunately, no straightforward extension to reconfigurable networks exists.

In the context of wireless sensor networks, the so-called *facility location problem* has been investigated throughout the last years. It is defined as follows: Facilities may be placed on a set of locations in a network and each facility produces a cost. Each task making use of the facility produces a cost which depends on the number of hops from the task to the facility location.

The objective is to minimize the cost depending on the placement of facilities and the number of each kind of facility. Among many algorithms for solving this problem, one distributed solution exists [14]. In another solution for binding functionality to computational nodes [18], a spanning tree is constructed and all nodes propagate their abilities to the root node of the tree. This root node decides on the binding of the functionality. This distribution of functionality is *self-organizing* and *self-stabilizing* [4, 5].

The remainder of this contribution is organized as follows: In section 11.2, we first define our model for a network. Then, in section 11.3, we present our two step approach, which consists of a *fast repair* phase and an *optimization* phase. We focus on online hardware/software partitioning for networked hardware/software reconfigurable controller architectures and present two distributed algorithmic solutions, one for tasks without data dependencies and one for tasks with data dependencies.

11.2 System model

Our system model is specified by a *network model*, which separates functionality from the network architecture. The network architecture is represented by a *topology graph* and the entire functionality of the network system is specified with so-called *sensor-controller-actuator chains*. The third component of the network model is the set of mapping edges, which denote the binding possibilities of the tasks onto the network nodes. We assume that the topology graph consists of hardware/software reconfigurable nodes, e.g., FPGAs together with a CPU. These nodes may integrate tasks implemented in software or hardware at run-time. Additionally, these FPGA-based nodes contain dedicated analog hardware for driving sensors and actuators which leads to a certain heterogeneity in the network.

Exemplarily, figure 11.1 shows a network topology with four *computational nodes* $n_i \in N$, *sensors* $s_i \in S$, *actuators* $a_i \in A$, and communication links represented by the edges between the nodes n_i. The sensors and actuators are not connected to all nodes in the network, but only to some. Thus, the methodologies presented in section 11.3 must be able to bind functionality to a heterogeneous network structure. Similar to the network structure, the functionality is modeled by the *sensor-controller-actuator chain* graph and distinguishes between *sensor* t_i^s, *controller* t_i^c and *actuator tasks* t_i^a. These tasks might be replicated to achieve fault tolerance and are called *replicas* $t_i'^{\{s,c,a\}}$. A replica takes over the functionality of a task in case of a node defect. This technique is called passive replication [1]. While sensor tasks produce data which is processed by one or more controller tasks, actuator tasks consume data entities. In figure 11.1, such a sensor-controller-actuator chain is represented by the gray nodes, which are connected by edges representing data dependencies. Note that the edges between the tasks t_i and replicas t_i' are necessary for keeping the state of a replica consistent.

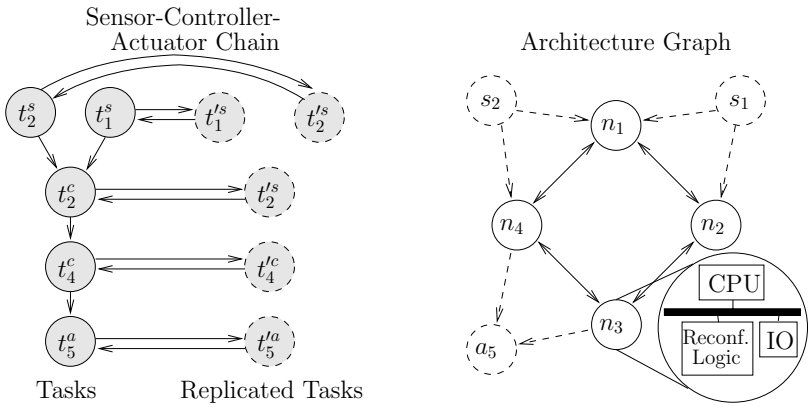


Fig. 11.1. Functionality is modeled by so-called *sensor-controller-actuator* chains. This functionality together with its replicas will be bound onto the nodes of the *topology graph*. The set of mapping edges (not shown here) denotes the binding possibilities.

Due to the heterogeneity caused by the sensors s_i and actuators a_i in the network topology, the binding of sensor tasks t_i^s and actuator tasks t_i^a is restricted. In particular, a sensor task t_i^s is only allowed to be bound onto a node $n_j \in N$ if s_i is a predecessor of n_j. In contrast, an actuator task t_i^a is only allowed to be bound onto a computational node n_j if a corresponding actuator node a_i is a successor of n_j. We assume that all controller tasks t_i^c may run on each computational node n_j. Considering figure 11.1, sensor task t_1^s may be bound onto n_1, but not onto n_2. Analogously, sensor task t_2^s can run on n_1 or n_4. For the controller tasks t_3^c, t_4^c, no binding restrictions exist. Thus, they are able to run on all network nodes (n_1, n_2, n_3, n_4). For binding the actuator task t_5^a, only the nodes n_3 and n_4 can be considered. The equivalent binding restrictions exist for the replicas $t_i'^{\{s,c,a\}}$.

The system model can be formally described as follows:

Definition 1 (Network Model). *The entire network model* $M(G^{tg}, G^{sca})$ *consists of a* topology graph G^{tg} *and a set of* sensor-controller-actuator chains G^{sca}.

Definition 2 (Topology Graph). *The topology graph* $G^{tg}(S^{tg}, C^{tg}, A^{tg}, E^{tg})$ *consists of sensor nodes* $s_i \in S^{tg}$, *computational nodes* $n_i \in N^{tg}$ *and actuator nodes* $a_i \in A^{tg}$. *The edges* $e_i^{tg} \in E^{tg} \subseteq S^{tg} \times N^{tg} \cup N^{tg} \times N^{tg} \cup N^{tg} \times A^{tg}$ *represent the connections between the nodes.*

The nodes of the topology graph can be defined as follows:

Definition 3 (Computational Node). *A computational node* $n_j \in N$ *has ports* $p_i \in P$ *with* $|P| = d_j + 1$ *and* d_j *is the degree of node* n_j. *While the*

ports $p_i : i = 1 \ldots d_j$ are dedicated for communication between sensor, computational or actuator nodes, the port p_0 is dedicated for node-internal communication.

For modeling the functionality, we define so-called *sensor-controller-actuator chains*.

Definition 4 (Sensor-Controller-Actuator Chain). *The sensor-controller-actuator chain $G^{sca}(T^s, T^c, T^a, E^{sca})$ consists of sensor $t_i^s \in T^s$, controller $t_i^c \in T^c$ and actuator tasks $t_i^a \in T^a$. The edges $e_i^{sca} \in E^{sca} \subseteq T^s \times T^c \cup T^c \times T^c \cup T^c \times T^a$ represent the data dependencies between the tasks.*

Edges and nodes can be annotated with different parameters, which need not be part of the model but can be application-specific.

Definition 5 (Binding Restrictions). *Binding restrictions are represented by binary variables $b_{i,j}$. Sensor tasks $t_i^s \in T^s$ may only be bound onto computational nodes $n_j \in N^{tg}$ iff it has a sensor $s_i \in S^{tg}$ as a predecessor:*

$$b_{i,j} = \begin{cases} 1 : \textit{if } (s_i, n_j) \in E^{tg} \\ 0 : \textit{else} \end{cases}$$

Actuator tasks $t_i^a \in T^a$ may only be bound onto computational nodes $n_j \in N^{tg}$ iff it has an actuator $a_i \in A^{tg}$ as a successor:

$$b_{i,j} = \begin{cases} 1 : \textit{if } (n_j, a_i) \in E^{tg} \\ 0 : \textit{else} \end{cases}$$

Computational tasks $t_i^c \in T^c$ may be bound onto all computational nodes $n_j \in N^{tg}$.

Note that sensor and actuator tasks may only run on certain nodes because they access dedicated interfaces. Also, additional binding restrictions may occur during the hardware/software partitioning process due to attributes annotated to edges or nodes in the graphs G^{tg} and G^{sca}.

Definition 6 (Binding). *The binding β is defined as a triple $(t_i^{\{s,c,a\}}, n_j, k)$, where k denotes the implementation style of a task $t_i^{\{s,c,a\}}$ which is executed by node n_j and can be either hardware (H) or software (S).*

11.3 Self-healing and self-adaptiveness in networks

In the context of self-adaptiveness, we aim at integrating new tasks into a running system. These tasks can be either platform-independent, i.e., executable on each node in the network or they can be executable only on a subset of all nodes in the network due to, e.g., I/O-interfaces or some inhomogeneity.

While *self-adaptiveness* treats changes in the entire functionality of a network in general, *self-healing* treats the problem of assuring that all currently running functionality keeps running correctly after topology changes in the network. All in all, the treatment of these changes have to happen in limited time. Thus, we propose a two-step strategy consisting of a so-called *fast repair phase* and an *optimization phase* (see figure 11.2). In case of a node or link defect, the fast repair phase activates replicated tasks and reestablishes the communication between the tasks. If new tasks arrive and need to be bound onto nodes in the network, the same strategy as used for placing replicas is applied. Then, communication between all tasks in the network is established if it does not exist yet. Afterwards, in the optimization phase, the problem of online hardware/software partitioning is solved with respect to different objectives. When the hardware/software partitioning phase finishes, replicas need to be placed in order to tolerate future defects.

In the following sections, we will focus on the online hardware/software partitioning part of the optimization phase. We will present two different algorithmic approaches for distributing the functionality between the nodes of the network: one which neglects data dependencies between tasks with the goal to balance the load between the nodes almost equally, and one which optimizes the task binding while respecting also inter-task communication. Moreover, we will show how newly arriving tasks and replicas will be bound onto network nodes. The explanations are based on the formal model introduced in the previous section.

11.3.1 Hardware/software balancing between reconfigurable network nodes

Our overall approach to online hardware/software partitioning tries to find a binding that optimizes the current binding β such that the work load on the resources is equally balanced. This goal has been chosen because of two main aspects:

- *Latency, Rate, Average Response Time* Beside the latency of a task or the execution rate in case of a periodic task, the average response time is an essential criterion for the quality of a schedule or a service. This average response time denotes the time interval from the arrival of a task until its completion.
- *Overhead* Due to task scheduling overhead occurs, which is caused by gathering task and system information, context switching as well as determination of a task schedule.

Therefore, we define the following three objectives for online hardware/software-partitioning:

1. *Load balance in the network:* With this objective, the load in the network is balanced between the nodes, whereby hardware and software tasks are

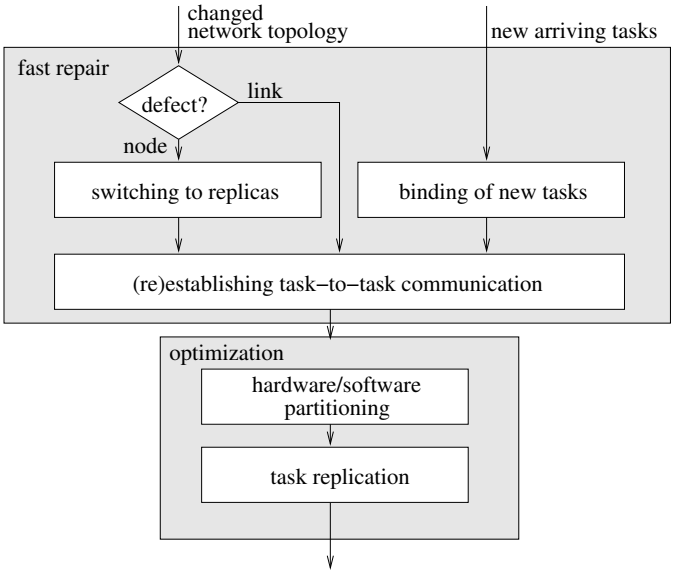


Fig. 11.2. In case of topology changes, the *fast repair* phase activates replicated tasks and communication between two tasks will be reestablished. If a new task arrives at one node in the network, the network decentrally tries to bind the task to one of its network nodes. The optimization phase optimizes the binding of tasks and creates new replicas.

treated separately. We try to minimize $\text{LBN} = \max(\max(w_i^S) - \min(w_i^S), \max(w_i^H) - \min(w_i^H))$ where w_i^S, w_i^H denote the software or hardware load of node n_i, respectively (see definitions 7 and 8). In order to minimize the risk of generating an overload on a node, this balance becomes important.

2. *Hardware/software load balance:* Here, we want to find a bi-partition such that the load is balanced between the hardware and software resources, i.e. we minimize $\text{HSB} = \left| \sum_{i=1}^{|N|} w_i^{\text{S}} - \sum_{i=1}^{|N|} w_i^{\text{H}} \right|$. With this strategy, a subsequent iterative *diffusion* [3] will then create most likely task assignments that enable a good *load reserve* on each active node which is important with respect to achieve fast repair times in case of unknown future node or link failures.
3. *Minimization of total load:* If this objective is not taken into account, solutions would be preferred which are optimal regarding objective 1 and 2, but cause unnecessary high computational load in the network. Therefore, we minimize $\text{MTL} = \sum_{i=1}^{|N|} (w_i^S + w_i^H)$

In order to minimize these three objectives simultaneously, we propose an online hardware/software partitioning strategy which is separated into a distributed load balancing phase and a bi-partitioning phase. While the load balancing step tries to find good solutions with respect to the first objective, the local bi-partitioning step searches for good solutions concerning the second and third objective.

The load of a task is defined in the objectives as follows:

Definition 7 (Software Load). *The software workload $w_i^S(t_j^c)$ on node n_i of task t_j^c implemented in software is the relation of its execution time to its period. This definition can be used for independent periodic and preemptable tasks. Buttazzo and Stankovic [2] proposed a load definition where the load is determined dynamically at runtime.*

Definition 8 (Hardware Load). *The hardware workload $w_i^H(t_j^c)$ on node n_i is defined as a fraction of required area and maximal available area, respectively logic elements for FPGA implementations.*

Definition 9 (Overall HW/SW Load). *The overall hardware/software load on a node n_i in the network is the sum of all loads of tasks bound onto this node, i.e.,* $w_i^S = \sum_{(t_j^c, n_i, S) \in \beta} w_i^S(t_j^c)$.

In this paper, we assume constant workload demands, i.e., $\forall i = 1...|N| : w^S(t_j^c) = w_i^S(t_j^c)$.

11.3.1.1 Load balancing

For the purpose of balancing the load of nodes, we propose a diffusion-based algorithm, which moves load entities along the edges in the network to other nodes. Characteristic to a diffusion-based algorithm, introduced first by Cybenko [3], is that iteratively, each node is allowed to move load of any size to each of its neighbors. Communication is only allowed along edges $e \in E$. The quality of such an algorithm may be measured in terms of the number of iterations required to achieve a balanced state and in terms of the total load moved over the edges of the graph. Diffusion algorithms have received a considerable amount of attention throughout the last couple of years, see, e.g., [6].

Definition 10. *A local iterative load balancing algorithm performs iterations on the nodes of $n \in N^{tg}$ determining load exchanges between adjacent computational nodes. On each computational node $n_i \in N^{tg}$, the following iteration is performed:*

$$\begin{aligned} y_e^{k-1} &= \alpha(w_i^{k-1} - w_j^{k-1}) \quad \forall e = \{n_i, n_j\} \in E \\ x_e^k &= x_e^{k-1} + y_e^{k-1} \quad \forall e = \{n_i, n_j\} \in E \\ w_i^k &= w_i^{k-1} - \sum_{e=\{n_i,n_j\} \in E} y_e^{k-1} \end{aligned} \tag{11.1}$$

In the above definition, we first calculate a real-valued load entity y_e^k in each iteration k which is sent via edge e. This amount of load is a fraction α of the load difference of two adjacent nodes connected by edge e. x_e^k is the amount of load sent via edge e until iteration k, and w_i^k is the load of node i after the k-th iteration. If arbitrary real-valued load portions may be sent at each iteration k, then it has been shown in [3] that the iteration converges to the average load $\bar{w}$. The number of iterations needed to obtain a certain error bound may be large and is in general not known a priori.

A slight modification of the above iteration scheme that works with changing values of α in each iteration k has shown that the convergence speed can be drastically improved to exactly $m-1$ iterations [6]. Simply choose $\alpha = \frac{1}{\lambda_k}$ in the k-th iteration of equation (11.1), where λ_k, $1 \leq k \leq m-1$ denotes an arbitrary numbering for the m non-zero eigenvalues of L. L is called the *Laplacian*-matrix of the network and defined as $L = D - B$ where D contains the node degrees as diagonal entries and B is the adjacency matrix of the network. Hence, $\alpha = \alpha_k = \frac{1}{\lambda_k}$, and in each iteration k, a node n_i adds a flow of $\frac{1}{\lambda_k}(w_i^{k-1} - w_j^{k-1})$ to the flow of edge $\{n_i, n_j\}$, choosing a different eigenvalue for each iteration. We obtain an equally balanced load distribution after exactly $m-1$ iterations, while the created flow is also l_2-optimal[1].

Note that after each topology change the eigenvalues of the Laplacian-matrix L have to be recalculated. This process is time consuming and may lead to instabilities of the numerical approximation algorithm. Due to this behavior, another diffusion scheme [15] called uniform diffusion replaces α with $\frac{1}{d_i+1}$ which is the reciprocal of the degree of node n_i increased by one.

To apply this diffusion algorithm in applications where we cannot migrate a real-valued part of a task from one node to another, an extension is needed. With this extension, we have to overcome two problems:

1. First of all, it is advisable not to split one task and distribute it to multiple nodes. This might produce a lot of data traffic in the network in addition to the inter-process communication.
2. Since the eigenvalue-based diffusion algorithm is an alternating iterative balancing scheme, it could occur that negative loads are assigned to computational nodes.

From now on, let $ycont_e^k$ be the real-valued continuous flow on edge e as determined by the continuous diffusion algorithm in iteration k. Also, let $ydisc_e^k$ be the discrete flow on edge e determined by the discrete diffusion algorithm. Analogously, all variables of the continuous diffusion algorithm are

[1] The l^2-norm is a vector norm defined by $\sqrt{\sum_{k=1}^{m-1} \sum_{e \in E} (y_e^k)^2}$

extended with the index *cont* and all variables of the discrete version are extended with *disc*. Then, in each iteration $k \in \{1, ..., m-1\}$ the discrete diffusion algorithm works as follows: In the first step of an iteration, the real-valued continuous flow $ycont_e^k$ is computed on all edges for all nodes. In the next step, each node tries to fulfill the resulting flow for its incident edges, sending or receiving tasks, respectively. Here, we encounter the optimization problem to choose number and size of tasks on each node in order to keep the optimality of the real-valued flow. This is an instance of the knapsack problem, which is known to be $\mathcal{NP}$-complete. Therefore, we randomly choose tasks to be sent via one edge as long as the discrete flow $ydisc_e^k$ does not exceed the continuous flow or no more tasks remain on the node:

$$ydisc_e^k \leq ycont_e^k + \Delta_e^{k-1} \qquad \text{with} \qquad \Delta_e^0 = 0 \tag{11.2}$$

In this equation, we already respect the error Δ_e^{k-1} made in the previous iteration by not fulfilling the optimality condition of the real-valued flow. The error $\Delta_e{}^k$ that occurred in the current iteration step is calculated as follows:

$$\Delta_e{}^k = ycont_e^k + \Delta_e^{k-1} - ydisc_e^k \quad \forall e = \{n_i, n_j\} \in E \tag{11.3}$$

In order to minimize the final error Δ_e^m, the error of the last iteration step is considered in a last additional iteration step, see equation (11.4). From here on, we start with a next iteration until the last iteration step $m-1$. After the last iteration step, the remaining error Δ_e^{m-1} is minimized in one additional adjustment step in which nodes exchange tasks according to the error after $m-1$ iterations:

$$\Delta_e^m = \Delta_e^{m-1} - y_e^{adj} \tag{11.4}$$

y_e^{adj} denotes the flow in this adjustment step.

Now, we can compare the behavior of our discrete diffusion algorithm with that of its continuous counterpart.

Theorem 1 *The discrete diffusion algorithm based on eigenvalues requires m steps.*

Proof. We introduced just one single adjustment step after the load balancing has completed. □

Theorem 2 *The overall congestion caused by the discrete diffusion algorithm is less than or equal to that caused by its continuous counterpart, which is known to be l_2-optimal.*

Proof. Let $xcont_e$ and $xdisc_e$ denote the whole load transmitted via edge e of the continuous and the discrete diffusion algorithm, respectively. Then we first show that

$$xcont_e \geq xdisc_e \tag{11.5}$$

no matter what network, initial load and edge e. With

$$xcont_e = \sum_{k=1}^{m-1} ycont_e^k \qquad \text{where} \qquad ycont_e^k \geq 0 \tag{11.6}$$

and

$$xdisc_e = \sum_{k=1}^{m-1} ydisc_e^k + y_e^{adj} \tag{11.7}$$

where y_e^{adj} is the flow via edge e in the last adjustment step which is always less than or equal to Δ_e^{m-1}, see equation 11.2:

$$\Delta_e^{m-1} \geq y_e^{adj} \tag{11.8}$$

Replacing $ycont_e^k$ in equation (11.6) with equation (11.3) leads to:

$$xcont_e = \sum_{k=1}^{m-1} \left(\Delta_e^k - \Delta_e^{k-1} + ydisc_e^k\right) \qquad \text{with} \qquad \Delta_e^0 = 0 \tag{11.9}$$

Inserting $xcont_e$ from equation (11.9) and $xdisc_e$ from equation (11.7) in equation (11.5) leads to:

$$\sum_{k=1}^{m-1} \left(\Delta_e^k - \Delta_e^{k-1} + ydisc_e^k\right) \geq \sum_{k=1}^{m-1} ydisc_e^k + y_e^{adj} \tag{11.10}$$

$$\Leftrightarrow \sum_{k=1}^{m-1} \left(\Delta_e^k - \Delta_e^{k-1}\right) \geq y_e^{adj} \tag{11.11}$$

$$\Leftrightarrow \Delta_e^{m-1} \geq y_e^{adj} \tag{11.12}$$

With respect to equation (11.8), we have proven that the overall edge congestion does not exceed the congestion of the continuous diffusion algorithm in the network for each edge e. □

Theorem 3 *The deviation of the discrete to the average load $\bar{w}$ on node n_i is bounded by the product of the maximal load S_{max} of a task $t^s \in T^s$ and the node degree $d_i = |P_i| - 1$:*

$$d_i \cdot S_{max} > |\bar{w} - w_i^m| \tag{11.13}$$

Proof. With

$$\bar{w} = w_i^0 - \sum_{e=1}^{d_i} \sum_{k=1}^{m-1} ycont_e^k \tag{11.14}$$

and

$$w_i^m = w_i^0 - \sum_{e=1}^{d_i} \left[\sum_{k=1}^{m-1} ydisc_e^k + y_e^{adj} \right] \tag{11.15}$$

we compute the average load $\bar{w}$ and the discrete load w_i^m after running the discrete diffusion algorithm for m iterations. In equations (11.14) and (11.15), we accumulate the flow on all edges of one node and add it to the load of the node n_i. We already mentioned the last adjustment step after the $m-1$ iterations. After this adjustment step, we require the final error Δ_e^m to be less than the maximal task size:

$$\Delta_e^m < S_{max} \tag{11.16}$$

Inserting equation (11.14) and (11.15) in (11.13) leads to:

$$d_i \cdot S_{max} > \left| w_i^0 - \sum_{e=1}^{d_i} \sum_{k=1}^{m-1} ycont_e^k - w_i^0 + \sum_{e=1}^{d_i} \left[\sum_{k=1}^{m-1} ydisc_e^k + y_e^{adj} \right] \right| \tag{11.17}$$

Replacing $ydisc_e^k$ with the help of equation (11.3), results in:

$$\begin{aligned} d_i \cdot S_{max} &> \left| -\sum_{e=1}^{d_i} \sum_{k=1}^{m-1} ycont_e^k + \sum_{e=1}^{d_i} \sum_{k=1}^{m-1} \left(ycont_e^k + \Delta_e^{k-1} - \Delta_e^k + y_e^{adj} \right) \right| \\ \Leftrightarrow d_i \cdot S_{max} &> \left| \sum_{e=1}^{d_i} \left[\sum_{k=1}^{m-1} \left(\Delta_e^{k-1} - \Delta_e^k \right) + y_e^{adj} \right] \right| \\ \Leftrightarrow d_i \cdot S_{max} &> \left| \sum_{e=1}^{d_i} -\Delta_e^{m-1} + y_e^{adj} \right| \end{aligned} \tag{11.18}$$

with equation (11.4), we obtain:

$$\Leftrightarrow d_i \cdot S_{max} > \left| \sum_{e=1}^{d_i} -\Delta_e^m \right| \tag{11.19}$$

According to equations (11.16) and (11.19), this assumption is correct. The maximal deviation between the continuous and the discrete load optimum is therefore smaller than the maximal size of a task times the degree of a node n_i, which is due to the nature of local iterative algorithms and thus, optimal in the discrete case. □

After having balanced the load between the nodes in the network, a local bi-partitioning phase on each single node $n_i \in N$ will determine the implementation style of each task.

11.3.1.2 Local bi-partitioning

The local bi-partitioning algorithm first determines the load ratio between a hardware and a software implementation for each new or arriving task $t_i \in T$, i.e., $w^H(t_i)/w^S(t_i)$. According to this ratio, the algorithm selects one task and implements it either in hardware or software. We then calculate the total software load and the total hardware load on each node. If the hardware load is less than the software load, the algorithm selects a task which will be implemented in hardware, and the other way round.

Due to these competing objectives, tasks with a ratio larger than one can be assigned to hardware and tasks with a ratio less than one are assigned to software. In the previous subsection, we mentioned that tasks which have to be sent via one edge are chosen randomly. From now on, we introduce two priority lists, one for software tasks and one for hardware tasks. In these lists, we collect all tasks in the reverse order of their assignment a hardware or software resource, respectively. Thus, e.g., the last task assigned to software is the first task to be diffused if the node has to send tasks via the network. Therefore, poorly partitioned tasks will have a higher mobility, which leads to an improvement concerning convergence behavior. Note that the hardware/software balance cannot be achieved by direct application of our discrete diffusion algorithm. The diffusion algorithm balances only the load between nodes, but as the load value of a task changes according to the assignment to hardware or software, it is not able to balance the hardware/software load.

11.3.1.3 Experimental evaluation

For a detailed evaluation of our approach to online hardware/software partitioning, we simulated our approach with different problem instances and scenarios. In total, nine different scenarios were generated each consisting of a sensor-controller-actuator-chain and a network topology. Out of these nine scenarios, three different scenarios were created with 40 tasks and 10 computational nodes. The next three scenarios had 80 tasks and 20 nodes and the last three scenarios had 200 tasks and 50 nodes. Our distributed approach started from an arbitrary initial binding of tasks to computational nodes. For each scenario, 10 initial bindings were determined such that in total 90 test cases were examined. Starting with an arbitrary binding of the tasks to the computational nodes of the network topology, the nodes try to improve the binding by executing the combined discrete diffusion and local bi-partitioning algorithm. Under the assumption that the diffusion algorithm runs synchronously on each node in the network and synchronously exchanges load chunks between the nodes in each iteration k, we calculate after each iteration the objectives (LBN, HSB, MTL) defined in section 11.3.1. After the load has been balanced between the nodes in the network, all nodes start the local bi-partitioning in parallel. For our evaluation, we let each node perform this bi-partitioning sequentially and calculate the objectives LBN, HSB and

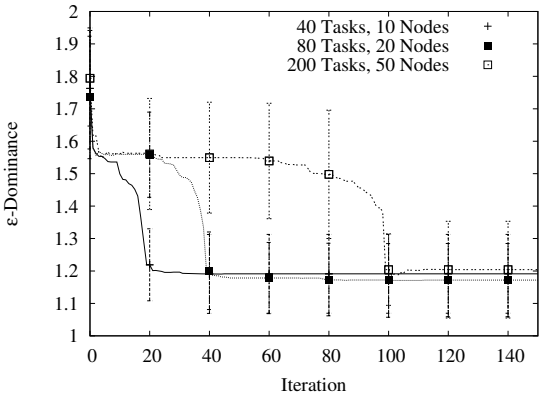


Fig. 11.3. ϵ-dominance and its standard deviation between a Pareto-optimal reference set R and the online partitioner over time.

MTL after each bi-partitioning. We compared the solutions $s_i \in S$ of each optimization run with a hardware/software partitioning algorithm based on Evolutionary Algorithms (EA) [7] that incorporates global knowledge. Our distributed algorithm tries to optimize the binding using only local knowledge. The EA-based approach determines a reference set of Pareto-optimal solutions R. For a comparison of the solutions $s \in S$, the ϵ-dominance $\epsilon(s)$ [11] between each $s \in S$ and R is calculated, which is defined as follows: A point r weakly ϵ-dominates a point s ($r \preceq_\epsilon s$) iff $r \preceq \epsilon \cdot s$. By scaling a point s by a factor ϵ, a point r is superior to point s. The ϵ-dominance, which is normalized between one and two indicates a better solution for low values. The ϵ-dominance and the standard deviation for the three different sizes of the problem instances are shown in figure 11.3. There, the first iteration steps lead to the highest improvement. The steps in the curves are caused by the alternating phases, i.e., the diffusion phase and the bi-partitioning phase.

11.3.2 Communication-aware hardware/software partitioning

With the hardware/software-partitioning algorithm of the previous section it is possible to balance the load between different network nodes. Unfortunately, this load distribution does not consider the communication traffic created on the links between the network nodes. Therefore, the minimization of computational load may lead to increased traffic in the network. Moreover, only one load definition can be used for the balancing of hardware tasks and software tasks. To solve this, we will present a generic task assignment protocol, which calculates *improvement values* and migrates tasks to neighbors leading to the highest overall improvement of the task binding. The following three improvement values are calculated for the determination of the overall improvement: a) *communication improvement*, that tries to cumulate functionality with data dependencies, b) *migration improvement*, which reduces the overhead caused

by task migration, and c) *partitioning improvement*, that tries to implement a task according to its favorite implementation style.

Communication improvement: The communication improvement $I_{i,j}^{com}$ is defined as the improvement for task t_i if it is migrated from node n_m over port p_j to a neighboring computational node n_j ($j \neq 0$):

$$I_{i,j}^{com} = \sum_{l=0}^{d_m} \sum_{k=1}^{|T|} T_{i,k} \cdot r_{k,l}$$

with

$$r_{k,l} = \begin{cases} -1 : \text{if traffic } T_{i,k} \text{ between } t_i \text{ and } t_k \text{ is routed via } p_l \ (l \neq j) \\ 1 : \text{if traffic } T_{i,k} \text{ between } t_i \text{ and } t_k \text{ is routed via } p_j \end{cases}$$

The outer sum over $d_i + 1$ terms considers not only the traffic over the external ports but also the node-internal.
Considering figure 11.4 as an example of a binding where the communication improvement $I_{1,3}^{com}$ for migrating task t_1 over port p_3 should be computed, we obtain: $I_{1,3}^{com} = -80 - 20 + 10 + 100$.
Afterwards, the communication improvement $I_{i,j}^{com}$ has to be normalized. For this normalization, the maximal absolute value of $I_{i,j}^{com}$ of all tasks t_i will be computed if migrated over a certain port p_j:

$$I_{max}^{com} = \max_{\forall t_i \text{ at } n_m, \forall p_j \in P} |I_{i,j}^{com}|$$

Migration improvement: For the determination of the migration improvement I_i^{mig}, the size M_i of the bit-stream and binary of task t_i^c which needs to be migrated is required. Then, the migration improvement for migrating task t_i^c over a port p_j is simply defined as:

$$I_i^{mig} = M_i$$

Note that this is not really an improvement of the task binding. It just avoids transferring huge data entities over the network's communication channels. Again the improvement I_i^{mig} needs to be normalized:

$$I_{max}^{mig} = \max_{\forall t_i^c \text{ at } c_m} |I_i^{mig}|$$

Partitioning improvement: Consideration of partitioning improvement $I_{i,j}^{par}$ is finally required for optimizing the implementation style (hardware/software) of a task t_i^c. For certain applications, e.g., video stream processing, it might be desirable to implement a task in hardware while alternatively, a state-machine might be efficiently executed in software. However, assuming that each task t_i^c has a favorite implementation style, a likelihood value $l_i \in \mathbb{R}$ with $0 \leq l_i \leq 1$ will be defined at design time.

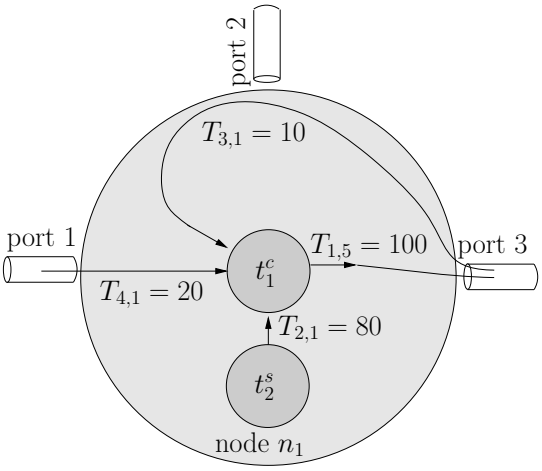


Fig. 11.4. Shown are two tasks t_1^c, t_2^s at a computational node c_1. The inter-task communication is denoted by directed edges to/from the ports or between t_1^c and t_2^s. Annotated to each edge is the traffic between two tasks.

The decision whether a task is better implemented in hardware or software can be taken based on resource utilization or a quality of service. The resulting improvement $I_{i,j}^{par}$ will be defined as:

$$I_{i,j}^{par} = l_i \cdot q_{i,j}$$

with

$$q_{i,j} = \begin{cases} 1 : \text{if } t_i^c \text{ was implemented in its non-favorite style and can be implemented in its favorite style after migration over } p_j \\ -1 : \text{if } t_i^c \text{ was implemented in its favorite style and can only be implemented in its non-favorite style after migration over } p_j \\ 0 : \text{else} \end{cases}$$

The resulting improvement $I_{i,j}$ for migrating a task t_i^c over port p_j to a neighboring computational node is then computed as:

$$I_{i,j} = \frac{I_{i,j}^{com}}{I_{max}^{com}} - \frac{I_i^{mig}}{I_{max}^{mig}} + I_{i,j}^{par}$$

As shown in figure 11.5, this improvement will be determined for all migratable tasks $t_i^c \in T_m^c \subseteq T$ and all ports p_j of node n_m. Some of the improvement values calculated for the migratable tasks, negative improvement values might be in the list and can impair the current binding. Therefore, two possibilities exist: a) remove all negative improvement values or b) allow

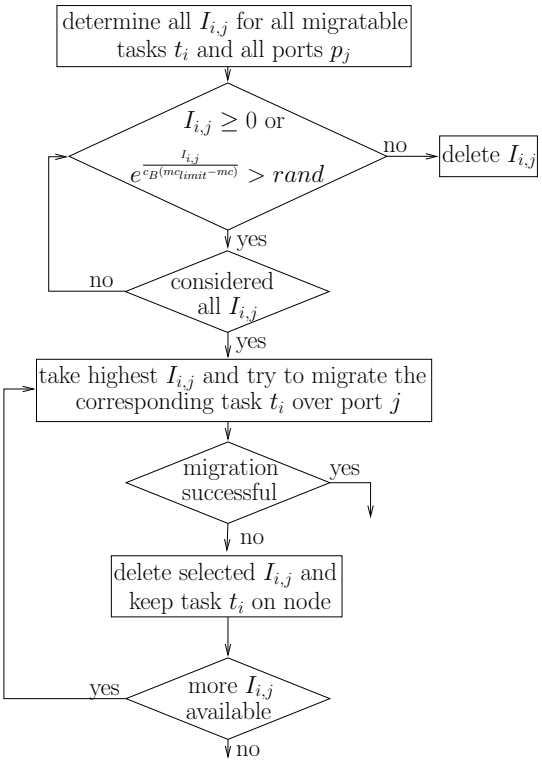


Fig. 11.5. The flow diagram shows the complete process of binding optimization that will be executed locally on each computational node $n_i \in N$ in the network.

for negative improvement values depending on the *migration counter* mc_i of task t_i^c. In the next step, the algorithm selects the task t_i^c with the highest improvement value $I_{i,j}$ and asks the neighboring computational node at port p_j if the task can be scheduled on the CPU or bound onto the reconfigurable hardware device, respectively (see figure 11.5). If enough resources are available for scheduling/placing the task it will be migrated and all improvement values $\forall p_j \in P : I_{i,j}$ will be deleted. Otherwise, only the improvement value $I_{i,j}$ for the considered port p_j and task t_i^c will be deleted. These two steps of selecting the task with the highest improvement value and trying to migrate it, is repeated locally until no improvement value $I_{i,j}$ remains. Note that the set of migratable tasks T_m contains only tasks with a migration counter below a given limit: $mc_i \leq mc_{limit}$. The counter mc_i is incremented after each migration of task t_i^c and reset after a node or link defect. With this constraint, the algorithm will terminate by preventing alternating behavior. All in all, our methodology runs asynchronously in the network, i.e., periodic migration rounds are not required. Since the routing needs to be fixed before calculating

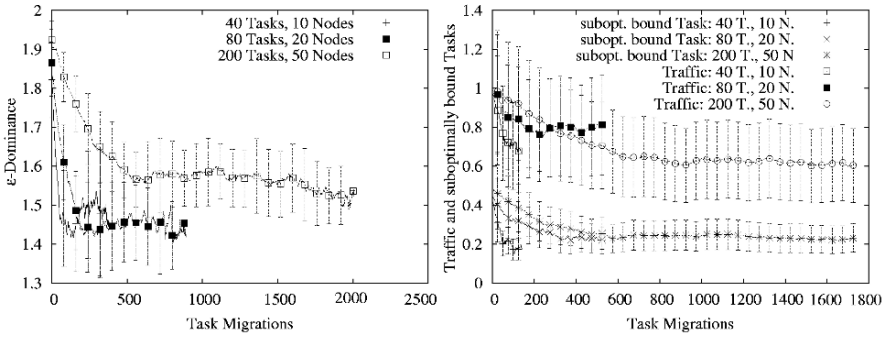


Fig. 11.6. a) ϵ-dominance and its standard deviation between a Pareto-optimal reference set R and the online partitioner over time (number of task migrations). b) Normalized traffic T and percentage of suboptimally bound tasks N over time (number of task migrations).

the improvement values of the tasks on a node, it is not possible to migrate tasks on different nodes simultaneously. Therefore, a token will be placed to an arbitrary node. If a node or link defect occurs, the node with the token will start with the calculation of improvement values and migrates a task to a neighboring node. Along with this migrated task a token will be transferred and the node which receives the token may start the calculation of improvement values. If the node will not migrate a task, the token will be passed to an arbitrary neighboring node. This strategy is derived from the class of hill climbing algorithms, where optimization runs are repeatedly started from arbitrary initial points. The algorithm stops after the token has been transferred a certain number of times.

11.3.2.1 Experimental evaluation

Setup and scenarios for the experimental evaluation of the communication-aware hardware/software partitioning algorithm are the same as before, but here, the edges in the problem graph have been annotated with demands. Starting again with an arbitrary binding of the tasks to the computational nodes of the network, the algorithm tries to improve the binding by migrating functionality between the hardware and software resources in the network. After each migration step, we determine the overall traffic T in the network and the fraction of tasks which are executed in their non-favorite implementation style F:

$$T = \sum_{i=1}^{|E_{\text{sca}}|} \sum_{j=1}^{|E_{\text{tg}}|} T_{\text{i,j}} \cdot b_{i,j}^{\text{traffic}}, \tag{11.20}$$

$$F = \frac{\sum_{i=1}^{|T^{\{sca\}}|} f_i}{|T^{\{sca\}}|} \tag{11.21}$$

with

$$f_i = \begin{cases} 1: & \text{if } t_i^{\{s,c,a\}} \text{ is implemented in its non-favorite style} \\ 0: & \text{else} \end{cases}$$

Again, we compared the solutions $s_i = (T, F), s_i \in S$ of each optimization run with a reference set R. The ϵ-dominance and the standard deviation for the three different problem sizes are shown in figure 11.6a). After each task migration the quality of the solution has been evaluated and it can be seen that the algorithm drastically improves the initial binding of the tasks, but it can also run into local minima. In figure 11.6b) the two normalized objectives are shown and it can be seen that, starting from an initial binding, the algorithm reduces the traffic by at least 20% and the number of suboptimally bound tasks by 50%.

11.4 Conclusions

In this contribution, a framework for self-healing and self-adaptive networks of hardware/software reconfigurable nodes has been presented. This framework consists of a *fast repair* and an *optimization phase*. While the *fast repair phase* may apply known mechanisms for activating tasks and reestablishing inter-task communication, the *optimization phase* is of central interest here. For this purpose, we presented algorithms and theoretical as well as empirical results for different task models (with and without data dependencies) on how to optimize the binding of active tasks to hardware/software reconfigurable resources in a network. As a proof of concept, we built up a real demonstrator [16] running the proposed methodology.

All in all, we expect that the presented approach is a promising step towards self-healing and self-adaptive networked embedded systems.

References

1. N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. *The primary-backup approach*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993.
2. G. C. Buttazzo and J. Stankovic. Adding Robustness in Dynamic Preemptive Scheduling. In *Responsive Computer Systems*, 1995.

3. G. Cybenko. Dynamic Load Balancing for Distributed Memory Multiprocessors. *Journal of Parallel and Distributed Computing*, 7:279–301, Oct. 1989.
4. E. W. Dijkstra. Self-stabilizing Systems in Spite of Distributed Control. *Communications of the ACM*, 17(11):643–644, Nov. 1974.
5. S. Dolev. *Self-stabilization*. MIT Press, Cambridge, MA, USA, 2000.
6. R. Elsässer, A. Frommer, B. Monien, and R. Preis. Optimal and Alternating-Direction Loadbalancing Schemes. In *Proc. of Euro-Par 99, Parallel Processing*, pages 280–290, 1999.
7. C. Haubelt. *Automatic Model-Based Design Space Exploration for Embedded Systems – A System Level Approach*. PhD thesis, University of Erlangen-Nuremberg, Germany, July 2005.
8. V. Izosimov, P. Pop, P. Eles, and Z. Peng. Design Optimization of Time- and Cost-Constrained Fault-Tolerant Distributed Embedded Systems. In *Proceedings of Design, Automation and Test in Europe*, Munich, Germany, Mar. 2005.
9. V. Kianzad and S. S. Bhattacharyya. CHARMED: A Multi-Objective Co-Synthesis Framework for Multi-Mode Embedded Systems. In *Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)*, pages 28–40, Galveston, U.S.A., Sept. 2004.
10. P. K. Lala. Self-Checking and Fault-Tolerant Digital Design. Technical report, San Francisco, 2001.
11. M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multi-objective optimization. *Evolutionary Computation*, 10(3):263–282, 2002.
12. M. López-Vallejo and J. C. López. On the Hardware-Software Partitioning Problem: System Modeling and Partitioning Techniques. *ACM Transactions on Design Automation of Electronic Systems*, 8(3):269–297, July 2003.
13. R. Lysecky and F. Vahid. A configurable logic architecture for dynamic hardware/software partitioning. In *Proceedings of the conference on Design, automation and test in Europe*, pages 480–485. IEEE Computer Society, 2004.
14. T. Moscibroda and R. Wattenhofer. Facility Location: Distributed Approximation. In *24th ACM Symposium on the Principles of Distributed Computing (PODC), Las Vegas, Nevada, USA*, July 2005.
15. Y. Rabani, A. Sinclair, and R. Wanka. Local Divergence of Markov Chains and the Analysis of Iterative Load-Balancing Schemes. In *Symp. on Foundations of Computer Science FOCS*, 1998.
16. *ReCoNets-Demonstrator*, 2006. www.reconets.de.
17. G. Sitt, R. Lysecky, and F. Vahid. Dynamic Hardware/Software Partitioning: A First Approach. In *Proceedings of Design Automation Conference 2003*, Anaheim, California, Germany, June 2003.
18. T. Weis, H. Parzyjegla, M. A. Jaeger, and G. Mühl. Self-Organizing and Self-Stabilizing Role Assignment in Sensor/Actuator Networks. In *The 8th International Symposium on Distributed Objects and Applications (DOA 2006)*, pages 1807–1824, Montpellier, France, Oct. 2006.

12

An Artificial Hormone System for Self-Organizing Real-Time Task Allocation in Organic Middleware

Uwe Brinkschulte, Mathias Pacher, and Alexander von Renteln

University of Karlsruhe, Bldg. 40.28, Engler-Bunte-Ring 8, 76131 Karlsruhe, Germany.
brinks@ira.uka.de, pacher@ira.uka.de, renteln@ira.uka.de

Summary. This article presents an artificial hormone system for a completely decentralized realization of self-organizing task allocation. We show that tight upper bounds for the real-time behavior of self-configuration, self-optimization and self-healing can be given. We also calculate the communication load produced by the hormone system and find it acceptable.

Key words: Decentralized control loops , real-time task allocation, task clustering, hormone simulator

12.1 Introduction

Today's computational systems are growing increasingly complex. They are built from large numbers of heterogeneous processing elements with highly dynamic interaction. Middleware is a common layer in such distributed systems, which manages the cooperation of tasks on the processing elements and hides the distribution from the application. It is responsible for seamless task interaction on distributed hardware. Like shown in figure 12.1, all tasks are interconnected by the middleware layer and are able to operate beyond processing element boundaries as if residing on a single hardware platform. To handle the complexity of today's and even more tomorrow's distributed systems, self-organization techniques are necessary. These systems should be able to find a suitable initial configuration by itself, to adapt or optimize itself to changing environmental and internal conditions, to heal itself in case of system failures or to protect itself against attacks. These so-called self-x features are essential for the idea of Organic Computing. Middleware is well-suited to realize such self-x features. By autonomously choosing an initial task allocation, which means finding the best initial processing element for each task, middleware can configure the distributed system. By changing the task allocation, middleware can optimize the system in case of changing environmental

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_12,

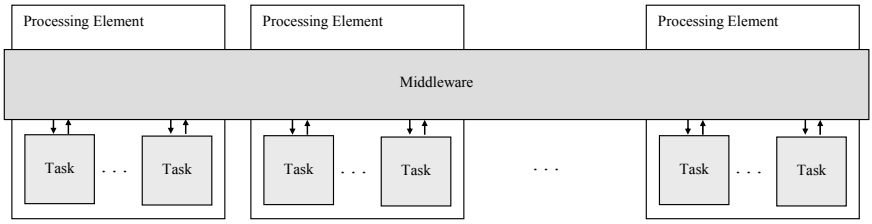


Fig. 12.1. Middleware in a distributed system.

conditions and heal it in case of processing element or task failures. Especially for self-healing, it is important that these organic features are decentralized to avoid single points of failure. This work presents an artificial hormone system for task allocation to heterogeneous processing elements. The proposed approach has the following properties:

- It is **completely decentralized**. There are no central decision making instances to determine the task allocation. Each processing element (PE) in the heterogeneous distributed system decides which tasks to take on the basis of simple local rules and information received from other processing elements.
- It is **self-organizing**. There is no external organization instance which influences the task allocation. This is done by the interaction of the PEs only.
- It is **self-configuring**. The presented approach determines an initial task allocation, which takes into account the capabilities (e.g. computational power, memory, etc.) and the state (e.g. operation temperature, energy level, etc.) of the heterogeneous PEs.
  The artificial hormone system is also able to respect related tasks (which often have a high communication rate) in order to cluster them close together, thus forming "organs".
- It is **self-optimizing**. The task allocation autonomously adapts to changing environmental conditions and states of the PEs (e.g. decreasing energy level, increasing temperature) during operation. Self-optimization also includes the assignment of newly arriving tasks to PEs.
- It is **self-healing**. Due to the lack of central instances and due to the capability of self-optimization the presented approach automatically compensates the effects of failing tasks or PEs by reordering the task allocation.
- It is **real-time** capable. There are tight upper time bounds for self-configuration and self-optimization. This bounds are partially valid for self-healing, too.
- It produces **limited communication overhead**, which is reasonable for embedded applications.

The term "artificial hormone system" was chosen because our approach was highly inspired by the hormone system of higher animals. There are several comparable properties between the hormone system in biology and our technical system:

- In biology, chemical signals called messengers or hormones are unspecifically spread to certain regions of the body or the whole body to cause some effects. The messengers (or hormones) of our artificial hormone system are also not addressed to a specific processing element (PE); rather they are spread in the neighborhood of a processing element or over the whole processor grid.
- The reaction of a cell to a hormone depends on the cell itself. In the same way, the reaction of a PE to a messenger in our system depends only on the specification of the PE itself (see properties mentioned above).
- A PE is able to react to a received messenger in different ways: It starts, stops, continues or quits the execution of a task. In reaction to this, the PE itself is also able to spread messengers over the system establishing a closed control loop, which stabilizes the system. Such loops can also be found in nature: the hormones T3 and T4 of the thyroid implement a closed loop controlling the body temperature.
- Like in the biological hormone system, these closed loops are completely decentralized. As for cells, removing PEs from the loop does not harm the system as long as there are enough PEs left to execute tasks and send or receive messengers.
- The hormones of higher animals are reduced by their metabolism, so they are not effective after some time (unless new ones are produced). In our implementation of the artificial hormone system, the effectiveness of the messengers is bounded by time stamps. If not renewed the messengers of our system expire, too.

It has to be stated that our "artificial hormone system" is not a copy of the biological hormone system, but it has been inspired by nature and its strategies. In biology, hormones are chemical objects transmitted via chemical processes and reactions. In our approach, the messengers are bits and bytes transferred via communication links. However, the effects and principles are similar. This is why we have called our messengers hormones as well.

In the following we will present our approach in detail and we will discuss and prove the enumerated properties.

12.2 An artificial hormone system for a decentralized realization of the self-x-properties

For task allocation, three types of hormones are used:

Eager value: This hormone determines how well a PE can execute a task. The higher the hormonal value the better the task is suited for the PE.

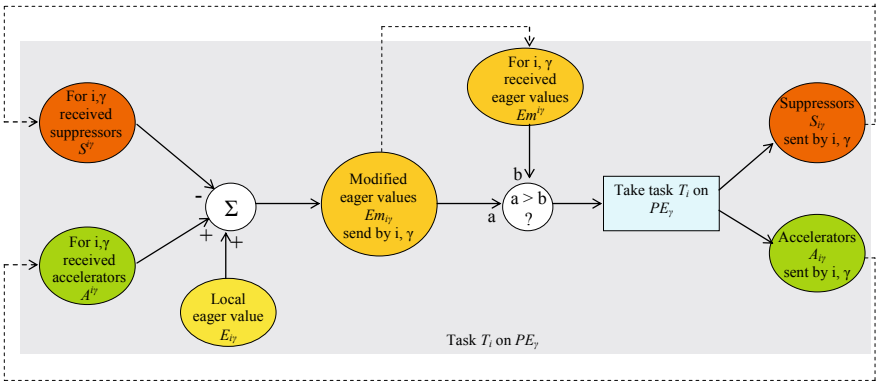


Fig. 12.2. Hormone-based control loop.

Suppressor: A suppressor represses the execution of a task on a PE. Suppressors are subtracted from eager values. They can be used to limit task execution and to indicate a degrading PE state.

Accelerator: An accelerator favors the execution of a task on a PE. Accelerators are added to eager values. They can be used to cluster related or cooperating tasks in the neighborhood (thus forming organs) or to indicate an improved PE state.

Figure 12.2 sketches the basic control loop used to assign a task T_i to a processing element. The notation scheme is as follows: $H^{i\gamma}$ means a hormone for task T_i executed on PE_γ and $H_{i\gamma}$ means a hormone from task T_i executed on PE_γ. Latin letters are task indices and Greek letters are processing element indices. This closed control loop is executed for every task on every processing element. Based on the level of the three hormone types it determines if a task T_i is executed on a processing element PE_γ or not. The local static eager value $E_{i\gamma}$ indicates how well task T_i executes on PE_γ. From this value, all suppressors $S^{i\gamma}$ received for task T_i on PE_γ are subtracted and all accelerators $A^{i\gamma}$ received for task T_i on PE_γ are added. The result of this calculation is a modified eager value $Em_{i\gamma}$ for task T_i on PE_γ. The modified eager value is sent to all other PEs in the system and compared to the modified eager values $Em^{i\gamma}$ received from all other PEs for this task. If $Em_{i\gamma}$ is greater than all received eager values $Em^{i\gamma}$, task T_i will be taken by PE_γ (in case of equality a second criterion, e.g. the position of a PE in the grid, is used to get an unambiguous decision). Now, task T_i on PE_γ sends suppressors $S_{i\gamma}$ and accelerators $A_{i\gamma}$ to the others. This procedure is repeated periodically.

At this point we emphasize that the initial strength of the hormone values is set by the applicants wanting to influence task allocation. The organic middleware evaluates the hormones to allocate the different tasks, but it does not set their initial strength.

12.2.1 Notation

Now we will define some basic indices and sets, which will be used frequently in the following sections. To allow an easy distinction, we use Latin lower case letters for task indices and Greek lower case letters for processing element indices (like already done in figure 12.2). Accordingly, we use upper case Latin letters for task sets and upper case Greek letters for sets of processing elements.
Let

Ω be the set of all processing elements in the system.
ω be the number of all processing elements in the system.

$$\omega = |\Omega|$$

$I\Omega$ be the set of indices of all processing elements.

$$I\Omega := \{1, ..., \omega\}$$

Thus, we obtain the set of all processing elements as

$$\Omega = \{PE_1, ..., PE_\omega\} = \{PE_\gamma \mid \gamma \in I\Omega\}.$$

Φ_γ be the set of processing elements which are neighbored to processing element PE_γ. Notice that this relation is reflexive.
Neighbored processing elements are able to communicate directly (hop count=0 or 1).

$$\Phi_\gamma := \{PE_\delta \mid \delta \in I\Omega \text{ and } PE_\delta \text{ neighbored to } PE_\gamma\}$$

φ_γ be the number of processing elements neighbored to PE_γ.

$$\varphi_\gamma := |\Phi_\gamma|$$

M be the set of all tasks in the system.
m be the number of all tasks in the system.

$$m := |M|$$

IM be the set of indices of all tasks.

$$IM := \{1, ..., m\}$$

Thus, we obtain the set of all tasks in the system as

$$M = \{T_1, ..., T_m\} = \{T_i \mid i \in IM\}.$$

V_i be the set of all tasks related to task T_i. Related tasks work on common problems and therefore have to cooperate closely.

$$V_i := \{T_j \mid j \in IM \text{ and } T_j \text{ related to } T_i\}$$

$\mathbf{v_i}$ be the number of all tasks related to task T_i.

$$v_i := |V_i|$$

$\mathbf{E}_\gamma$ be the set of tasks executed on processing element PE_γ.

$$E_\gamma := \{T_j \mid T_j \in M \text{ and } T_j \text{executed by } PE_\gamma\}$$

$\mathbf{e}_\gamma$ be the number of all tasks executed on PE_γ.

$$e_\gamma := |E_\gamma|$$

In the following sections, we describe the hormones in more detail. Several kinds of eager values, suppressors and accelerators have to be distinguished. Therefore, we extend the notation from figure 12.2 to specify the hormones:

$H_{i\gamma}^{j\delta}$: Hormone from task T_i running on PE_γ to be sent to task T_j running on PE_δ.

Hormones can be also sent to several tasks or PEs simultaneously. In that case, indices are replaced by the associated sets, e.g.:

$H_{i\gamma}^{M\Omega}$: Hormone from task T_i executed on PE_γ to be sent to all tasks on each processing element.

12.2.2 Different kinds of hormones

Using the notation introduced above we now describe the used hormones and their function in detail and start by explaining the eager values:

Local eager value $E_{i\gamma}$: This value states the initial suitability of PE_γ for task T_i. It assures that task allocation is adapted to the capabilities of the PEs.

Modified eager value $E_{i\gamma}^{i\Omega}$: This value is calculated by adding the received accelerators for task T_i on PE_γ and subtracting the received suppressors for task T_i on PE_γ from the local eager value $E_{i\gamma}$. It is sent to task T_i on all other PEs.

We used the following suppressors for the artificial hormone system:

Acquisition suppressor $Sa_{i\gamma}^{i\Omega}$: This suppressor is sent to task T_i on all other PEs in the system, as soon as PE_γ has taken task T_i. Therefore, this suppressor determines how often task T_i will be allocated in the overall system. A very strong acquisition suppressor enforces that task T_i is taken only once, while a weaker suppressor enables multiple allocation of this task.

Load suppressor $Sl_{i\gamma}^{M\gamma}$: This suppressor is sent only locally to that PE_γ which has taken task T_i. It affects not only task T_i, but all tasks on this PE. Thereby it determines how many tasks can be taken by a PE. A very strong load suppressor enforces, that a PE can take only one task, while a weaker one allows multiple tasks to be allocated on this PE.

Monitoring suppressor $Sm_{M\gamma}^{M\gamma}$: This suppressor is sent locally to a PE by local monitoring and affects all tasks on this PE. Thereby, the common state of a PE influences task allocation. E.g., the lower the energy level or the higher the temperature of a PE, the stronger this suppressor becomes.

We also used different kinds of accelerators for the artificial hormone system:

Organ accelerator $Ao_{i\gamma}^{V_i\Phi_\gamma}$: This accelerator is sent to all tasks V_i related to task T_i on the PEs Φ_γ neighbored to PE_γ, if PE_γ has taken task T_i. Thereby, this accelerator attracts tasks related to task T_i to settle on the same or neighbored PEs. The stronger the accelerator the stronger is the attraction. The basic idea behind this is that related tasks work on common problems and have to communicate frequently, making short communication distances useful. Related tasks form a kind of virtual organ, which works on a bigger problem.

Stay accelerator $As_{i\gamma}^{i\gamma}$: As soon as PE_γ has taken task T_i, this assignment is initially fixed. This leads to stable task allocation in the context of self-configuration. But to allow self-optimization, the possibility of changes in task allocation is necessary. Therefore, a task assigned to a PE can offer itself periodically for reallocation. To achieve this, the task suspends the transmission of its acquisition suppressor $Sa_{i\gamma}^{i\Omega}$ and starts sending its modified eager value $E_{i\gamma}^{i\Omega}$ again. This enables other PEs to take this task, if they are now more suitable. Such a task migration introduces costs expressed by the stay accelerator by means of favoring the stay of task T_i on PE_γ. It is sent from task T_i on PE_γ to itself (i, γ). The stronger the stay accelerator, the better another PE must be suited for task T_i to be able to take it from PE_γ.

Monitoring accelerator $Am_{M\gamma}^{M\gamma}$: This accelerator is sent locally to a PE by local monitoring and affects all tasks on the PE. It is the opponent of the monitoring suppressor. Therefore, the local monitoring can strengthen a PE if it is currently very powerful, e.g. due to a high energy level (solar cell in plain sun).

The described approach is completely decentralized, each PE is responsible for its own tasks, the communication to other PEs is realized by a unified hormone concept. Furthermore, it realizes the described self-x properties:

- The approach is **self-organizing**, because no external influence controls task allocation.
- It is **self-configuring**, as an initial task allocation is found by exchanging hormones. The self-configuration is finished as soon as all modified eager values become zero meaning no more tasks want to be taken. This is done by sending suppressors. Of course, the suppressors have to be chosen strong enough to inhibit an infinite task assignment (the suppressors must be stronger then the accelerators), otherwise the system would become instable.

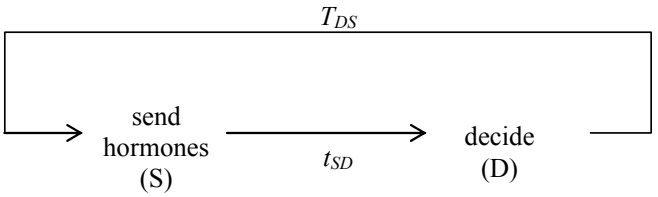


Fig. 12.3. Hormone cycle.

- The **self-optimization** is done by offering tasks. The point of time for such an offer is determined by the task or by the PE itself. It can be done periodically or at a point in time when the task or the PE is idle. Furthermore, an offered task continues its operation on the old PE as long as it is not taken by a new PE.
- The approach is **self-healing**. In case of a task or PE failure all related hormones are no longer sent, especially the acquisition suppressors. This initiates automatic reassignment of the task to the same PE (if it is still active) or another PE. The only additional requirement is a hormone $H_{i\gamma}^{j\delta}$ sent from task T_i on PE_γ to task T_j on PE_δ with an expiration time. If task T_j on PE_δ receives no new hormone value within this expiration time, the old value is discarded. This enables detection of missing hormones after the expiration time.

A detailed discussion of the real-time behavior, especially of upper time bounds for self-configuration, self-optimization, and self-healing can be found in the following sections. The communication overhead introduced will be analyzed there, too.

12.3 Dynamics of the artificial hormone system

In this section, the dynamics of the artificial hormone system and the conditions and rules for its correct operation will be presented. Figure 12.3 shows the cyclic sequence of sending out hormones and deciding on task allocation. The sequence starts with "send hormones" (S) to create the knowledge base for the first decision. At least the eager values need to be available. At time t_{SD} after sending the hormones, a decision (D) to allocate tasks is made based on the received hormones. This process is repeated after a time of t_{DS}.

12.3.1 Dynamics of task allocation

Let PE_γ be a processing element willing to run a task T_i. We need to distinguish three cases:

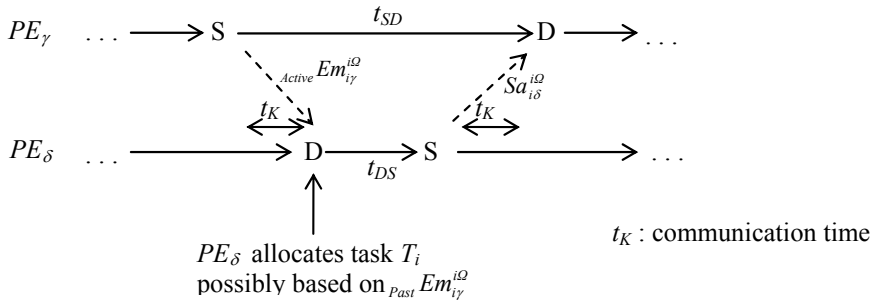


Fig. 12.4. Worst-case timing scenario of the hormone exchange for task allocation

Case 1: All eager values $Em_{i\gamma}^{i\Omega}$ of all processing elements $PE_\gamma \in \Omega$ for task T_i are constant and spread over the entire system. Thus, the system is in a steady state and all PEs make their decisions based on up-to-date and constant values. Then, PE_γ can allocate a task if it has the highest eager value or, in the case of equal eager values, a higher priority.

Case 2: The eager value $Em_{i\gamma}^{i\Omega}$ of processing element PE_γ for task T_i declines (e.g. by suppressor influence), i.e. $_{Active}Em_{i\gamma}^{i\Omega} < {}_{Past}Em_{i\gamma}^{i\Omega}$. In this case PE_γ may allocate the task T_i if the declined eager value $_{Active}Em_{i\gamma}^{i\Omega}$ is still sufficient. All the other PEs will not allocate the task, as they know either $_{Active}Em_{i\gamma}^{i\Omega}$ or $_{Past}Em_{i\gamma}^{i\Omega}$, and PE_γ wins with both values.

Case 3: The eager value $Em_{i\gamma}^{i\Omega}$ of the processing element PE_γ for task T_i increases (e.g. by accelerator influence), i.e. $_{Active}Em_{i\gamma}^{i\Omega} > {}_{Past}Em_{i\gamma}^{i\Omega}$. This case is critical if PE_γ becomes the winner by the increased eager value $_{Active}Em_{i\gamma}^{i\Omega}$, because other PEs might not yet know this increased eager value and therefore decide wrongly. Thus, PE_γ may only allocate the task T_i after the new eager value $_{Active}Em_{i\gamma}^{i\Omega}$ has successfully been submitted to all PEs and until PE_γ itself has received a possible acquisition suppressor $Sa_{i\delta}^{i\Omega}$ from another PE_δ ($\gamma \neq \delta$), which allocated the task T_i based on the old, lower eager value $_{Past}Em_{i\gamma}^{i\Omega}$.

But how long is the waiting time for PE_γ? Figure 12.4 shows the worst-case scenario, in which PE_δ allocated the task T_i just before the new eager value $_{Active}Em_{i\gamma}^{i\Omega}$ from PE_γ has been received. PE_γ may not come to a decision until it has received the possibly incoming suppressor $Sa_{i\delta}^{i\Omega}$ from PE_δ. The communication time t_K needed by a hormone to spread the whole system is very important. It is possible to establish the following rule for task allocation for increasing eager values as well as conditions for the times t_{DS} and t_{SD}.

Rule: If a processing element PE_γ is able to allocate a task T_i only by the increased eager value $_{Active}Em_{i\gamma}^{i\Omega}$ then it may not decide before the next communication cycle to allow the new eager value $_{Active}Em_{i\gamma}^{i\Omega}$ to

spread and to wait for potentially incoming suppressors. This is true if (follows directly from figure 12.4):

$$t_{SD} \geq t_{DS} + 2t_K$$

Thus, the cycle time results in:

$$t_C = t_{SD} + t_{DS}$$

Of course, the cycle time should be kept at a minimum, therefore
1) t_{DS} should be as small as possible, ideally 0.
2) $t_{SD} \geq t_{DS} + 2t_K$, ideally with $t_{DS} = 0$: $t_{SD} \geq 2t_K$

Conclusion: For the allocation of a task T_i by a processing element PE_γ the following cases can be distinguished:

1) ${}_{Past}Em_{i\gamma}^{i\Omega} = {}_{Active}Em_{i\gamma}^{i\Omega}$: The task can be allocated, if ${}_{Past}Em_{i\gamma}^{i\Omega} = {}_{Active}Em_{i\gamma}^{i\Omega}$ qualifies the processing element.
2) ${}_{Past}Em_{i\gamma}^{i\Omega} > {}_{Active}Em_{i\gamma}^{i\Omega}$: The task can be allocated, if ${}_{Active}Em_{i\gamma}^{i\Omega}$ qualifies the processing element.
3) ${}_{Past}Em_{i\gamma}^{i\Omega} < {}_{Active}Em_{i\gamma}^{i\Omega}$: The task can be allocated, if ${}_{Past}Em_{i\gamma}^{i\Omega}$ qualifies the processing element. Otherwise the decision has to be postponed until the following cycle.

♦

12.3.2 Self-configuration: worst case timing behavior

Figure 12.5 shows the detailed cycle of the hormone distribution and interpretation based on figure 12.3. First the hormones (eager values, suppressors and accelerators) for all tasks PE_γ is interested in are emitted by PE_γ. Therefore, we define

$$M_\gamma := \{T_j \mid T_j \in M \text{ and } PE_\gamma \text{ is interested in } T_j\}$$

After waiting the time t_{SD}, the decision for a task $T_i \in M_\gamma$ is made. Afterwards i is incremented and the next cycle starts ($t_{DS} = 0$). This way, in each cycle the hormones for all relevant tasks are emitted and the decision for exactly one task is made. To decide on only one task per cycle allows the hormones to take effect. If task allocation took place all at once for all available tasks, the accelerators emitted when a task is allocated would not have a chance to make an impact as all the tasks would already be allocated in the first cycle.

To calculate the worst case timing behavior of this allocation process, we make the following basic **assumption**: All tasks (m tasks) have to be distributed on all PEs and all PEs are interested in all tasks.

First we make a further assumption to simplify the scenario: Let all eager values be constant, i.e., there are no accelerators and suppressors. Then, all

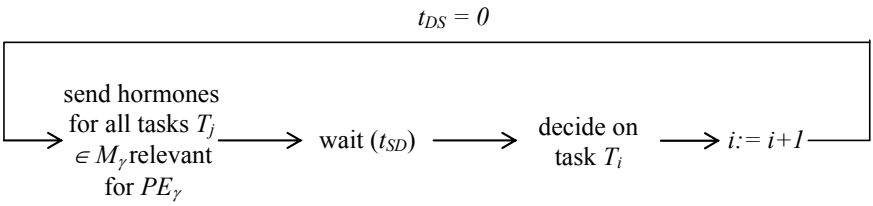


Fig. 12.5. Cycle of the hormone distribution and decision by PE_γ

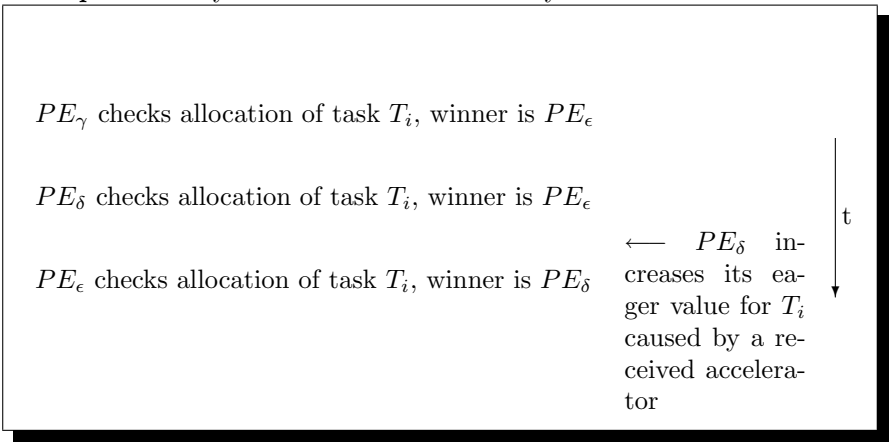
tasks have been handled by all PEs and have been allocated after m cycles and it follows:

$$\text{Worst Case Timing Behavior } = m \text{ cycles} \tag{12.1}$$

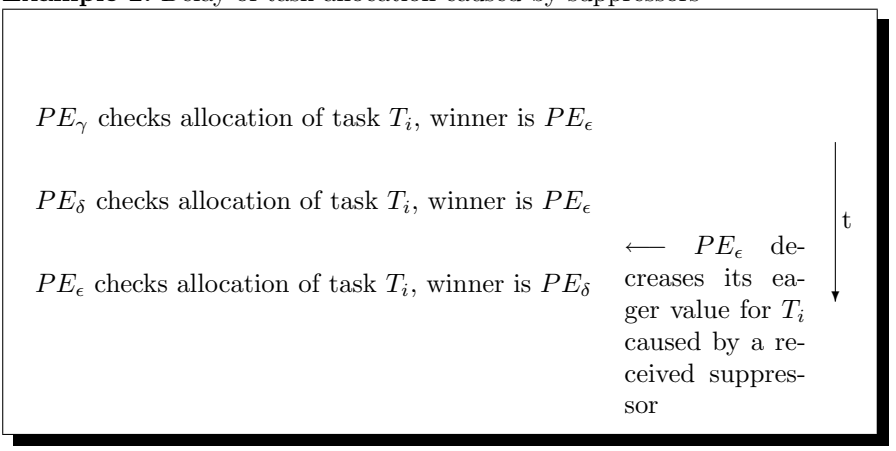
♦

In the following we remove the simplifying assumption of constant eager values and allow accelerators and suppressors. Now some tasks may not have been allocated after m cycles. This can be caused by accelerators and suppressors as shown in the following examples. In the first example three PEs are checking one after another the possibility to allocate task T_i. While PE_γ and PE_δ are checking, PE_ϵ still is the winner. After PE_δ has checked, it increases its eager value caused by a received accelerator. If afterwards PE_ϵ checks for allocation, PE_δ becomes the winner. However, PE_δ will not check again for allocation within the next m cycles. The second example shows a similar scenario, this time caused by an eager value decreased by a suppressor.

Example 1: Delay of task allocation caused by accelerators



Example 2: Delay of task allocation caused by suppressors



At worst in both cases task T_i will not be re-checked until a complete cycle of all other tasks, thus after m cycles. Afterwards, the same scenario could occur again. However, the maximal number of cycles is limited: A change of the eager value by suppressors or accelerators only takes place if a task has been allocated somewhere in the system (Assumption: Monitoring accelerators and suppressors are constant during the initial self-configuration). It follows that in each allocation cycle at least one task will be allocated. Thus, in the case of a variable eager value we get the following worst case timing behavior for the self-configuration:

$$\text{Worst Case Timing Behavior } = m^2 \text{ cycles} \tag{12.2}$$

♦

12.3.2.1 Improvement of the worst case timing behavior

By refining the algorithm presented in figure 12.5 it is possible to improve the timing behavior of the worst case scenario.

Refinement 1: If a PE_γ sends an eager value for a task T_k which was increased by an accelerator and this increased eager value is higher than all other values received for task T_k so far, then PE_γ exits the regular sequential decision cycle and checks for T_k instead.[1]

By using this refinement the following timing behavior results for the eager values increased by accelerators:

The worst case scenario is as follows: Assume, task T_i would be allocated by processing element PE_γ at the m^{th} cycle. Exactly in this cycle, the corresponding eager value $Em_{i\gamma}^{i\Omega}$ is incremented by an accelerator of another task.

[1] This is conform to the rule from section 12.3, that an interval of $t_{SD} \geq t_{DS} + 2t_K$ will be waited between sending and decision making.

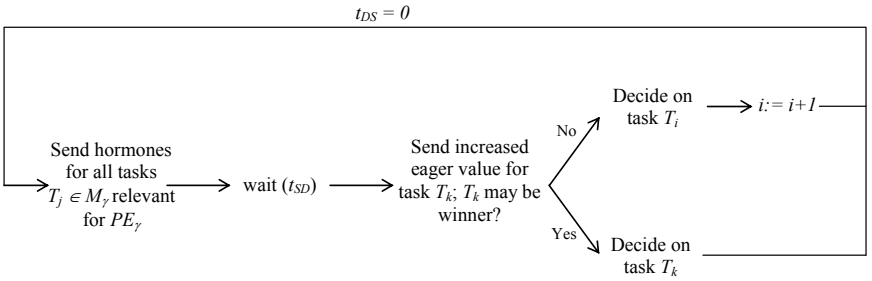


Fig. 12.6. Cycle of the hormone distribution and decision making for a PE_γ using the first refinement.

One of $m-1$ other tasks may be responsible for sending this accelerator by being allocated somewhere. As a result, T_i will not be allocated on PE_γ and will be re-checked in cycle $m+1$.

Further delay will arise if another accelerator is be sent in cycle $m+1$ and the eager value $Em_{i\gamma}^{i\Omega}$ for task T_i will be increased another time. Now, $m-2$ tasks may be responsible, one of which has been allocated. It follows that all tasks are assigned no later than $m+(m-1) = 2m-1$ cycles, which is also shown in the following scheme:

Cycle 1 :	$T_1\ T_2\ ...\ T_{m-2}\ T_{m-1}\ T_m$	
Cycle 2 :	$T_1\ T_2\ ...\ T_{m-2}\ T_{m-1}\ T_m$	
...	... ...	
...	... ...	
Cycle m :	$T_1\ T_2\ ...\ T_{m-2}\ T_{m-1}\ T_m$	$\leftarrow T_m$ assigned, accelerator sent
Cycle $m+1$:	$T_1\ T_2\ ...\ T_{m-2}\ T_{m-1}$	$\leftarrow T_{m-1}$ assigned, accelerator sent
Cycle $m+2$:	$T_1\ T_2\ ...\ T_{m-2}$	$\leftarrow T_{m-2}$ assigned, accelerator sent
...	... ...	...
...	... ...	...
Cycle $2m-2$:	$T_1\ T_2$	$\leftarrow T_2$ assigned, accelerator sent
Cycle $2m-1$:	T_1	$\leftarrow T_1$ assigned, accelerator sent

As a conclusion, we notice assuming that refinement 1 holds:

$$\text{Worst Case Timing Behavior } = 2m-1 \text{ cycles} \tag{12.3}$$

♦

Now we define a similar refinement for delays caused by suppressors:

Refinement 2: If a PE_γ receives an eager value for a task T_k which was decreased by a suppressor and therefore the own eager value is higher than

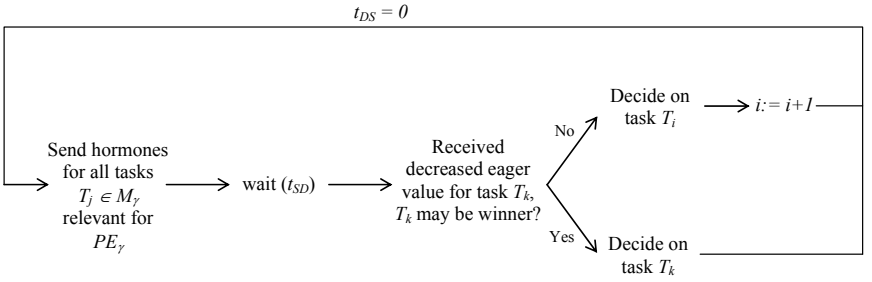


Fig. 12.7. Cycle of the hormone distribution and decision making for a PE_γ using the second refinement.

all other values received for task T_k so far, then PE_γ exits the regular sequential decision cycle and checks for T_k instead.

As a suppressor (similar like an accelerator) results from a task which has been allocated, the same worst-case timing behavior of $2m-1$ cycles will result from the same consideration as before. It should be noted that in our application refinement 2 can be omitted, because suppressors only affect the same tasks that created them. Therefore, a suppressor for a task is only emitted if this task has already been taken somewhere and need not be taken in the same cycle again.

12.3.2.2 Further improvements

Consequently, we narrow the scenario to the influence of accelerators on timing behavior and the worst-case timing behavior can be specified more precisely:

An accelerator is only sent to related tasks. Therefore a task T_i can not receive an accelerator from all the other $m-1$ tasks but only from the v_i tasks it is related to ($v_i \leq m-1$). Then, task allocation will be completed at the latest after

$$m + v_i \leq 2m - 1 \text{ cycles.}$$

Considering all tasks we get the following result:

$$\text{Worst Case Timing Behavior } = m + v_{\max} \text{ cycles} \tag{12.4}$$

where

$$v_{\max} := \max_{T_i \in M} \{v_i\}, \qquad \text{the largest number of related tasks.}$$

♦

Example 3: Differences caused by the improvements

We assume there are 10 tasks to be distributed in the system:

10 tasks: T_1 ... T_{10}, thus $m = 10$
Related tasks: T_1 ... T_4 and T_5 ... T_{10}, thus

$$v_{\max} = \max\{v_1, ..., v_{10}\} = \max\{4,\ 4,\ 4,\ 4,\ 6,\ 6,\ 6,\ 6,\ 6,\ 6\} = 6$$

The result is

- with $2m - 1 = 20 - 1 = 19$ cycles as an upper limit for the self-configuration (without any assumption about the largest number of related tasks).
- with $m + v_{\max} = 10 + 6 = 16$ cycles as an upper limit for the self-configuration (including the information about the largest number of related tasks).
- Furthermore it may happen that not all PEs apply for all tasks, but e.g. PE_1 and PE_2 for T_1 ... T_5 and PE_3 and PE_4 for T_6 ... T_{10}. Then

  $$m_{\max} = \max\{m_1,\ m_2,\ m_3,\ m_4\} = \{5,\ 5,\ 5,\ 5\} = 5$$

  Then, the upper limit for the self-configuration is

  $$m_{\max} + v_{\max} = 5 + 6 = 11 \text{ cycles.}$$

Further reductions of the worst-case timing behavior may take place, if not all PEs apply for all m tasks. If we release this basic assumption, the following timing behavior results for a PE_γ applying for a task T_i: The task allocation is finished at the latest after

$$m_\gamma + v_i \text{ cycles}$$

where

$$m_\gamma = |M_\gamma|, \qquad \text{the number of tasks for which } PE_\gamma \text{ applies.}$$

Extending this result to all tasks we obtain:

$$\text{Worst Case Timing Behavior } = m_{\max} + v_{\max} \text{ cycles} \tag{12.5}$$

where

$$m_{\max} := \max_{PE_\gamma \in \Omega}\{m_\gamma\}, \qquad \text{the largest number of tasks a PE is applying for.}$$

♦

Example 3 illustrates the differences of these improvements.

12.3.3 Self-optimization: worst case timing behavior

Self-optimization means the relocation of a task T_i from a processing element PE_γ to another processing element PE_δ. This relocation takes place only if PE_γ currently executing T_i offers this task. Thus, PE_γ chooses the point in time for the optimization, e.g., periodically or when T_i is idle. Additionally, PE_γ operates the task until it is completely transferred to PE_δ. Notice that even $\gamma = \delta$ is possible, which means the task execution is continued on PE_γ. This has the following consequences:

- There is no blackout time (except the time used to transfer the task state).
- Real-time behavior is guaranteed.

The time interval from offering the task until to the completion of the transfer is bounded. Thus, the time for self-optimization is also bounded.

In the worst case, all processing elements offer all tasks for self-optimization, which leads to the same time bounds like for self-configuration.

$$\text{Worst Case Timing Behavior} = m_{\max} + v_{\max} \text{ cycles} \tag{12.6}$$

♦

Notice that there is no interruption in task execution because relocated tasks are operated by the previous PEs until completely transferred to the new PEs.

If we assume that only one task is offered per cycle the time for self-optimization is decreased considerably. Let's assume PE_γ would offer T_i for self-optimization. Then, the eager value $Em_{i\delta}^{i\Omega}$ would increase on all processing elements $PE_\delta \in \Omega$ applying for T_i as the acquisition suppressor $Sa_{i\gamma}^{i\Omega}$ would be dropped. If we use refinement 1 (see section 12.3.2.1) all eligible processing elements check for T_i in the next cycle. Assuming that only one task is offered in this cycle there are no further modifications of the eager values. Thus, T_i is assigned to a new processing element PE_δ in the next cycle.

$$\text{Worst Case Timing Behavior} = \ 1 \text{ cycle} \equiv const. \tag{12.7}$$

♦

12.3.4 Self-healing: worst case timing behavior

In case of processing element failure, the execution of its tasks fails until they are reassigned. The point in time when the processing element fails is not predictable. Therefore, we only obtain limited real-time behavior. Nevertheless, we are able to compute time bounds for this case. The worst case is that all processing elements are failing simultaneously, which means there is no chance for self-healing. Thus, we exclude this case and assume that there are still enough processing elements operational to execute all tasks. Then,

almost the same upper bound as for self-optimization holds. We only have to add the time the operational processing elements need to recognize that other processing elements failed by the expiration of the hormones.

$$\text{Worst Case Timing Behavior} = m_{\max} + v_{\max} + a \text{ cycles} \tag{12.8}$$

where a: number of cycles after which a not updated hormone is considered to be too old and thus not valid any longer (expiration time, see section 12.2.2). ♦

If only one PE_γ fails and there is no self-optimization in parallel, the tasks $T_i \in E_\gamma$ running on PE_γ will be reassigned due to their vanishing acquisition suppressors. If we use refinement 1 and consider the emission of accelerators when a task is taken, we obtain

$$\text{Worst Case Timing Behavior} = e_\gamma + \max_{T_i \in E_\gamma} \{v_i\} + a \text{ cycles} \tag{12.9}$$

where

$\max_{T_i \in E_\gamma} \{v_i\}$: greatest number of related tasks to the tasks running on PE_γ.

♦

12.4 Communication load introduced by the artificial hormone system

Now, we calculate the communication load introduced by the artificial hormone system. A processing element PE_γ is sending per cycle:

Broadcast to all other PEs : 1 modified eager value $Em_{j\gamma}^{j\Omega}$ for each task T_j PE_γ applied for

1 acquisition suppressor $Sa_{i\gamma}^{i\Omega}$ for each task T_i PE_γ has taken

Multicast to neighbors : 1 organ accelerator $Ao_{i\gamma}^{V_i \Phi_\gamma}$ for each task related to a taken task T_i

All other kinds of hormones are sent or used locally (see section 12.2).

We also need to know the sender information additionally to the eager values, suppressors and accelerators to be able to refresh the hormone values of a sender. Thus, we propose the following structure for a sent hormone:

$$\underbrace{\text{Type of hormone}}_{\text{Eager value, accelerator, suppressor}}, \underbrace{\text{PE-identification, Task-identification}}_{\text{Sender information}}, \text{Value}$$

12.4.1 Hormone communication load per processing element

Each PE_γ causes the following hormone communication load:

Broadcast to all other processing elements:

$$Db_\gamma = De * k_\gamma + Ds * e_\gamma \tag{12.10}$$

where

Db_γ : broadcast data load caused by PE_γ
De : data load to send an eager value
Ds : data load to send a suppressors
k_γ : number of tasks PE_γ applied for, which are not yet completely taken in the system

Multicast to neighbors:

$$Dm_\gamma = Da * \sum_{T_i \in E_\gamma} v_i \tag{12.11}$$

where

Dm_γ : multicast data load caused by PE_γ
Da : data load to send an accelerator

Following from this, we obtain the hormone communication load from any processing element PE_γ at the beginning of the task allocation (self-configuration) and in the steady state (all tasks are allocated, only self-optimization and self-healing take place):

$$\left.\begin{aligned} {}_{\text{Start}}Db_\gamma &= De * k_\gamma \\ {}_{\text{Start}}Dm_\gamma &= 0 \end{aligned}\right\} \quad e_\gamma = 0 \text{ at the start} \tag{12.12}$$

and

$$\left.\begin{aligned} {}_{End}Db_\gamma &= Ds * e_\gamma \\ {}_{End}Dm_\gamma &= Da * \sum_{T_i \in E_\gamma} v_i \end{aligned}\right\} \quad k_\gamma = 0 \text{ at the end} \tag{12.13}$$

♦

12.4.2 Overall hormone communication load

The overall hormone communication load introduced by the artificial hormones at any processing element PE_γ results from the sum of the multicasts to its neighbors and the sum of broadcasts of all processing elements:

$$D_\gamma = \sum_{PE_\delta \in \Omega} Db_\delta + \sum_{PE_\delta \in \Phi_\gamma} Dm_\delta \tag{12.14}$$

where D_γ : overall communication load of PE_γ

As a result, we can compute the overall communication load at any processing element PE_γ at the beginning of the task allocation and in the steady state:

$$\begin{aligned} {}_{\text{Start}}D_\gamma &= \sum_{PE_\delta \in \Omega} {}_{\text{Start}}Db_\delta + \sum_{PE_\delta \in \Phi_\gamma} {}_{\text{Start}}Dm_\delta \\ &= \sum_{PE_\delta \in \Omega} De * k_\delta \end{aligned} \tag{12.15}$$

and

$$\begin{aligned} {}_{End}D_\gamma &= \sum_{PE_\delta \in \Omega} {}_{End}Db_\delta + \sum_{PE_\delta \in \Phi_\gamma} {}_{End}Dm_\delta \\ &= \sum_{PE_\delta \in \Omega} Ds * e_\delta + Da * \sum_{PE_\delta \in \Phi_\gamma} \sum_{T_i \in E_\delta} v_i \end{aligned} \tag{12.16}$$

Now, we can calculate an upper bound for the overall communication load at any processing element at the beginning of the task allocation and in the steady state. We estimate the sums of the individual communication load by multiplying the maximal communication load with the number of assigned processing elements.

Considering the broadcast, we estimate the sum of individual broadcast communication load by multiplying the greatest existing broadcast communication load of all processing elements with the number of processing elements.

Considering the multicast, we estimate the sum of the individual multicast communication load of neighbored processing elements by multiplying the greatest existing multicast communication load with the greatest number of neighbors existing in the scenario. We obtain for each $PE_\gamma \in \Omega$:

$$\begin{aligned} {}_{\text{Start}}D_\gamma &\leq \omega * \max_{PE_\delta \in \Omega} \{{}_{\text{Start}}Db_\delta\} \\ &= \omega * De * k_{\max} \end{aligned} \tag{12.17}$$

and for each $PE_\gamma \in \Omega$ holds:

$$\begin{aligned} {}_{End}D_\gamma &\leq \omega * \max_{PE_\delta \in \Omega} \{{}_{End}Db_\delta\} + \max_{PE_\delta \in \Omega} \{\varphi_\delta\} * \max_{PE_\delta \in \Omega} \{{}_{End}Dm_\delta\} \\ &\leq \omega * Ds * e_{\max} + \varphi_{\max} * Da * e_{\max} * v_{\max} \end{aligned} \tag{12.18}$$

where

$k_{\max} := \max_{PE_\delta \in \Omega} \{k_\delta\}$, maximum of all k_δ

$e_{\max} := \max_{PE_\delta \in \Omega} \{e_\delta\}$, maximum of all e_δ

$v_{\max} := \max_{T_i \in M} \{v_i\}$, greatest number of related tasks, see section 12.3.2.2

$\varphi_{\max} := \max_{PE_\delta \in \Omega} \{\varphi_\delta\}$, greatest number of all neighbored processing elements.

♦

12.4.3 Example

In this section, we calculate the data load introduced by the artificial hormones in a concrete scenario. First, we define the structure of the hormones and the resulting data load.

Eager values, suppressors	2 bit for the type of hormone	
	4 bit x-coordinate of PE 4 bit y-coordinate of PE	(ID of PE (256 PEs at maximum))
	7 bit for the task ID	(128 tasks at maximum)
	7 bit value	(128 nuances of a hormone)
$\sum$	24 bit	

Thus, it follows:

$$De = Ds = 24 \text{ bit}$$

Accelerators	2 bit for the type of hormone	
	4 bit x-coordinate of PE	
	4 bit y-coordinate of PE	
	7 bit for the task ID	
	7 bit for the ID of related tasks 7 bit value	(repeated v_i times)
$\sum$	$17 + v_i * 14$ bit	

To calculate the worst case, we assume $v_i = v_{\max}$. Thus, it follows:

$$Da = 17 + v_{\max} * 14 \text{ bit}$$

Now, we define the values for the number of processing elements, tasks and so on:

$$\begin{aligned} \omega &:= 64 && \text{(Number of processing elements)} \\ \varphi_{\max} &:= 9 && \text{(Number of PEs neighbored to a PE)} \\ k_{\max} &:= 32 && \text{(Maximal number of tasks a PE applied for)} \\ e_{\max} &:= 2 && \text{(Maximal number of tasks taken by a PE)} \\ v_{\max} &:= 8 && \text{(Maximal number of tasks related to a task)} \end{aligned}$$

Using these values, we obtain for each $PE_\gamma \in \Omega$:

$$_{\text{Start}}D_\gamma \leq 64 * 24 * 32 \text{ bit} = 49152 \text{ bit} = 6144 \text{ bytes} \tag{12.19}$$

$$_{End}D_\gamma \leq 64 * 24 * 2 + 9 * 2 * (17 + 8 * 14) \text{ bit} = 674.25 \text{ bytes} \tag{12.20}$$

Let's assume a cycle time of 100 ms ($t_{SD} + t_{DS}$). Then, we can compute the maximal data load caused by the artificial hormones for each $PE_\gamma \in \Omega$:

$$_{\text{Start}}DS_\gamma \leq 10 * 6144 \text{ bytes/sec} = 60 \text{ kBytes/sec} \tag{12.21}$$

$$_{End}DS_\gamma \leq 10 * 674.25 \text{ bytes/sec} \approx 6.58 \text{ kBytes/sec} \tag{12.22}$$

As it can be seen, the data load caused by the artificial hormones is significantly higher at the beginning than in the steady state. However, there is only a small amount of user data to be sent at the beginning because the tasks are not yet assigned. In the steady state, there is more user data to be sent and the data load caused by the artificial hormones is small. In the best case, both effects eliminate each other thus resulting in a constant data load caused by the artificial hormones.

12.5 Related work

There are several approaches for task allocation in middleware. In [2], the authors present a scheduling algorithm distributing tasks onto a grid. It is implemented in the Xavantes Grid Middleware and arranges the tasks in groups. This approach is completely different from ours because it uses central elements for the grouping: The Group Manager (GM), a Process Manager (PM) and the Activity Managers (AM). Here, the GM is a single point of failure because, if it fails there is no possibility to get group information from this group anymore. In our approach there is no central task distribution instance and therefore no single point of failure can occur.

Another approach is presented in [7]. The authors present two algorithms for task scheduling. The first algorithm, Fast Critical Path (FCP) makes sure

time constrains to be kept. The second one, Fast Load Balancing (FLB) schedules the tasks so that every processor will be used. Using this strategy - especially the last one - it is not guaranteed that related tasks are scheduled nearby each other. In contrast to our approach, these algorithms do not include the failing of processing elements.

In [6], a decentralized dynamic load balancing approach is presented. Tasks are considered as particles which are influenced by forces like e.g. a load balancing force (results from the load potential) and a communication force (based on the communication intensities between the tasks). In this approach, the tasks are distributed according to the resultant of the different types of forces. A main difference to our approach is that we are able to provide time bounds for the self-configuration. Besides our approach covers self-healing, which is absolutely not considered by this decentralized dynamic load balancing.

[8] presents a load balancing scheme for task allocation based on local workpiles (of PEs) storing the tasks to be executed. The authors propose to execute a load balancing algorithm between two PEs to balance their workload. The algorithm is executed with a probability inversely proportional to the length of the workpile of a PE. Although this approach is distributed it does not consider aspects like self-healing and real-time constraints.

Other approaches of load balancing are presented in [1, 3, 4, 5, 9]. None of them cover the whole spectrum of self-x-properties, task clustering, and real-time conditions like our approach.

12.6 Conclusion and further work

We presented an artificial hormone system to assign tasks to processing elements within a processor grid. The assignment is completely decentralized and holds self-x features. Besides, we showed that we can guarantee tight upper bounds for the real-time behavior of the artificial hormone system as well as for the data load induced by the artificial hormones.

We implemented a simulator including the presented algorithms and as future work, we will evaluate the time bounds received by the theoretical examinations of the hormone system.

Furthermore, we will investigate additional quality properties of the artificial hormone system like stability of the task assignment and rules for selecting the level of hormone values to, e.g., obtain organs. Another question in this scope is how to find an optimal task assignment (if it exists) by the artificial hormone system.

We will also investigate the artificial hormone system in the scope of a practical example, the DoDORG project, which deals with a grid of processing elements to be organized by an organic middleware using the artificial hormone system.

References

1. W. Becker. Dynamische adaptive Lastbalancierung für große, heterogen konkurrierende Anwendungen. Dissertation, Universität Stuttgart, Fakultät Informatik, Dezember 1995.
2. L. F. Bittencourt, E. R. M. Madeira, F. R. L. Cicerre, and L. E. Buzato. A path clustering heuristic for scheduling task graphs onto a grid. In *3rd International Workshop on Middleware for Grid Computing (MGC05)*, Grenoble, France, 2005.
3. T. Decker, R. Diekmann, R. Lüling, and B. Monien. Universelles dynamisches task-mapping. In *Konferenzband des PARS'95 Workshops in Stuttgart, PARS-Mitteilung 14*, pages 122–131, 1995.
4. J. Finke, K. M. Passino, and A. Sparks. Cooperative control via task load balancing for networked uninhabited autonomous vehicles. In *42nd IEEE Conference onDecision and Control, 2003. Proceedings*, volume 1, pages 31 – 36, 2003.
5. J. Finke, K. M. Passino, and A. Sparks. Stable task load balancing strategies for cooperative control of networked autonomous air vehicles. In *IEEE Transactions on Control Systems Technology*, volume 14, pages 789– 803, 2006.
6. H.-U. Heiss and M. Schmitz. Decentralized dynamic load balancing: The particles approach. In *Proc. 8th Int. Symp. on Computer and Information Sciences*, Istanbul, Turkey, November 1993.
7. A. Radulescu and A. J. C. van Gemund. Fast and effective task scheduling in heterogeneous systems. In *IEEE Computer - 9th Heterogeneous Computing Workshop*, Cancun, Mexico, 2000.
8. L. Rudolph, M. Slivkin-Allalouf, and E. Upfal. A simple load balancing scheme for task allocation in parallel machines. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 237–245, 1991.
9. C. Xu and F. Lau. Decentralized remapping of data parallel computations with the generalized dimension exchange method. In *Proceedings of Scalable High-Performance Computing Conference*, pages 414 – 421, 1994.

13

Bio-Inspired Networking — Self-Organizing Networked Embedded Systems

Falko Dressler

Autonomic Networking Group, Dept. of Computer Science 7, University of Erlangen, Martensstr. 3, 91058 Erlangen, Germany.
dressler@informatik.uni-erlangen.de

Summary. The turn to nature has brought us many unforeseen great concepts and solutions. This course seems to hold on for many research domains. In this article, we study the applicability of biological mechanisms and techniques in the domain of communications. In particular, we study the behavior and the challenges in networked embedded systems that are meant to self-organize in large groups of nodes. Application examples include wireless sensor networks and sensor/actuator networks. Based on a review of the needs and requirements in such networks, we study selected bio-inspired networking approaches that claim to outperform other methods in specific domains. We study mechanisms in swarm intelligence, the artificial immune system, and approaches based on investigations on the cellular signaling pathways. As a major conclusion, we derive that bio-inspired networking techniques do have advantages compared to engineering methods. Nevertheless, selection and employment must be done carefully to achieve the desired performance gains.

Key words: bio-inspired networking, autonomic networking, self-organization, networked embedded systems, bio-inspired algorithms

13.1 Introduction

The proliferation of wireless sensor networks (WSN) and similar ad hoc networks based on huge amounts of spontaneously interacting nodes is changing the world of telecommunications. In addition to the increasing number of communicating nodes, node mobility is an issue as addressed, for example, in sensor/actuator networks (SANET). Previously, controllability and determinism were the keywords during protocol development and network research. Based on the primary objectives of WSN, nodes communicate using a radio interface, they are battery-driven, small, and cover only few resources. Therefore, new key factors have been identified for developing communication methods. Above all, scalability of the employed mechanisms is required.

Researchers anticipate self-organization methods as the general solution to the depicted communication issues in WSN and SANET. Centralized

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_13,

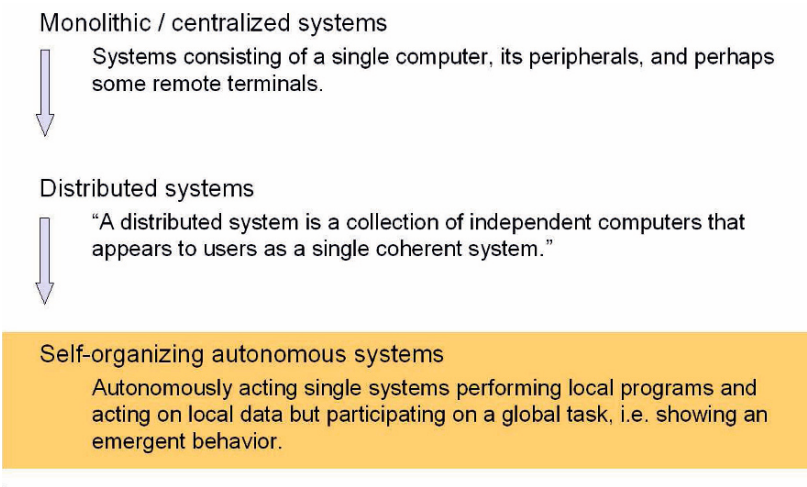


Fig. 13.1. The changing world: centralized systems, decentralized control, and self-organization [13].

management and optimized control will be replaced by methodologies that focus on local knowledge about the environment and adequate decision making processes. Similar problems are known and well-studied in nature. Therefore, such biological solutions should be analyzed for adaptation to the communication in ad hoc networks and WSN.

The goal of this article is to provide an overview of some bio-inspired networking mechanisms and to introduce the underlying biological functionality as well as the adaptation to technical processes. Even though it is not intended as a general review, it summarizes the best-known approaches and explains selected mechanisms in more detail.

13.2 Networked embedded systems

Networked embedded systems are used in many application scenarios. Above all, wireless sensor networks (WSN) are widely studied [3, 6]. Sensor networks consist of multiple, usually hundreds or even thousands of sensor nodes. Such networks do not have a predominant topology but are created dynamically, ad hoc on demand. The nodes themselves can be of any size. Nevertheless, most publications understand sensor nodes as small, battery-driven devices with limited processing power and memory, radio communication, and sensors to measure physical parameters such as the temperature.

Similarly, sensor/actuator networks (SANET) extend the idea of wireless sensor networks to mobile actuation systems, e.g. robot-like systems. In general, such SANET are built of cooperating mobile autonomous systems that allow some kind of actuation, e.g. handling, mobility [2].

With WSN and SANET, new issues appeared that are not covered by existing communication methods and protocols. Some of these issues are inherent in the idea of interconnecting thousands of networked embedded systems, others evolve based on particular application scenarios of WSN:

Node mobility: In general, sensor networks are believed to be stationary, i.e., to have a fixed topology – at least in terms of node location. Admittedly, node mobility is becoming a major concern of new application scenarios such as logistics. SANET, on the other hand, inherently include location dynamics and mobility.

Network size: In contrast to other networks, the number of nodes that are building a network on demand can be very high. Structured networks such as the Internet benefit from a hierarchical organization and a centralized management of subnetworks. WSN and SANET are infrastructureless networks facing scalability problems if too many nodes are concerned.

Deployment density: Depending on the application scenario, the node density in a WSN can be very high. This may break existing medium access control protocols and lead to energy exhaustion just for neighborhood detection.

Energy constraints: Instead of having unlimited energy for computation and communication, energy constraints are much more stringent than in fixed or cellular networks. Usually, sensor nodes are battery operated and in certain cases, recharging of the energy source is impossible. We distinguish replenishable power sources, e.g., for wearable sensors, non-replenishable power sources, e.g. for sensors deployed in remote, hazardous terrain, and regenerative power sources.

Data / information fusion: Limited bandwidth as well as the mentioned power constraints demand aggregation techniques. Each data packet that has to be transported through a WSN is expensive. Aggregated data reduce energy consumption and provide higher usefulness.

In summary, it can be said that self-organization mechanisms are needed for higher scalability in WSN/SANET communication [12]. The basic mechanisms available include neighborhood discovery, topology (re-)organization, and probabilistic approaches. Since optimization on a global level is no longer possible, there is always a discrepancy between multiple objectives. For example, the latency of path-finding with on-demand routing protocols may be too high and periodic routing overhead in a table-driven routing protocol may consume a significant amount of bandwidth [1]. On the other hand, the probability of successful transmission might be too low for stateless approaches. Therefore, hybrid architectures may improve the scalability and optimize the network behavior depending on the application scenario.

Figure 13.1 illustrates the control and management of systems consisting of multiple subsystems. Centralized control is primarily used to operate in an environment consisting of a few nodes. Using centralized information about all systems, optimized solutions for communication and task allocation can be

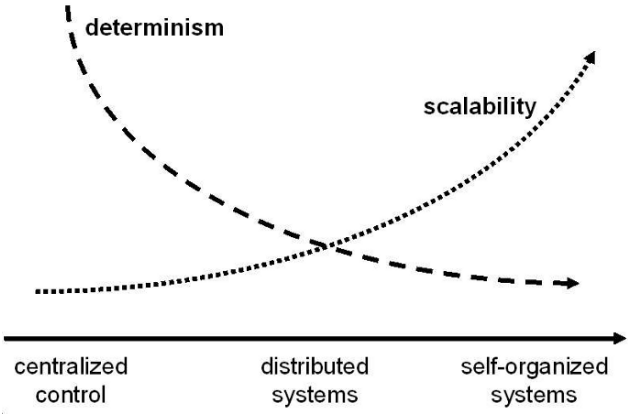


Fig. 13.2. Antagonism between determinism/controllability vs. scalability in system management and control [13].

derived. Examples are perfect schedules for medium access and real-time failure detection and repair. Distributed control allows to manage larger numbers of systems in a scalable way by preserving most systems characteristics such as controllability. Nevertheless, optimization becomes harder and predictability is reduced. Finally, self-organizing systems should help to overcome all scalability problems.

Unfortunately, determinism and controllability of the overall system are reduced. The relation between determinism and scalability is depicted in figure 13.2. Another issue is the challenge of programming such less predictable systems showing emergent behavior.

Referring to networked embedded systems and their management and control, self-organization mechanisms are needed in order to support a large amount of simultaneously intercommunicating nodes. In WSN and SANET, we need new methods to identify available communication paths, nodes, and their capabilities and resources. Additionally, data handling including storage, aggregation, and distribution must be changed and adapted to the new requirements. All mentioned operations should be possible without knowledge about the current network topology, available nodes, their addresses, their location, and others.

13.3 Self-organization: "from nature to engineering"

The turn to nature for solutions to technological problems has brought us many unforeseen great concepts. This encouraging course seems to hold on for many aspects in technology. First studies on biological self-organization and its possible adaptation to technical solutions date back to the 1960ies. Von Foerster [30] and Eigen and Schuster [16] proposed to employ self-organization

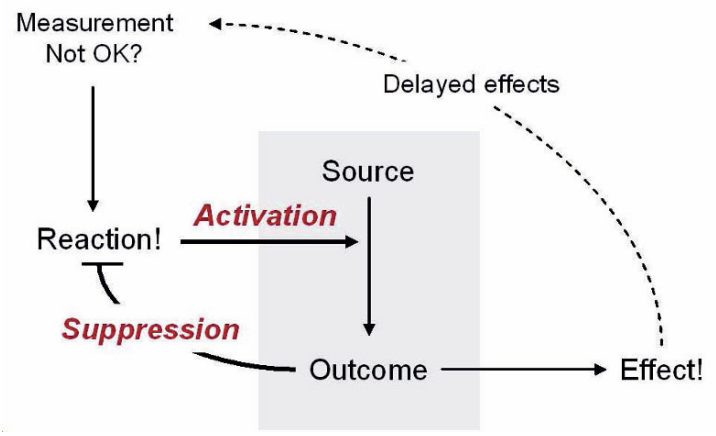


Fig. 13.3. System control using positive and negative feedback loops.

methods as known from many areas in biology. He saw the primary application in engineering in general. Nevertheless, it has been shown that communications can benefit from biologically inspired mechanisms as well.

13.3.1 Basic principles of self-organization

There are three major principles of self-organization mechanisms: feedback loops, local state evaluation, and interaction between individuals. Additionally, probabilistic methods that provide scalability and some degree of predictability can be found in nature and adapted to technology. This process needs careful consideration to prevent mistakes due to limited knowledge about the biological processes or due to the lack of correlation between the natural and the technical models [12].

Figure 13.3 depicts a system that employs all three principles. The main system is performing some action on a source to provide an outcome. Based on this system, the mentioned mechanisms for self-organization need to be discussed in more detail:

Feedback loops: One major component in understanding the interaction of components producing a complex pattern are positive and negative feedback loops. Positive feedback acts as an amplifier for a given effect. In order to prevent overreaction and misregulation, negative feedback is used to efficiently control the system behavior. An example for a positive feedback loop is depicted in figure 13.3, the activation of the processing step. Additionally, a negative feedback loop is included. The outcome directly suppresses an environmental reaction and, therefore, reduces the activation capabilities, i.e., the level of the system's inherent ability to become activated due to observed effects.

Local state: The second ingredient is the local state. This means that all subsystems are acquiring and acting upon locally stored information. Any global control or dependency is prevented in order to enable fully autonomous behavior embedded into a global context. The idea of using local state only is depicted in our example by missing external control processes.

Interactions: Information transfer between individuals is necessary to update the local state. There are two ways to conduct such interactions: direct interaction or communication between related subsystems and indirect information exchange by interacting with the environment. This process is also known as *stigmergic* [9]. The example in figure 13.3 includes stigmergic interactions. The system influences the environment (it produces some effect). This effect can be measured and directly increases or decreases the activation capabilities to the system behavior.

Probabilistic methods: In order to prevent synchronization problems and to increase the variety of application domains scalability is often achieved by random selection.

13.3.2 Bio-inspired techniques in technical systems

The development in the area of bio-inspired engineering is relying on various research fields including swarm intelligence, the artificial immune system, evolutionary and genetic algorithms, and cell and molecular biology based approaches. Some of the best known approaches should be summarized here whereas selected methods are depicted in more detail in the following section.

The behavior of large groups of interacting small insects such as ants and bees builds the basis for the field of ***swarm intelligence***. Simple and seemingly unrelated, autonomously working individuals are considered to compose complex cooperative tasks. Similar actions are required in various areas of engineering and computer science. Thus, swarm intelligence is forming a basis for building self-organizing systems [5, 19]. The focus lies on the formation of groups or clusters that allow efficient task allocation mechanisms. Successful application of swarm intelligence methods has been demonstrated in task allocation and control of multi-robot systems [24]. Recently, similar applicability has been shown in sensor networks [26].

The immune system of mammals builds the basis for research on the ***artificial immune system*** (AIS). The reaction of the immune system, even to unknown attacks, is a highly adaptive process. Therefore, it seems obvious to apply the same mechanisms for self-organization and self-healing operations in computer networks. In the last decade, several architectures for an AIS have been proposed [20, 17]. Application examples include autonomous communication [29] as well as ad hoc networking [27] Additionally, security scenarios including virus and intrusion detection already benefited from AIS approaches [22, 23].

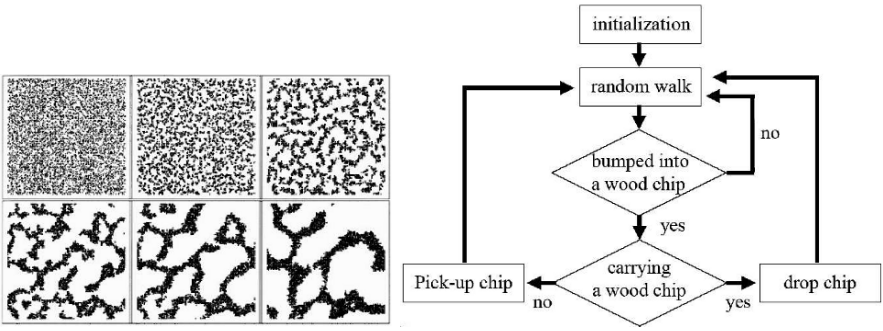


Fig. 13.4. The emergent collective intelligence of groups of simple agents [5].

Evolutionary algorithms (EA) are self-manipulating mechanisms. The evolution in nature is the basis for such methodologies. In particular, there are multiple ways for organisms to learn. A natural selection process (survival of the fittest) is going on letting only the optimal prepared organisms to survive and to reproduce. Changes appear for example by mutations. An overview to evolutionary algorithms is provided for example in [4, 7].

An emerging research area looks for ***cell and molecular biology*** based approaches. All organisms are built in the same way. They are composed of organs, which consist of tissues and finally of cells. This structure is very similar to computer networks, and so are the cellular signaling pathways. Therefore, research on methods in cell and molecular biology promises high potential for computer networking in general and adaptive sensor networks and network security in particular [14, 25].

While many advantages can be identified that make the use of bio-inspired techniques successful, we also need to comment the limitations of bio-inspired mechanisms. Biology always makes compromises between different goals and it is well known that biology sometimes fails. Additionally, some natural mechanisms are not well understood and well-defined problems may be solved by other means.

13.4 Bio-inspired networking

Primarily, the goal of this section is to demystify the concepts of bio-inspired networking. Based on selected approaches, the objectives and solution paths of biologically inspired methods are depicted in more detail.

13.4.1 Swarm intelligence

The collaborative work of a multitude of individual autonomous systems is necessary in many areas of engineering. Swarms of small insects such as bees

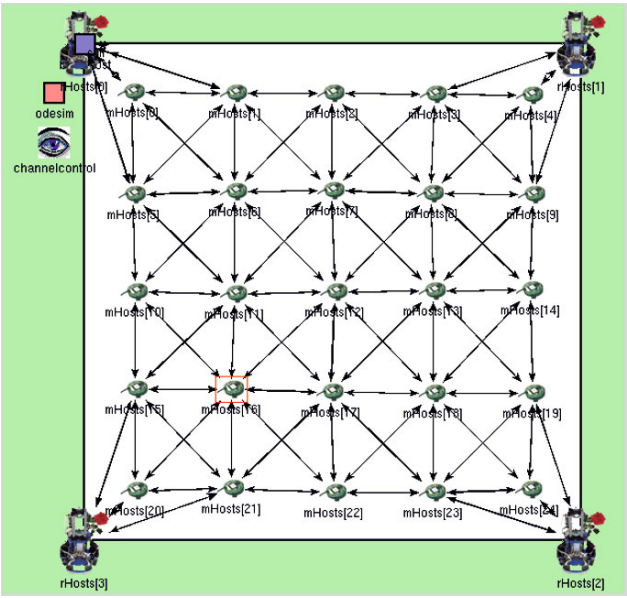


Fig. 13.5. Simulation setup for evaluation of the attractor based task allocation and routing.

or ants address similar issues. For example, ants solve complex tasks by simple local means. There is only indirect interaction between individuals through modification of the environment, e.g., pheromone trails are used for efficient foraging. Finally, the productivity of all involved ants is better than the sum of their single activities and ants are "grand masters" in search and exploration.

The basic principles are simple. All individuals – the systems that collaborate on an overall task – follow simple rules that lead to impressive global behavior, which emerges based on the simple rules and interactions between the systems, either directly or indirectly via the environment. An example is described in figure 13.4. The foraging algorithm used by termites to collect wood chips is shown on the left hand side. Using a simulation model, the overall visible behavior was studied [5]. Quickly, the chips are heaped together and structures emerge from the scene as shown on the right hand side.

Attractor-based routing and task allocation

As a specific example to demonstrate the capabilities of swarm intelligence methods in networking, we chose an attractor scheme for routing and task allocation [26]. In sensor networks supported by mobile robots, routing decisions usually need to be taken on demand because the network topology changes over time. Additionally, multiple tasks may be needed to be executed by different systems in the network. Usually, static programming or complex,

auction-based task allocation strategies are used, whereas those approaches fail in large scale and highly dynamic scenarios. The algorithm described here is based on the AntHocNet approach that enables self-organized routing control in ad hoc networks [10]. The pheromone trail mechanism is exploited to search for optimal paths through ad hoc networks. After a short learning phase, the optimal solution can be derived from messages transmitted previously over suboptimal paths.

The new approach is based on a probabilistic scheme. Each node performs a local decision process that provides the basis for task allocation and routing decisions. The basic idea is as simple as powerful. If a node successfully performed a particular task (whether forwarding a packet or anything else), its probability to perform this task again is increased. Similarly, the probability is decreased if the node failed for a particular task. Additionally, each node observes the behavior of the surrounding nodes to update its local behavior accordingly.

More formally, this algorithm can be written as follows. Each node n associates to a task T_i to an attractor τ_i with $i \in T$. At the moment of selecting a task to perform, the node computes a probability for choosing task T_i as follows:

$$P(i) = \frac{\tau_i^\beta}{\sum_{k \in T} \tau_k^\beta} \tag{13.1}$$

The parameter β was introduced to increase the exploitation of good paths. Each node initializes τ_i with τ_{init}. If the node successfully performed the given task i, τ_i is recalculated as follows: $\tau_i = \min\{\tau_{\max}, \tau_i + \Delta\tau\}$. Similarly, τ_i is reduced for unsuccessful operations: $\tau_i = \max\{\tau_{\min}, \tau_i - \Delta\tau\}$.

The complete algorithm, the corresponding calculations, and an in-depth evaluation can be found in [26]. In that paper, a set of experiments was performed to demonstrate the advantages of the attractor scheme. The simulation setup is shown in figure 13.5. 25 nodes were places in a grid on a playground of 500m×500m. Four different tasks were defined to be performed by all these nodes. Disregarding task allocation, we focus on the associated route selection in this network.

Figure 13.6 shows selected simulation results. On the left hand side, we show a typical snapshot of the distribution of tasks in the network. The plot refers to task T_3. It can be seen that when a node had high probability of performing T_3, its neighbors were likely to have a low one. The routes that were used to send the data to the base host are depicted in the same figure on the right. The network was split in two halves: there were few links between the top right triangle and the bottom left triangle. This figure does not represent the steady state of the network. The network reached a dynamic equilibrium, where things continually changed. This is especially true for the depicted routes, since the routing table entries were removed after a while, and new discoveries took place.

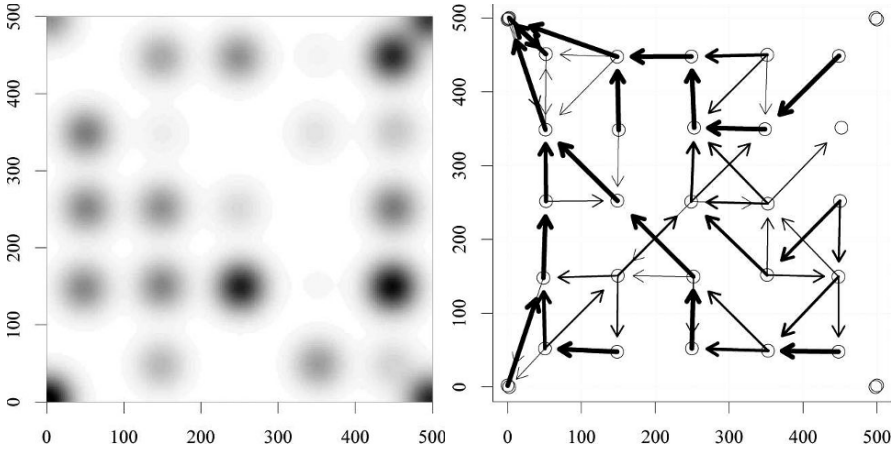


Fig. 13.6. Simulation results [26]. Left: Distribution of task T_3 among the nodes. The darker the circle, the higher is the probability that an agent performs T_3. Right: Routes to deliver the output of T_3 to the base host (in the upper left corner). The arrows show the known next hops for every node. Their thickness is proportional to the probability of choosing a node as next hop.

This example illustrates an architecture for attractor-based task allocation and routing. The nodes make use of solutions inspired by ants' behavior. The control architecture is based on strong interlayer and interagent interactions. The latter are local, meaning that they occur only between agents within a given range, smaller than the experimental area. The architecture is based on probabilistic decisions. During the lifetime of the network, the nodes adapt their probability to execute one task from a given set. The architecture exploits the interactions between agents, but only within a limited range. The local interactions are, however, sufficient to induce a global pattern, i.e., to provide a self-organizing behavior. No particular knowledge of the environment or of the other nodes' activity is required. Moreover, the architecture is based on a cross-layer design, in which application and network layers collaborate on a common objective.

13.4.2 Artificial immune system

Artificial immune systems are computational systems inspired by theoretical immunology and observed immune functions, principles and models, which are applied to complex problem domains [8]. The primary goal of an artificial immune system (AIS) is to efficiently detect changes in the environment or deviations from normal system behavior. The most impressing capabilities of the immune system are its recognition capabilities (anomaly detection, noise tolerance), its robustness, diversity, the capability of reinforcement learning,

and the possibility to memorize observations. These features allow to build self-optimizing and self-learning processes.

The role of the mammalian immune system can be summarized as follows. It should protect the body from infections. For this, two immune responses were identified. The primary one is to launch a response to invading pathogens leading to an unspecific response (using leucoytes). In contrast, the secondary immune response remembers past encounters, i.e., it represents the immunologic memory. It allows a faster response the second time around showing a very specific response (using B-cells and T-cells).

The immune recognition is based on the complementarity between the binding region of a receptor and a portion of an antigen called epitope. Antibodies have a single type of receptor, while antigens might show several epitopes. This means that different antibodies can recognize a single antigen. The immune system needs to be able to differentiate between self and non-self cells. Antigenic encounters may result in cell death; therefore, the immune system establishes some kind of positive and negative selection.

The scope of AIS is widespread. There are applications for fault and anomaly detection, data mining (machine learning, pattern recognition), agent-based systems, control, and robotics. In the mammalian immune system, the shape of the molecules defines the degree of binding. In an AIS, a similar distance measure is needed. Typically, antigens and antibodies are described in form of vectors, i.e. $Ab = \langle Ab_1, Ab_2, \ldots, Ab_L \rangle$ and $Ag = \langle Ag_1, Ag_2, \ldots, Ag_L \rangle$. Different shape spaces can be used depending on the current environment:

Real-valued shape space: the attribute strings are real-valued vectors.
Integer shape space: the attribute strings are composed of integer values.
Hamming shape space: composed of attribute strings built out of a finite alphabet of length k.
Symbolic shape space: usually composed of different types of attribute strings, such as a 'name', a 'color', etc.

Based on this definition, the matching of antigens to antibodies can be described using their *affinity*. The affinity is related to the distance. For example, the Euclidean distance can be used:

$$D = \sqrt{\sum_{i=1}^{L} (Ab_i - Ag_i)^2} \tag{13.2}$$

Other distance measures such as Hamming or Manhattan can be used as well. The main application in computer science and engineering is anomaly detection. The normal behavior of a system is often characterized by a series of observations over time. The problem of detecting novelties, or anomalies, can be viewed as finding deviations of a characteristic property in the system. For computer scientists, the identification of computational viruses and network intrusions is considered one of the most important anomaly detection tasks.

One of the first AIS was presented in [20]. Based on this work, misbehavior detection and attack or intrusion detection systems were developed according to the working principles of the natural immune system [22, 23, 27]. Besides network security applications, the operation and control of multi-robot systems was addressed by AIS approaches. The collaborative behavior of robots collecting objects in an environment is difficult to optimize without central control. It was shown that an emerging collective behavior through communicating robots using an AIS overcomes some of the problems. The immune network theory was used to suppress or encourage robots behavior [28].

Misbehavior detection in mobile ad hoc networks

In ad hoc networks, each node serves as both an end system and a router. This allows to build dynamic on demand network topologies supporting mobile systems as well. Various routing protocols for mobile ad hoc networks have been proposed focusing on the efficiency in terms of route detection and maintenance (time, overhead, etc). This dynamic behavior allows – on the one hand – to enable sophisticated mobile applications. On the other hand, such dynamics also open ways to attack the network on the routing protocol layer. Such attacks might be initiated for denial of service reasons as well as for taking over the ad hoc network for private services. A third reason for misbehavior in ad hoc networks is the occurrence of faulty nodes. Either the system might be erroneous or the routing protocol might be incorrectly implemented. A misbehavior detection scheme using an artificial immune system has been developed [27], which works for DSR (dynamic source routing), a particular ad hoc routing protocol. The goal was to build a system that, like its natural counterpart, automatically learns and detects new misbehavior. It employs negative selection, an algorithm used by the natural immune system. In the original paper, the mapping of the natural immune system concepts such as self, antigen and antibody to a mobile ad hoc network is defined and the resulting algorithm for misbehavior detection is presented. The following elements have been defined:

Body: the entire mobile ad-hoc network.
Self-Cells: well behaving nodes.
Non-Self Cells: misbehaving nodes.
Antigen: Sequence of observed DSR protocol events recognized in sequence of packet headers. Examples of events are “data packet sent”, “data packet received”, “data packet received followed by data packet sent”, “route request packet received followed by route reply sent”.
Antibody: A pattern with the same format as the compact representation of antigen
Negative Selection: Antibodies are created during an offline learning phase. In a deployed system this would be done in a testbed with nodes deployed by an operator.

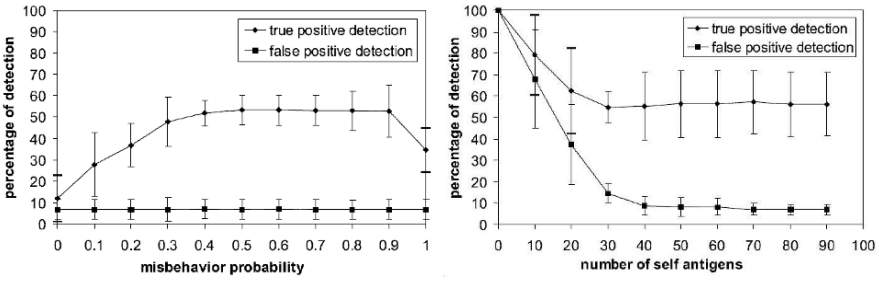


Fig. 13.7. Impact of misbehavior and parameter tuning [27]: Probability of correct detection of misbehaving nodes (true positive) and erroneous detection of well behaving nodes (false positive) vs. misbehavior probability for the misbehaving node (left) and number of self antigens collected for learning (right).

Since antigens represent traces of observed protocol events, such sequences would become very long in a short period of time. Therefore, all traces need to be limited by a time limit Δt for the observation interval. A typical sequence (the letters represent different protocol events) would look like this: $l_1 = (EAFBHHEDEBHDHDHDHD...)$. Then, a number of "genes" are defined. A gene is an atomic pattern used for matching. Typical examples are g_1=#E in sequence or g_2=#(H*D) in sequence. With this information, l_1 can be mapped to an antigen like this: l_2=(3 2 7 6). Finally, the antigens are encoded in binary representation. The numeric range of antigens is split into several intervals and the bit in the representation is set to one if the antigen belongs to this particular interval: l_3=(0000000010 0000000010 0000001000 0000001000).

As previously described, a matching function must be defined to associate antigens to antibodies. Antibodies have the same format as antigens (such as l_3), except that they may have any number of nucleotides equal to 1. An antibody matches an antigen if the antibody has a 1 in every position where the antigen has a 1. This approach has already been successfully demonstrated in [21]. It is used in this paper as a method that allows a detection system to have good coverage of a large set of possible non-self antigens with a relatively small number of antibodies. Antibodies are created randomly, uniformly over the set of possible antibodies. During negative selection, antibodies that match any self antigen are discarded.

The primary evaluation criteria for such detection approaches are the true positive detection rate and the false positive detection rate, i.e., the number of successfully identified misbehaving nodes and the number of accidentally mis-identified nodes, respectively. As shown in figure 13.7, the approach yields quite encouraging results.

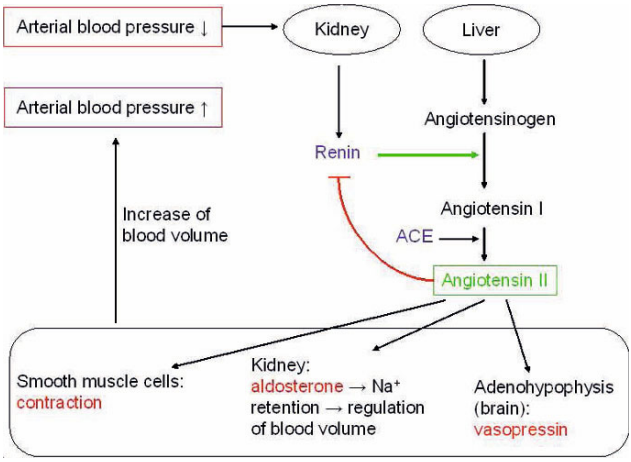


Fig. 13.8. Overview of the regulation of signaling cascades responsible for regulating the blood pressure [15].

13.4.3 Intercellular information exchange

Regarding efficient networking, investigations into the structure and organization of intercellular communication seem to be valuable. Molecular biology is the basis of all biological systems and features high specificity of information transfer. Interestingly, we find many similar structures in biology and in technology, especially in computer networking [25]. The primary concepts are intra- and intercellular signaling pathways and diffuse communication in large-scale structures. Considering the knowledge about molecular biology and its similarity to communication networks [14], it is possible to extract the following principles: efficient response to a request, shortening of information pathways, and directing of messages to an applicable destination.

The information pathways can be distinguished into local and remote. Local: a signal reaches only cells in the neighborhood. The signal induces a signaling cascade in each target cell resulting in a very specific response, which vice versa affects neighboring cells. Remote: a signal is released into the blood stream, which carries it to distant cells and induces a response in these cells, which then passes on the information or can activate helper cells (e.g. the immune system). Signals can appear in the form of particles, i.e., proteins and hormones, as well as of environmental conditions that can be observed and changed, e.g. the calcium concentration.

Inhibitors and promoters forming efficient feedback loops

An example for successful application of the described communication method in WSN is the feedback loop mechanism [15]. Here, the Angiotensin-based regulation process for the blood pressure was used to model the control loop

for an efficient regulatory process in an organism. In the case of decreasing arterial blood pressure, the kidney starts to produce a specific protein, renin. This protein initiates a cascade of conversions and activations, respectively. So it promotes the conversion of another protein (angiotensinogen) to a shorter one (now called angiotensin I), which is finally translated to angiotensin II. This protein represents the final response, which now has many effects on different cells in different organs in order to increase the blood pressure to its normal level. At the same time, a molecular negative feedback mechanism finishes the whole cellular reaction. If all receptor are bound by angiotensin II, the reaction is blocked, which in turn also blocks the primary conversion of angiotensinogen to angiotensin II in the way that the initial renin secretion is blocked. This process is shown in figure 13.8.

This process was adapted to work in a sensor network by using the following two concepts:

1. The density of the sensor network allows for alternative feedback loops via the environment: directly via the physical phenomena to be controlled by the infrastructure.
2. Indirect communication allows for more flexible organization of autonomous infrastructures and reduces the number of control messages.

In a sensor network, the control of activities requires information exchange between multiple nodes in the network. Such communication is needed for at least two reasons. First, the control information must be transported to the appropriate destination and, second, the destination must respond to the request by confirming the instructions. All conventionally designed network protocols for such a function follow the same principles. Transmission of a data packet destined for the particular target is initiated. State information is accumulated at several points in the network until a response packet is received which confirms the transaction. The paradigms for data transport in sensor networks are already changing. Directed diffusion, which was introduced in [18], has some interesting features: data-centric dissemination, reinforcement-based adaptation to the empirically best path, and in-network data aggregation and caching. Similar changes are expected for the control information flow which we are focusing on.

As learned from biology, a diffuse communication principle has been proposed [11, 15]. Each message to be sent is given a priority, which reflects the importance of achieving the particular task. Based on this priority, the message is sent to a percentage of the direct neighbors and an even lower percentage of remotely accessible nodes. This process is repeated until the desired job is confirmed running or the job is canceled globally. Thereby, a random factor is applied to the dispersion of information or, in particular, to the distribution of tasks. The benefit lies in better system efficiency and reliability, especially in unreliable multihop ad hoc wireless sensor networks.

13.5 Conclusion

In conclusion, it can be said that many approaches for bio-inspired networking have been studied and we can already see first impressive solutions and applications. Basically, the following mechanisms have been adapted to solve open issues in networking: feedback loops, i.e. positive feedback to initiate actuation or data aggregation, and negative feedback for network congestion control and smooth regulation; local state information for efficient data fusion, energy control, and clustering; and weighted probabilistic approaches for task allocation, controlled communication and congestion control. Finally, we are facing a multi-objective optimization process that balances between overhead (latency vs. energy) vs. predictability.

Self-organization mechanisms for communication and coordination between networked embedded systems need further research and development. There are many objectives and many directions, but similar solutions can be derived. Bio-inspired networking is a powerful approach among several others. Ongoing research objectives include efficient data dissemination, handling and storage in WSN as well as task allocation schemes and distributed control in SANET.

13.6 Acknowledgments

This work is partially a result of a collaborative research project conducted together with Dr. Bettina Krüger, Dept. of Physiology, University of Erlangen-Nuremberg, Germany.

References

1. K. Akkaya and M. Younis. Energy-aware routing of time-constrained traffic in wireless sensor networks. *Journal of Communication Systems, Special Issue on Service Differentiation and QoS in Ad Hoc Networks*, 17(6):663–687, 2004.
2. I. F. Akyildiz and I. H. Kasimoglu. Wireless sensor and actor networks: Research challenges. *Elsevier Ad Hoc Network Journal*, 2:351–367, October 2004.
3. I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–116, August 2002.
4. P. J. Bentley, T. Gordon, J. Kim, and S. Kumar. New trends in evolutionary computation. In *Congress on Evolutionary Computation (CEC-2001)*, pages 162–169, Seoul, Korea, May 2001.
5. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, 1999.
6. D. Culler, D. Estrin, and M. B. Srivastava. Overview of sensor networks. *Computer*, 37(8):41–49, August 2004.
7. S. K. Das, N. Banerjee, and A. Roy. Solving otimization problems in wireless networks using genetic algorithms. In *Handbook of Bio-inspired Algorithms*. 2004.

8. L. N. de Castro and J. Timmis. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer, Berlin, 2002.
9. G. Di Caro and M. Dorgio. AntNet: Distributed stigmergetic control for communication networks. *Journal of Artificial Intelligence Research*, 9:317–365, December 1998.
10. G. Di Caro, F. Ducatelle, and L. M. Gambardella. AntHocNet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Transactions on Telecommunications, Special Issue on Self-organization in Mobile Networking*, 16:443–455, 2005.
11. F. Dressler. Locality driven congestion control in self-organizing wireless sensor networks. In *3rd International Conference on Pervasive Computing (Pervasive 2005): International Workshop on Software Architectures for Self-Organization, and Software Techniques for Embedded and Pervasive Systems (SASO+STEPS 2005)*, Munich, Germany, May 2005.
12. F. Dressler. Self-organization in ad hoc networks: Overview and classification. Technical Report 02/06, University of Erlangen, Dept. of Computer Science 7, March 2006.
13. F. Dressler. *Self-Organization in Sensor and Actor Networks*. John Wiley & Sons Ltd., West Sussex, 2007.
14. F. Dressler and B. Krüger. Cell biology as a key to computer networking. In *German Conference on Bioinformatics 2004 (GCB'04), Poster Session*, Bielefeld, Germany, October 2004.
15. F. Dressler, B. Krüger, G. Fuchs, and R. German. Self-organization in sensor networks using bio-inspired mechanisms. In *18th ACM/GI/ITG International Conference on Architecture of Computing Systems - System Aspects in Organic and Pervasive Computing (ARCS'05): Workshop Self-Organization and Emergence*, pages 139–144, Innsbruck, Austria, March 2005.
16. M. Eigen and P. Schuster. *The Hypercycle: A Principle of Natural Self Organization*. Springer, Berlin, 1979.
17. S. Hofmeyer and S. Forrest. Architecture for an artificial immune system. *Evolutionary Computation*, 8(4):443–473, 2000.
18. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCOM'00)*, pages 56–67, Boston, MA, USA, August 2000.
19. J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, 2001.
20. J. O. Kephart. A biologically inspired immune system for computers. In *4th International Workshop on Synthesis and Simulation of Living Systems*, pages 130–139, Cambridge, Massachusetts, USA, 1994. MIT Press.
21. J. Kim and P. J. Bentley. The artificial immune system for network intrusion detection: An investigation of clonal selection with negative selection operator. In *Congress on Evoloutionary Computation (CEC-2001)*, pages 1244–1252, Seoul, Korea, July 2001.
22. J. Kim and P. J. Bentley. An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 1330–1337, San Francisco, CA, July 2001.

23. J. Kim and P. J. Bentley. Towards an artificial immune system for network intrusion detection. In *Congress on Evolutionary Computation (CEC-2002)*, pages 1015–1020, Honolulu, May 2002.
24. M. J. B. Krieger, J.-B. Billeter, and L. Keller. Ant-like task allocation and recruitment in cooperative robots. *Nature*, 406:992–995, August 2000.
25. B. Krüger and F. Dressler. Molecular processes as a basis for autonomous networking. *IPSI Transactions on Advances Research: Issues in Computer Science and Engineering*, 1(1):43–50, January 2005.
26. T. H. Labella and F. Dressler. A bio-inspired architecture for division of labour in sanets. In *1st IEEE/ACM International Conference on Bio-Inspired Models of Network, Information and Computing Systems (IEEE/ACM BIONETICS 2006)*, Cavalese, Italy, December 2006.
27. J.-Y. Le Boudec and S. Sarafijanovic. An artificial immune system approach to misbehavior detection in mobile ad-hoc networks. In *First International Workshop on Biologically Inspired Approaches to Advanced Information Technology (Bio-ADIT2004)*, pages 96–111, Lausanne, Switzerland, January 2004.
28. D. W. Lee, H. B. Jun, and K. B. Sim. Artificial immune system for realization of cooperative strategies and group behavior in collective autonomous mobile robots. In *4th International Symposium on Artificial Life and Robotics*, pages 232–235, 1999.
29. J. Suzuki and Y. Yamamoto. Biologically-inspired autonomous adaptability in a communication endsystem: An approach using an artificial immune network. *IEICE Transactions on Information and Systems*, E84-D(12):1782–1789, December 2001.
30. H. von Foerster. On self-organizing systems and their environments. In M. C. Yovitts and S. Cameron, editors, *Self-Organizing Systems*, pages 31–50. Pergamon Press, 1960.

14

Subspace Image Representation for Facial Expression Analysis and Face Recognition and its Relation to the Human Visual System

Ioan Buciu[1,2] and Ioannis Pitas[1]

[1] Department of Informatics, Aristotle University of Thessaloniki GR-541 24, Thessaloniki, Box 451, Greece.
pitas@zeus.csd.auth.gr

[2] Electronics Department, Faculty of Electrical Engineering and Information Technology, University of Oradea 410087, Universitatii 1, Romania.
ibuciu@uoradea.ro

Summary. Two main theories exist with respect to face encoding and representation in the human visual system (HVS). The first one refers to the dense (holistic) representation of the face, where faces have "holon"-like appearance. The second one claims that a more appropriate face representation is given by a sparse code, where only a small fraction of the neural cells corresponding to face encoding is activated. Theoretical and experimental evidence suggest that the HVS performs face analysis (encoding, storing, face recognition, facial expression recognition) in a structured and hierarchical way, where both representations have their own contribution and goal. According to neuropsychological experiments, it seems that encoding for face recognition, relies on holistic image representation, while a sparse image representation is used for facial expression analysis and classification. From the computer vision perspective, the techniques developed for automatic face and facial expression recognition fall into the same two representation types. Like in Neuroscience, the techniques which perform better for face recognition yield a holistic image representation, while those techniques suitable for facial expression recognition use a sparse or local image representation. The proposed mathematical models of image formation and encoding try to simulate the *efficient storing, organization* and *coding* of data in the human cortex. This is equivalent with embedding constraints in the model design regarding dimensionality reduction, redundant information minimization, mutual information minimization, non-negativity constraints, class information, etc. The presented techniques are applied as a feature extraction step followed by a classification method, which also heavily influences the recognition results.

Key words: Human Visual System; Dense, Sparse and Local Image Representation and Encoding, Face and Facial Expression Analysis and Recognition.

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_14,

14.1 Introduction

In the human visual system (HVS), the visual image propagates from retina to the inferotemporal (IT) cortex, where the visual signal is decoded and processed. The question of how the human brain stores the image patterns in its visual cortex and how many pattern-specific neurons are activated and respond to a specific visual stimulus is a fundamental problem of psychology. A huge amount of research has been done in the attempt to understand how information captured by sensory channels is represented in the brain at different levels. Nowadays, this task is not only a concern of psychologists but also of image processing and computer vision experts. The pattern features that must be extracted from the data is a task-dependent matter. Among the huge visual data our eyes are overwhelmed by, facial images receive particular attention, due to their biological and sociological significance. This fact explains why the face analysis enjoys an important status with psychologists, anthropologists, neuroscientists and computer scientists alike. It is well known that in social interaction the human face constitutes the primary source of information for person recognition. As far as the computer scientists are concerned, the development of an automated face recognition system is necessary in order to cope with a large and complex area of applications, such as biometrics for security, surveillance, banking, law enforcement, video indexing, human-computer interaction, etc.

Another aspect closely related to face analysis is provided by facial expressions. Emotions can typically be conveyed by facial expressions. Like for face recognition, the recognition of facial expressions is a subject of interdisciplinary research. From the psychological and anthropological perspectives the following questions are addressed: What information does a facial expression typically convey? Can there be emotions without facial expression? Can there be facial expression without emotions? How do individuals differ in their facial expression of emotions [23]? It is well known among psychologists that the social context is dominated by language. However, the language alone is insufficient when it comes to successful social interaction. Plenty of communication comes through non-verbal communication. As Mehrabian suggested in [40], people express only 7% of the messages through a linguistic language, 38% through voice, and 55% through facial expressions.

A good understanding of the underlying process that governs the appearance of expressions is necessary in order to develop an appropriate facial image representation. In a human-computer interaction task, this constitutes the input to a human facial expression recognition system with satisfactory classification performance and, eventually, to artificial facial expression synthesis on an avatar for friendlier human-computer interface.

This chapter is organized as follows. Face encoding in the HVS from the neuroscience perspective is described in section 14.2. It starts with the analysis of dense, sparse and local face image representation followed by examples of these representations for face and facial expression recognition. A com-

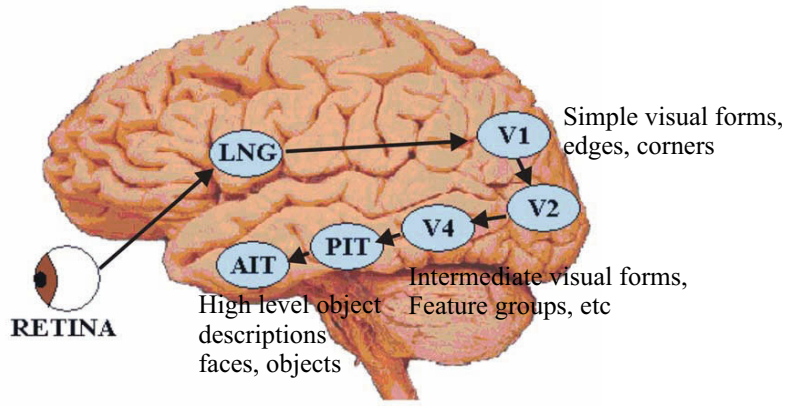


Fig. 14.1. Visual pathway in HVS. Information passes from the retina to the lateral geniculate nucleus (LGN) before arriving in cortical area V1. Further processing occurs in areas V2 and V4 and the posterior and anterior inferotemporal (IT) cortex (PIT and AIT).

puter vision analysis of face and facial expression recognition approaches is undertaken in section 14.3, where both dense and sparse image representation techniques are presented. The chapter ends with a discussion in section 14.4.

14.2 Face encoding in human visual system: a neuroscience view point

14.2.1 Dense and sparse image representation

How can we represent facial image information so that it can activate a representation in human memory under various conditions? Is human perception of a facial image based on its parts or it is viewed as a whole? Despite the huge amount of psychological research done in this respect, there is no general consensus in answering these questions. Rather, the answer to the problem of how the visual cortex understands complex objects, and, in particular human faces, is a controversial one. In recent years it has been argued from a visual neuroscience viewpoint that the architecture of the visual cortex suggests a hierarchical organization, in which neurons become selective to progressively more complex aspects of image structure.

Figure 14.1 depicts the visual pathway starting from the retina and ending at the two regions of inferotemporal cortex – IT (PIT and AIT). Multiple representations of the retinal space are mapped onto the cortex in a manner that preserves the visual topology. These representations define the visual modules: V1, V2, V4, IT. Whereas the earliest stages of the human visual system

(e.g. retina and V1 neurons) seem to produce a local distributed image representation, as we step into the higher visual system levels (such as V2, V4 areas or IT), the neurons have increasing receptive field sizes, being able to tackle increasingly complex stimuli [34]. Concerning neuroscience, the type of image encoding is related to the number of neurons that are active (respond) to a certain piece of information represented by a specific sensory stimulus caused by the image. We refer to a *local* image code when only a single individual specific cell is activated. We have a *dense* image code, when a large cell population with overlapping sensory input is activated and contributes to the image representation. The local code is "computed" very fast and occupies little memory. However, it cannot generalize (i.e., when trained with a sufficient number of samples, it achieves satisfactory results when tested on samples from the training set, but performs poorly on new test samples not belonging to the training set) [27]. This is caused by the fact that the input-output unit association (as in single-layer neural networks) is very weak and a new sample cannot be linked with the old association learned during the training process. On the other hand, a system based on a dense code suffers from slow training, requires heavy training and is likely to produce redundant image representations. However, it has a large capacity of making new associations. In between local and dense codes, we have the *sparse* image codes, where only a fraction of a large neuronal population is active. It is a trade-off between dense and local image codes, combining their advantages and trying to eliminate their drawbacks. Dense and local codes are closely related to holistic and local (component, or part-based) image representation and processing. The term *holistic* refers to an image representation which stores a face as a perceptual whole, without explicitly specifying its parts (components). The term *component* describes the separated parts of the face (e.g. eyes, nose, mouth, chin) that are perceived independently as distinct parts of the whole.

Atick and Redlich [1] support the idea of a dense image code within the HVS and argue for compact, densely decorrelated codes for image representation. They have demonstrated that receptive fields of retinal ganglion cells can be viewed as local "whitening" filters that remove second-order correlations between image pixels. Bandpass, multiscale and oriented receptive fields of V1 neurons may also be considered as filters that remove second-order correlation, the way Principal Component Analysis (PCA) does. Regarding human facial images, PCA has a certain appeal as a psychological model of face perception and memory. For example, the application of principal components is consistent with psychological evidence that the PCA of a set of face images accounts for some aspects of human memory performance, as shown by Valentine [56].

Ample evidence for sparse image coding within HVS has been collected by other researchers. They argue for a sparse image representation that leads to "efficient coding" in the visual cortex [26]. Since spatial receptive fields of simple cells (including V1 neurons) have been reasonably well described physiologically as being localized, oriented and bandpass, Olshausen and Field [42]

Fig. 14.2. Thatcher illusion [54]. The eyes and mouth of Margaret Thatcher (former Prime Minister of England whose face is depicted in the left hand image) have been inverted relative to the rest of her face (middle and right hand image). When the picture is viewed upright the face appears (middle image) highly grotesque. This strange distortion is much less evident when the face is turned upside-down (right hand image). Reproduced with permission from [54].

show that efficient image coding can be produced by considering an approach where the image is described by a small number of descriptors. These descriptors can be found by applying principles such as entropy minimization [3], which is equivalent to minimizing the mutual information in a such a way that the higher-order correlation between images is removed. Palmer [45] and Wachsmuth et al. [58] have drawn psychological and physiological evidence for parts-based object representations in the brain. Biederman came up with the theory of recognition-by-components (RBC) [7]. Empirical tests support his idea that complex objects are segmented into components called 'geons', which are further used by humans for image understanding. The "Thatcher illusion" presented in [54] suggests that parts of the face are processed independently. As depicted in figure 14.2 the rotated face seems to be processed by matching parts, which could be the reason why the face looks normal when turned upside-down.

Another sparse model of the neural receptive fields in early visual system was provided by Gabor functions [37]. A Gabor function is a sinusoid windowed with a Gaussian function. Its size, frequency and orientation can be manipulated to produce a wide range of different receptive field models. By convolving the image with the Gabor functions, a new image representation can be achieved with features that are sparse, oriented and localized.

Despite the large number of experiments and investigations, it is still unclear which approach (holistic or sparse) best fits for face image processing in terms of optimal image coding. This process is rather task-dependent [8, 16]. For instance, some evidence has been found that face identification and facial expression recognition are two independent tasks based on different repre-

sentations and processing mechanisms. This hypothesis comes from the dissociation of these two processes found in brain damaged patients. It leads to the hypothesis that multiple representations of faces may reside in the visual cortex. The IT area of the temporal lobe contains neurons whose receptive fields cover the entire visual space. It also contains specialized neurons (face cells) that are selectively tuned to faces. There are dedicated areas in temporal cortical lobe that are responsible for processing information about faces [20, 47, 33]. It was also found that in AIT areas neurons with responses related to facial identity recognition exist, while other neurons (located in the superior temporal sulcus) are specialized to respond only to facial expressions [28].

14.2.2 Face and facial expression recognition

Experimentally, evidence for both sparse and dense face representations in the HVS has been found by neurophysiologists. However, the contribution of each representation depends on the task to be processed. While face recognition seems to favor a dense image representation (hence producing a holistic appearance of the faces), a more sparse (or even local) image representation has been found to account for facial expression analysis. This difference has been noticed in several works. Tanaka and Farah presented evidence in favor of a holistic process involved in face recognition [51]. These findings are stressed by the work of Farah et al. [25], which brings new evidence that part-based shape representation for faces has less impact in recognition than the holistic one. Furthermore, their theory is emphasized by the work of Dailey and Cottrell [19].

Contrary to representation for face identification, the work of Ellison and Massaro [24] has revealed that facial expressions are better represented by facial parts, suggesting non-holistic representation. This is consistent with research results showing that human subjects respond to information around the eyes independently from variation around the mouth and they are able to recognize and distinguish isolated parts of faces. The dissociation between face and facial expression recognition is also noted by Cottrell et al. [16] who found that PCA (which produces eigenfaces) performs well for face recognition but eigeneyes and eigenmouth (nonholistic eigenfeatures) perform better in recognizing expressions than eigenfaces, suggesting that eigenfeatures might transmit facial expression information. One of the techniques successfully applied to classify facial actions related to facial expressions, was Independent Component Analysis (ICA), which looks for components as independent as possible from each other and produces image features that can mimic the output of V1 receptive fields with orientation selectivity, bandpass and scaling properties [6]. In a direct comparison between PCA and ICA, Draper et al. [22] found that facial identity recognition performance is better when the features are represented by a holistic approach (PCA) while an approach based on more localized features (ICA) performs better for facial action recognition.

14.3 Face and facial expression analysis: a computer-vision view point

The HVS often serves as an informal standard for evaluating systems. Therefore, not surprisingly, most face analysis approaches rely on biologically inspired models. To be plausible, these computer vision models have to share some characteristics and constraints with their organic models. A common characteristic of the proposed HVS models is the *dimensionality reduction* principle of image space. This physical constraint is easily understood if we consider, for instance, that an image of 64×64 pixels has dimensionality 4096. It is commonly accepted that the intrinsic dimensionality of the space of possible faces is much lower than that of the original image space. Basically, the latent variables incorporated there are discovered by decomposing (projecting) the image onto a linear (nonlinear) low dimensional image subspace. By reference to neuroscience, the receptive fields can be modeled by the basis images of the image subspace and their firing rates can be represented by the decomposition coefficients [42].

In order to generate the subspace image representation produced by the methods presented in this chapter, 164 image samples from the Cohn-Kanade AU-coded facial expression database [32] have been used. The number of basis images (subspace) was chosen to be 49.

14.3.1 Holistic image representations

As already mentioned, one of the most popular techniques for dimensionality reduction is PCA, which represents faces by their projection onto a set of orthogonal axes (also known as principal components, eigenvectors, eigenfaces, or basis images) pointing into the directions of maximal covariance in the facial image data. The basis images corresponding to PCA are ordered according to the decreasing amount of variance they represent, i.e., the respective eigenvalues. PCA-based Representations of human faces give us a dense code and the post-processed images have a holistic ("ghostlike") appearance, as can be seen from the first row of figure 14.3.

The principal components produce an image representation with minimal quadratic error. One of the proposed general organizational principles of the HVS refers to redundancy reduction. In PCA, this is achieved by imposing orthogonality among the basis images, thus redundancy is minimized. The nature of information encoded in the basis images was analyzed by O'Toole et al. [43] and Valentin and Abdi [57]. They found that the first basis images (containing low spatial frequency information) were most discriminative for classifying gender and race, while the basis images with small eigenvalues (corresponding to a middle range of spatial frequencies) contain valuable information for face recognition. This is coherent with findings that the face cells within HVS respond most strongly to face images containing energy within a middle range of spatial frequency between 4 and 32 cycles per image [49].

Fig. 14.3. Holistic subspace image representation. From top to bottom, each row depicts the first 10 basis images (out of 49) corresponding to PCA, FLD, ICA2 and NMF. For PCA the basis images are ordered by decreasing variance, for ICA2 and NMF by decreasing kurtosis.

Also, different components were found to be responsible for encoding identity and facial expression by Cottrell et al. in [15].

PCA has been successfully applied to face recognition [17], [5] and [55], and facial expression recognition, respectively [18], [44] and [14]. One statistical limitation of PCA is that it only decorrelates the input data (second-order statistics) without addressing higher-order statistics between image pixels. It is well known and accepted that, at least for natural stimuli, important information (e.g. lines, edges) is encoded in the higher-order statistics. Another limitation is related to the poor face recognition results for PCA when the faces are recorded under strong illumination variations.

Another holistic subspace image representation is obtained by a class-specific linear projection method based on Fisher's linear discriminant (FLD) [5]. This technique projects the images onto a subspace where the classes are maximally separated by maximizing the between-classes scatter matrix and minimizing the within-class scatter matrix at the same time. The basis images obtained through FLD are depicted in the second row of figure 14.3. This approach has been shown to be efficient in recognizing faces, outperforming PCA. Although this method seems to be more robust than PCA when small variation in illumination conditions appears, it fails in case of strong illumination changes. This is due to the assumption of linear separability of the classes. This assumption is violated, when strong changes in illumination occur. Another drawback of this method is that it needs a large number of training image samples for reasonable performance. Furthermore, the projection onto too few subspace dimensions does not guarantee the linear separability of the classes, hence the method will yield poor performance.

Along with redundancy reduction, another principle of HVS image coding mechanism is given by phase information encoding. It was shown by Field [26]

that methods relying only on second order statistics capture the amplitude spectrum of images but not the phase. The phase spectrum can be captured by employing higher order statistics. This has been proven to be accomplished by extracting independent image components [6]. There are several optimization principles taken into account when extracting independent components. The one described in [6] is based on the maximal information transfer between neurons and, among all the proposed ICA techniques, it seems to be the most plausible approach from the neuroscientific point of view.

Bartlett et al. [4] used two ICA configurations to represent faces for recognition. PCA was carried out prior to ICA for dimensionality reduction. An intermediate step for "whitening" the data has been introduced between PCA and ICA processing. The data were then decomposed into basis images and decomposition coefficients. Their second ICA configuration (ICA2) yields holistic basis images very similar to those produced by PCA. Such basis images are depicted in the third row of figure 14.3. In that case, ICA is applied to the projection matrix containing the principal components. Under this architecture, the linear decomposition coefficients are as independent as possible.

A recently proposed subspace image decomposition technique is Nonnegative Matrix Factorization (NMF) [36], which allows the data to be described as a combination of elementary features that involve only additive parts to form the whole. Both basis images and decomposition coefficients are constrained to be non-negative. Allowing only addition for recombining basis images to produce the original data is justified by the intuitive notion of combining parts to form the whole image. Another argument for imposing non-negativity constraints comes from neuroscience and is related to the non-negative firing rate of neurons. Finally, the positivity constraint arises in many real image processing applications. For example, the pixels in a grayscale image have non-negative intensities. Euclidean distance and Kullback-Leibler (KL) divergence were originally proposed as objective functions for minimizing the difference between the original image data and their decomposition product. Although, theoretically, the decomposition constraints tend to produce sparse image representations of basis images by composing the parts in an additive fashion, this is not always the case. It has been noticed in several works that, for some databases, the NMF decomposition rather produces a holistic image representation [38, 12, 30]. The representation could be affected by the imprecise image alignment procedure performed on the original database prior to NMF. It is known that the subspace techniques are generally sensitive to image alignment (registration). As noted in the last row of figure 14.3, for Cohn-Kanade database images, the basis images retrieved by NMF have a holistic appearance. A measure for quantifying the degree of sparseness in image representations is provided by the normalized kurtosis. If the basis images are stored as columns of a matrix $\mathbf{Z}$ the kurtosis of a base image $\mathbf{z}$ is defined as $k(\mathbf{z}) = \frac{\sum_i (z_i - \overline{z})^4}{(\sum_i (z_i - \overline{z})^2)^2} - 3$, where z_i are the the elements of $\mathbf{z}$ (pixels of base image) and $\overline{z}$ denotes the sample mean of $\mathbf{z}$. The average normalized kur-

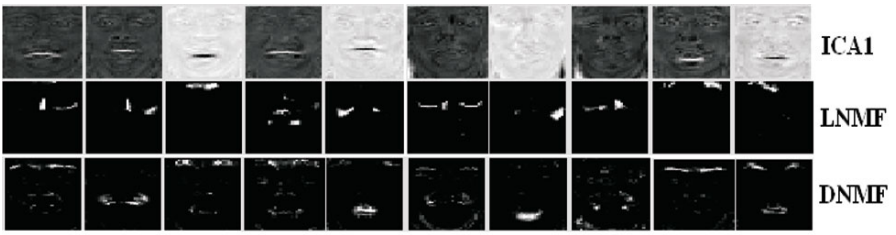


Fig. 14.4. Sparse subspace image representation. From top to bottom, each row depicts the first 10 basis images (out of 49) corresponding to ICA1, LNMF and DNMF, respectively. The basis images are ordered by decreasing kurtosis.

tosis for the 49 basis images are: $\overline{k}_{PCA} = 1.22$, $\overline{k}_{FLD} = 1.23$, $\overline{k}_{ICA2} = 0.93$, $\overline{k}_{NMF} = 5.93$. Thus, by far, NMF is the sparsest representation among ones represented in figure 14.3.

14.3.2 Sparse image representations

The first ICA (ICA1) configuration produces independent basis images [4]. In this case, ICA is applied to the projection coefficients of PCA. Entropy minimization leads to a highly kurtotic distribution of basis image pixels, most of them having zero value, thus producing a sparse representation, as can be seen in the first row of Figure 14.4. Another representation, Local Non negative Matrix Factorization (LNMF) [38] enhances the sparseness of basis images by generating much more sparse, even localized and oriented image features. The extremely sparse basis images resulting from the LNMF approach are depicted in the second row of figure 14.4. The proposed approach uses the KL divergence as objective function to be minimized [38]. In addition to the non negativity constraints imposed for both decomposition factors, the redundant information is minimized by adding orthogonality constraints in the basis images formation. Furthermore, two more terms are added to the objective function for maximizing both sparseness and total activity and retaining only the most "expressive" image components [38]. In direct comparison for face recognition LNMF outperformed NMF [38, 12]. Buciu and Pitas further modified the LNMF algorithm in [11] and proposed a supervised NMF approach called Discriminant Nonnegative Matrix Factorization (DNMF) for facial expression classification. Besides the common constraints borrowed from NMF and LNMF, its underlying objective function also contains terms referring to discriminant class information. The basis images found by running the algorithm on image samples are sparse, oriented and localized, as can be seen in the last row of figure 14.4. DNMF is differentiated from the other NMF algorithms in that its facial basis images emphasize the salient facial features (eyes, eyebrows, mouth), when the images are labeled according to

facial expression. These features convey the most discriminative information and are of great relevance for facial expression recognition. As can be seen by visual comparison of the basis images in figure 14.4, DNMF preserves local spatial information of salient facial features (that are almost absent in the case of LNMF) while it discards information less important for expression analysis (e.g., nose and chin, which is not the case for NMF) by incorporating class information. The average normalized kurtosis for the 49 basis images are: $\overline{k}_{ICA1} = 17.26$, $\overline{k}_{LNMF} = 49.16$, $\overline{k}_{DNMF} = 31.69$. The preservation of the spatial facial topology correlates well with the findings of Tanaka et al. [52], who argued that some face cells require the correct spatial feature configuration in order to be activated for facial expression recognition. Interestingly, DNMF seems to resemble many characteristics of the neural receptive fields [13]. The three NMF approaches have been applied to classify facial expressions [11] and to face recognition [10]. The DNMF approach was found to perform best for facial expression recognition, a fact that is indicating the role of sparse image representations. However, for face recognition, DNMF did not achieve the best performance compared to the other two approaches.

Local Feature Analysis (LFA) is another biologically inspired method that retrieves local image features [46]. To exploit image redundancy, LFA is used to extract a set of topographic local features defined by kernel filters that are optimally matched to the second-order statistics from the global PCA modes. They are found by minimizing the image reconstruction error and by using a process called sparsification [46]. To achieve this, a LFA neural network is employed, where the active units found by LFA are sparsely distributed. The selection of the spatial support of 100 filters found by LFA is shown in Figure 14.5 over a mean face from the experiment database.

One of the two most popular techniques for face recognition are known as "elastic graph matching" [35], and its relative, named "elastic bunch graph matching" [59]. "Elastic bunch graph matching" is based on applying a set of Gabor filters to special representative landmarks on the face (corners of the eyes and mouth, the contour of the face). Gabor filters represent the multiscale nature of receptive fields, as each component has a unique combination of orientation, frequency tuning and scale. The face is represented by a list of values that comprise the amount of contrast energy that is present at spatial frequencies, orientations and scales included in the jet. For recognition, each face is compared with any other one with a similarity metric that takes into account the spatial configuration of the landmarks. It has been noted that the similarity metric involved in this approach is in line with the one used by the HVS [8]. Similar Gabor filters have been used by Würtz in [60] to extract local features that are robust to translations, deformations, and background changes. Each image is convolved with a set of different Gabor kernels, followed by amplitude thresholding and discarding all units influenced by the background. Once the local features are extracted, four matching approaches (namely, multidimensional template matching, global matching, mapping refinement and phase alignment) are employed and combined to form corre-

Fig. 14.5. A set of 100 optimally localized topographic kernel filters found by Local Feature Analysis (LFA). The clusters of these filters are located around the fiducial points represented by eyes, eyebrows and mouth of the mean face. The algorithm from [46] has been applied to samples from the Cohn-Kanade database.

spondence maps used further for the face recognition task. To remove the weak correspondence points a relative similarity threshold is introduced, thus a final correspondence map is obtained. The combination of these matching approaches leads to a hierarchical structure of the algorithm with several decision levels, where the correspondence maps obtained by this method was proved to be very reliable. Furthermore, the Gabor functions have been successfully used for facial expression synthesis or recognition. The convolution of images with the set of Gabor filters can be performed either at the location of fiducial points (landmarks) [64] or, alternatively, the Gabor filters can be applied to the entire face image instead to specific face regions [9]. Figure 14.6 presents the result of the convolution of a set of 40 Gabor filters (5 frequencies and 8 orientations) with a sample image from the Cohn-Kanade database. The features extracted by the Gabor filters are localized and oriented [9].

One drawback of this feature extraction technique is the manual annotation of landmarks when the Gabor filters are applied to specific fiducial points. To overcome this issue, Heinrichs et al. [29] reduce the manual annotation to only one single image from which a self-organizing selection strategy builds up the bunches by adding the most similar face to the bunch graph and then the matching is recomputed. Another improvement is the replacement of the resulting Gabor wavelet bunches by principal components of the nodes of all training images. An enhancement in both the precision of landmark localization and face recognition accuracy was obtained with this new approach. An extension of "elastic bunch graph matching" with a new application was recently proposed by Tewes et al. in [53]. They have developed a flexible object model using Gabor-wavelet-labeled graphs to synthesize facial expression.

Fig. 14.6. The output of an image sample from the database convolved (filtered) with 40 Gabor filters (5 frequencies and 8 orientations).

The graphs are then parameterized to allow flexible facial expression generation, where the expression parameters are viewed as a graph function. An overview of "elastic bunch graph matching" approach and its relationship to the Organic Computing paradigms is presented in [61].

A representative research work for facial expression recognition was conducted by Donato et al. [21], who investigated several holistic and sparse image representation techniques and measured their performance. Their work shows that the extraction of sparse features from the entire face space by convolving each image with a set of Gabor filters having different frequencies and orientations can outperform other methods that invoke the holistic representation of the face, when it comes to classify facial actions, closely related to facial expressions. They achieved the best recognition results by using ICA and Gabor filters. However, they also found that other local spatial approaches, like local PCA and PCA jets provide worse recognition accuracy than, for example, Fisher Linear Discriminant (FLD), which is a holistic approach.

14.4 Discussion

It has been argued that the tuning of the temporal cortex neurons that respond preferentially to faces represents a trade-off between fully distributed

encoding (holistic or global representation, as PCA, FLD, ICA2, NMF result) and a grandmother cell type of encoding (local representation, achieved by LNMF) [50]. Psychophysiologically, one single pixel representation is similar to having a grandmother cell where a specific image is represented by one neuron. Among the approaches presented in this chapter, Gabor, ICA and DNMF turns out to be the most suitable biologically plausible models used in computer vision. However, it has to be noted that the DNMF approach is a relatively a new method and yet insufficiently investigated. In recent studies [11], it showed superior facial expression classification performance, when compared to Gabor, NMF, LNMF, and ICA approaches. However, its performance was not the one expected when applied to face recognition. More research has to be done in this regard. As we show in this chapter, one of the most popular techniques used for face and facial recognition tasks is PCA. When higher-order statistics are to be extracted and processed, ICA is chosen over PCA, which also seems to resemble the neuroscientific paradigms. However, a question related to ICA and PCA for face and facial expression recognition arises. Is ICA really better for these tasks than PCA? First of all, regardless of the feature extraction technique, the recognition results are not solely dependent on subspace image representation. It is also up to the classifier involved in the final step of the recognition task. A nearest neighbor classifier is usually chosen employing various similarity measures (distance metrics), such as L_1 (city block), L_2 (Euclidean), Mahalanobis or cosine distance. Several metrics favor the holistic representations, while others favor the sparse ones. This is mainly the reason, why, in the PCA-ICA debate, several works reported ICA outperforming PCA [21, 4, 22, 62, 39], while in other works ICA was found inferior to PCA [2], or, finally, no difference was found between them [41, 31]. Recently, new results on the ICA - PCA debate for face recognition have been revealed through the work conducted by Yang et al. [63]. They have repeated the experiments from [4] and have found that it is the "whitening" process (the intermediate step between PCA and ICA) that is responsible for the difference in the classification performance. Thus, as conclusion, ICA has an insignificant effect on the performance of face recognition . ICA was applied to cope with face recognition assuming that important information to discriminate between identities is contained in high-order image statistics, statistics that the PCA cannot retrieve. Interesting evidence that supports the observation that the elimination of high-order correlations between image pixels could not be so important for the neural receptive fields was brought by Petrov and Li [48]. They investigated local correlation and information redundancy in natural images and found that the removal of higher-order correlations between the image pixels increased the efficiency of image representation insignificantly. Accordingly, their results suggest that the reduction of higher-order redundancies than the second-order ones is not the main cause of receptive field properties of neurons in V1.

Still two main questions remain: Are the holistic subspace image representations more appropriate for face recognition and the sparse subspace image

representation more suitable for facial expression recognition? And, if it is so, which similarity measures should these representations be combined with in order to achieve the best recognition performance? Unfortunately, one of the shortcomings in neuroscience literature on face analysis is that no psychological measures for similarity of face image features (neither holistic nor sparse) exist. A large number of psychological studies are required in order to validate an existing subspace image representation model in combination with the optimal choice of the similarity metric.

Acknowledgments

This work has been conducted in conjunction with the "SIMILAR" European Network of Excellence on Multimodal Interfaces of the IST Program of the European Union (www.similar.cc).

References

1. J. J. Atick and A. N. Redlich. What does the retina know about the natural scene? *Neural Computation*, 4:196–210, 1992.
2. K. Baek, B. A. Draper, J. R. Beveridge, and K. She. PCA vs. ICA: A comparison on the FERET data set. *Joint Conference on Information Sciences, Durham, N.C.*, 2002.
3. H. Barlow. Unsupervised learning. *Neural Computation*, 1(3):295–311, 1989.
4. M. S. Bartlett, J. R. Movellan, and T. J. Sejnowski. Face recognition by independent component analysis. *IEEE Trans. Neural Networks*, 13(6):1450-1464, 2002.
5. P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman. Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(7):711-720, 1997.
6. A. J. Bell and T. J. Sejnowski. The "independent components" of natural scenes are edge filters. *Vision Research*, 37:3327–3338, 1997.
7. I. Biederman. Recognition-by-components: a theory of human image understanding. *Psychological Review*, 94(2):115–147, 1987.
8. I. Biederman and P. Kalocsai. Neurocomputational bases of object and face recognition. *Philosophical Transactions of the Royal Society of London*, 352(10):1203–1219, 1997.
9. I. Buciu, C. Kotropoulos, and I. Pitas. ICA and Gabor representation for facial expression recognition. in *Proc. 2003 IEEE Int. Conf. Image Processing*, pages 855–858, 2003.
10. I. Buciu, N. Nikolaidis, and I. Pitas. A comparative study of NMF, DNMF, and LNMF algorithms applied for face recognition. in Proc. *Second IEEE-EURASIP International Symposium on Control, Communications, and Signal Processing (ISCCSP)*, 2006.
11. I. Buciu and I. Pitas. A new sparse image representation algorithm applied to facial expression recognition. In *Proc. IEEE Workshop on Machine Learning for Signal Processing*, pages 539–548, 2004.

12. I. Buciu and I. Pitas. Application of non-negative and local non-negative matrix factorization to facial expression recognition. *Int. Conf. on Pattern Recognition*, pages 228–291, 2004.
13. I. Buciu and I. Pitas. DNMF modeling of neural receptive fields involved in human facial expression perception. *Journal of Visual Communication and Image Representation*, 17(5):958–969, Oct. 2006
14. A. J. Calder, A. M. Burton, P. Miller, A. W. Young, and S. Akamatsu. A principal component analysis of facial expressions. *Vision Research* , 41:1179–1208, 2001.
15. G. Cottrell, K. Branson, and A. J. Calder. Do Expression and Identity Need Separate Representations? In *Proc. of the 24th Annual Cognitive Science Conference*), pages 238–243, Dordrecht, Kluwer, 1990.
16. G. W. Cottrell, M. N. Dailey, C. Padgett, and R. Adolphs. Is all face processing holistic? The view from UCSD. In M. Wenger, J. Townsend (Eds.), *Computational, Geometric, and Process Perspectives on Facial Cognition: Contexts and Challenges*. Erlbaum, 2003.
17. G. Cottrell and M. Fleming. Face recognition using unsupervised feature extraction. *Proc. of Int. Neural Network Conference*, pages 322–325, Dordrecht, Kluwer, 1990.
18. G. Cottrell and J. Metcalfe. Face, gender and emotion recognition using holons. *Advances in Neural Information Processing Systems*, vol. 3, pages 564–571, 1991.
19. M. N. Dailey and G. W. Cottrell. Organization of face and object recognition in modular neural network models. *Neural Networks* 12:1053–1073, 1999.
20. R. Desimone. Face selective cells in the temporal cortex of monkey. *Journal of Cognitive Neuroscience*, 3:1–8, 1991.
21. G. Donato, M. S. Bartlett, J. C. Hager, P. Ekman, and T. J. Sejnowski. Classifying facial actions. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 21(10): 974–989, 1999.
22. B. A. Draper, K. Baek, M. S. Bartlett, and J. R. Beveridge, Recognizing faces with PCA and ICA . *Computer vision and image understanding*, 91:115–137, Special issue on Face Recognition, 2003.
23. P. Ekman. Facial expression and emotion. *American Psychologist*, 48(4):384–392, 1993.
24. J. W. Ellison and D. W. Massaro. Featural evaluation, integration, and judgment of facial affect. *Journal of Experimental Psychology: Human Experimental Psychology: Human Perception and Performance*, 23(1):213–226, 1997.
25. M. J. Farah, K. D. Wilson, M. Drain, and J. N. Tanaka. What is special about face perception. *Psychological Review*, 103(3):482–498, 1998.
26. D. Field. What is the goal of sensory coding? *Neural Computation*, 6(4):559–601, 1994.
27. P. Foldiak. Sparse coding in the primate cortex. *The Handbook of Brain Theory and Neural Networks*, Second Edition, pages 1064–1068, MIT Press, 2002.
28. M. E. Hasselmo, E. T. Rolls, G. C. Baylis, and V. Nalwa. The role of expression and identity in the face-selective responses of neurons in the temporal visual cortex of the monkey. *Behavioral Brain Research*, (32):203–218, 1989.
29. A. Heinrichs, M. K. Müller, A. H. J. Tewes, and R. P. Würtz. Graphs with Principal Components of Gabor Wavelet Features for Improved Face Recognition. in *Information Optics: 5th International Workshop on Information Optics; WIO'06*, pages 243–252, 2006.

30. P. O. Hoyer. Non-negative Matrix Factorization with sparseness constraints. *Journal of Machine Learning Research*, 5:1457–1469, 2002.
31. Z. Jin and F. Davoine. Orthogonal ICA representation of images, *Proceedings of 8th International Conference on Control, Automation, Robotics and Vision*, pages 369–374, 2004.
32. T. Kanade, J. Cohn, and Y. Tian. Comprehensive database for facial expression analysis. In *Proc. IEEE Intl. Conf. on Face and Gesture Recognition*, pages 46–53, 2000.
33. N. Kanwisher, J. McDermott, and M. M. Chun. The fusiform face area: A module in human extrastriate cortex specialized for face perception. *Journal of Neuroscience*, 17:4302–4311, 1997.
34. E. Kobatake and K. Tanaka. Neuronal selectivities to complex object features in the ventral visual pathway of the macaque cerebral cortex. *Journal of Neurophysiology*, 71(3):856–867, 1994.
35. M. Lades, J. C. Vorbrüggen, J. Buhmann, Jörg Lange, C. von der Malsburg, R. P. Würtz, and W. Konen. Distortion invariant object recognition in the dynamic link architecture. *IEEE Trans. on Computers*, 42(3):300–311, 1993.
36. D. D. Lee and H. S. Seung. Learning the parts of the objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.
37. T. Lee. Image representation using 2D Gabor wavelets. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(10):959–971, 1996.
38. S. Z. Li, X. W. Hou, and H. J. Zhang. Learning spatially localized, parts-based representation. *Intl. Conf. Computer Vision and Pattern Recognition*, pages 207–212, 2001.
39. C. Liu and H. Wechsler. Independent component analysis of Gabor features for face recognition. *IEEE Trans. Neural Networks*, 14(4):919–928, 2003.
40. A. Mehrabian. Communication without words. *Psychology Today*, 2(4):53–56, 1968.
41. B. Moghaddam. Principal manifolds and probabilistic subspaces for visual recognition. *IEEE Trans. Pattern Anal. Machine Intell.*, 24(6):780–788, 2002.
42. B. A. Olshausen and D. J. Field. Natural image statistics and efficient coding. *Network Computation in Neural Systems*, 7(2):333–339, 1996.
43. A. O'Toole, H. Abdi, K. Deffenbacher, and D. Valentin. Low-dimensional representation of faces in higher dimensions of the face space. *Journal of the Optical Society of America A*, 10(3):405–411, 1993.
44. C. Padgett and G. Cottrell. Representing face images for emotion classification. *Advances in Neural Information Processing Systems*, vol. 9, pages 894–900, 1997.
45. S. E. Palmer. Hierarchical structure in perceptual representation. *Cognitive Psychology*, 9:441–474, 1977.
46. P. S. Penev and J. J. Atick. Local feature analysis: A general statistical theory for object representation. *Neural Systems*, 7:477–500, 1996.
47. D. I. Perret, E. T. Rolls, and W. Caan. Visual neurons responsive to faces in the monkey temporal cortex. *Experimental Brain Research*, 47:329–342, 1982.
48. Y. Petrov and Z. Li. Local correlations, information redundancy, and the sufficient pixel depth in natural images. *Journal Optical Society of America A*, 20(1):56–66, 2003.
49. E. Rolls, G. Baylis, and M. Hasselmo. The responses of neurons in the cortex in the superior temporal sulcus of the monkey to band-pass spatial frequency filtered faces. *Vision Research*, 27(3):311–326, 1987.

50. E. T. Rolls and A. Treves. The relative advantages of sparse versus distributed encoding for associative neural networks in the brain. *Network* 1:407–421, 1990.
51. J. W. Tanaka and M. J. Farah. Parts and wholes in face recognition. *Quarterly Journal of Experimental Psychology: Human Experimental Psychology*, 46A:225–245, 1993.
52. K. Tanaka, C. Saito, Y. Fukada, and M. Moriya. Integration of form, texture, and color information in the inferotemporal cortex of the macaque. In *Vision, Memory and the Temporal Lobe*, pages 101–109, 1990.
53. A. Tewes, R. P. Würtz, and C. von der Malsburg. A flexible object model for recognizing and synthesizing facial expressions. In *Proc. of the Int. Conf. on Audio-and Video-based Biometric Person Authentication*, pages 81–90, 2005.
54. P. Thompson. Margaret Thatcher: a new illusion. *Perception*, 9:483–484, Pion Limited, London, 1980.
55. M. Turk and A. Pentland. Eigenfaces for recognition. *Cognitive Neuroscience*, 3(1):71–86, 1991.
56. T. Valentine. A unified account of the effects of distinctiveness, inversion, and race in face recognition. *Quarterly Journal of Experimental Psychology*, 43 A:161–204, 1991.
57. D. Valentin and H. Abdi. Can a linear auto-associator recognize faces from new orientations? *Journal of the Optical Society of America A*, 13:717–724, 1996.
58. E. Wachsmuth, M. W. Oram, and D. I. Perrett. Recognition of objects and their component parts: responses of single units in the temporal cortex of the macaque. *Cerebral Cortex*, 4(5):509–522, 1994.
59. L. Wiskott, J.-M. Fellous, N. Krüger, and C. von der Malsburg. Face recognition by elastic bunch graph matching. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(7):775–779, 1997.
60. R. P. Würtz. Object recognition robust under translations, deformations and changes in background. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(7):769–775, 1997.
61. R. P. Würtz. Organic Computing methods for face recognition. *it – Information Technology*, 47(4):207–211, 2005.
62. P. C. Yuen, and J. H. Lai. Face representation using independent component analysis. *Pattern Recognition*, 35(6):1247–1257, 2002.
63. J. Yang, D. Zhang, and J. Yang. Is ICA Significantly Better than PCA for Face Recognition? *10^{th} IEEE International Conference on Computer Vision*, 2005.
64. Z. Zhang, M. Lyons, M. Schuster, and S. Akamatsu. Comparison between geometry-based and Gabor-wavelets-based facial expression recognition using multi-layer perceptron. In *Proc. of Third IEEE Int. Conf. Automatic Face and Gesture Recognition, April 14-16, 1998, Nara, Japan*, pages 454–459, 1998.

15

Self-organized Evaluation of Dynamic Hand Gestures for Sign Language Recognition

Maximilian Krüger[1], Christoph von der Malsburg[2], and Rolf P. Würtz[1]

[1] Institut für Neuroinformatik, Ruhr-Universität, 44780 Bochum, Germany.
maximilian.krueger@neuroinformatik.rub.de,
rolf.wuertz@neuroinformatik.rub.de
[2] Frankfurt Institute for Advanced Studies, Max-von-Laue-Str. 1, 60438 Frankfurt a. M., Germany.
malsburg@fias.uni-frankfurt.de

15.1 Introduction

Human-computer interaction (HCI) is entering our everyday life. We are welcomed by robot guide-bots in Japan [2] and play computer games using Nintendo's nunchucks [1]. Nevertheless, the revolution is not finished and computer vision is still under development [21]. In this paper we present an organic computing approach to the recognition of gestures performed by a single person in front a monocular video camera.

Visual gesture recognition has to deal with many well-known problems of image processing, like camera noise, object tracking, object recognition and the recognition of a dynamic trajectory. Thus, a gesture recognition system has to show robust feature extraction and adaptation to a flexible environment and signer. It requires properties of an organic computing system, with different autonomous modules cooperating to solve the given problem.

Sign language is a good playground for gesture recognition research because it has a structure, which allows to develop and test methods on sign language recognition first before applying them on gesture recognition. Thus, here we restrict ourselves to working on signs of the British Sign Language (BSL) and concentrate on their manual part.

We have to consider that the projection of the 3D scene onto a 2D plane results in loss of depth information and therefore the reconstruction of the 3D-trajectory of the hand is not always possible. Also the position of the signer in front of the camera may vary. Movements like shifting in one direction or rotating around the body axis must be kept in mind, as well as the occlusion of some fingers or even a whole hand during signing.

Despite its constant structure each sign shows plenty of variation in time and space. Even if the same person performs the same sign twice, small changes in speed and position of the hand will occur. Generally, a sign is

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_15,

affected by the preceding and subsequent sign, an effect called *coarticulation*. Part of the coarticulation problem is that the system has to be able to detect sign boundaries automatically, thus the user is not required to segment the sign sentence into single signs.

Our approach follows the principles of organic computing [10]. We divide problems into different subtasks that are solved by autonomous subsystems. All subsystems are working on-line and therefore can help each other or can flexibly adapt to new situations. Integrating information from different sources, like hand shape, position and their temporal development present, beside the coordination of these processes, the main challenge for creating a recognition system. Our subsystems will autonomously solve part of the problem using organically inspired techniques like *democratic integration* for information merging, *bunch graph matching* for face/object recognition and a modified parallel *hidden Markov model* (HMM) for the recognition of the dynamic trajectories. Each of these techniques learns its knowledge from examples according to the organic computing approach. We explicitly separate gesture recognition into two main processes: feature extraction, which includes localization and tracking of body parts and the recognition process, which uses the selected features. Both processes will be performed during the performance of the sign.

To realize different autonomous units, their environment and communication between them in a software framework we designed a multi-agent system (MAS). Agents show self-x properties like dynamical adaptation to a changing environment (self-healing), perception of their environment and the capability to rate their action (self-reflection).

The structure of the paper is as follows: Section 15.2 gives an overview of previous work in the field of sign language recognition and motivates our ambition to use organic computing. The following section 15.3 describes the multi-agent system architecture, in particular the constructed agents and their use of organic computing methods. Visual tracking is presented in section 15.4 and our approach to sign language recognition in section 15.5. A description of the experiments undertaken and their results can be found in section 15.6. Finally, in section 15.7 conclusions are drawn and future work is outlined.

15.2 Related work

Sign languages, designed to be used by deaf people, are visual languages. They can be characterized by *manual* (hand shape, hand orientation, location and motion) and *non-manual* (trunk, head, gaze, facial expression, mouth) parameters. In this work, we concentrate on manual features and investigate *one-handed* signs performed by the dominant hand only, and *two-handed* signs, which can be performed symmetrically or non-symmetrically.

Sign language recognition (SLR) has to solve three problems, first the reliable tracking of the hands, second robust feature extraction, and third the

interpretation of the temporal feature sequence. In the following we present the approaches to these problems that have inspired our work.

Starner and Pentland [13] analyze sign gestures performed by one signer wearing colored gloves. After color segmentation and the extraction of position and shape of the hands their recognition is based on a continuous sequence of signs that are bound to a strict grammar using trained hidden Markov models. Bauer and Krais [3] introduce an HMM-based continuous sign language recognition system by splitting the signs into subunits to be recognized. Image segmentation and feature extraction are simplified by using colored gloves with different colors for fingers and palm. The extracted sequence of feature vectors reflects the manual sign parameters. The same group has built another recognition system that works with skin color segmentation and builds a multiple tracking hypothesis system [24, 20]. They are using HMM as well and extract geometric features like axis ratio, compactness and eccentricity of the hands segmented by skin color.

Instead of colored gloves Vogler and Metaxas [18] use 3D electrical tracking of the wrists. They propose a parallel HMM algorithm to model gesture components and recognize continuous signing sentences. Shape, movement, and location of the right hand along with movement and location of the left hand are represented by separate HMM channels, which are trained with relevant data and features. For recognition, individual HMM networks are built in each channel and a modified Viterbi decoding algorithm searches through all the networks in parallel. Path probabilities from each network that went through the same sequence of words were combined. Tanibata et al. [14] proposed a similar scheme where output probabilities from HMMs modeling the gesture data from right and left hand, were multiplied together for isolated word recognition in the Japanese Sign Language.

The group around Richard Bowden [4, 11, 7] structures the classification mode around a linguistic definition of signed words. This enables signs to be learned reliably from just a handful of training examples. Their classification process is divided into two stages. The first stage generates a description of hand shape and movement using skin color detection. This level of feature is based directly upon those used within sign linguistics to document signs. Its broad description supports generalization and therefore significantly reduces the requirements of further classification stages. In the second stage, Independent Component Analysis (ICA) is used to separate the channels of information from uncorrelated noise. Their final classification uses a bank of Markov chains to recognize the temporal transitions of individual words/signs.

All the presented work is very inspiring and has different interesting approaches to the problems of sign language recognition. Most of these systems are working offline, meaning they collect the feature sequence and do their recognition when the gesture is already performed.

In our approach we divide the problems into different subtasks that are solved by autonomous subsystems. Instead of color tracking we use self-organizing multi-cue tracking for the different body parts. Like in the pa-

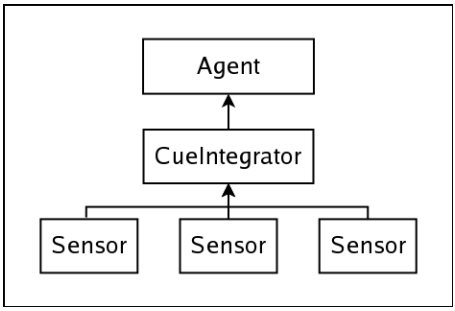


Fig. 15.1. An agent is based on three modules provided with as much flexibility as possible. The interface to the environment and the communication are included in the agent class. There is one cueIntegrator, a module that integrates and interprets the information provided by the sensors.

pers above we use an HMM approach for the temporal recognition, but we extended the idea of HMM by introducing self-organization properties.

15.3 System architecture

Organic computing systems consist of autonomous and cooperating subsystems. We build on a multi-agent system (MAS) developed earlier [9] as a framework for our task. The system consists of three base classes of objects, the **environment**, the **blackboard**, and the **agent**. While environment and blackboard are realized as singleton objects [5], there can be a multitude of different agents. These agents handle tasks ranging from coordination of subprocesses, tracking of an image point, up to the recognition of human extremities.

The information about the world is supplied by the **environment**. Based on the desired functionality of visual tracking and recognition, the environment provides access to image sequences, e.g., the current original color image and its processed versions, the gray value image and the difference image between two consecutive video frames.

Communication within the system is done via the **blackboard**. A message can be quite complex (e.g. carry an image) and has a defined lifetime. Each agent can write messages onto the blackboard and read the messages other agents have posted. Thus, the message handling allows the creation of new agents with specific properties, the change of properties and also the elimination of agents during run-time.

The **agent** is the most interesting entity, it shows the following self-x properties. Agents are autonomous and aware of their state. They perceive their surrounding, to which they can adapt and they communicate with other entities. To implement this behavior, agents have three layers, see figure 15.1. The top layer, called *agent* handles the communication, the fusion center,

called *cueIntegrator*, merges the information supplied by one or more *sensors*. Perception of the surrounding is twofold. On the one hand there is message handling via the blackboard, on the other hand an agent can receive information from its sensors, which filter incoming data. Based on the obtained information the agent reaches a decision about further actions.

Gesture recognition is split into the subproblems of object tracking and recognition (object and gesture). Each subproblem is solved by one or more agents. Hence teamwork and a observer/controller architecture are essential. There are three main classes of agents, tracking agents, agents for recognition and agents for control.

We designed **tracking agents** whose task is to follow an object. These agents merge different visual cues like color, texture, movement, etc. Cue fusion is done using democratic integration [16]. This technique offers a self-organized, flexible and robust way of tracking and will be explained in section 15.4. Agents that provide world knowledge stored in the system are called **recognition agents**. This includes knowledge for face recognition and static hand gesture recognition. Training the recognition agents, i.e. learning world knowledge from examples is also a crucial task requiring organic computing methods see [23]. As the system should act independently from user interaction **controlling agents** are responsible for solving the conflicts that might occur during execution.

15.4 Visual tracking

Visual tracking of head, left and right hand is done by a cooperation of globally and locally acting agents that are organized in a hierarchical network [9]. A global working agent scans the image for regions of interest, defined by skin colored and moving blobs. A controlling agent supervises the tracking. It collects the region of interest messages, checks whether they are already tracked and if not instantiates a new tracking agent. The visual appearance of the hand is a function of several factors, which hand classifier and tracking agents have to take into account, including pose, lighting, occlusion and intra/inter-signer variations.

Object tracking is performed by tracing an image point on the object. Tracking agents take on this task by scanning on the new frame the local surrounding of the last target position of the previous image. Hence they are called local agents. Due to lack of robustness of single cues, tracking should not rely on a single feature, thus each tracking agent integrates the results of four different information sources, namely pixel template, motion, motion prediction, and color, each realized in a sensor. The agent's cueIntegrator calculates the result as a weighted average of saliency maps derived from the different sensors. The result is fed back to the sensors and serves as the basis for two types of adaptation. First, the weights of the sensor are adapted according to their agreement with the overall result. Second, sensors are allowed to

Fig. 15.2. In this tracking sequence head and hands were found. The identity of the objects is visualized by the gray value of the rectangles, which delineate the search region of each tracking agent. Moving skin color in the background is ignored.

adapt their internal parameters in order to have their output better match the determined collective result. This integration scheme is called *democratic integration* [16] and will be explained below.

After tracking the object on the current frame, the tracking agent evaluates its success and posts a message containing the actual position, the contour and an image of the target. This information is passed to the recognition agents trying to identify the object. The face recognition agent, for instance, performs face detection using bunch graph matching [22]. Once a face has been found left and right hands are determined via their position relative to the face. The tracking agent adds its identification to its messages, see figure 15.2. To further support identification we added two recognition agents to identify the static hand gesture. The first recognition agent matches a gallery of learned bunch graphs on the image to identity the texture of the static hand gesture [17]; the second one matches the contour against a gallery of learned contours.

Since there might be skin colored moving blobs in the background of a real-world setting, which are not connected to a hand or the head, agents that track an unknown object over a period of time will delete themselves. This self-healing of the global system is also enforced if the agent is not content with its tracking results. After this analysis the tracking continues.

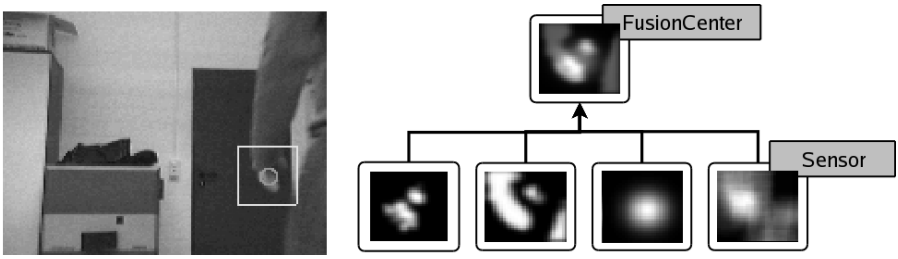


Fig. 15.3. Tracking agent in use, on the left we see the tracking result marked with the circle. The rectangle shows the border of the agent's search region. On the right we see the similarity maps created by the different sensors, from left to right: color, motion, motion prediction and pixel template. The fusion center shows the result of the information integration.

Using *democratic integration* the different cues, namely color, motion, motion prediction and pixel template, are integrated to agree on one result. After this decision each cue adapts toward the result agreed on. In particular, discordant cues are quickly suppressed and re-calibrated, while cues having been consistent with the result in the recent past are given a higher decision weight in future.

Integrating information using democratic integration relies on two assumptions. First, the cues must be statistically dependent, otherwise there is no point in trying to integrate them. Second, the environment must exhibit a certain temporal continuity. Without that, any adaptation would be useless.

As shown in figure 15.3 all cues are working on the two dimensional search region of the agent. Each sensor i provides a similarity map $\mathsf{M}_i(x,t)$ at time t, that shows the image similarity at each coordinate x with an agent-specific and adaptable prototype template $\mathsf{P}_i(t)$. To integrate the similarity maps to an overall map $\mathsf{R}(x,t)$, they are weighted and summed up

$$\mathsf{R}(x,t) = \sum_i r_i(t)\mathsf{M}_i(x,t), \tag{15.1}$$

The weights $r_i(t)$ are part of the self-controlling of each sensor and will further be called reliability. The reliabilities are normalized such that $\sum_i r_i(t) = 1$. The target position $\hat{x}(t)$ is found by scanning the overall similarity map for the maximal entry

$$\hat{x}(t) = \arg\max_x \{\mathsf{R}(x,t)\}\,. \tag{15.2}$$

To rate its action each tracking agent analyzes the similarity value at the target position $\hat{x}(t)$. If the value is above a threshold, the image point has been found, otherwise, the tracking agent has failed to track it. Using the information of the similarity value and the target position each sensor is able

to update its reliability and to adapt its prototype to the new situation. The adaptation depends on the tracking result — if the target was found in the image the prototype adapts towards the actual parameters at the target position $\hat{x}(t)$. Otherwise it adapts towards its initial values. We refer to [16] for a complete description of the update strategies.

Equation (15.2) has proven to work well on small objects [16, 8] with a unimodal similarity map. Larger objects can create a multimodal similarity map with more than one peak. Hence we modified the search of the target position by thresholding the map and from the remaining peaks we calculate the center of gravity. The new target position might be located outside the object or might be a bad point to track, but our experiments showed that this was not the case for different tracking scenarios and that tracking became more stable.

15.5 Recognition

In the previous section we presented the MAS system for visual tracking of head and hands. Tracking agents provide position and static hand gesture information of the object for nearly each frame. In this section we describe the subsystem that collects the information about the trajectory of left and right hand and the corresponding static hand gesture. The information for sign language recognition is merged by using an extended self-organizing hidden Markov model architecture. Recognition needs to be stable and robust enough tho deal with the changes in speed and position of a hand, which will even occur if the same person is performing the same sign twice. Hidden Markov models (HMMs) can solve these problems. Their ability to compensate time and amplitude variations of signals has been amply demonstrated for speech and character recognition. Before we discuss our approach to recognition using an extension to HMM we review the aspects of HMM theory relevant to this paper.

15.5.1 Theory of hidden Markov models

This section briefly discusses the theory of hidden Markov models (see figure 15.4 as an example of a left-right architecture). It follows the classic paper by Rabiner [12], which we recommend for a more detailed description of this topic. A hidden Markov model is characterized by the following:

1. N, the number of states in the model. We denote the individual states as $\mathbf{S} = \{S_1, S_2, \ldots, S_N\}$, and the state at time t as q_t.
2. M, the number of distinct observation symbols per state, i.e. the discrete alphabet size. The observation symbols correspond to the physical output of the system being modeled. We denote the individual symbols as $\mathbf{V} = \{v_1, v_2, \ldots, v_M\}$. If the observation is in a continuous space, M is replaced by an interval of possible observations.

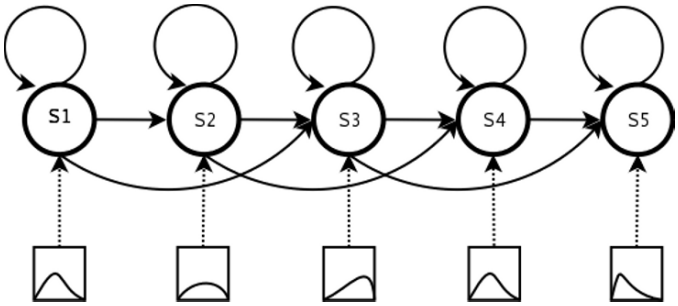


Fig. 15.4. HMM with the left-right (Bakis) topology, typically used in gesture and speech recognition. The solid lines denote the transition probabilities and set $a_{ij} = 0 \quad \forall \quad j < i \quad \wedge \quad j > i+2$. The dotted line connects a continuous observation distribution to the belonging state (circle).

3. The transition probability distribution $\mathsf{A} = \{a_{ij}\}$ where

$$a_{ij} = P\left(q_{t+1} = S_j \mid q_t = S_i\right), \quad 1 \leq i, j \leq N \tag{15.3}$$

and

$$\sum_j a_{ij} = 1. \tag{15.4}$$

Assuming that the state transition probability a_{ij} from state S_i to state S_j only depends on the preceding state (first order Markov process). For the special case that any state can reach any other state in a single step, we have $a_{ij} > 0$, for all i, j. For other types of HMM, we would have $a_{ij} = 0$ for one or more i, j pairs.

4. The observation probability distribution $\mathsf{B} = \{b_j(k)\}$ in state j, where

$$b_j(k) = P\left(v_k \text{ at } t \mid q_t = S_j\right), \quad 1 \leq j \leq N, \quad 1 \leq k \leq M \tag{15.5}$$

and

$$\sum_k b_j(k) = 1. \tag{15.6}$$

The observation probability distribution can be discrete or continuous.

5. The initial distribution $\boldsymbol{\pi} = \{\pi_i\}$ where

$$\pi_i = P\left(q_1 = S_i\right), \quad 1 \leq i \leq N. \tag{15.7}$$

A complete specification of a HMM consists of two model parameters (N and M), the specification of observation symbols, and the specification of the three probabilistic measures A, B and $\boldsymbol{\pi}$. For convenience, we use the compact notation

$$\boldsymbol{\lambda} = (\boldsymbol{\pi}, \mathsf{A}, \mathsf{B}) \tag{15.8}$$

to indicate the complete parameter set of the model.

Due to their doubly stochastic nature HMMs are very flexible and became quite famous in the gesture recognition community (section 15.2). The art of HMM design lies in the specification of their topology of allowed transitions, and the features to be observed. There are three basic problems of interest that must be solved for the HMM to be useful in real-world applications:

Evaluation problem: Given the observation sequence $\mathbf{O} = O_1, O_2, \ldots, O_T$, and the model $\boldsymbol{\lambda} = (\boldsymbol{\pi}, \mathsf{A}, \mathsf{B})$, how do we efficiently compute $P(\mathbf{O} \mid \boldsymbol{\lambda})$, the probability of generating the observation sequence given the model?

Decoding problem: Given the observation sequence $\mathbf{O} = O_1, O_2, \ldots, O_T$, and the model $\boldsymbol{\lambda}$, how do we choose a corresponding state sequence $\mathbf{Q} = q_1, q_2, \ldots, q_T$ which is meaningful in some sense (i.e., best "explains" the observations)?

Estimation problem: How do we train the HMM by adjusting the model parameters $\boldsymbol{\lambda} = (\boldsymbol{\pi}, \mathsf{A}, \mathsf{B})$ to maximize $P(\mathbf{O} \mid \boldsymbol{\lambda})$?

The standard way to recognize a gesture out of a set G is to train a HMM $\boldsymbol{\lambda}_g$ for every single gesture $g \in G$ and after the observation sequence is recorded, start the calculation of $P(\mathbf{O} \mid \boldsymbol{\lambda}_g)$ for every HMM $\boldsymbol{\lambda}_g$ (see section 15.5.3.1). The solution of the *evaluation problem* is used for recognition where the model $\boldsymbol{\lambda}_g$ which produces the highest probability of describing the observation sequences

$$g = \arg\max_g P(\mathbf{O} \mid \boldsymbol{\lambda}_g) \tag{15.9}$$

is deemed to be the recognized gesture.

15.5.2 Organic modification of the hidden Markov model

The topology of our HMMs is an extension of the Bakis model as seen in figure 15.4 and will be further explained in section 15.5.3.1.

The observed features (which we will call observations from now on) are provided by the tracking module described in section 15.4, namely position, texture and contour of left and right hand. Each kind of observation has a particular degree of uncertainty, the position can vary on the object, texture and contour might not be accurately determined due to blurring or erroneous segmentation. Thus, we use the organic computing principle that distributed information is advantageous for robustness and split the observations into different channels. This parallel HMM (PaHMM) structure had been used by Vogler [18] and Tanibata [14], who divided the observations for left and right hand and trained a HMM for each hand. The independence of the channels has been demonstrated in [19]. Consequently, we go one step further give the system better generalization power if every observation has its own HMM, instead of putting them into one observation vector. Therefore, in our system a gesture g will be represented by six channels that separate the position $\mathbf{y}$ relative to the head, texture $\boldsymbol{\tau}$ and contour $\mathbf{c}$ for left and for right hand. Another point is that if we assume that parallel HMMs model the parallel

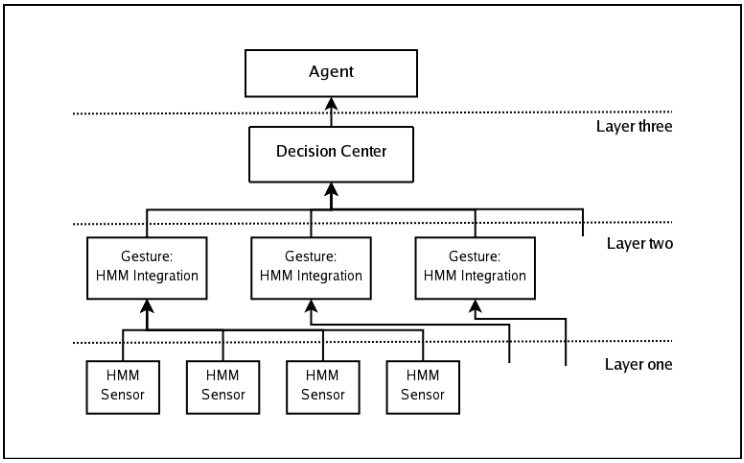


Fig. 15.5. Layout of the recognition agent. It is hierarchically organized, at the bottom we have the HMM sensor modules, four of them for each gesture. They collect the information and calculate their observation probability. The HMM sensor results are merged in the Gesture HMM Integration modules, of which there is one for each learned gesture. The Decision Center decides about the most probable gesture.

processes independently, they can also be trained independently, and do not require consideration of the different combinations at training time.

15.5.3 The HMM recognition agent

Administration of gesture recognition is hosted in the recognition agent. During a recognition cycle the agent starts collecting the observations. Gesture recognition is done from bottom to top (see figure 15.5 and algorithm box 1). Layer one consists of the trained HMM channels for each gesture. In terms of our MAS they are called HMM Sensors. They independently calculate the actual observation probability and pass this information to layer two. The Gesture HMM Integration modules in layer two represent the learned gestures. Each module fuses the information of its channels to compute a decision about the probability that the performed gesture is similar to the one represented by the Gesture HMM Integration module. Finally, in the Decision Center of layer three, the results of the Gesture HMM Integration modules are compared to determine which gesture is the most probable.

15.5.3.1 Layer one: modified HMM

Starting at the sensor layer we are mainly interested in solving the evaluation problem mentioned above. Before we present our extensions of the HMM idea, we want to outline the Forward-Backward approach to calculate $P(\mathbf{O} \mid \boldsymbol{\lambda})$ and motivate our modifications.

Forward-backward algorithm

The forward-backward algorithm [12] solves the evaluation problem by calculating the $P(\mathbf{O} \mid \boldsymbol{\lambda})$ using the forward variable $\alpha_t(j)$, which is defined as

$$\alpha_t(j) = P(O_1, O_2, \ldots, O_t, q_t = S_j \mid \boldsymbol{\lambda}), \tag{15.10}$$

the probability of the partial observation sequence $O_1, O_2, \ldots, O_t$ and state S_j at time t, given the model $\boldsymbol{\lambda}$. The forward variable is solved inductively, as follows:

1. Initialization:
$$\alpha_1(j) = \pi_j b_j(O_1), \quad 1 \leq j \leq N. \tag{15.11}$$
2. Induction:
$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}), \quad 1 \leq t \leq T-1 \quad 1 \leq j \leq N. \tag{15.12}$$
3. Termination:
$$P(\mathbf{O} \mid \boldsymbol{\lambda}) = \sum_{i=1}^{N} \alpha_T(i). \tag{15.13}$$

The backward variable $\beta_t(i)$ is calculated in a similar manner and is defined as the probability of the partial observation sequence $t+1$ to the end, given state S_i at time t. Each of these variables or a combination of them can be used to solve the *evaluation problem*.

Using the forward-backward algorithm and especially the multiplication of probabilities in equation (15.12), recognition would not be robust to:

1. a missing observation,
2. a wrongly classified static hand gesture that was not in the training data[1],
3. the observation sequence taking longer than the learned ones.

Problem one can be solved by having perfect tracking and perfect classification of static hand gestures. The other two problems become less crucial when collecting more training data. But it is our aim to develop a system that works robustly in a self-organized way under real-world conditions, the conditions of sparse data.

HMM sensor

The flexibility of the HMM depends on the training data, which determines the transition probabilities a_{ij} and the observation probability distributions $\mathsf{B} = \{b_j(k)\}$ of the model. Under real world conditions the HMM will be confronted with unknown, not learned, observations or variations in the dynamics caused

[1] Using a discrete observation distribution, a zero will be returned for the observation probability of a symbol that has not been learned for the particular state.

by, e.g., missing tracking information, blurring, etc. To face these problems we split up the doubly probabilistic method of the HMM by introducing a self-organized transition between the states. In our approach we are using a strict left-right model with $\pi_1 = 1, \pi_i = 0$ for $i \neq 1$ and $a_{ij} = 1$ if $j = i + 1$, $a_{ij} = 0$ else. The number of states N is equal to the longest learning sequence of the gesture's training set. Instead of the transition probability matrix A where the transitions are learned, the a_{ij} are replaced by a weighting function $w_t(u)$. Equation (15.12) will become

$$\alpha_{t+1}(j) = \alpha_t(i) \underbrace{a_{ij}}_{=1} w_t(u) b_j(O_{t+1}), \tag{15.14}$$

and therefore equation (15.13) becomes

$$P(\mathbf{O} \mid \boldsymbol{\lambda}) = \alpha_T(N). \tag{15.15}$$

Equation (15.14) allows the HMM to perform on-line gesture recognition and computes its probability on every frame.

The weighting function of each channel is Gaussian

$$w(u) = \exp\left(-\frac{u^2}{2\sigma}\right), \tag{15.16}$$

where $u = [0, \infty]$ is a measure of uncertainty. Starting with a maximal certainty of $u = 0$ at the beginning of the recognition, the modified HMM (mHMM) checks whether the received observation O_t is presented in the observation distribution $b_i(O_t)$ of the actual state i. If the result is not satisfactory, i.e., below a recognition threshold, the mHMM can pass the observation to the next state $i+1$, check again, and pass it further to state $i+2$ if necessary. If the observation does not even match at state $i + 2$ it will be ignored. Each of these transitions is punished by increasing the uncertainty u and thus lowering the weighting function. To reinstall its certainty, the mHMM recovers with every recognized observation by decreasing u. If the observation has been recognized the system switches to the next state.

To come back to our HMM recognition agent, the task of the HMM sensor (layer one in figure 15.5) is to calculate its weighted observation probability $w_t(u)b_j(O_t)$ for the actual frame.

15.5.3.2 Layer two: gesture HMM integration

Each learned gesture g has a Gesture HMM Integration unit in layer two. By merging the information of its six sensors the Gesture HMM Integration unit computes the quality Q_g of the gesture g matching the observation sequence.

Due to the nature of the observation, the HMM sensors have different probability distributions. The position information is stored in a continuous

Algorithm 1: Recognition is hierarchically organized using three layers. The characteristic of each layer is its information integration. Layer one, the HMM Sensor, compares the received observation with its observation probability function. Layer two comprise the HMM Integration unit of each learned gesture and integrates the information received from layer one. The top layer compares the results from the HMM Integration units. The Decision Center determines the most probable gesture and manages the inhibition.

```
1  while not at end of gesture sequence do
       /* ********************************************************* */
       /* Layer one: HMM sensor                                     */
       /* ********************************************************* */
2      foreach HMM sensor do
3          calculate observation probabilities;
4      end
       /* ********************************************************* */
       /* Layer two: HMM Integration unit                           */
       /* ********************************************************* */
5      foreach HMM Integration unit do
6          compute ϱ to fuse the information of position, texture and contour;
7          calculate the actual quality Q_a;
8          update the overall quality Q_g;
9          control the activation using Q_g, ξ_start and ξ_stop ;
10     end
       /* ********************************************************* */
       /* Layer three: Decision Center                              */
       /* ********************************************************* */
11     if ∃ HMM Integration unit that reached its ζ_min then
12         choose HMM Integration unit with highest Q_g
13           as current winner;
14     end
15     if ζ_winner == 1 then
16         reset all HMM Integration units;
17     end
18     else
           /* inhibit all gestures                                  */
19         search for the maximal quality Q_max;
20         foreach HMM Integration unit do
21             subtract Q_max from Q_g;
22         end
23     end
24 end
```

Result: last winner will be chosen as recognized gesture.

distribution using Gaussian mixtures. The information of the static hand gesture, the identified most similar contour and bunch graph, are stored separately using a discrete probability distribution, which is realized as histogram based on the appearance of the elements of the contour alphabet or bunch graph alphabet, respectively.

The continuous distribution offers a more flexible way to evaluate the observation, as we have a Euclidean distance for our position observations. In our discrete feature space the concept of similarity, or distance, cannot be assumed to be Euclidean. Therefore, we use position as the basis for our recognition. The aim of sensor integration is the computation of the overall quality Q_g for the single gesture g. At the beginning or after a reset (see below), the Q_g is initialized with zero. To get rid of possible multiplications with small numbers when estimating $\alpha_{t+1}(j)$ using equation (15.14), we will work with the logarithms of the probabilities sent by the sensors and therefore obtain:

$$\alpha_{t+1}(j) = \alpha_t(i) + \log(w_t(u)b_j(O_{t+1})). \tag{15.17}$$

Thus, for every frame we receive the log probability l of the left hand position $l_{lh}(\mathbf{y})$, left hand contour $l_{lh}(\mathbf{c})$, left hand texture $l_{lh}(\boldsymbol{\tau})$ and right hand $l_{rh}(\mathbf{y})$, $l_{rh}(\mathbf{c})$, $l_{rh}(\boldsymbol{\tau})$. To calculate the actual gesture quality Q_a of the current frame, we first weight the position probabilities of the two hands and add them to Q_a

$$Q_a = w_{lh}l_{lh}(\mathbf{y}) + w_{rh}l_{rh}(\mathbf{y}). \tag{15.18}$$

Thereby we focus on the dominant hand by setting $w_{rh} = 0.7$ and $w_{lh} = 0.3$. Although position is already a good observation for gesture recognition we have to add the static hand gesture information to obtain better results. But as mentioned above, recognition of the static hand gesture might not be stable on every frame, especially when the hand is moving. Hence, we decided to integrate the bunch graph and contour information using a rewarding function ϱ. This function rewards only if position and static hand gesture information are correlated. Correlation does not necessarily mean that the mHMMs have to be in the same state i. For each hand the $l(\mathbf{c})$ and $l(\mathbf{y})$ or $l(\boldsymbol{\tau})$ and $l(\mathbf{y})$ just have to be above a threshold θ. The reward is linked to the probability for the static hand recognition $l(\mathbf{c})$, $l(\boldsymbol{\tau})$ respectively

$$\varrho(x) = (x - \theta)H(x - \theta)\ ; \quad H : \text{Heaviside step function} \tag{15.19}$$

and will be added to Q_a. After computing Q_a we update the Q_g by adding Q_a. Without the static hand gesture information the Q_g would decrease with increasing gesture length. By introducing ϱ we allow the Q_a and Q_g to become positive and therefore Q_g cannot be transferred into a probability again.

Each Gesture HMM Integration unit has two states, active and inactive. In the active state the gesture is certain that it could match the data and by increasing the states of the HMM Sensor the recognition is continuously following the incoming observations. Increase of the state of the HMM Sensor

is a cue for the similarity of the learned gesture to the performed sign. A gesture becomes active if the Q_a of the first state is above the threshold ξ_{start}. Otherwise, the gesture is inactive, which means that all the connected HMM Sensors are set to their initial state and all the parameters like the uncertainty u of each sensor and the Q_g are reset to zero. An active gesture can become inactive if the Q_g drops below a threshold ξ_{stop}. ξ_{start} and ξ_{stop} have global values and allow the system to reset a gesture autonomously to restart the recognition during the performance of the sign. We developed this active/inactive mode to handle the problem of coarticulation (the frames between two gestures) and the case where we have similar beginning for one or more gestures and only the following frames will decide which gesture is performed.

15.5.3.3 Layer three: decision center

Only active gestures will receive the attention of the Decision Center in layer three. The Decision Center compares the results of the Gesture HMM Integration units and determines which gesture is the most probable so far.

The autonomy of the Gesture HMM Integration units in choosing a starting time prohibits the Decision Center from using equation (15.9) directly and declare the gesture g with the highest value of Q_g the recognized one. In that case the Decision Center would wrongly favor gestures that just started over gestures that already accumulated similarity. Thus, we coupled the recognition to the progress, the actual state, of the HMM Sensor by means of a confidence value ζ, which is computed by the ratio of the actual state of the sensor to the maximal number of states N of the mHMM. This confidence value is a measure of certainty. Only gestures that are above a threshold of ζ_{min} will be handled by the Decision Center. This minimal confidence ζ_{min} is individual for each gesture and is computed as the ratio of its shortest to its longest sequence in the training set. Out of the gestures that reached their ζ_{min} the Decision Center chooses the one with the highest Q_g to be the most probable gesture that represents the observation sequence so far. This method favors short gestures that only need a small amount of recognized frames to reach their ζ_{min}. Therefore, all gestures are inhibited by the gesture with the highest Q_g to become inactive before they reach the needed confidence value. If a gesture reaches a confidence value of one, it is deemed recognized already before termination of the sequence, and a reset signal is sent to all connected Gesture HMM Integration units.

15.6 Experiments and results

15.6.1 Sign language data

Our training and testing data consist of signs of the British Sign Language (BSL) that were kindly provided by Richard Bowden. The data is a continuous

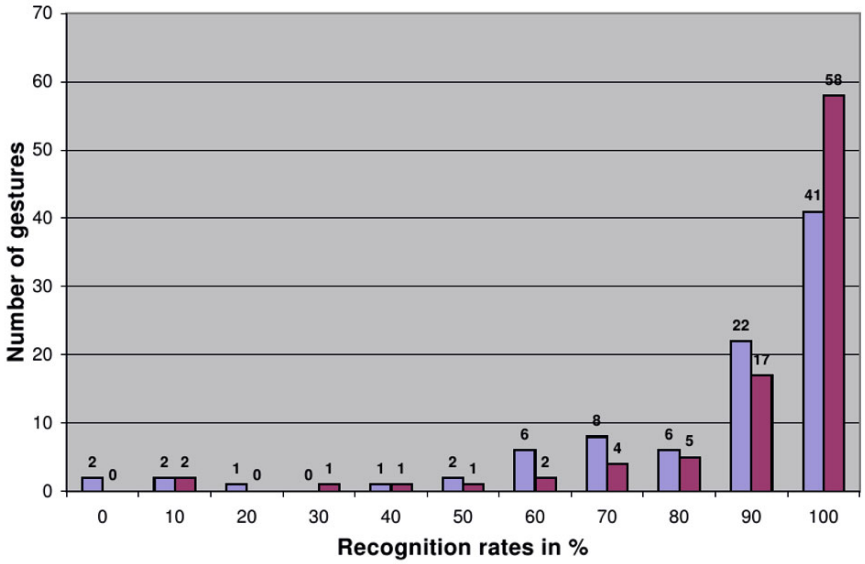


Fig. 15.6. Recognition started with the beginning of the gesture. The histogram shows the result for the experiment using only position and contour information in light color and in dark color the result when integrating position, contour and texture information.

movie with ground truth information about the beginning and the end of each gesture. We have 91 different signs performed with 10 repetitions by one signer, a total of 29219 images to be processed. The sequence length ranges from 11 to 81 frames for the gestures and even within the gestures the sequence length shows differences of around 50 percent, e.g., the length of gesture "live" ranges from 18 to 41 frames. The signer is wearing colored gloves, hence for training the exact position of the hand (the center of gravity), the texture and the shape contour could be automatically determined. To calculate relative positions for the hands a bunch graph face detection was run on the images. The segmented hands allow the automatic creation of bunch graphs for left and right hand for each frame. They where clustered by matching each bunch graph on the other images and adding the image if the matching similarity was above a certain threshold. The extracted contours were clustered using standard vector quantization as described in Gray [6] to gain an alphabet of representative hand shapes. As a result we obtain observation sequences for relative hand position and static hand gesture, which are used to train the mHMM.

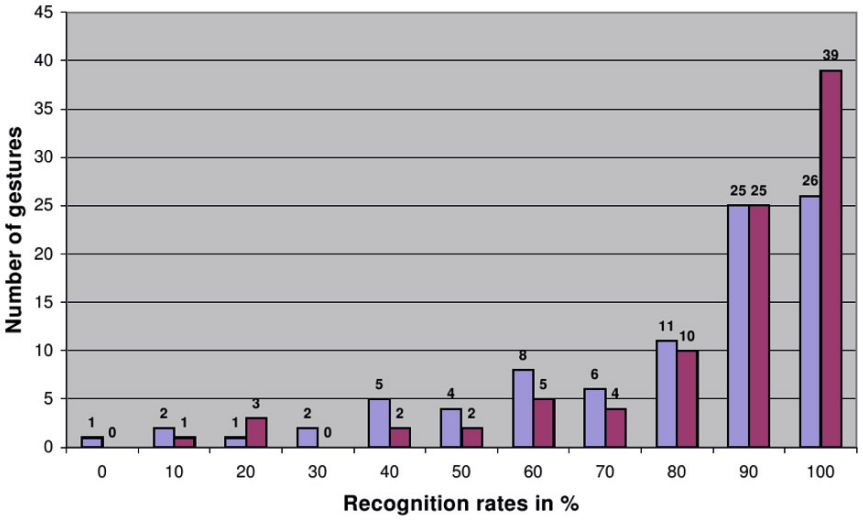


Fig. 15.7. Recognition started 10 frames in front of the the beginning of the gesture to examine the effect of coarticulation. The histogram shows the result for the experiment using only position and contour information in light color and in dark color the result when integrating position, contour and texture information

15.6.2 Experiments

The recognition experiments were performed using a leave-one-out procedure, where for the testing gesture all sequences excluding the one that is tested were used to build the mHMMs. Therefore, we perform ten recognition experiments per gesture. At the end of each performance, the final most likely gesture is deemed to be the overall recognized gesture.

We tested our system in two sets of recognition experiments. In the first set recognition starts at the known beginning of the sign, while in the second recognition experiment we included the coarticulation of a previous gesture. To simulate coarticulation we started the recognition 10 frames before the ground truth starting time.

On both sets we tested the benefit of multi-cue integration and the stability of of the system concerning missing data by running each set two times. In the first run we integrated position, contour and texture information for recognition and then we dropped the texture information and only integrated position and contour information in the second run.

The distribution of the recognition rates is shown in figure 15.6 for the given start of the gesture and figure 15.7 for the coarticulation. The results are presented in histogram style, where we plotted the recognition rate (light bars for the recognition experiment applying only position and dark bars for adding texture information) against the number of recognized gestures. For

Fig. 15.8. The trajectories and static hand gestures of the signs "excited_interested" (left) and "live" (right) are very similar. Therefore, the shorter sign live dominates the recognition of the excited_interested performance. The integration of non-manual observation like a grammar or facial expression should help to differentiate between similar signs.

example, using figure 15.6 we have recognized eight gestures with a recognition rate of 70% when only using position and contour information.

Given the start of the gesture we achieve an average recognition rate of 90% if we integrate position contour and texture. The mean recognition rate is reduced to 84% if we exclude the texture information. Taking the first run we receive a mean recognition rate of 90% and higher for over the half of our gestures. By analyzing the gestures with lower recognition rates of 0 to 10%, these gestures were mainly dominated by a very similar gesture that have a lower sequence length. For example the "excited_interested" gesture is dominated by the shorter "live" gesture, that shows a very similar trajectory and similar static hand gestures as can be seen in figure 15.8. The difference of the trajectories is small compared to the inter-sign variations that can occur in other sign like the "different" and the "bat" gesture trajectories that are plotted in figure 15.9. This misclassification trap is caused by the self-organizing property of the system. All known gestures are in a loop and are waiting to become active by passing the activation threshold ξ_{start}. Therefore as seen for the low recognized gestures they are likely to be dominated by similar shorter gestures and this might be a good reason to include grammar or other non-manual observation like facial expression to future systems.

The benefit of this autonomy to start the recognition becomes obvious in our second set. Running the experiment with the same data and parameters we achieve a mean recognition rate of 85% or 78% respectively for the second run. The recognition system shows just a 5% difference between a fixed and a self-organized start of the recognition.

Comparing recognition rates for sign language recognition is a difficult task, because every group has its own data. Nevertheless we have to admit that our recognition rates are below the results of von Agris et al [20] with 97.9%

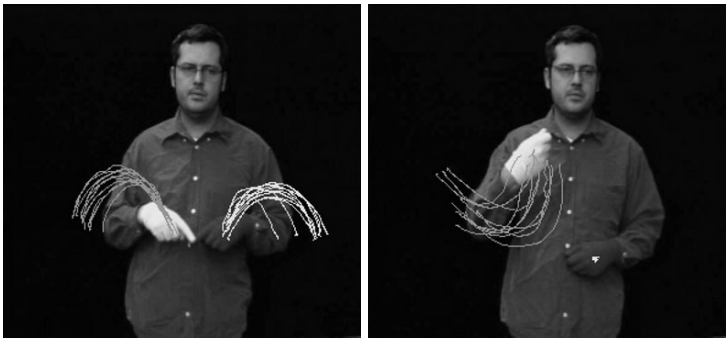


Fig. 15.9. The signs "different" (left) and "bat" (right) shown with the trajectory differences of ten repetitions by the same signer.

for a database of 153 isolated signs and Kadir et al [7] who have a recognition rate of 92% for a lexicon of 164 words. The strength of our system is the autonomy of the recognition process to handle the effect of coarticulation, which have not been investigated by von Agris and Kadir.

15.7 Conclusion

We have presented an approach to gesture recognition by organic computing technology. We built a software framework to design and test multi-agent systems. The characteristics of a multi-agent system are autonomous and cooperating units. Organic computing principles like divide and conquer, learning from examples and self-control have been used for object tracking and sign language recognition. Both systems are running simultaneously.

For gesture recognition we modified a standard HMM architecture by introducing two types of information, a more reliable channel as a basis and a weaker one. Both are integrated by using a correlation and rewarding scheme. Another innovation is the competition of the learned gestures during the recognition process. In addition to satisfying recognition results the autonomy of the system allows to handle the problem of coarticulation.

Only simple features like the position, contour and texture of the hands have been applied, we resigned to grammar or a high level description. To learn a grammar or a high level description would be an interesting challenge for future projects. In the near future we plan to integrate facial expression recognition of Tewes et al [15] as a new HMM sensor and to run the system on more data to examine its signer independence ability.

Acknowledgments

We thank Richard Bowden from the University of Surrey for providing his sign language data. Funding by the DFG in the SPP "Organic Computing" (MA 697/5-1 and WU 314/5-2) is gratefully acknowledged.

References

1. http://de.wii.com/, 2006.
2. http://www.kokoro-dreams.co.jp/english/robot/act/index.html, 2006.
3. B. Bauer and K.-F. Kraiss. Video-based sign recognition using self-organizing subunits. In *ICPR (2)*, pages 434–437, 2002.
4. R. Bowden, A. Zisserman, T. Kadir, and M. Brady. Vision based interpretation of natural sign languages. In *International Conference on Computer Vision Systems, Graz. Austria*, 2003.
5. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Sons, August 1996.
6. R. M. Gray. Vector quantization. In A. Waibel and K.-F. Lee, editors, *Readings in Speech Recognition*, pages 75–100. Kaufmann, San Mateo, CA, 1990.
7. T. Kadir, R. Bowden, E. J. Ong, and A. Zisserman. Minimal training, large lexicon, unconstrained sign language recognition. In *Proceedings of the 15th British Machine Vision Conference, Kingston*, 2004.
8. O. Kähler and J. Denzler. Self-organizing and adaptive data fusion for 3d object tracking. In U. Brinkschulte, J. Becker, D. Fey, C. Hochberger, T. Martinetz, C. Müller-Schloer, H. Schmeck, T. Ungerer, and R. Würtz, editors, *ARCS 2005 – System Aspects in Organic and Pervasive Computing – Workshops Procedeedings, Innsbruck Austria, March 14–17*, pages 109–116. VDE Verlag, Berlin, Offenbach, 2005.
9. M. Krüger, A. Schäfer, A. Tewes, and R. P. Würtz. Communicating agents architecture with applications in multimodal human computer interaction. In P. Dadam and M. Reichert, editors, *Informatik 2004*, volume 2, pages 641–645. Gesellschaft für Informatik, 2004.
10. C. Müller-Schloer, C. von der Malsburg, and R. P. Würtz. Aktuelles Schlagwort: Organic Computing. *Informatik Spektrum*, 27(4):332–336, 2004.
11. E. Ong and R. Bowden. A boosted classifier tree for hand shape detection. In *Face and Gesture Recognition*, pages 889–894, 2004.
12. L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb. 1989.
13. T. Starner and A. Pentland. Visual recognition of american sign language using hidden markov models. In *International Workshop on Automatic Face and Gesture Recognition*, pages 189–194, 1995.
14. N. Tanibata, N. Shimada, and Y. Shirai. Extraction of hand features for recognition of sign language words. In *Proceedings International Conference on Vision Interfaces*, pages 391–398, 2002.
15. A. Tewes, R. P. Würtz, and C. von der Malsburg. A flexible object model for recognising and synthesising facial expressions. In T. Kanade, N. Ratha, and A. Jain, editors, *Proceedings of the International Conference on Audio- and*

Video-based Biometric Person Authentication, LNCS, pages 81–90. Springer, 2005.
16. J. Triesch and C. von der Malsburg. Democratic integration: Self-organized integration of adaptive cues. *Neural Computation*, 13(9):2049–2074, Sept. 2001.
17. J. Triesch and C. von der Malsburg. Classification of hand postures against complex backgrounds using elastic graph matching. *Image and VisionComputing*, 20(13):937–943, 2002.
18. C. Vogler and D. N. Metaxas. Parallel hidden markov models for american sign language recognition. In *ICCV (1)*, pages 116–122, 1999.
19. C. Vogler and D. N. Metaxas. Handshapes and movements: Multiple-channel american sign language recognition. In *Gesture Workshop*, pages 247–258, 2003.
20. U. von Agris, D. Schneider, J. Zieren, and K.-F. Kraiss. Rapid signer adaptation for isolated sign language recognition. In *CVPRW '06: Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop*, page 159, Washington, DC, USA, 2006. IEEE Computer Society.
21. C. von der Malsburg. Vision as an exercise in Organic Computing. In P. Dadam and M. Reichert, editors, *Informatik 2004*, volume 2, pages 631–635, 2004.
22. L. Wiskott, J.-M. Fellous, N. Krüger, and C. von der Malsburg. Face recognition by elastic bunch graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):775–779, 1997.
23. R. P. Würtz. Organic Computing methods for face recognition. *it – Information Technology*, 47(4):207–211, 2005.
24. J. Zieren and K.-F. Kraiss. Robust person-independent visual sign language recognition. In J. S. Marques, N. Pérez de la Blanca, and P. E. Pina, editors, *Pattern Recognition and Image Analysis, Second Iberian Conference IbPRIA, Estoril, Portugal*, volume 3522 of *LNCS*, pages 520–528. Springer, 2005.

Index