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

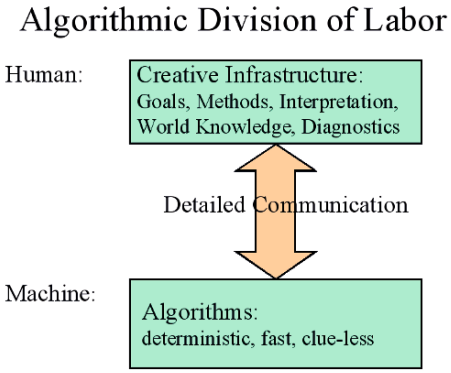

Fig. 2.1. Classic algorithmic computing entails a division of labor between man and machine. Creative infrastructure resides mainly in the human domain (upper box), the machine (lower box) blindly following commands. The two domains are coupled by detailed communication so that the programmer can inspect,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

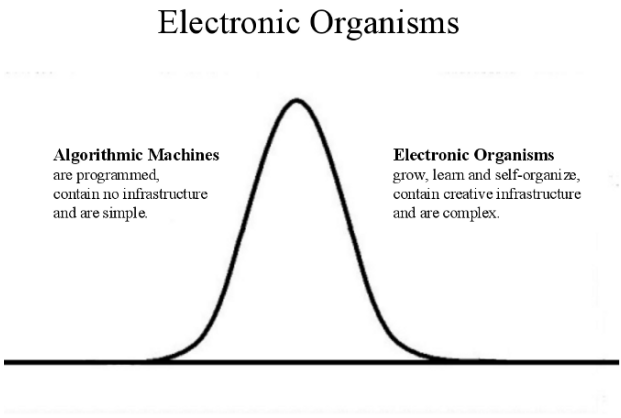

Fig. 2.2. To reach the realm of organic computing, a potential mound has to be crossed. Classically, software *has to be simple* in order to be intelligible by the human programmer, and it *may be simple*, containing little or no creative infrastructure. Electronic organisms contain much creative infrastructure 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-

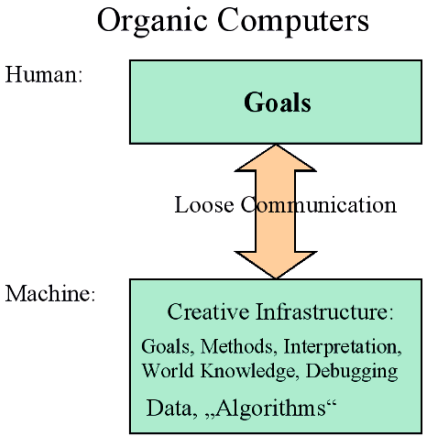

Fig. 2.3. In organic computing, the only task humans hold on to is the setting of goals. As the machine is autonomously organizing, detailed communication between programmer and machine is restricted to the fundamental algorithm, which is realizing system organization. Application-oriented mechanisms 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.