Evolutionary and Embryogenic Approaches to Autonomic Systems [*]

Daniele Miorandi
CREATE-NET
via Alla Cascata 56/c
IT — 38100
Povo, Trento, Italy
daniele.miorandi@create-net.org

Lidia Yamamoto
Computer Science Department
Bernoullistrasse, 16
CH — 4056
Basel, Switzerland
Lidia.Yamamoto@unibas.ch

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; F.1.1 [**Computation by Abstract Devices**]: Models of Computation; I.2 [**Computing Methodologies**]: Artificial Intelligence

General Terms

Algorithms, Design

Keywords

Evolutionary Computation, Genetic Programming, Genetic Algorithm, Artificial Embryogenies, Chemical Computing

ABSTRACT

In this paper we present a review of state-of-the-art techniques for automated creation and evolution of software. The focus is on bio-inspired bottom-up approaches, in which complexity is grown from interactions among simpler units. First, we review Evolutionary Computation (EC) techniques, highlighting their potential application to the automated optimization of computer programs in an online, dynamic environment. Then, we survey approaches inspired by embryology, in which artificial entities undergo a developmental process. We introduce the concept of *EmbryoWare* to refer to software that can be modified via an embryogenic process. We refer to *Evolutionary Developmental Computation* as the combined evo-devo approach in software, and describe its constituent elements. The paper concludes with a short discussion and outlook for applications of the aforementioned techniques to autonomic computing and communication systems.

[*]This work has been partially funded by the European Commission within the framework of the BIONETS project EU-IST-FET-SAC-FP6-027748, `www.bionets.eu`.

1. INTRODUCTION

Building software that is able to continuously improve itself automatically is a common goal in artificial intelligence, software engineering, and other areas of computer science, including, more recently, autonomic systems and organic computing. The dream is to bring to computers the ability to constantly seek to learn and adapt, driven by a concrete purpose and motivation coming from the interaction with the real world [51]. This is also meant to automate a large number of processes in computer systems, reducing human intervention while improving the system's performance and its robustness, reducing costs related to design and management.

Efforts in this direction follow a top-down or a bottom-up approach: Top-down approaches attempt to automate the reasoning process used in software engineering and design, from user requirements down to the code implementation. These include automatic program and protocol synthesis from specifications [36, 47] and more recently, derivation of policy rules from high-level representations closer to natural language [67, 62]. Bottom-up approaches look at how higher-level software functionality would emerge from lower-level interactions among simpler system units. Artificial Life (ALife), Evolutionary Computation, Swarm Intelligence, Amorphous Computing and other areas focus on such bottom-up approach.

While the top-down approach seeks a formal model of software construction by humans, the bottom-up approach is essentially biologically-inspired. Even the most elementary life forms possess a level of robustness and adaptation far beyond current artificial systems, therefore it seems worthwhile to learn from biology in order to draw inspiration for the design of new systems.

In this paper we provide a survey of bio-inspired approaches to such bottom-up creation of software functionality. Our focus is on dynamic, online processes where evolution and adaptation must happen continuously, during the operation of the system, as opposed to offline, design-time optimization approaches. We investigate the potential of bio-inspired algorithms to obtain systems that are able to continuously pursue an optimum operation point without ever stopping. Such online optimization process involves the ability to self-organize into structures at multiple scales, analogous to cells, multicellular organisms, up to artificial ecosystems of interacting parts. Particular attention will be devoted to distributed systems, where extra challenges arise due to the need of working with partial (and in many cases delayed) information.

Numerous bio-inspired systems are available. A classification was proposed in [55], which positions them in a 3-D space defined by three axes, related to evolution of functionality, structural growth, and learning ability, respectively. We focus on the first two axes, represented mainly by evolutionary computation and developmental approaches related to embryology.

This chapter is organized as follows. In Sec. 2 we position our context within the POE (Phylogenesis, Ontogenesis, Epigenesis) framework introduced in [55]. In Sec. 3 we review the state-of-the-art in evolutionary computation with focus on online and dynamic environments. In Sec. 4 we present the main research lines inspired by embryology: embryonics and artificial embryogenies. Sec. 5 presents a critical discussion on the possible combination of the aforementioned approaches. Sec. 6 discusses possible research directions on embryonic software systems. Sec. 7 concludes the chapter pointing out possible applications to the automated generation and optimization of software in the context of autonomic computing and communications.

2. CONTEXT: THE PO-PLANE

A classification of bio-inspired systems was proposed in [55], positioning them in a 3-D space defined by three orthogonal axes: Phylogeny (P), Ontogeny (O), and Epigenesis (E), forming the POE framework. Although it was proposed more than ten years ago for hardware, its concepts remain valid today, and apply to software as well. We focus on two of the the three axes, namely Phylogeny and Ontogeny, resulting in what we refer to as the PO-Plane. The third axis from the original POE framework is related to learning, and covers techniques such as artificial neural networks and artificial immune systems, which are outside the scope of the present survey. The two remaining axes are defined as follows:

- *Phylogeny* or *phylogenesis* is the process of genetic evolution of species. Phylogenetic mechanisms are essentially non-deterministic, with mutation and recombination as major variation triggers. Artificial systems along this axis perform Artificial Evolution (AE) either in hardware (Evolvable Hardware) or in software (Evolutionary Computation). The latter will be described in Section 3.

- *Ontogeny* or *ontogenesis* is the process of growth and development of a multicellular organism from the fertilized egg to its mature form. Ontogeny is studied in developmental biology, which covers the genetic control mechanisms of cell growth, differentiation and morphogenesis. Artificial systems here go from simple replicators and self-reproducing systems to embryonics (mostly in hardware) and artificial embryogeny (mostly in software). These will be described in Section 4.

The POE classification is represented in Fig. 1, where some of the techniques which will be treated in this paper are positioned on the PO-Plane.

As predicted in [55], today combinations of phylogenetic and ontogenetic approaches, forming the so-called PO-Plane, are becoming more and more common. On the one hand, an indirect encoding followed by a developmental process has shown to increase the scalability of evolutionary computing for complex problems [23]. On the other hand, evolution enhances embryogenic systems with the potential of finding new solutions that were not preprogrammed. We conjecture that a combination of both is probably also essential to achieve the goal of online dynamic optimization, which is the focus of the present paper: due to its highly non-deterministic nature, evolution alone would be too slow or insufficient to achieve this goal, while ontogenetic processes alone would lack the creation potential necessary to face new situations in a dynamic online environment. The potential of such combined PO-Plane approaches will be discussed in Section 5.

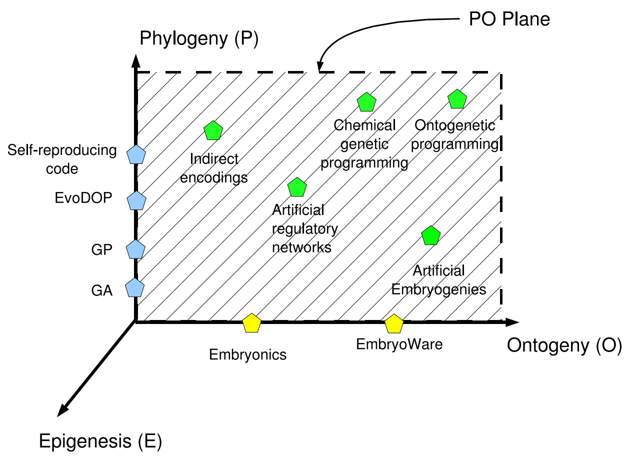

Figure 1: The POE classification and some of the phylogenetic and ontogenetic approaches that will be treated in this paper.

3. EVOLUTIONARY COMPUTATION

Evolutionary Computation (EC) [23, 20] derives optimization algorithms inspired by biological evolution principles such as genetics and natural selection. *Evolutionary Algorithms* (EAs) are meta-heuristics that can be applied to a variety of search and optimization problems. Existing EAs include: *Genetic Algorithms* (GAs), *Genetic Programming* (GP), *Evolutionary Programming* (EP) and *Evolution Strategies* (ES). They all model candidate solutions as a population of individuals with a genotype that is iteratively transformed, evaluated against a given fitness criterion, and selected according to the "survival of the fittest" principle, until an optimal solution is found. The difference among them lies in the way candidate solutions are represented, and on the search operators applied to obtain new solutions.

Recently, these existing iterative approaches are referred to as *Artificial Evolution* (AE) [8], in which biology concepts are applied in a very simplified way. In [8] the authors propose a new term *Computational Evolution* (CE) to reflect a new generation of bio-inspired computing [65] that builds upon new knowledge from biology and increased synergies between biologists and computer scientists.

AE is largely based on the so-called "Central Dogma of Artificial Evolution", analogous to the "central dogma" of biology, in which information flows unidirectionally from

DNA to proteins. This dogma is known today to be an over-simplification of reality. In CE, instead, one looks at the complex interactions that occur within the cell and beyond, such as genetic regulation and various other regulatory mechanisms in the cell, the effects of interactions with the environment, symbiosis and competition in artificial ecosystems, and other highly dynamic processes which occur in many real-life problems. CE is of particular interest in online dynamic scenarios which are the focus of this survey. Note that there is no clear-cut border between AE and CE, but rather a gradual transition. For instance, the combination of phylogenetic and ontogenetic mechanisms positioned on the PO-Plane can be seen as a movement in the CE direction.

3.1 Genetic Algorithms

In a Genetic Algorithm (GA) [27] candidate solutions are represented as a population of individuals whose genotype is a string of solution elements (bits, characters, symbols, etc.). Strings typically have a fixed or bounded length, such that the size of the search space can be constrained. The goal of a GA is to find the optimum value of such string that optimises a given fitness criterion. An initial population of candidate strings is generated and evaluated against the fitness criterion. An intermediate population is created by choosing, usually with a probability which depends on the respective fitness level (through mechanisms such as, e.g., roulette wheel selection and tournament selection) a set of potentially promising genotypes. Elements from such intermediate population are then used to generate, by applying genetic operators such as cross-over and mutation, the next generation.

GAs have been successfully applied to a variety of problems in science and engineering, and turn out to work well in many situations. The level of performance offered by GA depends on the particular fitness landscape of the problem to be optimized. In particular, GAs are known to work well in the case of rather smooth fitness landscape. GAs have been designed to work offline and in a centralized manner. In particular, they require the ability to evaluate synchronously and exactly the fitness level of all genotypes in the current population. Further, GAs are not meant for online evolution, as they typically stop once the optimal solution is found.

3.2 Genetic Programming

Genetic Programming (GP) [29, 10, 32] applies the GA idea to evolve computer programs automatically. A GP algorithm is essentially the same as a GA, but the candidate solutions encode computer programs, such that they can solve all instances of a problem, instead of optimizing for a particular instance as in GA.

GP typically evolves programs encoded in a linear (similar to assembly language) or tree representation (similar to functional languages such as LISP). Other representations are also possible, such as graphs [46, 39], finite state machines [3, 54, 53], neural networks [40], and more recently, chemical programs [37, 70].

When solving a problem by GP, one generally does not know the maximum size of the target solution program. Therefore, the genotype representation in GP generally allows for variable-length programs with unbounded size. The size of the search space in this case is infinite, so programs can in principle grow indefinitely. The *bloat* phenomenon was discovered early in the GP history, and refers to the fact that programs evolved by GP (especially tree-based GP) tend indeed to grow very large, with obvious shortcomings in terms of memory usage and execution efficiency. The bloat phenomenon is generally accompanied by an *intron growth* phenomenon, in which non-coding regions emerge, that have no effect on the program outcome. Although some authors pointed out that this phenomenon may also have positive effects, such as some protection against destructive crossover, it was mandatory to control code growth. Several methods were proposed for this purpose, such as parsimony pressure [34].

Recently, special attention has been devoted to indirect representations in which a genotype encoding is mapped onto a different phenotype representation. The goal is make GP solutions scale to complex problems without corresponding growth in program size. Indirect encodings may also provide additional robustness and evolvability, via redundant representations in which one phenotype may be expressed by more than one genotype, and via neutrality in representations, in which mutations in the genotype do not immediately affect the corresponding phenotype. This is especially important for online evolution, as will be discussed in Section 3.6.

Most GP approaches can be placed along the “P” axis in the POE framework (Fig. 1). Some indirect encodings include a growth process which positions them on the PO-plane. An example is an Artificial Embryogeny, which will be discussed in Section 4.2.

3.3 Evolution in Dynamic Environments

EC techniques have been widely used for solving optimization problems in dynamic environments, in which the problem instance or the constraints may vary over time [28]. The aim is to introduce mechanisms able to “track” the optimal solution. This field is referred to as *Evolutionary Computation for Dynamic Optimization Problems* (EvoDOP). EC provides a natural framework for dynamic optimization, in that natural evolution is a continuous process. Most approaches in EC for dynamic optimization are based on the assumption that the changes in the problem settings are gradual, such that the previous population can be reused to search for the new optimum, without having to restart from scratch.

One of the main problems in EvoDOP is premature convergence: the population quickly converges to the optimum and tends to become very uniform, i.e. all solutions resemble the best one. In a static environment this is not an issue, since one can stop searching once a satisfactory solution is found. In a dynamic environment, premature convergence hinders the ability to search for new solutions. Proposed solutions for this problem include:

- Generate diversity after a change, e.g. through *Hypermutation*, i.e. artificially high mutation rates in response to a change [17].
- Maintain diversity throughout the run, e.g. through *random immigrants*, individuals that move between subpopulations [71].
- Explicit memory: previously good solutions are stored in memory, and retrieved when the system encounters a previous situation for which the solution applied.

- Implicit memory: usually takes the form of a redundant representation such as *diploid* or *polyploid* individuals with a dominance mechanism. In nature, many organisms have diploid cells in which chromosomes occur in pairs, one coming from each parent. Some species are polyploid (e.g. some plants), with several pairs of homologous chromosomes. In their artificial counterpart, multiple alternative solutions occur in a genotype, and a dominance mechanism decides which one is actually used (expressed). The solutions that are not expressed remain "dormant" as a form of memory, potentially to be reactivated later when the conditions become favorable. However it is difficult to exploit such memory, due to the lack of explicit correlation between the expressed solution, the conditions of the environment at that time, and the obtained fitness.

- Multi-population: Different subpopulations are spawned from a main population and assigned a subspace in which to track local optima. Several different promising subspaces can then be explored simultaneously.

- Anticipation and prediction: These are recent methods that attempt to predict the consequences of current decisions on the future of the system, such that informed decisions can be taken which will lead to improved results with high probability [13].

Although much has been done in EvoDOP in the GA domain, little has been explored in the GP domain. In [72] the evolvability of programs under environmental variations is studied. The author shows that neutrality (in that context, different programs with the same fitness) plays a key role in ensuring evolvability under changing conditions. In [16] the authors show a multi-chromosome approach to GP based on Prolog programs. Multi-chromosomal GP is a polyploidy mechanism, thus a variant of implicit memory, which has been shown to achieve only mitigated results in EvoDOP. Indeed, the approach [16] is not applied to a dynamic environment. In nature, however, polyploidy mechanisms are extremely helpful, therefore it would be interesting to see how to improve the analogous artificial mechanisms to achieve comparable performance. Another interesting research line would be to bring the most promising EvoDOP approaches to the GP domain, namely multi-population and anticipation.

EvoDOP is especially important for autonomic systems, where evolution must face online changing conditions. Applying these techniques to distributed systems involves several challenges, such as: devise reliable decentralized ways to measure diversity across several nodes; coordinate subpopulations; distribute implicit or explicit memory.

3.4 Self-Replication and Self-Reproduction

Much attention has been paid in EC on self-replicating and self-reproducing code. Although in many cases replication and reproduction are treated as synonymous, we adopt the distinction proposed in [55]: Replication is an ontogenetic, developmental process, involving no genetic operators, resulting in an exact duplicate of the parent organism. Reproduction, on the other hand, is a phylogenetic (evolutionary) process, involving genetic operators such as crossover and mutation, thereby giving rise to variety and ultimately to evolution.

The study of self-replicating software can be traced back to the pioneering work of John von Neumann in the late 40s on self-replicating automata. He set the basis for a mathematically rigorous study of self-replicating artificial machines based on cellular automata. Since then, several examples of self-replicating machines have been shown and elaborated [24]. Such machines are able to produce an identical copy of themselves, which means that the copy must also contain the part of the code that is able to produce a further copy of itself. Errors in the replication process are usually not allowed, and recovery from copy errors are thus in general not provided.

Self-reproducing code, on the other hand, involves a variation mechanism by which the new individual is not an exact copy of its parent. Self-reproduction thus requires some form of self-modification, which will be discussed below. Moreover it must include resilience to harmful replication errors in the form of a self-repair mechanism, or a selection mechanism able to detect and discard harmful code.

3.5 Self-Modifying Code

In a system that is required to constantly evolve and adapt, the ability to automatically modify or update its own code parts is essential. Since reliable and secure self-modification of software is still an open issue, self-modifying code has been banished from good practice software engineering.

However, self-modifying code plays a key role in EC and ALife. In the case of EC, only the best programs which have been thoroughly tested via multiple fitness cases can be safely used. In the case of Digital Evolution in ALife [38], programs are "digital organisms" which must survive in a virtual world, where their operations do not present any risk for the end user. Programs in the Avida system [38] run on top of a virtual CPU which executes instructions from an assembly-like language that manipulates the virtual machines' memory locations, registers, and stacks. Programs in such language are easily self-modifiable: one can write on memory positions that include the own memory location of the code. Moreover, the instruction set is designed in a way that any arbitrary mutation results in a valid program. For instance, memory locations are addressed by pattern matching as opposed to absolute or relative address positions. Instructions have no explicit arguments, using the adjacent instructions as arguments when needed. Avida has shown to evolve programs in several domains, including modelling of biological systems, distributed algorithms, and more recently, sensor networks, mobile devices, and model-driven development [38].

In the GP context, the *Push* family of programming languages [56] is designed for a stack-based virtual machine in which code can be pushed to a stack and therefore be manipulated as data. A variant of Push was used in *Autoconstructive Evolution* [57], where individuals take care of their own reproduction, and the reproduction mechanism itself can evolve (showing self-modification at the level of reproduction strategies). Recently [56], an enhancement of the language permitting the explicit manipulation of an execution stack has been introduced. It has been shown to evolve iterative and recursive function which are non-trivial to be evolved in GP.

Ontogenetic Programming [59] is a developmental approach to GP in which the generated programs include self-modification instructions that enable them to change during the run. This is foreseen as an advantage for adaptation to the environment. To illustrate the concept, in [58] Ontogenetic Programming is applied to a virtual world game in which agents must find gold and survive multiple dangers and obstacles. It is shown that the ontogenetic version is able to evolve correct solutions to the game, where traditional GP fails to do so. This is an interesting example of hybrid approach located along the PO-Plane.

3.6 Indirect Encodings

It is well known in Evolutionary Computation that the representation of candidate solutions and the genetic operators applied to it play a key role in the performance of the evolutionary process. The genotype to phenotype mapping scheme is included in this representation problem, and it is well known in GP that indirect encodings like Cartesian GP [39] and Grammatical Evolution [41] can greatly help in obtaining viable individuals. Moreover they present a potential for encoding *neutrality*.

Neutrality occurs when small mutations in the genotype are likely not to affect the fitness of the corresponding individual. Such "silent" mutations, which modify the genotype while leaving the fitness unchanged, are called *neutral mutations*. Since the resulting changes are not subject to selection, their immediate impact is invisible. At first sight, they slow down evolution. However, over the long run, as neutral mutations accumulate, some genotypes may end up expressing a different solution with a potentially higher fitness. Neutrality provides a "smooth" way to explore the search space, and has been shown to potentially increase the evolvability of a population.

Indirect encodings may also be used to enhance the scalability of EC to complex problems: a compact genotype can express a large number of different phenotypes, such that the number of genes required to specify a phenotype may be orders of magnitude less than the number of structural units composing the phenotype. If coupled with developmental approaches (embryogeny, morphogenesis) it can encode phenotypes that grow from simple structures to more complex ones. Many indirect encoding approaches include such a developmental process and can therefore be positioned on the PO-plane of the POE framework.

3.7 Approaches Based on Gene Expression

Many indirect encoding approaches have taken inspiration from gene expression in order to improve the performance of EC, especially GP. In these approaches, the process of decoding a genotype into a phenotype is analogous to expressing genes, and is controlled by a regulation or feedback mechanism.

Artificial Regulatory Networks have been shown to model the biological regulatory mechanisms in both natural [33] and artificial systems [31]. In [33] a genetic network exhibiting stochastic dynamics is evolved using a set-based encoding of systems of biochemical reactions. In [31] the regulatory network is represented with a genotype/phenotype binary encoding in which genes express proteins, which in turn control the expression of genes, unleashing large reaction networks that evolve by gene duplication and divergence. These networks are able to compute functions, such as a sigmoid and a decaying exponential.

Chemical Genetic Programming [44] proposes a feedback-based dynamic genotype to phenotype translation mechanism inspired by a cell's dual step transcription-translation process from DNA to RNA and then to proteins. Using a chemical reaction mechanism, it dynamically builds the rewriting rules of a grammar used to translate a linear genotype into a tree phenotype. This leads to a highly dynamic and evolutive genotype to phenotype mapping: starting from a pool of simple grammar rules, the system evolves more complex ones and discards those that are not useful. While the concept itself seems promising, the encoding used and the algorithm itself are relatively complex, albeit applied to comparatively simple problems so far.

Epigenetic Programming [63] associates a developmental process to GP, in which an Epigenetic Learning (EL) algorithm activates or silences certain parts of the genetic code depending on the current environmental conditions.

This is said to protect individuals from destructive crossover by silencing certain genotypic combinations and explicitly activating them only when they lead to beneficial phenotypic traits. The authors show a 2-fold improvement in computational effort with respect to GP, on a predator-prey pursuit problem. Although in this approach a potentially large number of phenotypes can be expressed from a single genotype, this apparent increase in complexity is misleading, since all phenotypes are subsets of the original genotype.

Gene Expression Programming (GEP) [22] uses a linear genotype representation in the form of a chromosome with multiple genes. Each gene is translated into an expression tree, and trees are connected together by a linking function. Although inspired by gene expression, this approach does not include a dynamic feedback mechanism.

3.8 Chemical Computing

Chemical computing models [6, 15, 43, 19] express computations as chemical reactions that consume and produce computation objects (data or code). Objects are represented as elements in a *multiset*, an unordered set within which elements may occur more than once.

We believe that chemical models have a great potential for the class of online dynamic software optimization problems that we are addressing. This is due to their inherent parallelism and multiset model, which permits several copies of instructions to be present simultaneously. We conjecture that a chemical language can express programs that can be more easily transformed and can become more robust to disruptions due to alternative execution paths enabled by a multiset model.

In this section we discuss some work in evolving programs using a chemical representation, which presents a new challenge for GP.

An *Algorithmic Chemistry* [9, 68] expresses algorithms as multisets of instructions in a reaction vessel in which instructions are executed in random order. In [9] the power of GP applied to an algorithmic chemistry on evolving solutions to specific problems is shown. The authors point out the importance of the concentration of instructions, rather than their sequence. They start from a nearly unpredictable system in which execution of instructions at a random order leads to a random program output. This system is set to evolve by GP, including crossover and mutation of instructions placed in registers. After some generations, some order can be observed, and at the end of the evolutionary process a highly reproducible output is obtained, in spite of the random execution order.

The emergence of evolution in a chemical computing system is investigated in [37], using organization theory. Artificial biochemical signaling networks are evolved in [18] to compute several mathematical functions, using an evolutionary computation method based on asexual reproduction and mutations. Although the networks evolved in [18] show computational capacity, it does not seem trivial to extend their capabilities from mathematical functions to generic software actions.

With such chemical systems it becomes possible to quantitatively regulate the behaviour of programs for evolution or adaptation purposes. An example of that is *Chorus* [5], a grammar-based GP system which uses a concentration table to keep track of concentrations of rules in the system. The rule with the highest concentration is picked for execution. The purpose is to obtain a system in which the absolute position of a gene (encoding a grammar rule number) does not matter. Such a system is then more resilient to genetic operators. In [69] we have proposed a code regulation system based on the control of the concentration of signals that activate or inhibit the expression of given genotypes according to their fitness. While [5] chooses the rule with the highest concentration, in [69] the choice is probabilistic: the chance of a variant being picked for execution is proportional to the concentration of its expression signals. While [69] is explicitly intended for online problems, to the best of our knowledge [5] has not been applied in this context.

4. EMBRYOLOGY

Embryology, in general, is a branch of developmental biology focusing on embryogeny, i.e., the process by which the embryo is formed and develops, from fertilization to mitotic divisions and cellular differentiation. The ability of embryos to generate complexity starting from a basic entity has generated a lot of attention in the computing field, since replicating such process *in silico* could break the complexity ceiling which limits conventional EC.

The application of ideas from embryology (or, better: embryogenies) to artificial systems has been following two main research directions. One is *embryonics* (embryology plus electronics), an approach to improve fault tolerance in evolvable hardware by using a cellular architecture presenting dynamic self-repair and reproduction properties. Another one is *artificial embryogeny*, which aims at extending evolutionary computation with a developmental process inspired by embryo growth and cell differentiation, such that relatively simple genotypes with a compact representation may express a wide range of phenotypes or behaviours. These two directions reflect just different communities (hardware vs. software) rather than a clear conceptual partition, since the underlying concepts are common to both. Indeed, embryonics can be considered as a branch of artificial embryogeny which focuses on cell differentiation as an error handling mechanism in reconfigurable hardware, without necessarily covering evolutionary aspects.

4.1 Embryonics

The main goal of *embryonics* [14, 48, 42, 64] is to embed extreme fault tolerance into electronic devices (e.g., FPGA arrays) while maintaining the redundancy (e.g., number of "spare" columns/rows in FPGA arrays) at acceptable levels. Approaches in this area have mostly focused on the use of *artificial stem cells*, i.e., cells which are able to differentiate into any specific kind of cell required for the organism to work. The approach is based on the flexibility offered by embryology-based mechanisms, in which there is no need to specify *a priori* the actions to be undertaken as a consequence of a fault detected. In FPGA arrays, specifying the reaction to each possible fault configuration would lead to poorly scalable designs, while at the same time resulting in a large overhead, due to the need of maintaining a large number of spare rows/columns. The systems devised in embryonics are based on the following two principles:

- Each cell (understood as the smallest indivisible building block of the system) contains the whole genome, i.e., has the complete set of rules necessary for the organism to work. Each cell is totipotent, i.e., can differentiate into any specific function and decides, based on the interaction with neighboring cells, which functionalities (genes) need to be expressed.

- The system possesses self-organizing properties. Each cell monitors its neighborhood and, upon detection of a faulty component, can return to the stem cell state and differentiate into another type of cell to repair the fault. (Some works have proposed to use solutions inspired by the mammalian immune system to implement this second functionality [14].) This step involves the availability of "spare" cells, which provide resources necessary to replace the faulty component.

The main difference between the embryonics approach to fault tolerance and classical approaches is that classical fault tolerance techniques tend to focus on simple replication as a way to achieve redundancy that can be used to recover from failures. In embryonics the information stored in neighboring cells that might have differentiated to perform other functions may be used to recreate a lost functionality by re-expressing the corresponding genes that may be dormant in other cells. Such flexibility to switch functionality adds another level of robustness, as now not only cells with identical functionality can be used as backup or template to repair a failure, but also other cells with different functionalities can be used to recreate a lost one. In evolvable hardware, functionality is mainly expressed by the state of a cell (e.g. in the form of a configuration register). A self-replication mechanism is also supported, in order to enhance robustness in the case of massive faults. Any of the cell tends indeed to replicate to spare cells, if any is found present. In this way, the whole organism can replicate in a different portion of the system whenever necessary.
The Embryonics approach does not encompass any evolutionary aspect. Therefore, in terms of the classification introduced in Sec. 2, we can position it as a pure ontogenetic approach.

4.2 Artificial Embryogeny

Artificial Embryogeny [60] is a branch of Evolutionary Computation (EC) in which compact genotypes are expressed into phenotypes that go through a developmental phase that may cause them to differentiate to perform specific functions. Indeed, researchers have recognized that "conventional" EC techniques (like GA, GP, Evolutionary Strategies, etc.) present scalability problems when dealing with problems of relevant complexity [23]. The issue is that the size of the genotype representing possible solutions in the search space turns out to grow fast as the complexity of the organism/behaviour to be optimized grows. One solution studied in such approach has been to add one more level of abstraction. In such case, the genotype does not code the solution itself, but it codes recipes for building solutions (i.e., phenotypes). In this way, a genotype change does not imply a direct change in the solution, but in the way solutions are decoded from the genotype and further grown from an initial "seed" (the embryo).

In the case of GP, such indirect genotype encodings play an important role in obtaining viable individuals (i.e., syntactically correct programs suitable to be executed) via genetic transformations such as crossover and mutation. Approaches in the GP area are classified according to the genotype representation and decoding: generative or grammar-based, and ontogenetic GP [23]. In the first case, the genotype is a grammar that comprises a set of rewriting rules which are applied until a complete phenotype is obtained. A prominent example in the GP area is Grammatical Evolution [41]. Since grammar production rules are applied to obtain the program, the derived program is syntactically correct by construction. In the second case, the genotype contains a set of instructions/transformations which are applied repeatedly on an embryonic entity to obtain a full organism. An example of the second case is Ontogenetic Programming [59] (see Section 3.5), which produces self-modifying programs that are highly adaptive.

Note that although grammatical evolution is an indirect encoding approach, it is not performing embryogeny per se, as the generated individuals do not necessarily continue to develop after the phenotype is expressed. One could imagine a grammar to express self-modifying programs (for instance, a grammar that encodes for the stack-based linear programs in [59, 58]) such that grammatical evolution then becomes part of the full cycle of evolution, gene expression, development and adaptation. However this is orthogonal to the developmental process implied in artificial embryogeny.

Other grammar approaches outside the GP context have an inherent growth model which has been associated with artificial embryogeny. Chapter 2.1 of [60] presents a survey of these grammatical approaches to artificial embryogeny. A simple one is to use L-Systems. L-Systems, or Lindenmayer Systems, express fractal-like objects using a grammar where the production rules may or may not contain parameters that determine how the structure will grow. Why is this form of grammar closer to embryogeny than grammatical evolution? Since L-Systems encode structures as opposed to executable programs, any intermediate step in the expansion of an L-System is a valid structure (thus a valid individual in growth process), while in grammatical evolution the first valid program that can be executed is one in which all production symbols (non-terminals) have been rewritten into terminal ones. On the other hand, L-Systems have not been designed with evolution in mind, although they have been later used for this purpose.

The impact of embryogenies in an optimization process is still not fully understood. Some studies [11] have reported a clear advantage of an indirect encoding over a direct one such as tree-based GP. On the other hand, further experiments [30] report that the indirect approach actually takes much longer to run, and show cases where tree-based GP outperforms the indirect approach and vice-versa. The fact that a developmental phase can slow down evolution has been later confirmed and extensively analyzed by [66].

What remains consistent across different experiments is that in general, indirect encodings perform best when the problem is complex. There is therefore a trade-off between the computational resources needed for performing embryogeny and the complexity of the problem to be tackled, which needs to be carefully accounted for, especially in the presence of resource-constrained devices.

Artificial embryogenies may encompass both an ontogenetic as well as a phylogenetic aspect: as such, they lie in the PO–plane.

5. EC AND EMBRYOLOGY: COMMON SYNERGIES

While Evolutionary Computation and Embryology-based approaches pertain both —broadly speaking— to the same research area, and while there is a considerable overlap in the research communities involved (with the notable exception of embryonics), there are many synergies which have not been exploited so far. A thorough understanding of all the combinations possible on the PO-Plane is still missing.

Such combination could complement genetic-based EC techniques by providing a different level of system dynamics. While, indeed, one of the advantages of artificial embryogenies is related to the possibility of encoding complex behaviours in a parsimonious way (thus tackling the scalability problems encountered by standard GA/GP techniques), it seems to us that another advantage would be the possibility of having a much faster system dynamics, obtained through a fast growth process. Further, embryology-inspired approaches can sustain interactions with the environment in a natural way (embedding a self–repair mechanism in most cases, and a form of adaptation in some cases), therefore complementing the natural selection process at the heart of EC techniques.

5.1 Evo-Devo Computation

In analogy to *Evolutionary Developmental Biology* (Evo-Devo), we refer to *Evolutionary Developmental Computation* as the emerging area of Artificial Embryology combined with Evolutionary Computation and Embryonics. It consists of the following elements:

- *genotype encoding*: an alphabet ($\mathcal{A}$) of B symbols, and a structure (typically linear, i.e. string of symbols from $\mathcal{A}$, but can also be a multiset, such as compositional genomes [50]); e.g. DNA: $\mathcal{A} = \{A, C, T, G\}$, $B = 4$. The genome of an individual may be composed of several genes that can be expressed independently.

- *genotype-phenotype mapping = genetic code*: a translation table that gives the corresponding phenotype unit for each codeword (codon) in the genetic code.

  code length (L): length of each codeword, in number of atoms of the alphabet (e.g. in DNA, one codon has $L = 3$ nucleotides)

  code size (S): maximum number of different phenotype units (e.g. aminoacids) that can be encoded with a code of length L:

  $S = B^L$, e.g. $S = 4^3 = 64$ in the case of codon-aminoacid code

Several types of mapping are possible:

- 1-to-1 mapping: there is a one-to-one correspondence between each codeword and each distinct phenotype unit; this simple mapping is also known as *direct encoding*, and is typically associated to a non-developmental approach.
- n-to-1 mapping = redundant: several codewords may map to the same phenotype unit (e.g. several codons map to the same amino-acid)
- 1-to-n mapping = ambiguous, non-deterministic: one codeword may map to multiple phenotypes, in which case one of them is chosen at random. (e.g. Chemical Genetic Programming [44])
- n-to-n mapping: combining n-to-1 and 1-to-n: this is a more rare and complicated one. We believe that the coevolution of translation tables in Chemical GP [44] may fall within this category.

The mappings other than 1-to-1 are collectively referred to as *indirect encoding* (see Section 3.6). Indirect encodings typically lead to *non-linear* genotype-phenotype mappings [4], since there is a non-linear relationship between the complexity of the genotype and that of the corresponding phenotype; for instance, a small change in the genotype length might lead to a multiple-fold increase in phenotype complexity.

- *gene expression*: the process of translating a genotype into a phenotype, possibly involving a regulation process.
- *regulation*: feedback process that expresses some genes in the genotype more intensely than others, according to some influence from the environment on some of the generated phenotypes (transcription factors), which then cause the expression activity of certain genes to be activated or inhibited.
- *differentiation*: this is applicable in multicellular settings where multiple cells (e.g. in a grid or cellular automaton) have the same genotype, which is expressed into different phenotypes in response to signals (e.g. concentrations of certain chemicals) present in the environment surrounding the cells.

  A differentiation process requires some form of feedback from the environment in order to activate or inhibit certain genes; thus differentiation requires regulation.
- *growth*: the process by which the phenotype continues the developmental process and "grows" into a bigger and more complex structure than originally encoded in the genotype. For instance, a multicellular structure may grow more cells by replication.
- *morphogenesis*: the process by which a multicellular phenotype acquires a given target shape driven by *morphogens*. Morphogens are chemicals that control cell differentiation and growth.

The regulation of gene expression may lead to potentially large *Genetic Regulatory Networks (GRNs)*, in which the product of a gene may activate or inhibit the expression of other genes, which themselves might regulate other genes, and so on. The potential of GRNs for artificial evolution is only now starting to be understood, as several artificial GRN approaches to genetic and developmental systems are proposed [12, 7, 31].

A phenotype can also grow by *self-assembly* [25]. In this case, the genotype contains a set of building blocks which bind to each other with a certain affinity. Those building blocks that fit well together are then repeatedly reused to build the solution pattern.

Independent studies report *emergent self-repair* in developmental systems [2, 21, 49]: upon damage in large areas, the phenotype is able to regrow to its functional shape, even in the absence of explicit selection towards self-repair. Others explicitly select for self-repair [26]. Both emergent and explicit self-repair are interesting properties for autonomic systems which needs to be further explored.

6. RESEARCH DIRECTIONS: EMBRYONIC SOFTWARE

In this section, we aim at pointing out one possible application of the surveyed techniques to the design of autonomic computing/communication systems. Such novel concept, which we call EmbryoWare (Embryonic Software), focuses on the application of tools and techniques developed in embryonics (hence for hardware applications) to distributed software and service systems. The goal is, as in embryonics, to provide extreme fault tolerance and reconfiguration capabilities, but in this case to software systems.

Let us consider a distributed service, i.e., a service whose outcome comes from the interaction of different components running on different machines. Distributed services are prone to errors related to the possible faults of one (or more) of the machines where the components reside and run. This is particularly important in open uncontrolled environments, where the resources used for providing the service do not reside on dedicated servers but are the spare resources possibly present in user's desktop or even mobile devices. (Example of such scenarios are the SETI@Home initiative [52] and, to a given extent, PlanetLab [45].) In this context, there is a clear need to employ mechanisms ensuring system's ability to detect faults and recover automatically. It is not enough to rely purely on classical fault tolerance techniques, where failure modes must be pre-engineered into the system. Neither can we rely exclusively on evolutionary mechanisms for recovery purposes, as they tend to be slow and unreliable, requiring a resilience mechanism of their own. Clearly, embryonics provides a middle ground in which diversity can be exploited for quick repair and re-adaptation, without changing the underlying (potentially evolvable) genotype.

Such approach would involve the construction of artificial (in our case: software) stem cells, in the form of (concise) representation of the *complete* service process to be perfomed. In biological terms, the stem cell should contain the whole genome. Such artificial stem cells shall be spread in the network, where they shall differentiate (for example following a reaction-diffusion pattern) into the various components needed for performing the service. Adequate signalling mechanisms shall be provisioned, so that cells could exchange information about the state of their neighbours. Upon detection of a fault in a neighbouring cell, they could re-enter the embryo state, for differentiating again into the

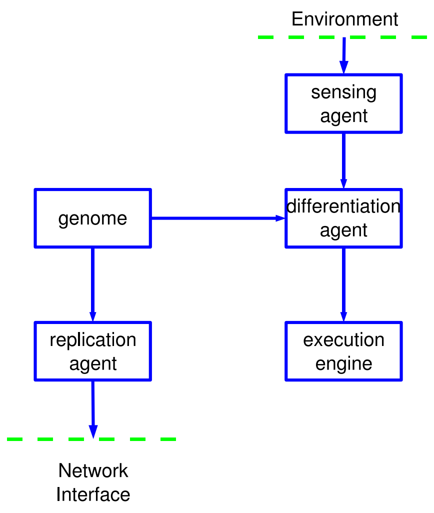

Figure 2: Possible architecture of an EmbryoWare cell.

required functionalities, expressing the necessary genes. A basic architecture of a cell in EmbryoWare is depicted in Fig. 2.

This mechanism can be complemented with some form of distributed monitoring for ensuring the expression of the correct functionalities by each single cell. This can be done by implementing some form of "monitoring agents", whose functioning can be based on Artificial Immune Systems theory [14].

From a research perspective, the most challenging issue seems to be the development of an adequate representation for distributed software/service systems, which is at the same time concise and expressive. Conciseness is important in order to encode highly complex services in short genotypes that can be easily disseminated over several nodes for distributed evolution. Expressiveness is important to maintain all the semantics necessary to express the whole service functionalities.

In order to provide insight into the functioning of an EmbryoWare–like system, we consider an example analogous to the BioWatch [61] developed by researchers at EPFL active in embryonics. The problem can be stated as follows: given a linear disposition of machines, each one equipped with a screen, find the "best" way to display the current time (as `hh:mm:ss`). Five basic components are necessary to build such system in software using the EmbryoWare concept:

1. A separator;
2. A system call to the 'second' field of the current time;
3. A system call to the 'minute' field of the current time;
4. A system call to the 'hour' field of the current time;
5. A system call to the current time (expressed as `hh:mm:ss`).

Each of these components is mapped to a cell state, using, e.g., integers in the range $[1; 5]$. State -1 will be used to identify faulty cells (i.e., cells which cannot perform any operations). An additional state, 0 is added to denote a cell that is not performing any task. All cells are assumed to be in state 0 at system bootstrap. Differently from embryonics, we assume that faulty cells cannot simply be bypassed in software; we believe it is indeed more realistic to assume that they represent "barriers" separating groups of active cells. This is because a malfunctioning piece of software must not be trusted to perform any useful functionality, not even relaying signals to a nearby neighbour. The sensing agent periodically queries nearby cells (on the left and right) to check their state (`leftState` and `rightState` respectively). The genome contains a table mapping the triplet `<currentState,leftState,rightState>` to the new state, `nextState`. A routine is periodically run by the differentiation agent to decide on the next state. The genome needs to be engineered in such a way to let the running system (phenotype) present the following behaviour:

1. if three cells in a row are active and available, display the hours on the first one, the minutes on the second one and the seconds on the third one, then add a separator;
2. if two cells in a row are active and available, display `hh:mm:ss` on both of them;
3. if one cell in a row is active and available, display `hh:mm:ss`.

The problems are related to the fact that cells have visibility of the state of their neighbours only, so that they do not know directly the state of a cell two hops away. This has to be inferred by implementing a (simple) self–organization process, which triggers reactions between nearby cells. A possible representation of the genome for the EmbryoWare–based BioWatch is given in Table 1. The symbol '*' is used to indicate the lack of dependency on the value of a given parameter. The symbol '!x' is used to indicate a value different from x. Such set of rules ensures that the desired behaviour is attained.If a cell fails, the system reacts in such a way to ensure that, after a few steps, the desired behaviour is restored.

Table 1: Possible genome representation of an EmbryoWare–based BioWatch.

currentState	leftState	rightState	nextState
-1	*	*	-1
!-1	5	-1	5
!-1	5	!-1	0
!-1	4	-1	5
!-1	4	!-1	3
!-1	3	*	2
!-1	2	*	1
!-1	0	0	4
!-1	0	-1	5
!-1	0	3	4
!-1	0	1,2,4,5	1
!-1	-1	-1	5
!-1	-1	5	5
!-1	-1	1,2,3,4	4
!-1	1	-1	5
!-1	1	5	5
!-1	1	1,2,3,4	4

Deriving the genome, even for a simple application like the one we are considering, is a non–trivial task. This relates to the fact that the behaviour depends on the state of cells that are more than one–hop away, i.e., out of the sensing range. Hence, a careful design is necessary to devise reaction-diffusion patterns able to attain the desired behaviour. This contrasts sharply with the approach (based on Ordered Binary Decision Diagrams [1]) that can be used to design genomes in embryonics [35]. Such approach is not suitable to our framework due to the fact that we do not consider the possibility of shortcircuiting faulty cells, but, rather, they constitute borders blocking the propagation of signals. The definition of a framework for automating the programming of genomes for EmbryoWare–like systems, given a high–level definition of the desired system behaviour, is currently an open research challenge.

7. CONCLUSION

In this paper, we have presented a survey of existing approaches in Evolutionary Computation and Embryology-inspired techniques and discussed their applicability for building autonomic computing systems. While these techniques have been proven useful in a variety of problems, we are still far from the application of such techniques for creating and evolving software in an online and dynamic way, which is at the very hearth of autonomic computing systems.

There are indeed issues of fundamental nature to be addressed before moving to the application of such techniques to autonomic and distributed computing/communication systems. In particular, such fundamental issues include:

- *Complexity:* there is a clear need to envisage approaches able to solve problems of considerable complexity. This is especially relevant to research in EC, where many approaches are known to suffer (in terms, e.g., of convergence time, computing/memory requirements etc.) when facing complex problems as those commonly found in autonomic computing and communication systems.

- *On-Line:* we need to envisage approaches for solving problems in an on–line fashion. This means that the intermediate solutions achieved during evolutionary/developmental processes need to be valid ones, i.e., able to ensure a minimum set of functionalities without disrupting the system's operations.

- *Distributed:* as many current applications are built around distributed software systems, we need to detail mechanisms for distributed, cooperative evolution/development. In such a view, components running on different machines and performing different tasks should change in a coordinated way, ensuring an improvement of the global system operations.

Notwithstanding such challenges, there is a considerable body of works in the area that can provide insight into the design of novel, bio-inspired techniques. The aim is to overcome conventional static solutions by enriching them with the possibility of evolving new configurations in an unsupervised manner. This shall provide additional degrees of flexibility, robustness and resilience to the system as a whole.

At the same time, Computational Evolution and the forthcoming second generation of bio-inspired systems [8, 65] (which are looking more closely into biology than EC in order to draw inspiration from accurate biological models) bring with them the promises of moving one step further in the creation of suitable computational models for self–creating and self–improving software. While this may take long to come, it is our belief that the area of bio-inspired solutions to autonomic computing represents the most promising and viable approach to address such problems.

8. REFERENCES

[1] Andersen, H. R. An introduction to binary decision diagrams, 1999. http://www.itu.dk/people/hra/bdd-eap.pdf.

[2] Andersen, T., Newman, R., and Otter, T. Development of virtual embryos with emergent self-repair. In *Technical Report FS-06-03, Developmental Systems, AAAI Fall Symposium* (2006).

[3] Araújo, S. G., Pedroza, A. C. P., and Mesquita, A. C. Evolutionary Synthesis of Communication Protocols. *10th International Conference on Telecommunications (ICT 2003) 2* (Feb.-Mar. 2003), 986–993.

[4] Attolini, C. S.-O. *From Molecular Systems to Simple Cells: a Study of the Genotype-Phenotype Map.* PhD dissertation, University of Vienna, Nov. 2005.

[5] Azad, R. M. A. *A Position Independent Representation for Evolutionary Automatic Programming Al gorithms - The Chorus System.* PhD dissertation, University of Limerick, 2003.

[6] Banâtre, J.-P., Fradet, P., and Radenac, Y. A Generalized Higher-Order Chemical Computation Model with Infinite and Hybrid Multisets. In *1st International Workshop on New Developments in Computational Models (DCM'05)* (2005), pp. 5–14. To appear in ENTCS (Elsevier).

[7] Banzhaf, W. Artificial Regulatory Networks and Genetic Programming. In *Genetic Programming - Theory and Applications, R. Riolo, B. Worzel (Eds.).* Kluwer Academic, Boston, MA, 2003, ch. 4, pp. 43–61.

[8] Banzhaf, W., Beslon, G., Christensen, S., Foster, J., Képès, F., Lefort, V., Miller, J., Radman, M., and Ramsden, J. From Artificial Evolution to Computational Evolution: A Research Agenda. *Nature Reviews Genetics* (2006), 729 – 735.

[9] Banzhaf, W., and Lasarczyk, C. Genetic Programming of an Algorithmic Chemistry. In *Genetic Programming Theory and Practice II, O'Reilly et al. (Eds.)*, vol. 8. Kluwer/Springer, 2004, ch. 11, pp. 175–190.

[10] Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. *Genetic Programming, An Introduction.* ISBN 155860510X. Morgan Kaufmann Publishers, Inc., 1998.

[11] Bentley, P. J., and Kumar, S. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)* (Orlando, Florida USA, 1999), pp. 35–43.

[12] Bongard, J. C. Evolving Modular Genetic Regulatory Networks. In *Proceedings of the IEEE*

2002 Congress on Evolutionary Computation (CEC2002) (2002), vol. 2, pp. 1872–1877.

[13] Bosman, P. A. N. Learning, anticipation and time-deception in evolutionary online dynamic optimization. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 2005)* (Washington DC, USA, June 2005), pp. 39–47.

[14] Bradley, D., Ortega-Sanchez, C., and Tyrrell, A. Embryonics + immunotronics: a bio-inspired approach to fault tolerance. In *Proc. of NASA/DoD Workshop on Evolv. Hardw.* (2000), pp. 215–223.

[15] Calude, C. S., and Paun, G. *Computing with Cells and Atoms: An Introduction to Quantum, DNA and Membrane Computing.* Taylor & Francis, 2001.

[16] Cavill, R., Smith, S., and Tyrrell, A. Multi-Chromosomal Genetic Programming. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 2005)* (Washington DC, USA, June 2005), pp. 1753–1759.

[17] Cobb, H. G. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Tech. Rep. AIC-90-001, Naval Research Laboratory, Washington, USA, 1990.

[18] Deckard, A., and Sauro, H. M. Preliminary Studies on the In Silico Evolution of Biochemical Networks. *ChemBioChem 5*, 10 (2004), 1423–1431.

[19] Dittrich, P. Chemical Computing. In *Unconventional Programming Paradigms (UPP 2004), Springer LNCS 3566* (2005), pp. 19–32.

[20] Eiben, A., and Smith, J. *Introduction to Evolutionary Computing.* Springer, 2003.

[21] Federici, D., and Downing, K. Evolution and Development of a Multicellular Organism: Scalability, Resilience, and Neutral Complexification. *Artificial Life 12*, 3 (2006), 381–409.

[22] Ferreira, C. Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. *Complex Systems 13*, 2 (2001), 87–129.

[23] Foster, J. A. Evolutionary Computation. *Nature Reviews Genetics* (June 2001), 428–436.

[24] Freitas Jr., R. A., and Merkle, R. C. *Kinematic Self-Replicating Machines.* Landes Bioscience, Georgetown, TX, USA, 2004. available online http://www.molecularassembler.com/KSRM.htm.

[25] Fuechslin, R. M., Maeke, T., Tangen, U., and McCaskill, J. S. Evolving inductive generalization via genetic self-assembly. *Adv. Complex Systems 9*, 1-2 (2006), 1–29.

[26] Grajdeanu, A. Methods for open-box analysis in artificial development. In *9th Genetic and Evolutionary Computation Conference (GECCO 2007)* (New York, NY, USA, 2007), ACM, pp. 1005–1012.

[27] Holland, J. *Adaptation in Natural and Artificial Systems.* MIT Press, 1992. First Edition 1975.

[28] Jin, Y., and Branke, J. Evolutionary Optimization in Uncertain Environments - A Survey. *IEEE Transactions on Evolutionary Computation 9*, 3 (June 2005), 303–317.

[29] Koza, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[30] Kumar, S., and Bentley, P. J. Computational embryology: Past, present and future. In *Advances in Evolutionary Computing, Theory and Applications*, Ghosh and Tsutsui, Eds. Springer, 2003, pp. 461–478.

[31] Kuo, P., Banzhaf, W., and Leier, A. Network topology and the evolution of dynamics in an artificial genetic regulatory network model created by whole genome duplication and divergence. *Biosystems 85* (2006), 177–200.

[32] Langdon, W. B., and Poli, R. *Foundations of Genetic Programming.* Springer, 2002.

[33] Leier, A., Kuo, P. D., Banzhaf, W., and Burrage, K. Evolving Noisy Oscillatory Dynamics in Genetic Regulatory Networks. In *Proc. 9th European Conference on Genetic Programming* (Budapest, Hungary, Apr. 2006), P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt (Eds.) Springer LNCS 3905, pp. 290–299.

[34] Luke, S., and Panait, L. Fighting Bloat With Nonparametric Parsimony Pressure. In *Parallel Problem Solving from Nature (PPSN VII), LNCS 2439* (2002), pp. 411–421.

[35] Mange, D., Stauffer, A., and Tempesti, G. Embryonics: a microscopic view of the molecular architecture. In *Proc. of ICES* (Lausanne, CH, 1998), pp. 185–195.

[36] Manna, Z., and Waldinger, R. Fundamentals of Deductive Program Synthesis. *IEEE Transactions on Software Engineering 18*, 8 (Aug. 1992), 674 – 704.

[37] Matsumaru, N., di Fenizio, P. S., Centler, F., and Dittrich, P. On the Evolution of Chemical Organizations. In *Proc. 7th German Workshop on Artificial Life* (2006), pp. 135–146.

[38] McKinley, P., Cheng, B., Ofria, C., Knoester, D., Beckmann, B., and Goldsby, H. Harnessing Digital Evolution. *IEEE Computer 41*, 1 (Jan. 2008), 54–63.

[39] Miller, J. F., and Thomson, P. Cartesian Genetic Programming. In *Genetic Programming, Proceedings of EuroGP'2000* (Edinburgh, Apr. 2000), R. P. et al., Ed., vol. 1802 of *LNCS*, pp. 121–132.

[40] Nolfi, S., and Floreano, D. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines.* MIT Press, 2000.

[41] O'Neill, M., and Ryan, C. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language.* Kluwer Academic Publishers, 2003.

[42] Ortega-Sanchez, C., Mange, D., Smith, S., and Tyrrell, A. Embryonics: a bio-inspired cellular architecture with fault-tolerant properties. *Genetic Programming and Evolvable Machines 1* (2000), 187–215.

[43] Paun, G. Computing with Membranes. *Journal of Computer and System Sciences 61*, 1 (2000), 108–143.

[44] Piaseczny, W., Suzuki, H., and Sawai, H. Chemical Genetic Programming - The Effect of Evolving Amino Acids. In *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference (GECCO 2004)* (July 2004).

[45] PlanetLab Initiative. http://www.planet-lab.org/.

[46] Poli, R. Parallel Distributed Genetic Programming. In *New Ideas in Optimization*. McGraw-Hill, Maidenhead, Berkshire, England, 1999, ch. 27, pp. 403–431.

[47] Probert, R. L., and Saleh, K. Synthesis of Communication Protocols: Survey and Assessment. *IEEE Transactions on Computers 40*, 4 (Apr. 1991), 468 – 476.

[48] Prodan, L., Tempesti, G., Mange, D., and Stauffer, A. Embryonics: artificial stem cells. In *Proc. of ALife VIII* (2002), pp. 101–105.

[49] Roggen, D., Federici, D., and Floreano, D. Evolutionary morphogenesis for multi-cellular systems. *Genetic Programming and Evolvable Machines 8*, 1 (Mar. 2007), 61–96.

[50] Segré, D., and Lancet, D. Composing Life. *EMBO Reports, European Molecular Biology Organization 1*, 3 (2000), 217–222.

[51] Selfridge, O. G. Learning and Education: A Continuing Frontier for AI. *IEEE Intelligent Systems 21*, 3 (May-June 2006).

[52] SETI@Home Initiative. http://setiathome.ssl.berkeley.edu/.

[53] Sharples, N. *Evolutionary Approaches to Adaptive Protocol Design*. PhD dissertation, University of Sussex, UK, Aug. 2001.

[54] Sharples, N., and Wakeman, I. Protocol construction using genetic search techniques. In *Real-World Applications of Evolutionary Computing – EvoWorkshops 2000* (Edinburgh, Scotland, Apr. 2000), Springer LNCS 1803.

[55] Sipper, M., Sanchez, E., Mange, D., Tomassini, M., Perez-Uribe, A., and Stauffer, A. A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems. *IEEE Transactions on Evolutionary Computation 1*, 1 (Apr. 1997).

[56] Spector, L., Klein, J., and Keijzer, M. The Push3 execution stack and the evolution of control. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 2005)* (2005), pp. 1689–1696.

[57] Spector, L., and Robinson, A. Genetic Programming and Autoconstructive Evolution with the Push Programming Language. *Genetic Programming and Evolvable Machines 3*, 1 (2002), 7–40.

[58] Spector, L., and Stoffel, K. Automatic generation of adaptive programs. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior: From animals to animats 4* (Cape Code, USA, 9-13 Sept. 1996), P. Maes, M. J. Mataric, J.-A. Meyer, J. Pollack, and S. W. Wilson, Eds., MIT Press, pp. 476–483.

[59] Spector, L., and Stoffel, K. Ontogenetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference* (Stanford University, CA, USA, 28–31 July 1996), J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds., MIT Press, pp. 394–399.

[60] Stanley, K. O., and Miikkulainen, R. A taxonomy for artificial embryogeny. *Artif. Life 9* (2003), 93–130.

[61] Stauffer, A., Mange, D., Tempesti, G., and Teuscher, C. A Self-Repairing and Self-Healing Electronic Watch: The BioWatch. In *Evolvable Systems: From Biology to Hardware* (2001), Springer LNCS Volume 2210, pp. 112–127.

[62] Sterritt, R., Hinchey, M. G., Rash, J. L., Truszkowski, W., Rouff, C., and Gracanin, D. Towards Formal Specification and Generation of Autonomic Policies. In *1st IFIP Workshop on Trusted and Autonomic Ubiquitious and Embedded Systems* (Nagasaki, Japan, Dec. 2005).

[63] Tanev, I., and Yuta, K. Epigenetic Programming: an Approach of Embedding Epigenetic Learning via Modification of Histones in Genetic Programming. In *Proc. Congress on Evolutionary Computation (CEC)* (2003), pp. 2580–2587.

[64] Tempesti, G., Mange, D., and Stauffer, A. Bio-inspired computing architectures: the *embryionics* approach. In *Proc. of IEEE CAMP* (2005).

[65] Timmis, J., Amos, M., Banzhaf, W., and Tyrrell, A. Going back to our Roots: Second Generation Biocomputing. *International Journal on Unconventional Computing 2*, 4 (2006), 349–382.

[66] Viswanathan, S., and Pollack, J. How Artificial Ontogenies Can Retard Evolution. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 2005)* (Washington DC, USA, June 2005), pp. 273–280.

[67] Weeds, J., Keller, B., Weir, D., Wakeman, I., Rimmer, J., and Owen, T. Natural Language Expression of User Policies in Pervasive Computing Environments. In *Proc. LREC Workshop on Ontologies and Lexical Resources in Distributed Environments (OntoLex)* (2004).

[68] W.G.Lasarczyk, C., and Banzhaf, W. An Algorithmic Chemistry for Genetic Programming. In *Proc. 8th European Conference on Genetic Programming* (Lausanne, Switzerland, Apr. 2005), M. Keijzer, A. Tettamanzi, P. Collet, M. Tomassini, J. van Hemert (Eds.) Springer LNCS 3447, pp. 1–129.

[69] Yamamoto, L. Code Regulation in Open Ended Evolution. In *Proceedings of the 10th European Conference on Genetic Programming (EuroGP 2007)* (Valencia, Spain, Apr. 2007), Ebner et al., Ed., vol. 4445 of *LNCS*, pp. 271–280. poster presentation.

[70] Yamamoto, L., and Tschudin, C. Experiments on the Automatic Evolution of Protocols using Genetic Programming. In *Proc. 2nd Workshop on Autonomic Communication (WAC)* (Athens, Greece, Oct. 2005), pp. 13–28.

[71] Yang, S. Memory-based immigrants for genetic algorithms in dynamic environments. In *Proc. of ACM GECCO* (2005), pp. 1115–1122.

[72] Yu, T. Program Evolvablility under Environmental Variations and Neutrality. In *Proceedings of the 9th European Conference on Artificial Life (ECAL 2007)* (Lisbon, Portugal, Sept. 2007).