AN APPROACH FOR SELF-CREATING SOFTWARE CODE IN BIONETS WITH ARTIFICIAL EMBRYOGENY

Ramaraju.S.V.S.V.Palla[1]

1 Assistant Professor, CSE Dept., DIET, Anakapalle, Andhra Pradesh, India, ***ramarajusvsvp@gmail.com***

Abstract

In this we present a review of techniques for optimized automatic evolution and creation of software. The focus is on new class of computer thinking with bio-inspired method using engineering technology process known as bottom-up approaches, in which complexity increases with interactions among smaller and simpler building block units. First, we study Evolutionary Computing (EC) techniques, focusing and presenting the advantages of their potential application to the automatic optimization of computer programs in a constant changing environment. Then, we inspect the approaches inspired by embryology, in which the artificial entities go through a developmental process. Then at the end concludes with a crucial discussion and positive outlook for applications of these techniques to the environment of BIONETS.

Index Terms: *Evolutionary Computing (EC), PO-Plane, Artificial Life, Artificial Evolution (AE), Intron, Embryogeny.*

----------***----------

1. INTRODUCTION

Structuring software is able to continuously improve itself automatically is a common goal in artificial intelligence, software engineering, and other areas of computer science, even in, autonomic systems and organic computing. The vision is to bring the ability of constantly searching for learning and adapt in computer.

Generally in this area we make use either 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 and more recently, derivation of policy rules from high-level representations closer to natural language. Bottom-up approaches look at how higher-level software functionality would emerge from lower-level interactions among simpler system units. Artificial Life (A Life), Evolutionary Computation, Swarm Intelligence 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 have 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 we provide a survey of bio-inspired approaches to such bottom-up creation of software functionality. Our focus is on dynamic, on-line processes where evolution and adaptation must happen endlessly, during the operation of the system, as opposed to off-line, 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 on-line 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. Numerous bio-inspired systems are available.

A classification was proposed in [1], 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 is organized as follows. In Sec. 2 we position our context within the classification adopted from [1]. In Sec. 3 we review the state-of-the-art in evolutionary computing with focus on online and dynamic environments. In Sec. 4 we present the two 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 concludes the chapter pointing out possible applications to BIONETS.

2. CONTEXT: THE PO-PLANE

A classification of bio-inspired systems was proposed in [1], positioning them in a 3-D space defined by three orthogonal axes. Although it was proposed ten years ago for hardware, its concepts remain valid today, and apply to software as well. We focus on two of the three axes, namely Phylogeny and Ontogeny. The third axis (Epigenesis), related to learning,

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 [1], today combinations of both approaches, forming the so-called PO-Plane, are becoming more and more common. On one hand, an indirect encoding followed by a developmental process has shown to increase the scalability of evolutionary computing for complex problems. 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 on-line dynamic optimization, which is the focus of the present chapter: 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 on-line environment. The potential of such combined PO-Plane approaches will be discussed in Section 5.

3. EVOLUTIONARY COMPUTING

Evolutionary Computing or Evolutionary Computation (EC) [2, 5] 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) [19], in which biology concepts are applied in a very simplified way. In [19] the authors propose a new term Computational Evolution (CE) to reflect a new generation of bio-inspired computing [20] that builds upon new knowledge from biology and increased synergies between biologists and computer scientists.

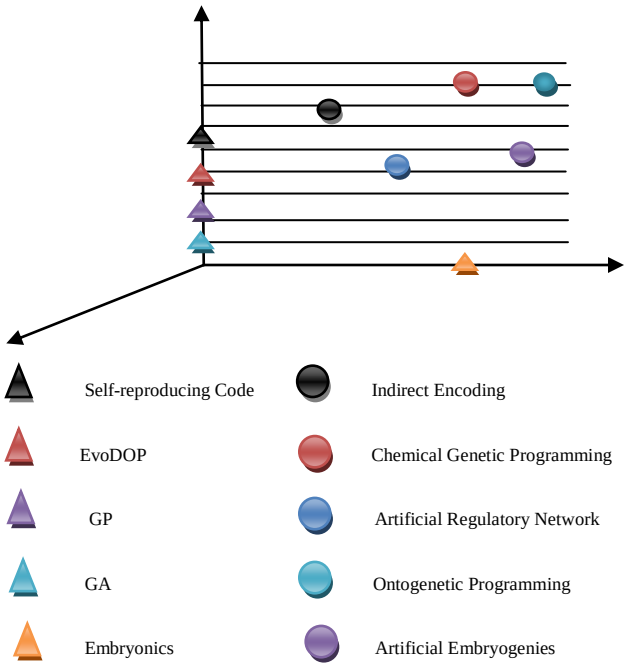

Fig1. The POE classification and some of the phylogenetic and ontogenetic approaches that will be treated in this paper

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 regulation 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 on-line 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) 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 optimizes a given fitness criterion. An initial population of candidate strings is generated and evaluated against the fitness criterion. Multiple candidate solutions are then chosen (usually with a probability which depends on the respective fitness level) for giving rise to the next generation. This is accomplished by applying genetic operators (e.g., crossover and mutation) to such candidate solutions in order to create new variants.

3.2 Genetic Programming

Genetic Programming (GP) [4] 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, finite state machines, neural networks, and more recently, chemical programs [21,15].

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 has 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.

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 on-line evolution. Indirect encodings will be briefly 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 Evolutionary Computing for Dynamic Optimization

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. The aim is to introduce mechanisms able to "track" the optimal solution. This field is referred to as Evolutionary Computing 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 Hyper-mutation, i.e. artificially high mutation rates in response to a change.
- Maintain diversity throughout the run, e.g. through random immigrants, individuals that move between sub-populations.
- Implicit memory: usually takes the form of a redundant representation such as diploid or polyploidy individuals with a dominance mechanism.
- Explicit memory: previously good solutions are stored in memory, and retrieved when the system encounters a previous situation for which the solution applied.
- Multi-population: Different sub-populations 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.

Although much has been done in EvoDOP in the GA domain, little has been explored in the GP domain. In [17] 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 [17] 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 an equivalent performance. Another research line would be to bring the most promising EvoDOP approaches to the GP domain, namely multi-population and anticipation. EvoDOP techniques are mainly situated along the "P" axis in the POE framework.

3.4 Self-Replicating and Self-Reproducing Code

Much attention has been paid in EC on self-replicating and self-reproducing code. In some cases, replication and reproduction have actually been considered synonymous, which they are not [1]. 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. 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 recoveries 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 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, where evolution still occurs mostly in a simulated environment. 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 ALife, the main role of programs is just to survive, and since they remain in a virtual world there is no risk for the end user.

Evolvable instructions set virtual machines are used in most well-known ALife systems, such as Tierra and Avida. They resemble assembly language, which is easily self-modifiable: one can write on memory positions that include the own memory location of the code. This is used to evolve software that self reproduces, adapts, seeks to survive, etc. A precursor of such machine language approach was Core Wars.

In the GP context, the Push family of programming languages [16] 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 Auto constructive Evolution, 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 [16], 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 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 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 in Evolutionary Computing

It is well known in Evolutionary Computing 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 and Grammatical Evolution [6] 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 [22] and artificial systems [18]. In [22] a genetic network exhibiting stochastic dynamics is evolved using a set-based encoding of systems of biochemical reactions. In [18] 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 mutations. These networks are able to compute functions, such as a sigmoid and a decaying exponential.

Chemical Genetic Programming [10] proposes a feedback-based dynamic genotype to phenotype translation mechanism inspired by a cell's dual step transcription-translation process from DNA 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 relatively simple problems. Epigenetic Programming associates a developmental process to GP, in which an Epigenetic Learning (EL) algorithm activates or silences certain parts of the genetic code. 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) 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 any dynamic feedback mechanism.

3.8 Chemical Computing Models and Evolution

Chemical computing models [3,13] express computations as chemical reactions that consume and produce computation objects (data or code). Objects are represented as elements in a multi set, an unordered set within which elements may occur more than one. The number of occurrences of a given element within the multi set is called the multiplicity of the element.

We believe that chemical models have a great potential for the class of on-line dynamic software optimization problems that we are aiming at. This is due to their inherent parallelism and multi set 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 multi set 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 [12] is a reaction vessel in which instructions are executed in random order. In [12] the power of GP applied to an algorithmic chemistry on evolving solutions 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 [21], using organization theory. Artificial biochemical signaling networks are evolved in [11] to compute several mathematical functions, using an evolutionary computation method based on asexual reproduction and mutations. Although the networks evolved in [11] show computational capacity, it does not seem trivial to extend their capabilities from mathematical functions to generic software actions.

We are working on our own chemical programming language [15,23] which we are extending with generic computation capacity and an evolution framework in which evolution will occur in an intrinsic and asynchronous manner as in nature. Variation operations such as mutation and recombination will occur within the individuals themselves as self-modifying code.

With such chemical systems it becomes possible to quantitatively regulate the behavior of programs for evolution or adaptation purposes. An example of that is Chorus [7], 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 [23] 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 [7] chooses the rule with the highest concentration, in [23] the choice is probabilistic: the chance of a variant being picked for execution is proportional to the concentration of its expression signals. While [23] is explicitly intended for on-line problems, to the best of our knowledge [7] 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 the ability to replicate in silico such process would enable researchers to break the complexity ceiling which limits the ability of conventional EC techniques.

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 computing 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 behaviors. 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 key objective of embryonics [14] is to implant great fault tolerance into electronic devices (e.g., FPGA arrays) while maintaining the redundancy at acceptable levels. Approaches in this area have mostly focused on the use of artificial stem cells,. 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 result 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:

- Every unit/cell (smallest building block) contains the entire genome. Every cell is totipotent, 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 damaged component, can return to the stem cell state and differentiate into another type of cell to repair the damage. This step involves the availability of “spare” cells, which provide resources needed to change the faulty component.

The major variation among 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), while in software it could also take the

form of a computer program. Let us, for example, 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. A reactive and efficient self-repair or self-healing ability is essential in this context. It is not enough to rely purely on classical fault tolerance, where failure modes must be pre-engineered into the system. Neither can we rely exclusively on evolutionary mechanisms, which 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. This will involve the construction of artificial stem cells, in the form of representation of the complete service instructions to be used. Such artificial stem cells shall be spread in the network, where they shall differentiate (for example following a reaction-diffusion pattern) into the different components needed for performing the service. Upon detection of a fault, they could re-enter the embryo state, for differentiating again into the required functionalities, expressing the necessary genes.

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 [9] is a branch of Evolutionary Computing (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 [2]. 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 cross over and mutation. Approaches in the GP area are classified according to the genotype representation and decoding: grammatical evolution and developmental / ontogenetic GP [2]. 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 [6]. 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. One of the most prominent examples of the second case is Ontogenetic Programming (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 easily imagine a grammar to express self-modifying programs (for instance, a grammar that encodes for the stack-based linear programs in 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 [9] 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.

Studies on artificial embryogeny have reported a clear advantage of an indirect encoding over a direct one such as tree-based GP. On the other hand, further experiments [8] 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. 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 incorporate both an ontogenetic as well as a phylogenetic feature: as such, they lie in the PO–plane.

5. DISCUSSION: EC AND EMBRYOLOGY:

COMMON SYNERGIES

EC and Embryology-based approaches affect both the same research area, and a considerable overlap in the research communities involved with the outstanding exception of embryonics, there are many interactions which have not been exploited so far. Definitely, embryology-based approaches are still in their beginning, having been mostly applied to solve simple problems. At the same time, it is worth remarking that a full understanding of all the combinations possible on the PO-Plane is still absent. Nevertheless, embryology–based approaches have the prospective to complement genetic-based EC techniques by providing a different level of system dynamics. One of the advantages of artificial embryogenies is related to the possibility of encoding complex behaviours in a parsimonious way (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. This is extremely important in applications requiring near real–time flexibility, a feature badly supported by pure evolutionary approaches. Such an issue is extremely important in the perspective of the BIONETS scheme, where services are expected to be able to self–organize and adapt to unpredictable working conditions in an extremely timely manner.

Embryology-inspired methods can withstand the interactions with the environment in a natural way (a self-repair mechanism), so supplementing the natural choice process at the hearth of EC techniques. Which is a particularly significant when EC techniques are applied in an on–line fashion.

Indeed, EC technique scan easily lead to the making the evolutionary paths of organisms which are not able to perform the expected operations. In conventional EC applications this is not a problem, as midway solutions are not turned into a working system but only the concluding result of the evolutionary process is used. On the opposing, in an on–line evolution also midway solutions are used for executing the system's operations. The capability to repair automatically and/or to withstand the presence of faulty components becomes a critical goal for ensuring purposeful system operations. We trust that grouping of the two methods can be recycled to successfully design distributed, autonomic software systems as addressed within the BIONETS development.

CONCLUSIONS

In this, we have represented a survey of existing approaches in Artificial Embryology-inspired techniques with evolutionary computing. Which are useful in a different categories of problems, but we are still away from the application of such techniques for evolving and creating software in a non-line and dynamic way.

In the viewpoint of the BIONETS development, there are issues of fundamental nature which need to be began before moving to the application of such techniques for solving the problems related to autonomic computing and communication systems.

Nevertheless, there is a considerable body of works in the area which can provide vision into the design of new, bio-inspired techniques which may prove to overcome straight static solutions by inspiring them with the option of growing in an unverified manner new configurations, providing greater elasticity, toughness and spirit.

At the same time, CE and the future second generation of bio-inspired systems [50] brings the promises of moving one step further in the making of suitable computational models for self-creating software code. While this may take long to come, it is our trust that the area of bio-inspired solutions to autonomic computing characterizes the most talented and feasible approach to attack such problems.

REFERENCES

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

[2] J. A. Foster. "Evolutionary Computation". Nature Reviews Genetics, pages 428–436, June 2001.

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

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

[5] A.Eiben and J. Smith. "Introduction to Evolutionary Computing". Springer, 2003.

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

[7] R. M. A. Azad. "A Position Independent Representation for Evolutionary Automatic Programming Algorithms – The Chorus System". PhD dissertation, University of Limerick, 2003.

[8] S. Kumar and P. J. Bentley. "Computational Embryology: Past, Present and Future". In Ghosh and Tsutsui, editors, Advances in Evolutionary Computing,

Theory and Applications, pages 461–478. Springer, 2003.
[9] K. O. Stanley and R. Miikkulainen. "A taxonomy for artificial embryogeny". Artif. Life, 9:93–130, 2003.
[10] W. Piaseczny, H. Suzuki, and H. Sawai. "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
[11] A.Deckard and H. M. Sauro. "Preliminary Studies on the In Silico Evolution of Biochemical Networks". ChemBioChem,5(10):1423–1431, 2004.
[12] W. Banzhaf and C. Lasarczyk. "Genetic Programming of an Algorithmic Chemistry". In Genetic Programming The oryand Practice II, O'Reilly et al. (Eds.), volume 8, chapter 11, pages 175–190. Kluwer/Springer, 2004.
[13] P. Dittrich. "Chemical Computing". In Unconventional Programming Paradigms (UPP 2004), Springer LNCS 3566, pages19–32, 2005.
[14] G. Tempesti, D. Mange, and A. Stauffer. "Bio-inspired computing architectures: the Embryionics approach". In Proc. Of IEEE CAMP, 2005.
[15] L. Yamamoto and C. Tschudin. "Experiments on the Automatic Evolution of Protocols using Genetic Programming". In Proc. 2nd Workshop on Autonomic Communication (WAC), pages 13–28, Athens, Greece, October 2005.
[16] L. Spector, J. Klein, and M. Keijzer. "The Push3 execution stack and the evolution of control". In Proc. Genetic and Evolutionary Computation Conference (GECCO 2005), pages 1689–1696, 2005.
[17] R. Cavill, S. Smith, and A. Tyrrell. "Multi-Chromosomal Genetic Programming". In Proc. Genetic and Evolutionary Computation Conference (GECCO 2005), pages 1753–1759, Washington DC, USA, June 2005.
[18] P. Kuo, W. Banzhaf, and A. Leier. "Network topology and the evolution of dynamics in an artificial genetic regulatory network model created by whole genome duplication and divergence". Bio systems, 85:177–200, 2006.
[19] W. Banzhaf, G. Beslon, S. Christensen, J. Foster, F. Képès, V. Lefort, J. Miller, M. Radman, and J. Ramsden. "From Artificial Evolution to Computational Evolution: A Research Agenda". Nature Reviews Genetics, pages 729 – 735, 2006.
[20] J. Timmis, M. Amos, W. Banzhaf, and A. Tyrrell. "Going back to our Roots: Second Generation Biocomputing". International Journal on Unconventional Computing, 2(4):349–382, 2006.
[21] N. Matsumaru, P. S. di Fenizio, F. Centler, and P. Dittrich. "On the Evolution of Chemical Organizations". In Proc. 7thGerman Workshop on Artificial Life, pages 135–146, 2006.
[22] A.Leier, P. D. Kuo,W. Banzhaf, and K. Burrage. "Evolving Noisy Oscillatory Dynamics in Genetic Regulatory Networks" .In Proc. 9th European Conference on Genetic Programming, P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt(Eds.) Springer LNCS 3905, pages 290–299, Budapest, Hungary, April 2006.
[23] L. Yamamoto. "Code Regulation in Open Ended Evolution". In Ebner et al., editor, Proceedings of the 10th European Conference on Genetic Programming (EuroGP 2007), volume 4445 of LNCS, pages 271–280, Valencia, Spain, April 2007.poster presentation

BIOGRAPHIE:

Ramaraju.S.V.S.V.Palla received the M.Tech. from Andhra University, Andhra Pradesh, in 2005, pursuing P.hD in JNTU, Kakinada, Andhra Pradesh. His research interests include bio-informatics and bio-networks with security and communication realated areas.