Gardening Cyber-Physical Systems

Susan Stepney, Ada Diaconescu, René Doursat, Jean-Louis Giavitto, Taras Kowaliw, Ottoline Leyser, Bruce Maclennan, Olivier Michel, Julian F. Miller, Igor Nikolic, et al.

▶ **To cite this version:**

Susan Stepney, Ada Diaconescu, René Doursat, Jean-Louis Giavitto, Taras Kowaliw, et al.. Gardening Cyber-Physical Systems. Unconventionnal Computation and Natural Computation (UCNC'2012), Sep 2012, Orléans, France. pp.1-1. hal-01161014

HAL Id: hal-01161014
https://hal.science/hal-01161014v1

Submitted on 13 Jul 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Gardening Cyber-Physical Systems

Susan Stepney[1], Ada Diaconescu[2], René Doursat[3,4], Jean-Louis Giavitto[5], Taras Kowaliw[4], Ottoline Leyser[6], Bruce MacLennan[7], Olivier Michel[8], Julian F. Miller[9], Igor Nikolic[10], Antoine Spicher[8], Christof Teuscher[11], Gunnar Tufte[12], Francisco J. Vico[3], Lidia Yamamoto[13]

Depts of [1] Computer Science and [9] Electronics, University of York, UK
[2]CNRS LTCI, Télécom-ParisTech, France
[3]Research Group in Biomimetics (GEB), Universidad de Málaga, Spain
[4]Institut des Systèmes Complexes (ISC-PIF), CNRS, Paris, France
[5]UMR STMS 9912, IRCAM - CNRS, France
[6]Sainsbury Laboratory, University of Cambridge, UK
[7]Electrical Engineering and Computer Science, University of Tennessee, Knoxville, USA
[8]LACL, Université Paris-Est Créteil, France
[10]Faculty of Technology, Policy and Management, Technical University of Delft, Netherlands
[11]Dept Electrical and Computer Engineering, Portland State University, USA
[12]Dept Computer and Information Science, NTNU, Trondheim, Norway
[13]University of Strasbourg, France

Abstract. Today's artefacts, from small devices to buildings and cities, are, or are becoming, cyber-physical socio-technical systems, with tightly interwoven material and computational parts. Currently, we have to laboriously build such systems, component by component, and the results are often difficult to maintain, adapt, and reconfigure. Even "soft"ware is brittle and non-trivial to adapt and change. If we look to nature, however, large complex organisms grow, adapt to their environment, and repair themselves when damaged.

In this position paper, we present Gro-CyPhy, an unconventional computational framework for *growing* cyber-physical systems from *computational seeds*, and *gardening* the growing systems, in order to adapt them to specific needs.

The Gro-CyPhy architecture comprises: a *Seed Factory*, a process for designing specific computational seeds to meet cyber-physical system requirements; a *Growth Engine*, providing the computational processes that grow seeds in simulation; and a *Computational Garden*, where multiple seeds can be planted and grown in concert, and where a high-level gardener can shape them into complex cyber-physical systems.

We outline how the Gro-CyPhy architecture might be applied to a significant exemplar application: a (simulated) *skyscraper*, comprising several mutually interdependent physical and virtual subsystems, such as the shell of exterior and interior walls, electrical power and data networks, plumbing and rain-water harvesting, heating and air-conditioning systems, and building management control systems.

1 Introduction

Our artefacts, from small devices to buildings and cities, are, or are becoming, cyber-physical socio-technical systems, with tightly interwoven physical material and computational parts. We build such systems, laboriously placing material components, laboriously programming computational ones, laboriously integrating the parts, laboriously maintaining the resulting structures. In contrast, trees grow, adapting their form and function to the environmental conditions, and trees self-repair, using the same mechanisms as for growth. These properties allow trees to be gardened—planted, fed, pruned, trained—to meet human needs.

Our vision is of *construction by directed growth*, through gardening macroscopic cyber-physical artefacts formed from a growing, integrated combination of material and virtual subsystems. This is a novel approach to designing, implementing, and maintaining the wealth of cyber-physical artefacts that comprise our "built environment", including: housing, schools, hospitals, shops, and factories; transport, power, and communication infrastructures. The objective is an unconventional embodied computational process that will produce autonomous, adaptive, robust systems in a controllable and cost-effective manner.

The overall objective of computational growth is to be able to "grow" macroscopic objects that comprise both physical and computational aspects (cyber-physical systems, "smart" materials, etc). The related field of physical self-assembly focuses on the material side: designing nano- and micro-scale assemblers, or micro-scale artificial cells, that can manipulate and position matter. The computational control of such devices (how and where to position the relevant material; how and when to divide and replicate) is typically assumed to be a relatively trivial matter of programming. Little consideration is given to scalability, adaptability, correctness, or other computer science issues relating to the macro scale of cyber-physical artefacts. Here we consider the matters in the context of an unconventional computational framework.

The structure of rest of the paper is as follows. In §2 we introduce our Gro-CyPhy architecture for designing, growing, and gardening complex cyber-physical systems. In §3 we outline how this process would work for a substantial application: a skyscraper. In §4 we conclude with a discussion of how the Gro-CyPhy approach changes our ideas about artefact construction.

2 Gro-CyPhy Architecture

2.1 Overview

There are several visions of assembling or growing macroscopic artefacts from the bottom up. These include nanoscale robots manipulating material at the molecular level [7], which are an artificial analogue of termite and other social insect *construction* processes, and synthetic protocells dividing and differentiating [29], which are an artificial analogue of plant and animal *growth* processes.

Past work on exploiting growth metaphors has tended to concentrate on either purely virtual or purely physical structures. Concentrating on only the virtual loses the advantages of embodied physical reality [33, 34], and allows issues of reality constraints, such as the cost of replication and other such operations, to be neglected. Concentrating on only the physical ignores the computational issues of programming the desired growth, and the necessity of integrating physical and virtual subsystems.

Here we take the software perspective, and outline a computational growth process that, given the necessary physical assembly devices (nanobots or protocells), can program these to construct macroscopic cyber-physical structures. Additionally, the resulting growth process technology will be able to grow purely virtual computational structures in virtual environments, such as those running on computer networks and clouds.

In the current absence of such physical assembly devices, initial work is necessarily in simulation. However, the aim is for a process where a given program could either be embodied in a physical constructor that would provide the growth mechanisms, or placed in a virtual "growth engine" that would simulate the growth process. This dual approach also allows close integration and co-growth of purely virtual software components such as software control systems.

The focus of our approach is on the design and growth of a complex artefact with several interrelated subsystems. These subsystems must co-develop to produce an integrated whole. Additionally, we want our cyber-physical artefacts to be responsive and adaptive to their environments. For a growth approach, it makes sense to consider parallels with plants rather than with animals: plants are more "plastic" in their morphological responses. There is evidence that this sort of plasticity can be exploited in artificial developmental systems [16, 17, 35]. This plasticity admits a new possibility: deliberately and selectively applying environmental stimuli to direct growth in a desired direction—or *gardening* the growing artefact. To this end, we have designed the Gro-CyPhy architecture, which comprises three major components (figure 1):

- a *Seed Factory*, a process for designing specific computational seeds to meet cyber-physical system requirements
- a *Growth Engine*, providing the computational processes that grow physical seeds in simulation, and grow virtual seeds into software
- a *Computational Garden*, where multiple seeds can be planted and grown in concert, where virtual seeds can be interfaced with embodied growth processes, and where a high-level gardener can shape the whole into complex cyber-physical systems.

The gardening metaphor guiding this approach is an attempt at drastically changing the way we build cyber-physical systems. Traditionally, such systems are produced by a tightly coupled team of engineers following a life cycle that separates phases of design, production and use. In contrast, plants develop continuously. As a "product", a plant already has an innate ability to seamlessly adapt to its environmental conditions (within reasonable limits). Additionally,

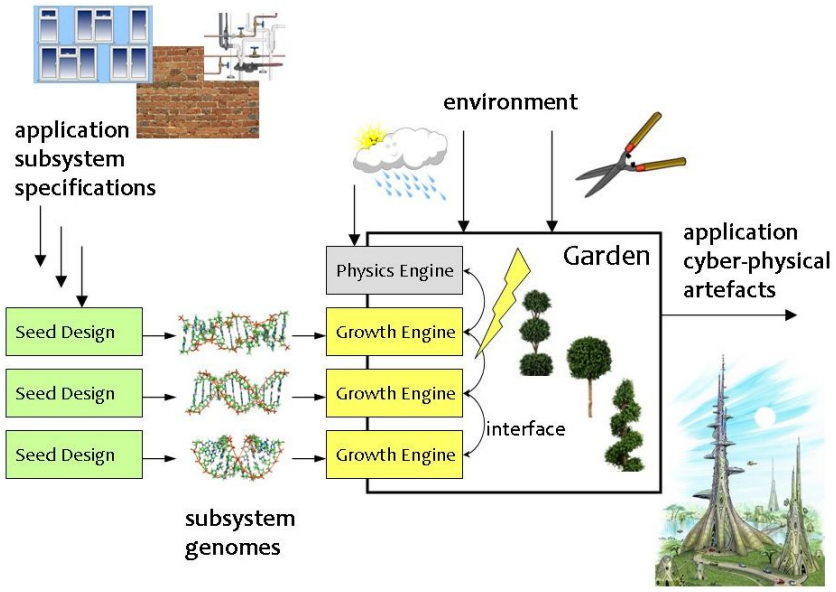

Fig. 1. Conceptual overview of the GRO-CYPHY architecture in application (here, a skyscraper): application-specific subsystem seed specifications are developed (wall, windows, plumbing, etc); the *Seed Factory* develops the relevant seeds (subsystem genomes); *Growth Engines* grow these seeds in the *Computational Garden* where the application specific gardening takes place, delivering the cyber-physical system end product. See following text for a full explanation of the various components and their relationships.

the shape and metabolism of a plant—its "purpose"—can be artificially adjusted and tailored at various stages:

1. before the growth phase, by choosing an adequate genome: this genome is typically programmed or designed by directed evolution (in the Seed Factory) which does not require an understanding of the lower-level processes (implemented in the Growth Engine)
2. at the beginning of the growth phase, by configuring the initial conditions of the seed, including the internal "metabolism" (growth preferences and parameter values), and possibly by introducing a pre-configured collection of initial components (a "seedling")
3. during the growth phase, by taking into account the various actions of the gardener and the interactions with the environment.

Even if the organism reaches a steady state, the growth phase never stops: there is no distinction between the "building phase" of the artefact, the "functional phase", or the "repair and maintenance" phase.

2.2 Seed Factory

The Seed Factory develops the seeds (genomes, programs) that are implanted either in physical devices (nanobots, protocells) or in virtual Growth Engines, to grow into the desired artefacts.

Seeds are tightly packed data structures that contain the necessary instructions (genome G) and initial conditions (internal state S) needed to grow a given target structure (the organism phenotype P). By "tightly packed" we mean that the actual phenotypic shape adopted after growth is not explicitly represented in the seed, but will unfold and reveal itself as the seed develops.

Several underlying technologies for defining and implementing seeds could be supported simultaneously in a GRO-CYPHY system, and should be chosen as appropriate for growing different kinds of structures. Those for physical technologies such as nanobots or protocells need to be tailored for the physical capabilities of the embodying systems. Example of virtual technologies include Morphogenetic Engineering (ME) [5, 6, 26], Developmental Cellular Automata (DCA) [20, 22], Generative Systems (GS) [11–13, 23, 24, 27], Self-Modifying Cartesian Genetic Programming (SMCGP) [8, 21], and Spatial Computing (SC) [4, 30, 31].

Several seed design methodologies are applicable to various types of computational cyber-physical structures. For example, simple to moderately sophisticated structures could grow out of simple seeds produced by a composition of elementary "rules of thumb" for seed design [1, 36]. More elaborate organisms could be difficult to engineer "by hand" in this way, and would require automatic search and optimisation methods to sweep a potentially vast space of possible seeds, in search of good ones able to grow into the desired structures (figure 2).

When "planted" in the computational garden and grown by the relevant Growth Engine, seeds should reliably provide the intended functionality, in a sufficiently flexible and fault-tolerant way such that a gardener can steer their growth towards customised organisms. The resulting organisms should have the features of genuinely adaptable components, which can be used at a higher level in large-scale systems, such as a building's physical architecture, technical networks, or distributed software.

Designing seeds deals with the mapping from a genome to a phenotype, $G \rightarrow P$. Although the growth of a given genotype into a fixed phenotype is already a stimulating challenge in itself, a greater goal is to design genomes that can support *adaptable* growth on multiple different levels, corresponding to the forms of interaction present in the growing models.

One form of adaptability involves modification of the phenotype via environmental influence; for an environment E_i we have $(G, E_i) \rightarrow P_i$. That is, on the ontogenetic timescale, and under a given genotype-phenotype mapping, the development process should be sensitive to, and modifiable by, environmental conditions in the garden. External conditions or stimuli encountered by one individual during its growth, whether of a mechanical or signalling type, should be able to influence the outcome. This is the level of P, or rather the genotype-phenotype mapping, for which natural analogies can be found more readily in the

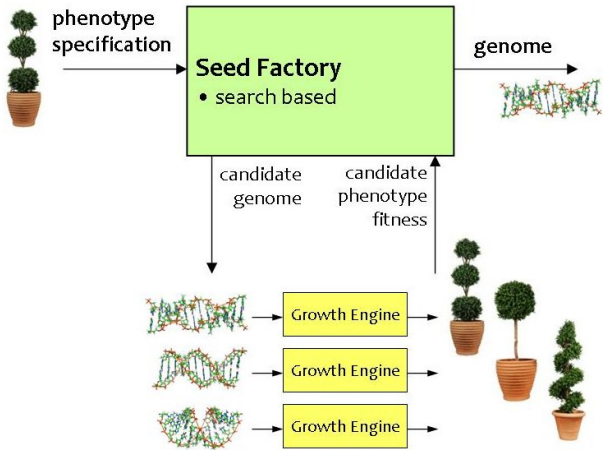

Fig. 2. Conceptual overview of the search process in the Seed Factory: high-level phenotype (grown) specifications are input; the search process develops the relevant seeds (subsystem genomes); it uses the Growth Engine to grow candidate seeds into phenotypes, which it evaluates against the specification, and feeds the information back into its search process.

plant kingdom than the animal kingdom. Plants and trees can be pruned, bent, arranged, sculpted, etc., whether intentionally by a human gardener (bonsais, espaliers, topiaries) or spontaneously when faced with adverse or favourable conditions (wind strength, rocky obstacles, soil composition, light intensity). This is precisely what computational gardeners expect to be doing. Therefore, the seeds that they "order" from the Seed Factory's species engineers should provide appropriately workable material.

A second form of adaptability involves *symbiosis*, or the growing of different phenotypes or individuals together. This includes both individuals from the same "species", but also individuals from different species as well (ecosystemic interactions). We may denote this $(G_i + G_j, E) \rightarrow (P_i, P_j)$.

To qualify as true "components", grown organisms must also be literally "symbiotic" with each other. Whether they belong to the same breed or species, or to different species, their shape and physiology should offer certain anchor points to be able to attach and function together into a larger architecture. Think of interfaces and method-calling in software objects; prefabricated walls and rooms in trailer homes; clusters and routing in networks. A cyber-physical system such as a skyscraper needs several different species of seeds to grow together: walls, plumbing, power, control systems, etc.

2.3 Growth Engine

A Growth Engine provides the computational mechanisms to grow a seed. This might be required to grow in simulation a seed intended for a physical device,

or to grow the seed of a virtual component such as a software control system. There are several types of growth engine, corresponding to the several types of seeds, based on different underlying physical technologies such as nanobots or protocells, or on the virtual technologies such as ME, DCA, GS, SMCGP, SC mentioned earlier. For example, a seed defined as a set of rewrite grammar rules would require a growth engine that could implement the grammar rewriting system (among other things, discussed below).

The Gro-CyPhy approach contains a conceptual switch from the notion of "machine" to the notion of "organism". An organism is an organised ensemble of components. This ensemble is derived from one generic component, the seed (or fertilised egg in animals), by duplication and differentiation, or alternatively by the self-assembly of components produced elsewhere. Each component has some autonomy, but it is also coupled and interacts with the other components and with the environment. Contrary to the relations between parts of a machine, the coupling between components of an organism is not the result of deliberate design: two components interact because they are neighbours (e.g. in physical space), not because these components have been pre-made to interact. This new type of relationship, natural and spatially explicit, requires the components to be generic: they are not designed for one specific and dedicated purpose but must be able to react to a wide variety of interactions. Furthermore, a complete artefact, classically several "machines", is now seen as several "organisms", grown together cooperatively in a garden, from several seeds.

The Growth Engine executes a programmable and reproducible indirect mapping from "genotypes" G (the local rules of growth and self-assembly followed by the components at the microscopic level) to "phenotypes" P (the global structure and function of the emergent systems at the macroscopic level), in the context of an environment E. Calculating the transformation from a given G to a resulting P corresponds to developing an organism. Solving the inverse problem, of finding an appropriate G (or family of Gs) given a desired P, is the challenge of searching and designing useful species, which occurs in the Seed Factory. The growing organism interacts with co-growing subsystems and environmental influences, in the Computational Garden.

The generic component that grows, reacts, and interacts has two parts:

1. an internal state S that develops over time
2. a program (genome) G that specifies the internal dynamics and the results of the potential interactions with the other components and with the environment.

Program G is inherited from the (components in the) seed through its successive division steps or during self-assembly. State S includes the physical characteristics of the component (size, shape, position, etc.) as well as a set of internal variables corresponding to some gene-regulatory and metabolic activity.

The development of the whole organism is implemented by the Growth Engine. This engine:

– takes care of the interactions between components and with the environment

– updates the state of each component of an organism in accordance with their Genome
– manages the organism as a whole.

Several organisms co-develop in interaction in a garden, achieving an ecosystem of developmental artefacts. Each organism is under the control of its own Growth Engine instance, and these individual Growth Engines interact with each other and with the wider environment.

2.4 Computational Garden

The computational garden is where the various seeds are planted and grow together, responding to their environment, into the resultant artefact. The idea of several "plastic" organisms growing together and adapting to their environment is one path to our gardening metaphor. Another path that also leads us to the garden springs from a disappointment with progress in software development. Programming still essentially occurs at the individual statement level: there is the necessity of managing an overwhelming amount of low-level detail. Moreover, the resulting software is very *brittle*: the smallest change or error can have a devastating effect. The garden provides us with a higher-level metaphor: high-level guiding of a robust complex growing system, rather than low-level engineering of the precise placement of every cell or particle.

The majority of the computational components in a garden will be growing seeds; the remainder will be supplied by external software such as physics engines, and (interfaces to) growing embodied physical systems. The gardener needs to manipulate these components. Keeping with the gardening metaphor, we require operations such as:

– planting seeds (starting the growth of a seed at a particular location)
– moving growing organisms (transplanting)
– feeding: applying nutrients, energy, and their virtual analogues, to affect growth rates (enhance growth in a particular direction)
– pruning growing organisms (stop growth in a particular direction)
– training along a given direction (change the direction of growth)
– grafting physical and/or virtual organisms together
– manipulating environmental variables and other parameter values
– recording and rerunning gardening "scripts".

In order to raise the level of the programming task, these operations need to be controlled via some high-level, intuitive visual metaphor, including facilities for:

– various graphical views of the garden
  • 3D graphical visualisation (for physical systems)
  • other standard graphical visualisations (e.g. graphs), for virtual systems (and for physical systems if desired)
  • visual widgets indicating the state or desirability of the computational output of the garden (since computation underlies the Garden, the graphical elements need not be too sophisticated, as it is the computation they perform that is most important)

    - ability to create customised views, using "drag and drop" from the visual widget library
- user navigation through the garden, to explore, view and manipulate the various components growing there.

It should also be possible to embed user-defined entities into the garden, that is, to place non-"living" but functional objects into the garden. This allows a combination of traditionally engineered and growing parts within the final artefact. For example, in some circumstances one might want to provide a "pre-cast" shell for a building, and only grow the utilities and control systems within it.

3 Illustrative Application – a Skyscraper

There is a sub-discipline of "morphogenetic architecture" [9, 10, 25], where researchers attempt to "grow" the form of a building. However, these researchers tend to focus on the aesthetic architecture of the physical shell, and tend not to consider the simultaneous growth of the other physical subsystems such as plumbing, or the virtual components such as the control systems.

The generation of useful architectural and related sub-system design is often realized via parametrised geometries; these realisations are sometimes used in real-world construction [3, 28]. There have been several successful approaches to the generation of structural designs via developmental techniques, usually cellular growth procedures [15, 18, 32, 37]. [14] argues in favour of such life-like representations as a means of incorporating features such as scalability, robustness, evolvability, and self-adaptation. Once such example of robustness has been demonstrated in structural design: [17] shows the use of a developmental procedure in the creation of truss designs which led to self-adaptation to differing scales and geometric environments, implying a sort of artificial polymorphism.

A skyscraper provides a good exemplar application, being a non-trivial cyber-physical system with multiple physical (structural) and virtual (control) components.

There are several issues to address in a skyscraper application. These include:

1. how various subsystems can be grown in concert such that they integrate effectively, and influence each other's growth, for example: plumbing adapting to the shell; plumbing and wiring adapting so as not to interfere)
2. how various subsystems can be grown in the context of a physical environment, allowing some of a system to be built conventionally and the remainder to be grown through it, and an analysis of the tradeoffs
3. how much of the morphological requirements can be placed in the seed, and how much in the subsequent gardening environment and regime (genericity *v* specificity tradeoffs)
4. how to stop the systems growing where necessary [20], potentially based on the mechanisms plants use to stop growing, which process can be triggered either by environmental conditions, such as winter, or by internally regulated developmental programmes [19]:

- A plant (or parts of it) can go into *dormancy*, for example, over winter, or as a seed, or axillary buds.
- A plant (or parts of it) can *senesce* and die: leaves in winter, whole plant for annual plants that only last a year. This involves active suicide of tissues in a very regulated process, for example, remobilising all the nutrients to the roots before shedding leaves.
- Plants grow mostly from meristems (root and shoot tips). These are usually indeterminate: they can go on for ever maintaining themselves while producing new cells to build the plant, but they can also become *determinate* and use themselves up making things. For example, the shoot apical meristem becomes a terminal flower, so the shoot stops growing.

The next phase of work is to use the exemplar skyscraper application to drive the implementation of a prototype Gro-CyPhy system conforming to the architectural design described above.

4 Discussion and conclusions

Currently most of the research effort in growing software has been concentrated either on growing structures or shapes with no obvious computational purpose, or on growing programs from (grammar) rules. Gro-CyPhy is a process to grow both aspects together in an integrated fashion: the shapes will be needed to perform the functions, and programs will be embedded in the structure to make it adaptable and responsive.

Visual programming has been around for a long time. For example, most graphical user interfaces (GUIs) have some development toolkit that allows programmers to create GUI layouts in a visual and intuitive way. Furthermore, complex programs can be specified with the help of UML diagrams. However, such visual programming has mainly been used to build software in a static way, following a traditional software development lifecycle: software is rarely manipulated in this way while executing (model-driven software adaptation [2] is one exception); growing software in this way is not yet a reality. Gro-CyPhy is a radically new way of conceiving of software: as something that grows, and therefore modifies itself during execution. Such software will naturally be able to withstand external operations on itself during execution, such as pruning and grafting by a "gardener". The gardener will be a high-level programmer, and ultimately can even be the user directly.

Gro-CyPhy is a vision of "programmed organisms", software intensive embodied systems or cyber-physical systems that are grown in a garden, where they are autonomous and yet responsive and programmable at a high level. Such programmed organisms help blur the artificial distinction between the abstract software programs and the substrate where the software actuates (execution hardware plus supporting physical structures).

References

1. Beal, J.: Functional blueprints: an approach to modularity in grown systems. Swarm Intelligence 5, 257–281 (2011)
2. Bencomo, N., Blair, G., France, R.: Model-driven software adaptation. In: ECOOP 2007, LNCS, vol. 4906, pp. 132–141. Springer (2008)
3. von Buelow, P., Falk, A., Turrin, M.: Optimization of structural form using a genetic algorithm to search associative parametric geometry. In: Proceedings of the International Conference on Structures Architecture (ICSA 2010) (2010)
4. DeHon, A., Giavitto, J.L., Gruau, F.: 06361 executive report – computing media languages for space-oriented computation. In: DeHon, A., Giavitto, J.L., Gruau, F. (eds.) Computing Media and Languages for Space-Oriented Computation. No. 06361 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany (2007), http://drops.dagstuhl.de/opus/volltexte/2007/1025
5. Doursat, R.: Organically grown architectures: Creating decentralized, autonomous systems by embryomorphic engineering. In: Würtz, R.P. (ed.) Organic Computing. pp. 167–200. Springer (2008)
6. Doursat, R., Sayama, H., Michel, O. (eds.): Morphogenetic Engineering: Toward Programmable Complex Systems. Springer (2012), (to appear)
7. Drexler, K.E.: Engines of Creation. Doubleday (1986)
8. Harding, S., Miller, J.F., Banzhaf, W.: Developments in cartesian genetic programming: self-modifying CGP. Genetic Programming and Evolvable Machines 11(3-4), 397–439 (2010)
9. Hensel, M., Menges, A., Weinstock, M. (eds.): Emergence: Morphogenetic Design Strategies. Wiley (2004)
10. Hensel, M., Menges, A., Weinstock, M. (eds.): Techniques and Technologies in Morphogenetic Design. Wiley (2006)
11. Hornby, G.S., Pollack, J.B.: The advantages of generative grammatical encodings for physical design. In: CEC 2001, vol. 1. pp. 600–607. IEEE (2001)
12. Jacob, C.: Genetic L-system programming. In: PPSN III. LNCS, vol. 866, pp. 333–343. Springer (1994)
13. Jacob, C.: Evolving evolution programs: Genetic programming and L-systems. In: Proceedings of the First Annual Conference on Genetic Programming. pp. 107–115. MIT Press (1996)
14. Jin, Y., Sendhoff, B.: A systems approach to evolutionary multiobjective structural optimization and beyond. IEEE Computational Intelligence Magazine 4, 62–76 (August 2009)
15. Kicinger, R., Arciszewski, T., Jong, K.D.: Evolutionary design of steel structures in tall buildings. Journal of Computing in Civil Engineering 19(3), 223–238 (2005)
16. Kowaliw, T., Banzhaf, W.: Augmenting artificial development with local fitness. In: CEC 2009. pp. 316–323. IEEE (2009)
17. Kowaliw, T., Grogono, P., Kharma, N.: Environment as a spatial constraint on the growth of structural form. In: GECCO '07. pp. 1037–1044. ACM (2007)
18. Kowaliw, T., Grogono, P., Kharma, N.: The evolution of structural form through artificial embryogeny. In: IEEE Symposium Series on Computational Intelligence (IEEE-ALIFE '07). pp. 425–432 (2007)
19. Leyser, O., Day, S.: Mechanisms in Plant Development. Blackwell (2003)
20. Miller, J.F.: Evolving a self-repairing, self-regulating, French flag organism. In: Proc. Genetic and Evolutionary Computation Conference. LNCS, vol. 3102, pp. 129–139. Springer (2004)

21. Miller, J.F. (ed.): Cartesian Genetic Programming. Springer (2011)
22. Miller, J.F., Thomson, P.: Beyond the complexity ceiling: Evolution, emergence and regeneration. In: Proc. GECCO 2004 Workshop on Regeneration and Learning in Developmental Systems (2004)
23. Mock, K.J.: Wildwood: The evolution of L-system plants for virtual environments. In: Proceedings of the 1998 IEEE World Congress on Computational Intelligence. pp. 476–480. IEEE Press (1998)
24. Monks, M., Oh, B.M., Dorsey, J.: Audioptimization: goal-based acoustic design. IEEE Computer Graphics and Applications 20(3), 76–90 (2000)
25. Roudavski, S.: Towards morphogenesis in architecture. International Journal of Architectural Computing 7(3), 345–374 (2009)
26. Sayama, H.: Swarm chemistry. Artificial Life 15, 105–114 (2009)
27. Shea, K., Smith, I.F.C.: Improving full-scale transmission tower design through topology and shape optimization. Journal of Structural Engineering 132(5), 781–790 (2006)
28. Shepherd, P.: Aviva stadium – the use of parametric modelling in structural design. The Structural Engineer 89(23), 28–34 (2010)
29. Solé, R.V., Munteanu, A., Rodriguez-Caso, C., Macía, J.: Synthetic protocell biology: from reproduction to computation. Philos Trans R Soc Lond B Biol Sci 362(1486), 1727–1739 (2007)
30. Spicher, A., Michel, O.: Using rewriting techniques in the simulation of dynamical systems: Application to the modeling of sperm crawling. In: ICCS'05, part I. LNCS, vol. 3514, pp. 820–827. Springer (2005)
31. Spicher, A., Michel, O., Giavitto, J.L.: Declarative mesh subdivision using topological rewriting in MGS. In: ICGT 2010. LNCS, vol. 6372, pp. 298–313. Springer (2010)
32. Steiner, T., Jin, Y., Sendhoff, B.: A cellular model for the evolutionary development of lightweight material with an inner structure. In: GECCO '08. pp. 851–858. ACM (2008)
33. Stepney, S.: Embodiment. In: Flower, D., Timmis, J. (eds.) In Silico Immunology, chap. 12, pp. 265–288. Springer (2007)
34. Stepney, S.: The neglected pillar of material computation. Physica D: Nonlinear Phenomena 237(9), 1157–1164 (2008)
35. Tufte, G., Haddow, P.C.: Extending artificial development: Exploiting environmental information for the achievement of phenotypic plasticity. In: ICES 2007. LNCS, vol. 4684, pp. 297–308. Springer (2007)
36. Werfel, J.: Biologically realistic primitives for engineered morphogenesis. In: ANTS 2010. LNCS, vol. 6234, pp. 131–142. Springer (2010)
37. Yogev, O., Shapiro, A.A., Antonsson, E.K.: Computational evolutionary embryogeny. IEEE Transactions on Evolutionary Computation 14(2), 301–325 (2010)