Engineered Self-Organization Approaches to Adaptive Design

Jacob Beal*
Raytheon BBN Technologies, 10 Moulton Street, Cambridge, MA 02138, USA
jakebeal@bbn.com

Abstract
Assumptions are "compiled into" designs as a natural part of the engineering process, losing information about the reasons particular design decisions were made. This leads to systems that are fragile in the face of failures, changing requirements over time, and changing contexts of use. Engineered self-organization techniques can help mitigate this by making more of the system design implicit, thereby transferring aspects of system integration from human engineers to the system itself. This paper reviews three engineered self-organization techniques currently being investigated: manifold abstractions of networks, stochastic coordination, and functional blueprints, illustrating them with recent results from my work on networked sensing and control, grid-scale power management, and electromechanical design.

Keywords: Self-Adaptation; Engineered Self-Organization; Spatial Computing; Morphogenetic Engineering; Functional Blueprints; Stochastic Control

1 INTRODUCTION

As the complexity of engineered systems grows, and as more and more of those systems operate as part of an ecosystem of interacting systems, we are facing an increasing problem of design complexity and maintainability. Over the lifetime of a system, it is likely that our requirements of it will evolve, the environment in which it operates will change, and it will need to be upgraded, serviced, and customized. For infrastructural systems, these changes often must happen even while the system continues to operate: the trains must run, drinking water must flow, and Google can't go down.

Standard engineering processes, however, are not well suited to these challenges. As a design develops from concept to implementation, assumptions naturally become "compiled" into its structure, resulting in sets of system elements that depend closely on one another, but retain little information about how and why that dependency came about. Changing any element of such a set is likely to require correlated changes in the others, and onward to elements that those elements may be linked to, etc., yet the knowledge about such linkages is generally not readily available to the system itself. This leads to systems that are fragile in the face of failures, changing requirements over time, and changing contexts of use.

One promising approach to mitigating this problem is by using engineered self-organization. *Self-organization* is aggregate structure or behavior that arises from local interactions [1, 2, 3]. Any particular self-organization mechanism may be viewed as an instance where the infamous "local-to-global" problem has been solved. We may then define *engineered self-organization* as design that predictably leads to a specified self-organization of elements. This can be accomplished by identifying equivalent global and local rules for composing and modulating self-organization mechanisms, thereby allowing the designer to work only with aggregate models while the system is implemented by a "global to local to global" flow as illustrated in Figure 1.

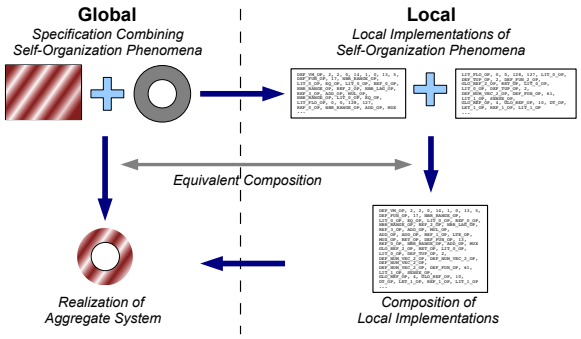


Figure 1: Self-organization mechanisms are specific instances where the "local-to-global" problem has been solved. Engineered self-organization builds on these mechanisms with composition rules that are valid in both the aggregate and local views, allowing "global-to-local-to-global" system design.

Engineered self-organization approaches can help with the problems of design by transferring aspects of sys-

*Partially sponsored by DARPA; views and conclusions contained in this document are those of the authors and not DARPA or the U.S. Gov't.

tem integration from human engineers at design time to the system itself at runtime. Many engineered self-organization processes can adapt to operate will across a broad range of environments. A human engineer building on top of such processes should then need to fix only the aggregate behavior at design time, leaving the details to be adaptively implemented by the self-organization methods.

In some ways, this is quite similar to other self-adaptation approaches, such as autonomic computing [4, 5], organic computing [6, 7, 8], or other similar forms of self-reconfiguration (e.g., [9, 10, 11]). These other approaches, however, typically require some degree of centralization or other restrictive pre-defined structure, which is often not the case for self-organization.

Applying an engineered self-organization approach does that the drawback that it often makes initial system design more difficult, as it frequently forces the designer to confront design adaptation problems that might otherwise be postponed or overlooked. The reward for this investment, however, is systems that are inherently adaptable across a range of circumstances and that often degrade gracefully when taken beyond their limits. Engineered self-organization is not a silver bullet, nor is there any one overarching technique for self-organization, any more than there is a single technique for object orientation in programming. Rather, we may expect to accumulate techniques over time, each of which applies over a certain family of design problems and broadens the scope of adaptivity that can be made routine.

As a starting point and illustrative example, the remainder of this paper reviews three engineered self-organization techniques that are subjects of current research, each of which addresses a different aspect of adaptability:

- Abstracting a network as a continuous space-time manifold forces algorithms to build in scalability, as well as providing implicit adaptability to local and global topology changes.
- Stochastic coordination methods force control systems to deal with uncertain progress and encourage the use of self-stabilization.
- Functional blueprints explicitly encode the goals of design decisions, along with instructions for incremental design change when those goals are not being satisfied, allowing integration to be maintained automatically in response to design changes.

These techniques will be illustrated with recent results from the application domains of networked sensing and control, grid-scale power demand management, and electromechanical design, adapted primarily from [12], [13], and [14] respectively, along with other recent related publications.

2 NETWORKS AS MANIFOLDS

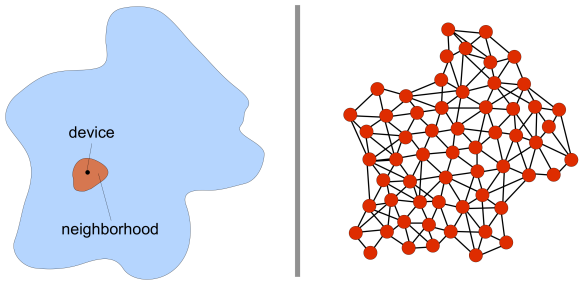


Figure 2: The amorphous medium abstraction allows a message-passing network (right) to be viewed as a discrete approximation of a manifold (left) where every point is a computing device that can access the recent pas state of its neighbors (Figure from [12]).

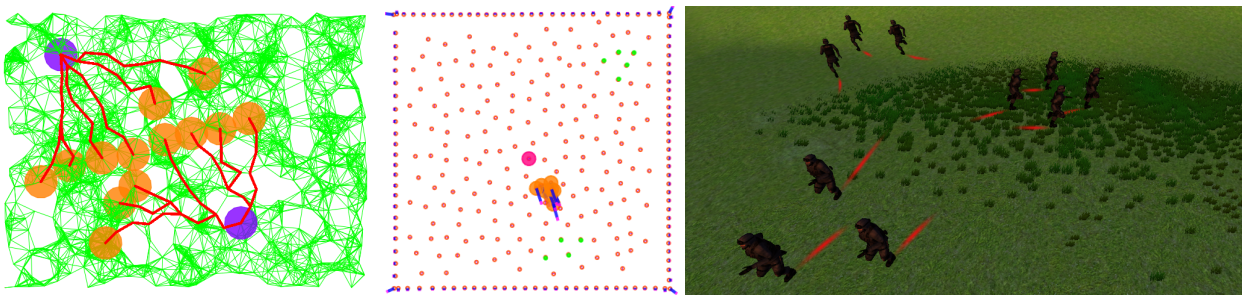
One way to bridge the gap between local and global is by means of geometric abstractions, as described in [15] and [12]. Many networked systems are tied closely to the space through which the networked devices are distributed. The tie may be physical, such as for devices that can only communicate with other devices nearby, or may arise from the purposes of the network, such as a sensor network intended to monitor a physical space. Such systems are known as *spatial computers*, and they can be programmed at the aggregate level by viewing the network as an approximation of a continuous space-time manifold.

The *amorphous medium* abstraction [16] formalizes this view as a Riemannian manifold[1] with a computational device at every point, where every device knows the recent past state of all other devices in its local neighborhood (Figure 2). We may then view any spatial computer as a discrete approximation of an amorphous medium, implementing it approximately with message passing and local measurements.

The amorphous medium abstraction may then serve as a rendezvous point between global and local models. In Proto [15, 17], a programmer specifies aggregate behavior as a purely functional composition of geometric and information-flow operations over a manifold. This is compiled from global specification to an amorphous medium program, which may then be approximated on real hardware, e.g., by means of scripts executed on a virtual machine. Purely functional composition and a careful choice of four families of primitives (pointwise, restriction, feedback, and neighborhood) ensures that global and local composition are equivalent.

At the same time, this formulation makes Proto programs implicitly adaptable at two levels. First, programs written in terms of physical units such as meters and seconds are resilient to changes in the discrete approximation. When programs are formulated respecting the amorphous medium abstraction—that

[1] A manifold is locally equivalent to Euclidean space, but may have non-Euclidean global structure, such as the surface of a sphere or a torus. Riemannian manifolds also support other useful geometric properties such as angle, distance, integrals, and derivatives.



(a) Distributed Publish/Subscribe (b) Search and Rescue (c) Simulated Maneuvers

Figure 3: Examples of complex self-organization produced by functional composition of self-organization mechanisms implemented using a continuous manifold abstraction: distributed publish-subscribe (a, from [18]), search and rescue by robot swarms (b, from [19]), and simulation of tactical maneuvers (c, from [20]).

is, in terms of a continuum of devices with a neighborhood of unknown diameter, most changes in the set of devices only affect the local quality of the continuous approximation. This allows programs to adapt implicitly to a large class of networks and network changes, so long as the approximation can be maintained and remains at a fine enough quality for the program's requirements. Second, the use of geometric and information-flow operations over a manifold means that program behavior is derived adaptively from the gross structure of the network. For example, the shortest path between two locations depends on the shape of the manifold between them. Thus, correct value for a geometric property such as shortest path changes along with the network, implicitly causing any self-repairing computation to adapt to the new configuration.

The continuous space abstraction is not itself self-organizing except in the very loose sense of local network approximations collectively representing a continuous manifold. A number of self-organization mechanisms have been implemented on it, however, such as fast self-stabilizing gradients [21, 22], gradient-based broadcast [19] and flocking [19]. Once such self-organizing mechanisms are implemented in Proto, its continuous manifold model of functional composition ensures that such mechanisms have a predictable behavior under any feed-forward composition: as each element converges, it provides a stable environment in which mechanisms downstream of it may also converge.

Thus, abstracting a network as a continuous manifold is one way to achieve the "global-to-local-to-global" flow of Figure 1. This has allowed the construction of many complex composite self-organizing behaviors from the functional composition of self-organization mechanisms. Figure 3 shows several examples from network and swarm control, each of which is a composition of some mixture of gradient, broadcast, and flocking self-organization mechanisms.

3 STOCHASTIC COORDINATION

The gap between global and local can also be bridged by a stochastic approach to coordination. Under this approach, the global model is of distributions, with a specification of the goal distribution to be achieved and one or more models of the current distribution of states over the aggregate of local devices. We can then view the global problem as a control problem to bring these two distributions into alignment, computing a distribution over control actions from the difference between goal and current distributions.

Correspondingly, in the local view, each device is a sample of these distributions. The current distribution is computed as a distributed aggregation over devices, thereby summarizing the state of the all devices in a much simpler global model. Each device can then compare its view of the current distribution with the target, and randomly choose an action from the distribution over control actions or from any other distribution with the same expected effect on the global model. The aggregate of local actions thus accumulate to produce a global action with expected behavior equal to the desired global distribution over control actions.

For example, Berman et al. [23] design controllers for a large swarm of aerial robots by abstracting the swarm as a chemical concentration function over space. The desired motion of the swarm is then specified as a partial differential equation, which can be solved using numerical approximation methods. This global continuous model can then mapped to its particle model equivalent, where the random movement of the individual robots is specified by the diffusion term of the particle equations. A similar partial differential equation model is set forth in [24].

Adopting a stochastic coordination approach has the consequence that the progress of a system towards any particular goal is never certain. This is particularly true when the distribution is sparse, such that the law of large numbers cannot apply, such as in the case of leader election. This approach does, however, have the advantage of encouraging a view of the system in terms of control, or more generally of self-stabilization, rather than distinct modes such as initial construction and repair of errors.

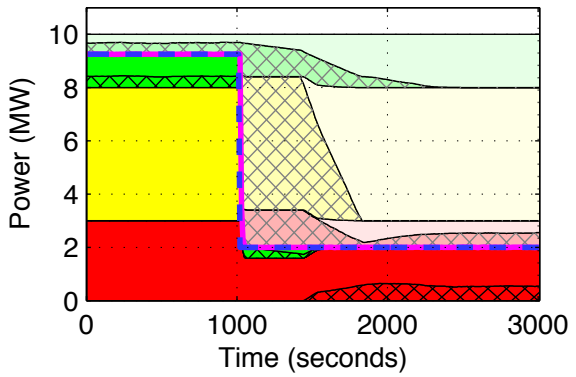


Figure 4: Example from [13] of priority-based composition of stochastic coordination, applied to control energy demand in a simulated network of 10,000 devices. When the target changes, applying the distribution to satisfy this highest priority constraint may cause violations in the others, which are later repaired in order of priority.

Complex self-organizing systems can then be constructed by composing distributions together. For example, the algorithm presented in [13] matches energy demand to a moving target in a highly constrained system of energy-consuming devices, each marked with a qualitative flexibility class. These devices become temporarily uncontrollable when they are switched on or off, and their control is subject to a set of constraints (target following, flexibility rank, and cycling) that often cannot be simultaneously satisfied. In this case, the controller is constructed by composing distributions for controlling each constraint in a strict priority order. This mean that first the target following distribution is applied. If any expected controllability remains in the system, the priority order distribution is applied to the remaining controllability, and any remainder from that to the cycling distribution.

Figure 4 shows an example of demand control in this system, where the stochastic approach initially satisfies the first constraint (target following) by violating the other two, then repairs each of the others in priority order. In Figure 4, demand is showed as a stacked graph of cumulative power demand in order of flexibility from least flexible (red) to most flexible (green), with saturated colors for devices that are currently on, faded colors for devices that are off, and cross-hatching for devices that cannot be controlled since they have recently switched states. When the target demand drops (blue dashes), first the target-following distribution adjusts the total power draw to follow (magenta line), but in this case that causes the system to violate both of the other constraints. As devices become controllable again, first the flexibility rank is restored (trading green devices for red) and then the cycling constraint (balancing controllable and uncontrollable red devices).

Thus stochastic control is another approach for designing self-organizing systems, where the local actions are samples of a global distribution, and coordination between devices is enabled by updating shared summary models of the global distribution. Complex /self-organizing systems can then be defined by composing distributions together.

4 FUNCTIONAL BLUEPRINTS

The previous two techniques that we have discussed each address the problems of design and maintainability indirectly, by using abstractions that encourage flexibility and adaptiveness in design. The final technique that we consider, functional blueprints [14], directly addresses the question of design adaptation using ideas derived from the study of natural morphogenesis.

Natural biological organisms are remarkable in their ability to adapt to changes in their structure or environment, both within the development of an individual and also on an evolutionary time scale. This self-adaptation to produce a viable organism, known as canalization [25], creates a close link between morphogenesis (the development of shape in an individual) and evolution, which has been much studied in recent years [26, 27].

Functional blueprints aim to replicate this flexibility with a representation based on results from the study of natural morphogenesis. The enabling insight for this approach is that in engineered systems, the majority of the parameters describing a design are implicit in the geometric and topological relations between system components. Typically, there are a relatively small number of key parameters that are linked tightly to the functional goals of the design, and the remainder may be derived from geometric, topological, or functional relationships between key parameters.

Functional blueprints exploit this insight regarding key parameters with two levels of specification. First, there are the functional blueprints themselves, which specify how key parameters should be adapted in order to preserve system functionality in the face of changes to the specification or environment. A functional blueprint, as defined in [14], consists of four elements:

- a *system behavior* that degrades gracefully across some range of viability,
- a *stress metric* quantifying the degree and direction of stress on the system when its behavior is degraded,
- an *incremental program* that relieves this stress by small changes to key parameters, and
- a program to construct an initial viable *minimal system*.

A collection of functional blueprints is then integrated by means of a developmental program that encodes geometric and topological relations between key parameters [28]. This effectively provides a reference

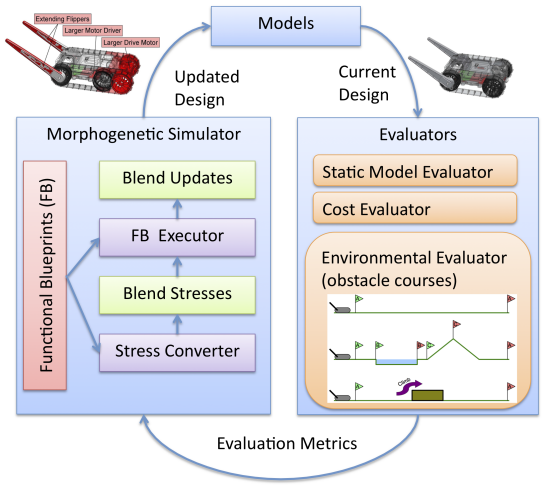


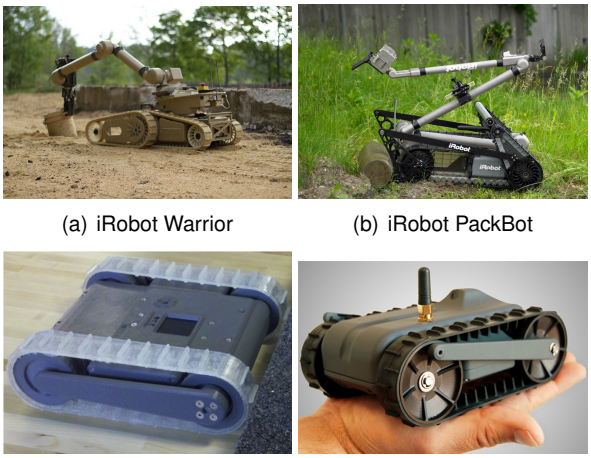
Figure 5: Design loop for iterative adaptation based on functional blueprints (from [29]): (1) evaluation of system behavior, (2) stress in evaluation leads to functional blueprint adjustment of key parameters, and (3) execution of developmental program to produce an incrementally modified model.

architecture for the system, with the ordering and linkages between elements capturing design decisions that would ordinarily be lost when the system blueprints are fully specified and all implicit parameters become explicit.

Functional blueprints can be applied to the problem of design adaptation using an incremental update loop such as that developed for electromechanical systems in [29] (Figure 5). First, the functionality of the current design is assessed by a collection of evaluators, whose outputs are combined to produce a stress on each of the system's functional blueprints. Stressed functional blueprints produce incremental updates for the key parameters they control, which are combined to produce the actual update for each key parameter and damped based on overall system stress. Finally, the design's developmental program is run with the new key parameter values, producing an incrementally adapted design ready for evaluation.

To see how this approach can work, let us consider the set of robots in the iRobot PackBot family, some of which are shown in Figure 6. This family of robots all share a common body plan despite vast differences in scale and capabilities, indicating that this family of designs is likely a good target for further adaptation via functional blueprints.

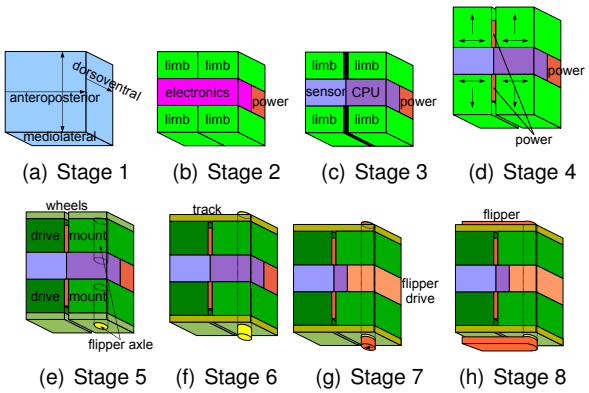
One of the salient elements of the body plan is a pair of flippers, which are attached coaxially with one of the pairs of wheels allow maneuvers such as climbing over obstacles. To climb over a bigger obstacle, the robot needs longer flippers. The length of a flipper is thus a key parameter, regulated by a functional blueprint for climbing obstacles. When climbing begins to be difficult, as measured by the angle of the robot during a critical phase of the climbing maneuver, the flipper increases in length.



(a) iRobot Warrior (b) iRobot PackBot

(c) iRobot miniDroid (d) iRobot LANdroid

Figure 6: iRobot PackBot family of robots, which all share a base body plan, including symmetric two-wheel treads, flippers coaxial with one wheel, and a top-mounted sensor/manipulator package.



(a) Stage 1 (b) Stage 2 (c) Stage 3 (d) Stage 4

(e) Stage 5 (f) Stage 6 (g) Stage 7 (h) Stage 8

Figure 7: Approximate developmental stages of a miniDroid (or other PackBot family member) body plan (from [28]).

When a flipper is lengthened, however, should it be lengthened from the center, from one of the two sides, or from some combination of that? This ambiguity is resolved by the developmental model. The developmental model begins with a homogeneous "egg" and executes a program of manifold operations to develop the structure of the design. For example, under the developmental model for the body plan of a PackBot-family robot shown in Figure 7 (from [28]), the differentiation and growth of the flipper in stages 5-8 constrains the flipper attachment point to be coaxial with the rear wheel and sets the key parameter of flipper length to be measured away from that coaxial point.

Together, the developmental program and the functional blueprints form a network of locally adapting elements that interact to produce the desired global functionalities under changed conditions. As in our prior discussion of stochastic coordination, there is a simple summary model that provides a coordination signal for all of these local elements—in this case the global stress level. As the global stress level increases, the size of adjustments made by the func-

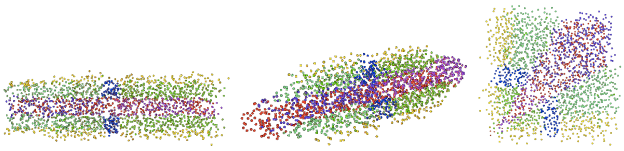


Figure 8: Specifying a developmental program using manifold operators allows implicit adaptation to changes in the space where portions of the program execute, as shown by these particle simulations of the developmental program in Figure 7 executed through Stage 5 on a flat egg, an ovoid egg, and twisted coordinates (from [28]).

tional blueprints is damped, in order to make it less likely that the system will produce a non-viable design (though determining good relative damping rates is still an open problem). When increments are sufficiently conservative, stress can diffuse rapidly through the network of functional blueprints and key parameters, allowing rapid adaptation, as shown in [29].

Likewise, as we have seen previously in the discussion of networks abstracted as manifolds, specifying the developmental program using manifold operators allows subprograms to adapt implicitly to changes in the space on which they are executed, as shown in [28]. For example, Figure 8 shows the developmental program from Figure 7 executed through Stage 5 with a particle simulation on variously distorted spaces: in all cases, the design produced is an equivalently distorted version of the original body plan.

Functional blueprints thus appear to offer a high potential for directly addressing the problem of design adaptation. This technique is less mature than the others discussed, however, with many open questions still to be resolved.

5 SUMMARY

Self-organization is a powerful tool for transforming explicit design elements that must be fixed at design time into implicit elements that can be determined dynamically as they are needed, thereby providing an engineered system with increased flexibility. This review has illustrated this point by discussing three self-organization techniques—manifold abstraction of networks, stochastic coordination, and functional blueprints—along with recent results from their use in a number of areas.

These three techniques are merely the tip of the iceberg for potentially useful self-organization techniques, with many others possible. For example, [30] identifies a taxonomy of other frequently used biologically-inspired self-organization primitives and begins systematizing them with the aim of expanding the toolkit of reliable engineering techniques. Looking to the future, there are three main strands in the continued development of this area of research:

- Discovery or adaptation of additional self-organization phenomena that provide additional useful global behaviors.
- Refinement of self-organization phenomena into engineering techniques suitable for routine use.
- Application of self-organization techniques to real-world problems of systems design and maintenance.

The challenges of this research area are great, but so is the potential reward: a major increase of reliability and efficacy in our ability to build, maintain, and manage complex engineered systems.

6 REFERENCES

[1] De Wolf, T. and Holvoet, T. (2005) Emergence versus self-organisation: Different concepts but promising when combined. Brueckner, S., Di Marzo Serugendo, G., Karageorgos, A., and Nagpal, R. (eds.), *Engineering Self-Organising Systems*, vol. 3464 of *Lecture Notes in Computer Science*, pp. 77–91, Springer Berlin / Heidelberg.

[2] Haken, H. (2008) Self-organization. *Scholarpedia*, **3**, 1401.

[3] Heylighen, F. (1999) The science of self-organization and adaptivity. *The Encyclopedia of Life Support Systems*, pp. 253–280, Eolss Publishers.

[4] Kephart, J. (2003) The vision of autonomic computing. *Computer*, **36**, 41–50.

[5] Parashar, M. and Hariri, S. (2005) Autonomic computing: An overview. Banatre, J.-P., Fradet, P., Giavitto, J.-L., and Michel, O. (eds.), *Unconventional Programming Paradigms*, vol. 3566 of *Lecture Notes in Computer Science*, pp. 97–97, Springer Berlin / Heidelberg.

[6] Schmeck, H. (2005) Organic computing - a new vision for distributed embedded systems. *8th Int'l Symposiumn on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, May, pp. 201–203.

[7] Bellman, K., Herkersdorf, A., and Hinchey, M. G. (2011) Organic Computing - Design of Self-Organizing Systems (Dagstuhl Seminar 11181). *Dagstuhl Reports*, **1**, 1–28.

[8] Wurtz, R. (2008) *Organic computing*. Springer-Verlag.

[9] Berns, A. and Ghosh, S. (2009) Dissecting self-* properties. *Self-Adaptive and Self-Organizing Systems, 2009. SASO '09. Third IEEE International Conference on*, September, pp. 10–19.

[10] Salazar, N., Rodriguez-Aguilar, J., and Arcos, J. (2008) An infection-based mechanism for self-adaptation in multi-agent complex networks. *Self-Adaptive and Self-Organizing Systems, 2008. SASO '08. Second IEEE International Conference on*, October, pp. 161–170.

[11] Bernauer, A., Bringmann, O., and Rosenstiel, W. (2009) Generic self-adaptation to reduce design effort for system-on-chip. *Self-Adaptive and Self-Organizing Systems, 2009. SASO '09. Third IEEE International Conference on*, sept., pp. 126 –135.

[12] Beal, J. and Schantz, R. (2010) A spatial computing approach to distributed algorithms. *45th Asilomar Conference on Signals, Systems, and Computers*, November.

[13] Beal, J., Berliner, J., and Hunter, K. (2012) Fast precise distributed control for energy demand management. *6th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2012)*, to appear September.

[14] Beal, J. (2011) Functional blueprints: An approach to modularity in grown systems. *Swarm Intelligence*, **5**.

[15] Beal, J. and Bachrach, J. (2006) Infrastructure for engineered emergence in sensor/actuator networks. *IEEE Intelligent Systems*, pp. 10–19.

[16] Beal, J. (2004) Programming an amorphous computational medium. *Unconventional Programming Paradigms International Workshop*, September.

[17] (Retrieved September 16, 2012), MIT Proto. software available at `http://mitproto.net`.

[18] Usbeck, K. and Beal, J. (2011) An agent framework for agent societies. *Proceedings of Systems, Programming, Languages and Applications: Software for Humanity*, Portland, Oregon, USA, Oct, pp. 201–212, SPLASH.

[19] Bachrach, J., Beal, J., and McLurkin, J. (2010) Composable continuous space programs for robotic swarms. *Neural Computing and Applications*, **19**, 825–847.

[20] Beal, J., Usbeck, K., and Krisler, B. (2012) Lightweight simulation scripting with proto. *Spatial Computing Workshop*, June.

[21] Beal, J., Bachrach, J., Vickery, D., and Tobenkin, M. (2008) Fast self-healing gradients. *ACM Symposium on Applied Computing*, New York, NY, USA, March, ACM.

[22] Beal, J. (2009) Flexible self-healing gradients. *ACM Symposium on Applied Computing*, New York, NY, USA, March, pp. 1197–1201, ACM.

[23] Berman, S., Kumar, V., and Nagpal, R. (2011) Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination. *2011 IEEE International Conference on Robotics and Automation (ICRA'11)*.

[24] Hamann, H. and Worn, H. (2008) A framework of space-time continuous models for algorithm design in swarm robotics. *Swarm Intelligence*, **2**, 209–239.

[25] Waddington, C. H. (1942) Canalization of Development and the Inheritance of Acquired Characters. *Nature*, **150**, 563–565.

[26] Carroll, S. B. (2005) *Endless Forms Most Beautiful*. W. W. Norton & Company.

[27] Kirschner, M. W. and Norton, J. C. (2005) *The Plausibility of Life: Resolving Darwin's Dilemma*. Yale University Press.

[28] Beal, J., Mostafa, H., Axelrod, B., Mozeika, A., Adler, A., Markiewicz, G., and Usbeck, K. (2012) A manifold operator representation for adaptive design. *GECCO 2012*, July.

[29] Adler, A., Yaman, F., Cleveland, J., and Beal, J. (2011) Morphogenetically assisted design variation. *Int'l Conference on Morphological Computation*.

[30] Fernandez-Marquez, J., Di Marzo Serugendo, G., Montagna, S., Viroli, M., and Arcos, J. (2012) Description and composition of bio-inspired design patterns: a complete overview. *Natural Computing*, pp. 1–25.