Interdisciplinary Research: Roles for Self-Organization

No generally accepted principles and guidelines currently exist to help engineers design local interaction mechanisms that result in a desired global behavior. However, several communities have developed ways of approaching this problem in the context of niched application areas. Because the ideas underlying these approaches are often obscured or underemphasized in technical papers, we invited representatives of several communities to review the role of self-organization in their work. These short articles round out this special issue by drawing a better picture of the status of the emerging field of self-organizing systems.

Radhika Nagpal reviews the motivations, results, and possible future of *amorphous computing*. This research intends to lay the foundations of self-organization by developing principles and programming languages. It has a wide range of potential applications including microfabrication and cellular engineering. Nagpal uses the adaptive, flexible coordination of mobile robots to illustrate this approach. The feature article by Jacob Beal and Jonathan Bachrach (pp. 10–19) describes a practical way to implement the abstract ideas she presents in real systems.

Franco Zambonelli offers thought-provoking ideas on what he calls the *ecological approach* to self-management. In this approach, a system is controlled by a small proportion of integral components rather than logically separate managers. Zambonelli also proposes considering humans as information system components—a view that the feature article by Sergi Valverde and his colleagues (pp. 36–40) supports as well.

Emin Gün Sirer describes important insights into a key problem: how to design local interactions to achieve good or optimal global performance in a predictable, informed way under different constraints. He argues that designers can and should apply mathematical optimization in distributed systems as opposed to "rule of thumb" heuristics. The feature article by David Hales and Stefano Arteconi (pp. 29–35) also targets optimality as a central issue with the system adapting its performance on the fly, and Tracy Mullen, Viswanath Avasarala, and David L. Hall (pp. 41–49) explicitly perform optimization in real time.

Finally, Hakima Chaouchi and Mikhail Smirnov walk us through the bold vision of the *autonomic communication* community. They project a future networking infrastructure that is fully adaptive and capable of learning and evolving, as opposed to the currently deployed design based on strictly isolated protocol layers. This vision isn't about methods so much as a specific application area—networking. All approaches and ideas in this issue are relevant to fulfilling it.

—Márk Jelasity, Ozalp Babaoglu, and Robert Laddaga

Self-Organizing Shape and Pattern: From Cells to Robots

Radhika Nagpal, *Harvard University*

A starfish is an amazing creature. Like many multicellular organisms, it begins life as a single-cell egg that divides and develops through a complex program executed by identically programmed cells. Throughout its life, the starfish functions as a whole, even though it's essentially a colony of cells that are constantly dying and being replaced. But even more remarkable is its ability to self-repair. Most starfish can grow back a severed limb, and some species can even grow back a body from a limb.

If we wanted to create such a system, what would we tell the cells to do? Many people are interested in this question in different forms for different reasons. For example, it's important to embedded systems composed of many identical parts, such as reconfigurable robots built from identical modules and network systems built from smart sensors scattered in the environment.

Here I would like to address this question in the context of self-organizing shapes and patterns. This work began as part of the Massachusetts Institute of Technology's Amorphous Computing project.[1] This project has investigated many systems—from cells to robots. We've come to understand a great deal about how to engineer shape and even self-repair, but many things still remain unknown.

Shape and pattern on an amorphous computer

We can think of an amorphous computer as a cellular automata with two main differences:

- The cells or "agents" are not perfectly placed on a regular lattice; instead they're randomly distributed and communicate with other agents within a small local radius.
- Although the agents have similar clock speeds, they don't operate synchronously.

The agents are identically programmed, can store state, have no preexisting notion of position or orientation, and

1541-1672/06/$20.00 © 2006 IEEE
Published by the IEEE Computer Society

have random-number generators. They can also receive some simple initial state.

The Amorphous Computing project posed a simple question: given a field of agents and a global goal—say, a particular pattern of lines—how do we derive local agent rules that will produce it? A complementary metal-oxide semiconductor (CMOS) is the canonical case for testing answers to this question because it has an asymmetrical pattern and clear constraints, so it's hard to produce and failure is obvious.

The first answer came from work by Daniel Coore on the Growing Point Language (GPL).[1] He showed that you could describe inverter-like patterns as a construction of lines that grew toward (or away from) points and created new points. The new points, in turn, then grew new lines, and so on. Later, I showed how you could use similar ideas to program a simulated foldable sheet and to fold it into different forms.[2] The simulated sheet is composed of embedded actuator agents, and the desired shape is specified as a global program using a program language based on origami—specifically, the Origami Shape Language. OSL is then compiled to generate the agent-level program. This work showed that we could achieve a kind of global-to-local compilation.

Attila Kondacs further showed how you could start from a single agent and grow an arbitrary shape—such as a caricature of the starfish—by generating the agent program directly from a picture of the shape.[2] All these systems achieved the basic tenets of amorphous computing, such as robustness to varying agent numbers and random placement, agent death and addition, asynchronous updates, and so on.

The systems yielded three major lessons. First, it's possible to create global-to-local compilers for shape and pattern. The trick is to find generative grammars that can describe large classes of shapes using a small rule set and then to develop a compiler that translates each grammar rule to local agent rules.

Second, the type of global representation can dramatically affect the way the pattern adapts to different conditions. For example, consider the CMOS inverter pattern generated by GPL and OSL. Given a larger agent field, the GPL program fills the space with repeating patterns, while the OSL program "stretches" the pattern (see figure 1). This surprisingly different adaptation occurs because GPL generates a pattern by sequentially laying down the pieces, while OSL operates by segmenting space. Other shape description languages could encode other global adaptation concepts, including self-repair.

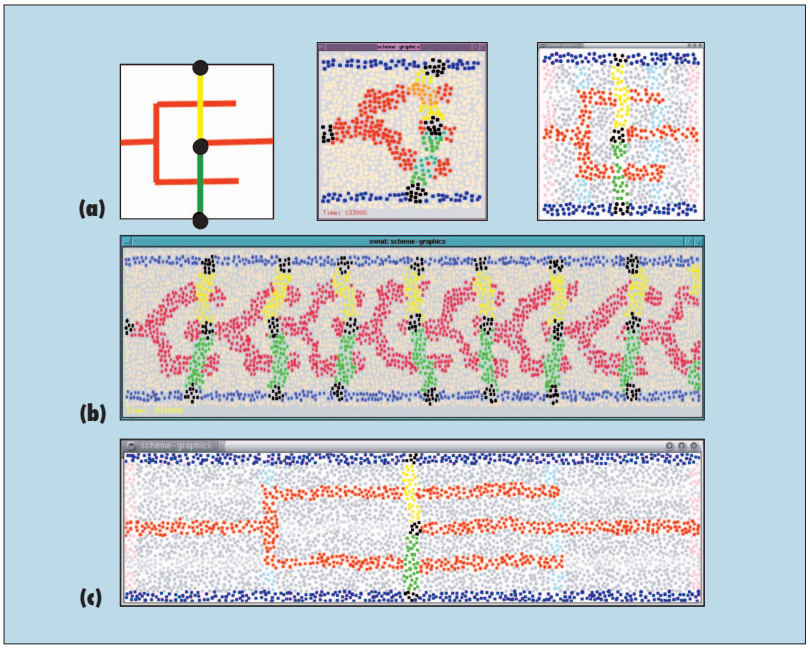

Figure 1. (a) A desired abstract pattern and its realizations on an amorphous computer using the Growing Point Language and the Origami Shape Language. The two languages react differently to changes in the initial conditions: (b) GPL creates space-filling structure, while (c) OSL scales proportionally.

Third, while all these systems were developed with different classes of shapes in mind and different agent assumptions, they all rely on a surprisingly small set of agent-level primitives—most prominently, coin-flipping to break local symmetry, small amounts of agent state, and morphogen gradients that propagate information across an agent field.

Shape and pattern in robotics

Three examples from robotics can illustrate practical applications that concern shape and pattern: mobile-robot formations, modular robot self-reconfiguration, and collective construction by robots. As in the amorphous computing scenario, these applications prespecify an arbitrary shape to be formed and require many of the same robustness criteria. The lessons from amorphous computing should apply. However, researchers solved most of these systems using a very different principle: they programmed the agents to robustly self-organize a coordinate system.

Consider the problem of programming a mobile-agent swarm to aggregate into a particular formation (see figure 2). My colleagues and I recently showed a simple decentralized mechanism for this task.[3] The mechanism uses three group behaviors: gas expansion (that is, local repulsion), local trilateration, and filling a shape container. Together, these behaviors cause agents to develop a consensus global-coordinate system, while spreading out evenly within the desired formation. The system has some interesting global properties. It automatically adjusts to the number of agents by adjusting density, which also results in automatic self-repair and error-correction if a region of agents is removed or displaced. While these formations are static, simple flocking-like rules could allow the swarm to move in formation and "flow" through obstacles in its way.

Now consider a similar problem in reconfiguring a modular robot. We can model a modular robot as a set of connected cubic agents that move by sliding around one another—much like tiles in the 15 Puzzle but in three dimensions. Each agent can

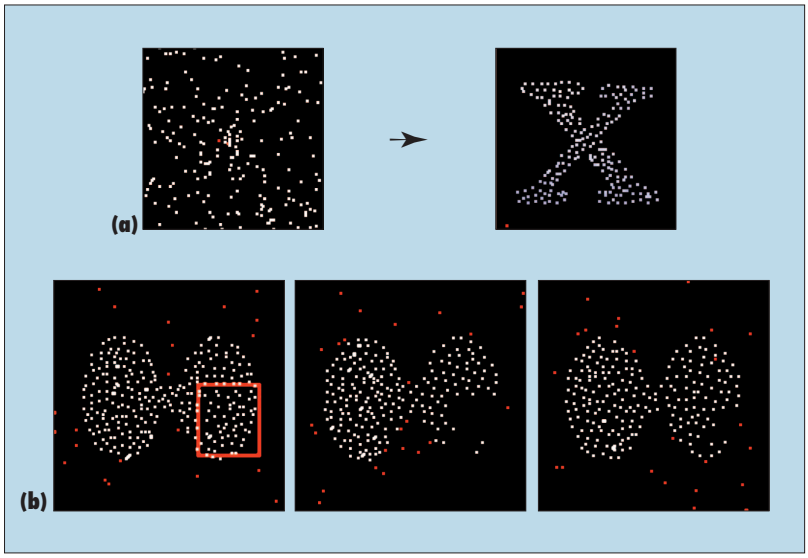

Figure 2. (a) A mobile-agent swarm aggregates into a prespecified shape. (b) A shape can recover from death and displacement.

communicate with its physical neighbors through their shared face. Given a desired shape, the goal is to generate the agent program such that the agents would move to end up approximating the shape, starting from any initial configuration and without becoming disconnected.

One solution relies on a strategy similar to self-organizing a coordinate system through local interactions.[4] Given an arbitrary agent as a seed that knows its location, this agent can "grow" locally to include neighboring agents and give each neighbor a coordinate. Neighboring agents then behave the same way. If an agent can't find a neighbor where it needs to grow, it generates a recruiting morphogen gradient. Wandering agents move so as to climb this gradient until they become part of the structure. Thus the shape forms through a directed growth process, where the local coordinate and the shape map determine the growth direction. The local rules are robust to asynchronous agent timing and remain fundamentally the same irrespective of starting and ending shape.

Collective construction is an analogous problem. This case involves two agent types —mobile robots and immobile blocks—and two different constraint sets, but the goal of creating a predetermined shape remains the same. Again, a strategy of self-organizing a coordinate system works well.[5]

These examples show that we can implement user-specified global goals through simple local control, and the resulting systems achieve a significant level of robustness to features such as variation in number of agents, lack of control over agent timing, agent loss and addition, or message loss and movement error. In each case, the constraints and agent capabilities differ, but the overall strategy is the same.

Yet the strategies are strikingly different from the amorphous computing examples. Why is this so? One probable answer is that it's much easier in these engineered systems to create and store arbitrary information, making a coordinate system a feasible generic solution. The downside is the potential loss of the shape's natural ability to adapt, which is a compelling property in the amorphously created patterns.

Shape and pattern in cells

While cells formed the inspiration for the amorphous computing concepts, they will also eventually constitute the substrate on which the concepts are applied. Synthetic biology is an effort to develop methodologies for engineering cells by creating genetic programs.[6] Researchers have already demonstrated several simple circuits, from toggle switches to a ring oscillator. Still, individual cells are only so robust or precise, and the excitement will finally come from being able to program populations of cells.

Can we program a field of bacteria to take on different patterns? The answer is yes. Most recently, Ron Weiss and his lab have demonstrated how to put some simple and powerful primitives, such as gradients, under engineered control.[7] Amorphous computing tells us that the distance from simple bull's eye patterns and polka dots to the caricature CMOS is not that large. In the future, we can imagine programming living cells to create a complex tissue or human-specified material.

Toward adaptive structures

All termite mounds looks similar, but they aren't the same. Branching structures, such as capillaries in our body, are made of similar multicellular tubes, but they form different networks depending on environmental cues. Amorphous computing has shown how to control some cues, such as scale, through the environment. In general, however, how do we create—or even describe—shapes with some parts that are predetermined while other parts adapt and optimize for the environment?

For example, what if we want a modular robot to form a stair whose height is unknown or a table that is level with respect to the ground or a snake that fits through a hole. Can we program cells to form a sieve (mesh) that fits inside a damaged artery? We still haven't tackled these kinds of shapes in any systematic way. There's still a lot to learn from the starfish.

References

1. H. Abelson et al., "Amorphous Computing," *Comm. ACM*, vol. 43, no. 5, 2000, pp. 74–82.
2. R. Nagpal, A. Kondacs, and C. Chang, "Programming Methodology for Biologically Inspired Self-Assembling Systems," *Computational Synthesis: From Basic Building Blocks to High-Level Functionality*," tech. report SS-03-02, AAAI, Mar. 2003.
3. J. Cheng, W. Cheng, and R. Nagpal, "Robust and Self-Repairing Formation Control for Swarms of Mobile Agents," *Proc. 20th Nat'l Conf. Artificial Intelligence* (AAAI 05), AAAI Press, 2005, pp. 59–64.
4. K. Støy and R. Nagpal, "Self-Reconfiguration Using Directed Growth," *Proc. 7th Int'l Symp. Distributed Autonomous Robotic Systems* (DARS 05), Springer, 2005.
5. J. Werfel, Y. Bar-Yam, and R. Nagpal, "Building Patterned Structures with Robot Swarms," *Proc. 19th Int'l Joint Conf. Artificial Intelligence* (IJCAI 05), 2005, pp. 1495–1502.

6. P. Silver and J. Way, "Cells by Design," *The Scientist*, vol. 18, no. 18, 2004, p. 30.

7. S. Basu et al, "A Synthetic Multicellular System for Programmed Pattern Formation," *Nature*, vol. 434, 28 Apr. 2005, pp. 1130–1134.

Radhika Nagpal is an assistant professor of computer science at Harvard University. Her research interest is in biologically inspired approaches to multiagent and distributed systems. She received her PhD in computer science from MIT. Contact her at Harvard Univ., 33 Oxford St., Cambridge, MA 02138; rad@eecs.harvard.edu.

Self-Management and the Many Facets of "Nonself"

Franco Zambonelli,
Università di Modena e Reggio Emilia

The difficulties in dealing with increasingly complex information systems that operate in dynamic operational environments calls for self-management properties. More generally, we could say that they call for the integration of "self-*" features—self-configuration, self-adaptation, self-healing, and so on—in software and information systems.

Nearly all self-* approaches consider human beings as "nonself" in relation to the system (see figure 1a). Indeed, all approaches share the key goal of moving humans out of the loop and having information systems autonomously perform all the costly and often very complex configuration and maintenance activities that will keep them working properly under all conditions. Still, we can identify several perspectives on what to consider "self" and what, besides humans, to consider "nonself" in these systems.

Autonomic computing perspective

IBM's autonomic computing initiative exemplifies the applied industrial perspective on self-management, which tends to view humans as the only nonself system components.[1] The basic idea is to replace humans with digital surrogates that will perform those monitoring, configuration, and maintenance activities formerly performed by humans.

At the level of either whole information systems or individual components, the autonomic computing perspective couples software managers with the system. These managers are in charge of monitoring what's happening and autonomously planning actions to reconfigure the system as needed, in a continuous control loop (figure 1b). At the distributed systems level, the perspective can also consider a set of distributed managers, each associated to different distributed elements. The managers exchange information with each other and orchestrate their actions to ensure specific functional and nonfunctional properties in the overall distributed system behavior (figure 1c).

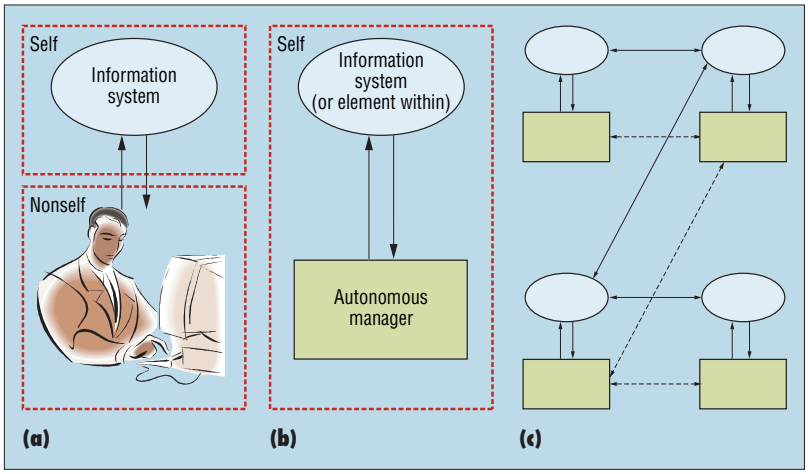

Figure 1. Moving humans out of the loop. (a) Humans as information system managers are considered "nonself" from the self-management perspective. (b) The autonomic computing perspective considers substituting those "nonself" human managers with "self" surrogates in the form of digital autonomous managers. (c) Multiple distributed autonomous managers can interact with each other to enforce distributed self-management activities.

Such an approach leads to a conceptually clean architecture for self-managing systems, well grounded in lessons learned from research in operating and distributed systems. Coupling traditional monitoring and resource management approaches with AI planning and knowledge management techniques as well as multiagent automated negotiation techniques might lead to the near-term release of seemingly self-managing systems. However, an architecture based on autonomous managers that are logically separated from the components they control introduces several potential drawbacks. In fact, accounting for all possible contingencies and appropriate reactions to ensure continuous functioning could result in a heavyweight architecture or in slow, inappropriate reactions that undermine self-management efficiency.

Self-organization perspective

To some extent, the limitations of the autonomic computing perspective derive mainly from inheriting the architecture of traditional human-based management approaches. Even if humans are no longer in the loop, the autonomous managers of figures 1b and 1c are essentially digital nonselfs, alien to the information system.

In self-organization approaches to self-management—exemplified by the research articles in this issue of *IEEE Intelligent Systems*—a self-managing system should be intrinsically self-managing, not externally managed by "nonself" entities—digital or otherwise. To this purpose, self-organization approaches take inspiration from natural adaptive systems and their intrinsic capabilities to organize global activities into highly adaptive functional patterns. Systems such as bacterial colonies, insect colonies, embryos, and organs exhibit globally functional activity patterns. These patterns emerge autonomously from simple local activity rules and local intercomponent interactions. The systems are robust with regard to both internal contingencies, such as components' deaths, and external contingencies, such as environmental perturbations. This flexibly ensures the preservation of the global functional pattern.

Because several of these natural phenomena find a natural mapping to functional problems in modern and distributed information systems,[2] they can enable the engineering of robust self-managing information