# 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 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

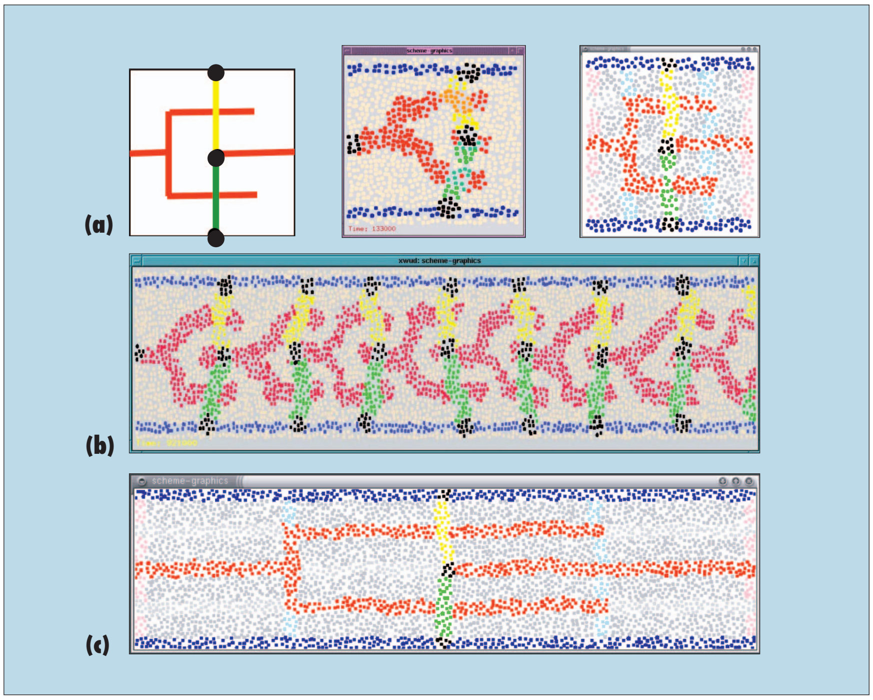

**Figure 1. (a) A desired abstract pattern and its realizations on an amorphous computer using the Growing Point Language and the Origami Shape Language. The two languages react differently to changes in the initial conditions: (b) GPL creates space-filling structure, while (c) OSL scales proportionally.**

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.

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

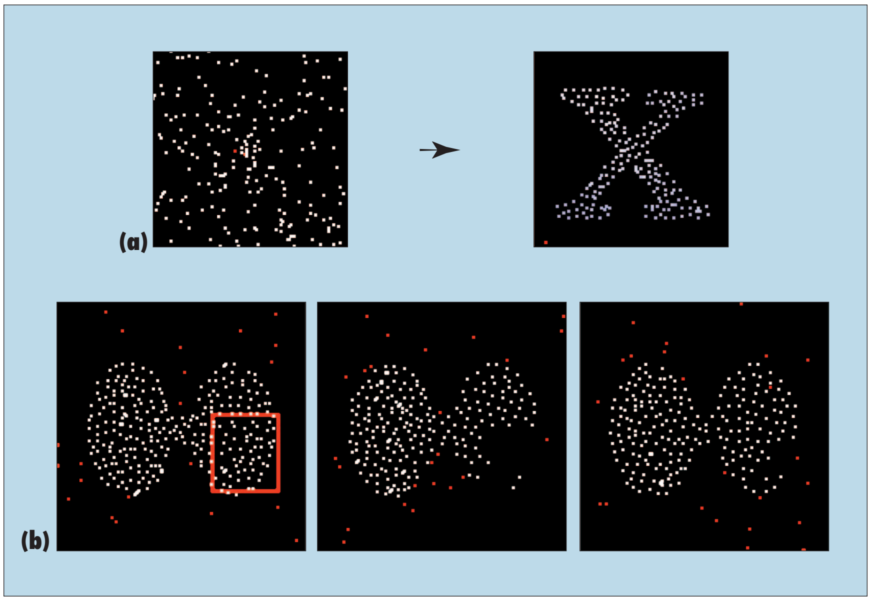

**Figure 2. (a) A mobile-agent swarm aggregates into a prespecified shape. (b) A shape can recover from death and displacement.**

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).

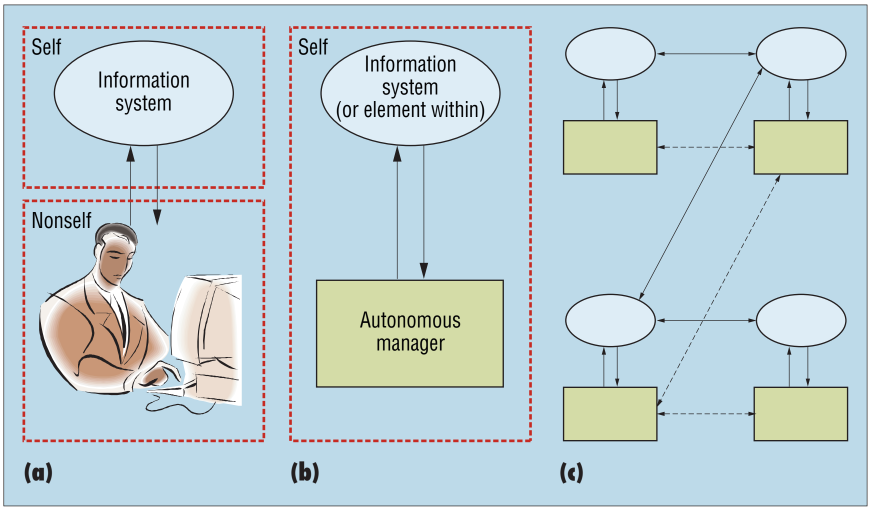

**Figure 1. Moving humans out of the loop. (a) Humans as information system managers are considered "nonself" from the self-management perspective. (b) The autonomic computing perspective considers substituting those "nonself" human managers with "self" surrogates in the form of digital autonomous managers. (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

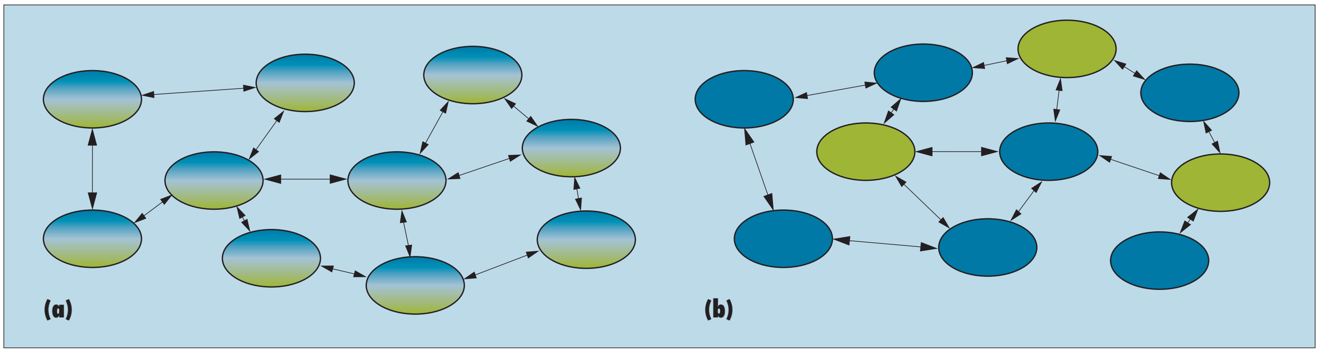

**Figure 2. Self-organization versus ecological self-management approaches. (a) In self-organization, a uniform set of self-organizing components interact locally with each other and act as both functional and management components. (b) In an ecological approach, self-organizing components live and interact with each other and with additional "manager" components that might somehow control and direct the overall system behavior.**

systems that integrate the functional and management parts seamlessly into the same components (see figure 2a).

Self-organization's key advantage for building self-managing systems is an overall simpler and lighter-weight system architecture than autonomic computing offers. Also, because self-management properties are intrinsic to the system, they don't require complex planning or knowledge management activities to react properly to specific or unforeseen contingencies.

However, despite the increasing number of success stories, self-organization isn't a self-management panacea, and it has several limitations. First, the rich catalog of natural phenomena that could apply to modern information systems doesn't eradicate the "solutions in search of a problem" approach to engineering. This approach focuses on reverse-engineering natural phenomena for mapping into useful distributed applications, but it's still missing the general methodologies for direct-engineering a self-organizing system that solves specific problems.

Second, most current self-organization approaches can enforce a single or limited set of self-managing functionalities—specifically, those directly related to the self-organizing functional pattern. However, these approaches fail in properly accounting for the diverse functionalities and possibly competing needs that occur in complex real-world information systems.

## Ecological perspective

Limitations of current self-organization approaches suggest that most real-world self-managing information systems would still require external managers to achieve a global view of the system and all its needs. However, an alternative approach might combine the autonomic and self-organization perspectives to enforce effective management and, at the same time, avoid introducing complex nonself external managers. We could mitigate the "nonselfness" of autonomous managers by making them first-class citizens in self-organizing systems.

The basic idea is to inject additional "manager" components in a self-organizing system. These components would live inside the system and interact with other components as if they were native to the system (figure 2b). They wouldn't undermine the basic self-organizing (and so self-managing) nature of the system, but they would have knowledge and abilities beyond those available to normal components. In this way, these components could direct the system's evolution toward a specific configuration out of the many toward which the self-organizing system alone might have evolved. Such control capabilities can be useful in accommodating the system's diverse competing needs as a whole and in improving its effectiveness in reacting to contingencies.

Our research group at the Universitá di Modena e Reggio Emilia has conducted preliminary experiments in this direction. On the one hand, we've shown how to globally direct the dynamical evolution of cellular automata by simply modifying a very small percentage of the cells.[3] On the other hand, in field-based coordinated systems, we've shown how to escape from suboptimal configurations by having some components that can reason about local field shapes, rather than simply react to them.[4] The Service D'Ecologie Sociale of the Université Libre de Brussels is also performing interesting experiments that show how a few robotic ants in a real ant colony can affect the whole colony's behavior.[5]

Beyond the horizon, we imagine a scenario in which our networks will be like large ecosystems, hosting multiple "specimens" of complex self-organizing systems, coexisting over the same resources, and interacting with each other in entirely new ways. These systems will be decentralized, without clearly identifiable stakeholders, and will run continuously. The only way to enforce control over them so that each individual system's self-management features coexist properly with more global forms of self-management will be to populate the ecosystem with additional manager components. This approach is already at work in agriculture, where parasites are contained by introducing natural predators. It also occurs in marketing, where "opinion leaders" are commonly recruited to promote specific products from within communities. In any case, an ecological approach to managing complex information systems still requires a lot of research and suitable engineering tools.

An interesting consequence of the ecological perspective is its potential to undermine the basic initial assumption of human "nonselfness." As information systems become more pervasive and integrated with both the physical and social worlds, the information system ecology will necessarily include humans as an integral component. Human activities and behaviors will directly affect the overall system behavior and self-management properties. In other words, humans will get back into the management loop as first-class "self" entities, even though they might lack explicit management responsibilities.

As a simple example, consider students accessing a campus Wi-Fi mesh with their laptops. While the mesh self-organizes its activities and redistributes connections to provide everybody a suitable quality of service, some students will likely try to optimize their own positions on campus to get better connectivity. To some extent, we could say that these students and the system implicitly cooperate for the network's optimal self-management. An analysis of the implications of these aspects, though, would require much more room than the few pages of this article and much more interdisciplinary competencies than I actually have.

## Unconcluding remarks

The spectrum of possible self-management perspectives makes it very hard to predict what the future will be. If I had to, I would bet on autonomic computing approaches to prevail in the short term, gradual integration with self-organizing approaches in the medium term, and subsumption by ecological approaches in the long term. Whatever the case, there's plenty of room for exciting research along the way.

## References

1. J. Kephart and D.M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, 2003, pp. 41–50.
2. O. Babaoglu et al., "Design Patterns from Biology for Distributed Computing," *Proc. European Conf. Complex Systems*, 2005; www.cs.unibo.it/babaoglu/papers/eccs05.pdf.
3. M. Mamei, A. Roli, and F. Zambonelli, "Emergence and Control of Macro Spatial Structures in Perturbed Cellular Automata, and Implications for Pervasive Computing Systems," *IEEE Trans. Systems, Man, and Cybernetics—Part A*, vol. 36, no. 5, 2005, pp. 337–348.
4. M. Mamei and F. Zambonelli, *Field-Based Coordination for Pervasive Multiagent Systems*, Springer, 2006.
5. G. Caprari et al., "Animal and Robot Mixed Societies—Building Cooperation between Microrobots and Cockroaches," *IEEE Robotics & Automation*, vol.12, no. 2, 2005, pp. 58–65.

**Franco Zambonelli** is a professor of computer science at the Università di Modena e Reggio Emilia. His research interests include pervasive computing, self-organization, and multiagent systems. He received his PhD in computer science from the University of Bologna. He is a member of the IEEE and the ACM. Contact him at Via Allegri 13, 42100 Reggio Emilia, Italy; franco.zambonelli@unimore.it.

## Heuristics Considered Harmful: Mathematical Optimization for Resource Management in Distributed Systems

Emin Gün Sirer, *Cornell University*

At the core of many distributed systems lies a difficult resource management problem: partitioning a critical resource—such as bandwidth, storage, or computational elements—among competing tasks. Such trade-offs are encountered in content distribution networks (CDNs), Grid resource management,[1] distributed cache management, data distributions in large-scale storage systems, high-performance publish-subscribe systems, as well as many other infrastructure services where performance is a function of resources and resources are limited.

Distributed systems designers often resort to ad hoc heuristics to address allocation problems. A particularly common technique is to use locally managed, independent resource managers that follow simple strategies at each node with no coordination. For instance, a CDN might use an independent least-recently-used cache manager in each node. These heuristics are typically validated using limited traces collected from the field. While some heuristics might fit some traces well, heuristic techniques are neither robust to fluctuations in load characteristics nor do they enable the system designer to reason definitively about a system's emergent properties after deployment.

Heuristics perform surprisingly well on a given trace. After all, they're effectively a way of fitting a function to a workload, and there's no reason—besides the designer's internal motivation to keep the system simple—why the fit can't incorporate the entire workload and thus be perfect. The question to ask of heuristics-based approaches, then, is not how well they perform but for how broad a set of workloads they perform well. This characterization is often difficult to formalize: heuristics might work, but there's often no telling when they will stop working.

Mathematical optimization offers a more principled approach to resource management in distributed systems. The term *optimization* is commonly used in computer science to refer to incremental program transformations designed to improve performance, but *mathematical optimization* is a process for finding the true optimal point in a given function, subject to optional constraints. It offers a general approach to resolving resource allocation problems in distributed systems. The pillars of this approach are analytical modeling to capture the core trade-off, analytical and numerical techniques for determining the optimal solution, and limited runtime aggregation for estimating parameters in the solution. This technique is quite general, and my group at Cornell University has applied it to diverse problems including the design of high-performance, scalable infrastructure services such as CDNs, publish-subscribe systems, and large-scale name systems.

## A principled approach

Our approach to finding the optimal resource allocation for competing tasks consists of four steps.

***Capture the trade-off.*** The first step analytically captures the relationship between the amount of resources awarded and the resulting performance achieved. This process requires an articulation of the performance metrics of interest and a formulation of the metrics as a function of the resources. We call this the *performance equation*.

***Express the constraints.*** The second step captures the constraints on critical resources. Typically, two kinds of constraints exist: *resource constraints* and *performance targets*. Resource constraints arise naturally whenever a finite resource is being partitioned. For example, the sum of all bandwidths allocated to competing clients in a CDN can't exceed line speed. Resource constraints force the system to achieve the best possible performance while remaining within an upper bound on resource consumption.

In contrast, performance targets pose a lower bound on the performance equation that the system must achieve. For instance, a publish-subscribe system might want to ensure that the average time to propagate new information to subscribers is below a particular threshold. Such performance targets force the system to achieve the desired performance level while minimizing resource consumption.

***Solve the system.*** Having expressed the performance equation and the constraints, we can now solve the system. We can solve a system with performance equation $f$ and constraint equation $g$ by introducing the

Lagrange multiplier $\lambda$, and solving for $\nabla f = \lambda \nabla g$. Often, this solution will require differentiation with respect to independent resource allocations $x_i$. The system will typically be analytically tractable if the system of equations is independent in $x_i$. Analytical solutions are desirable because they lead to formulas that an implementation can evaluate efficiently.

In cases where analytical solutions aren't tractable, you can use numerical techniques to solve for $\lambda$ and $x_i^*$, the optimal resource allotment for each competing task.

***Implement the solution.*** Translating the optimal solution into a concrete implementation is often nontrivial. The solutions, whether analytical or numerical, require you to determine parameter values for the system of equations. For instance, optimal bandwidth allocation for object replication in a CDN will typically require a relative ranking of objects by popularity. Determining this order is difficult; done naively, it requires global information. At this stage, various domain-specific design considerations might be used to reduce the amount of communication and to replace global computations with limited, local data aggregation over existing channels. For instance, the structure provided by a distributed hash table can simplify the task of propagating such aggregate information on the relative popularity of objects.

## Applications

My group recently applied this approach to the construction of three infrastructure services.

CoDoNS is a high-performance, failure-resilient, and scalable name service for the Internet. It serves as both a short-term safety net and a long-term replacement for the legacy Domain Name System.[2,3] Mathematical optimization enables CoDoNS to provide strong optimality guarantees. Specifically, the system can achieve $O(1)$ lookups on top of an $O(\log N)$ peer-to-peer overlay. The result is surprising because heavy-tailed distributions, which occur frequently in distributed systems such as DNS,[4] the Web,[5] and RSS,[6] were long thought to pose difficult performance problems, since typical heuristics perform poorly on such inputs.[7] Formally expressing the trade-offs as a mathematical optimization problem enabled us to build a system that can achieve very low lookup latencies and respond to sudden changes in object popularity, as in the so-called "slashdot effect."

CobWeb is an open-access CDN that can deliver Web pages quickly and efficiently. CobWeb operates as a ring of cooperative proxy servers, each of which can serve any HTTP request. When a client requests Web objects, CobWeb fetches them from their origin servers and inserts them into the ring of cooperating proxies. Through an analysis of Web object popularity, size, and update rate, CobWeb then computes an optimal replication strategy for each object to provide low lookup latency while minimizing overhead. This system is similar to Akamai's CDN but uses no heuristics and provides a strong performance guarantee.

Corona is a publish-subscribe system for quick dissemination of Web micronews.[8] It is a replacement for RSS, the currently dominant technology by which clients monitor sources, such as Web sites, blogs, and news, for recent updates. The core optimization that Corona performs differs from CoDoNS and CobWeb in that the constraints it addresses are not flat constants but vary with the client population. Specifically, the system places no more load on the network than what plain RSS would place if it were used instead, but it improves update latency by three orders of magnitude. Clients that used to receive aggregated updates every hour can receive them within seconds, with no additional load on the network.

The use of mathematical optimization in system design enables strong performance guarantees and provides assurance under a wide, well-characterized set of workloads.

Overall, the use of a principled resource allocation framework in these systems provides strong confidence in system robustness. In other research,[2] we've also shown that formally capturing and optimizing the central resource trade-off enables qualitative improvements in system performance.

## Benefits

Mathematical optimization is a promising, principled way to approach problems that are too often resolved via ad hoc heuristics, and the approach outlined here is widely applicable. For instance, my group recently examined failure detectors, which are simple building-block components found in virtually every distributed system. A failure detector is simply a bandwidth allocator whose goal is to minimize failure detection time without exceeding a given bandwidth budget. Early simulations based on data from PlanetLab indicate that mathematical optimization can improve failure detection latencies by a factor of two without increasing bandwidth consumption. Similar improvements are possible for energy consumption in sensor networks, bandwidth consumption in software distribution, and even for processing overhead in a secure operating system through the judicious selection of optimal chunk size in data transfers.[9]

The use of mathematical optimization in system design enables strong performance guarantees and provides assurance under a wide, well-characterized set of workloads. We call on system designers to abolish unreliable heuristics in favor of a more principled approach to resolving difficult resource management problems.

## Acknowledgments

I thank Venugopalan Ramasubramanian, Yee Jiun Song, Ryan Peterson, Bernard Wong, Kelvin So, Dan Williams, Alan Shieh, and Rohan N. Murty for designing, building, and deploying systems based on the approach described in this article. This work was supported in part by US National Science Foundation grants CAREER 0430161 and CCF-0424422.

## References

1. J. Nabrzyski, J.M. Schopf, and J. Weglarz, eds., *Grid Resource Management: State of the Art and Future Trends*, Kluwer Academic, 2004.

2. V. Ramasubramanian and E.G. Sirer, "Beehive: Exploiting Power Law Query Distributions for $O(1)$ Lookup Performance in Peer-to-Peer Overlays," *Proc. Networked Systems Design and Implementation* (NSDI 04), Usenix, 2004, pp. 99–112.

3. V. Ramasubramanian and E.G. Sirer, "The Design and Implementation of a Next Gener

ation Name Service for the Internet," *Proc. Sigcomm*, ACM Press, 2004, pp. 331–343.

4. J. Jung et al., "DNS Performance and Effectiveness of Caching," *Proc. ACM Sigcomm Internet Measurement Workshop*, 2001; www.imconf.net/imw-2001/proceedings. html.
5. L. Breslau et al., "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. 18th Ann. Joint Conf. IEEE Computer and Communications Societies* (Infocom 99), vol. 1, IEEE Press, 1999, pp. 126–134.
6. H. Liu and E.G. Sirer, "Client Behavior and Feed Characteristics of RSS, a Publish-Subscribe System for Web Micronews," *Proc. Internet Measurement Conf.*, Usenix, 2005; www.usenix.org/events/imc05/tech/full_papers.liu_hongzhou/liu_hongzhou_html.
7. A. Wolman et al., "On the Scale and Performance of Cooperative Web Proxy Caching," *Proc. 17th ACM Symp. Operating System Principles (SOSP)*, ACM Press, 1999, pp. 16–31.
8. V. Ramasubramanian, R. Peterson, and E.G. Sirer, "Corona: A High-Performance Publish-Subscribe System for the World Wide Web," to be published in *Proc. 3rd Usenix/ACM Symp. Networked System Design and Implementation* (NSDI 06), 2006.
9. D. Williams and E.G. Sirer, "Optimal Parameter Selection for Efficient Memory Integrity Verification Using Merkle Hash Trees," *Proc. 3rd Int'l Symp. Network Computing*, IEEE CS Press, 2004, pp. 383–388.

**Emin Gün Sirer** is an assistant professor of computer science at Cornell University. His research interests include self-organizing peer-to-peer systems, reputation systems, and a new operating system for trusted computing. He received his PhD in computer science from the University of Washington. Contact him at Cornell Univ., Dept. of Computer Science, 4119A Upson Hall, Ithaca, NY 14853; egs@cs.cornell.edu.

## Autonomic Communication: Business-Driven Revolution

Hakima Chaouchi, *National Institute of Telecommunication*
Mikhail Smirnov, *Fraunhofer Fokus*

The communications world has seen a tremendous variety of technologies, mechanisms, and architectures over the past several decades—from circuit- and packet-switching to service-switching, from fixed to wireless networks such as mobile, ad hoc, or sensor networks, and so on. However, the enormous deployment of TCP/IP networks—a result of their internetworking simplicity—is currently hindering further development of the underlying network infrastructure.

The problem arises from disincentives for carriers to make infrastructure investments that they find it hard to charge customers for. Recently described as the *Internet profit dilemma*,[1] the problem has three interdependent causes:

- *best-effort TCP service model*—adding capacity for premium services also improves best-effort services, so end users are less motivated to upgrade to the premium services;
- *lack of settlement interfaces*—Internet service providers don't have commercial-grade interfaces to bill each other for anything more complex than aggregates of best-effort traffic, so any premium service degrades to best-effort at interdomain boundaries; and
- *convergence*—as traditional carrier services converge on the Internet, the revenues from them are disappearing.

The enormous deployment of TCP/IP networks—a result of their internetwork simplicity—is currently hindering further development of the underlying network infrastructure.

### Automation solutions

The Internet Engineering Task Force recognized the lack of intercarrier settlement interfaces for IP traffic as a problem when it began quality-of-service (QoS) work on *differentiated services* (www.ietf.org/html. charters/OLD/diffserv-charter.html). This standardization effort assumed that contracted access to the Internet is feasible. It proposed access automation as a self-acting process at the business boundary. Carriers would negotiate intercarrier *service-level agreements*, and SLA technical specifications would configure QoS network elements accordingly.

The IETF failed to standardize straightforward automated access for two reasons.

First, even with automated access, it's impossible to ensure *invariance under aggregation*[2] without coordinated automation of other in-network processes, such as traffic engineering. (A network achieves invariance under aggregation if it can handle a packet marked with a DiffServ service class on every one of its links regardless of link utilization and state. This is possible only with runtime traffic engineering in place—a complex task that's still far away.

Second, automating access at both retail (end-user access) and wholesale (service settlement) business boundaries requires access, service, and resource control mediation[3] that's impossible to achieve in a single layer of a protocol stack.

### Adaptation solutions

IP's designers succeeded in finding the least common functionality required for forwarding, address resolution, and routing of packetized media over any link layer technology. They systematically designed the protocol with self-adaptation in mind. Routing experience proves that fairly complex and intelligent functionality can be safely placed inside the datagram network, but only if the functionality can act self-adaptively by, for example, recovering from topology changes.

The *end-to-end principle*, a term coined to prevent the in-network placement of functions and intelligence that couldn't recover automatically, is currently in question.[4] Its relaxation for any other feature will require either a self-adaptation internetworking layer like IP or, much more feasible, cross-layer optimization of a protocol stack.

### Reconfiguration approaches

The Wireless World Research Forum (www.wireless-world-research.org) sees reconfigurability as a way to enable a seamless experience in all-IP infrastructures. In the wireless domain, reconfigurability expands the principles of software-defined radio beyond the composite-radio environments tailored to the telecom sector.[5] The need for end-to-end reconfiguration adds an abstraction layer to an already complicated network management architecture. The extension requires cross-layer integration, but only at the wireless ends of the communication channel.

In more general settings, reconfigurability appears as a business layer that controls business relationships,[1] not networks. The business layer forces traditional services to move to the overlay as much as the current abstraction requires. In both cases, the reconfiguration pushes the complexity of either radio or trade to a dedicated layer, while the underlying network mechanisms stay largely unaware of the reconfiguration process.

## Evolution

The global information infrastructure evolves concurrently in many dimensions and increases protocol stack diversity. Automation, adaptation, and reconfigurability contribute to a seamless service experience. However, they can't properly address self-adaptation based on network context within a highly heterogeneous networking environment.[6]

A recently proposed *network pluralism architecture* embraces heterogeneity in the hope of allowing radical innovation.[7] This architecture doesn't abandon the homogeneous Internet architecture but retains it as one architecture among many. The architecture divides the world into homogeneous contexts understood as sets of bindings. Interstitial functions interconnect distinct contexts; endpoints can belong to different contexts simultaneously and dynamically. This view seemingly dissects the Internet effort, unless we assume that the endpoint's protocol stack enables the hypothetical interstitial functions.

The *autonomic communication* vision sees a stack instance as being dynamically composed on demand and shaped by available network contexts. This doesn't mean that all interfacing functions are limited to endpoints only. We see a control within autonomic communication being instantiated as a tree rooted at an endpoint and having as much in-network support as current context can provide. The true self-management then will be the result of autonomic decision-making in collaboration with other endpoints and network infrastructure when available. The continuously evolving complexity in the communication system makes it necessary to design a protocol stack that can learn and evolve to fill a newly identified need. This contrasts with providing a "face lift" for a communication system designed to be simple whenever a new need is expressed. Such an approach will have only limited impact.

## Revolution

We claim that the future of communication is being defined now by autonomic communication research and experimentation. This work aims to replace a network architecture of predefined, layered functionality with a ready-to-evolve architecture that will include three sets of functionalities:

- *basic functionalities*, which can be layered or cross-layered;
- *advanced functionalities*, which will emerge as the network architecture evolves; and
- *evolving functionalities*, which will be capable of learning and adding new properties and treatments in the network.

A ready-to-evolve architecture will be open to include new technologies, mechanisms, and functionalities that the network can handle differently. The evolving functionality allows network architecture to be seen as a program.

The autonomic communication vision sees a protocol stack instance as being dynamically composed on demand and shaped by available network contexts.

An autonomic communication roadmap would identify the vital relationships between evolving functionality and the set of basic network functionalities, the evolving process, and the programming languages for building functionality that assumes programmers might not necessarily be humans. Network programming will necessarily be context-driven, where context will influence both "code" and its requirements.

The necessary technology is already out there in bits and pieces. The targeted research must build a coherent "programming environment," in which coprogramming entities are as natural as coexisting elements within an ecosystem. This "Net ecology" will revolutionize the way we communicate and do business over the Net. It might also require us to learn how best to coexist with a new ecosystem. ■

## References

1. T. Nolle, "A New Business Layer For IP Networks," *Business Communications Rev.*, July 2005, pp. 24–29.
2. K. Nichols and B. Carpenter, *Definition of Differentiated Services per Domain Behaviors and Rules for Their Specification*, IETF RFC 3086, Apr. 2001; www.ietf.org/rfc.rfc3086.txt.
3. G. Cortese et al., "CADENUS: Creation and Deployment of End-User Services in Premium IP Networks," *IEEE Communications*, vol. 41, no 1, 2003, pp. 54–60.
4. D.D. Clark et al., "Making the World (of Communication) a Different Place," *ACM Computer Communication Rev.*, vol. 35, no. 2, 2005, pp. 91–96.
5. P. Demestichas et al., "Evolution in Wireless Systems Management Concepts: From Composite Radio Environments to Reconfigurability," *IEEE Communications*, vol. 42, no. 5, 2004, pp. 90–98.
6. N. Niebert et al., "Ambient Networks—An Architecture for Communication Networks Beyond 3G," *IEEE Wireless Communications*, vol. 11, no. 2, 2004, pp. 14–22.
7. J. Crowcroft et al., "Plutarch: An Argument for Network Pluralism," *ACM Computer Communication Rev.*, vol. 33, no. 4, 2003, pp. 258–266.

**Hakima Chaouchi** is an associate professor at the National Institute of Telecommunication in France. Her research interests are in wireless and mobile networks. She received her PhD in networking from Paris VI University. She is cochair of the Autonomic Communication Forum's self-ware group and a member of UNESCO's International College to help deploy new technologies in the third world. Contact her at National Inst. of Telecommunication, 9 rue Charles Fournier, 91011 Evry, France; hakima.chaouchi@int-evry.fr.

**Mikhail Smirnov** is a member of the board of directors of Fraunhofer FOKUS Research Institute for Open Communication Systems and an adjunct professor at Technical University Berlin. His research interests include advanced Internet services, distributed and autonomic communication, scalable group communication, and policy-based programmability. He received his PhD in computer science from St. Petersburg Electrotechnical University, Russia. He is vice chair of Infocom 2006. Contact him at Fraunhofer FOKUS, Kaiserin-Augusta-Allee 31, Berlin 10589, Germany; mikhail.smirnov@ fokus.fraunhofer.de.