

---

# Autonomy in Spatial Computing

---

Jonathan Bachrach, Jacob Beal

JRB@CSAIL.MIT.EDU, JAKEBEAL@MIT.EDU

An increasing number of distributed systems may be viewed as *spatial computers*—collections of devices distributed to fill a physical space, and where the difficulty of moving information between any two devices is strongly dependent on the distance between them. Examples include peer-to-peer wireless networks, robotic swarms, wireless sensor networks, and reconfigurable computing platforms (e.g. FPGAs), as well as natural systems like animal swarms and cells during morphogenesis.

Spatial computers pose a major autonomic computing challenge due to their potential scale and radical decentralization. First, the scale of a spatial computer can range across many orders of magnitude. For example, consumer electronics with peer-to-peer wireless have the potential to form networks ranging from a handful of devices in a personal network to thousands or millions of interconnected devices across an urban area. Second, spatial computers are radically decentralized: the cost of long-distance communication is high, and aggregates often have high diameter and no backbone infrastructure. Nevertheless, they must be able to cope with frequent changes in the set of participating devices and the structure of the network, and do so with little or no human management, due to the large numbers of devices involved or the circumstances of their deployment. For example, a consumer electronics network is effectively unmanaged because most cell-phone owners are not sysadmins. Spatial computing is thus a domain with serious autonomic computing challenges.

**Continuous-Space Approach** One promising approach to the challenges of spatial computing is to focus not on the network of devices, but on the continuous space that they occupy, using the *amorphous medium* abstraction. An amorphous medium[2] is a manifold with a computational device at every point, where every device knows the recent past state of all other devices in its neighborhood (Figure 1). While an amorphous medium cannot, of course, be constructed, it can be approximated on the discrete network of a spatial computer. Our language, Proto[2], uses the amorphous medium abstraction to factor programming a spatial computer into three loosely coupled subproblems: global descriptions of programs, compilation from global to local execution on an amorphous medium, and discrete approximation of an amorphous medium by a real network.

Proto is a functional language that is interpreted to pro-

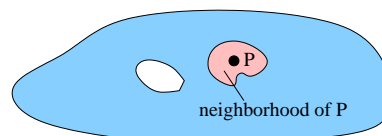


Figure 1. An amorphous medium is a manifold where every point is a device that knows its neighbors' recent past state.

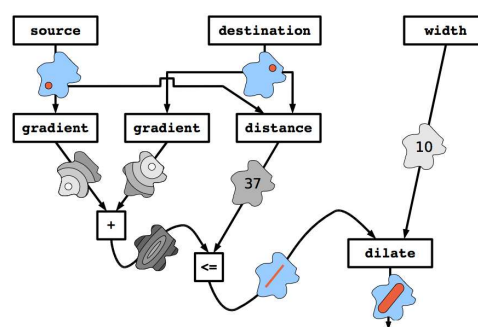


Figure 3. A Proto program—here creating a channel between two regions—specifies a dataflow graph of operations on fields. The program is shown evaluated on an irregularly shaped space, with scalar fields grey (lighter is less) and boolean colored (**true** is red).

duce a dataflow graph of operations on fields (Figure 3). This program is then evaluated against a manifold to produce a field with values that evolve over time. Proto uses four families of operations: pointwise operations like + that involve neither space nor time, restriction operations that limit execution to a subspace, feedback operations that establish state and evolve it in continuous time, and neighborhood operations that compute over neighbor state and space-time measures and summarize the neighbors with a bulk operation like integral or minimum.

With appropriate operators, compilation and discrete approximation are straightforward. Thus, Proto makes it easy for a programmer to carry out complicated spatial computations using simple geometric programs that are robust to changes in the network and self-scale to networks with different shape, diameter, density of nodes, and execution and communication properties[1].

**Other Approaches** Another major approach to spatial computing are viral code systems, (e.g. TOTA co-fields[6]). Closer to Proto is Regiment[8], which gath-

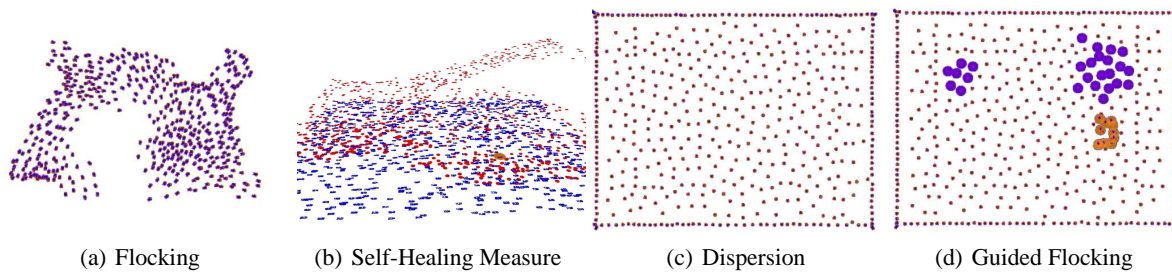


Figure 2. Three emergence primitives—flocking (a) self-healing distance measure (b) and dispersion (c)—combined using Proto to form guided flocking (d), where devices disperse to guide a mobile flock (orange) to the nearest goal region (purple).

ers streaming data from spatial regions. Others include Kairos[5], which operates on abstract graphs, pattern languages like Origami Shape Language[7] and Growing Point Language[3], and modular robotics systems (e.g. [9]).

**Libraries for Engineered Emergence** Emergent phenomena in natural systems have long been a source of inspiration for spatial algorithms—examples include plant tropism[3], ant foraging[4], and epithelial folding during morphogenesis[7]. Although these phenomena are often robust and cheap to implement, it is notoriously hard to predict what will happen when they are used in a new situation or used in a larger system.

Proto, however, is well-suited to building a library of emergent phenomena that can be deployed and composed to produce predictable behavior: programming for continuous space reveals scaling problems and increases portability, functional composition limits the opportunities for phenomena to interfere with one another, and the restriction operator allows the behavior of an emergent phenomenon to be modulated by modulating the space in which it is running.

For example, Figure 2 shows three emergent phenomena captured in Proto: flocking, self-healing distance measures (based on morphogenetic gradients), and dispersion. These take only 13, 14, and 3 lines of code, respectively. We can combine these predictably to produce guided flocking, with devices dispersing to guide a flock down a distance gradient to a goal region. Doing so takes only 8 lines of Proto code, for a total of a mere 38 lines of code to describe a predictable composite of three emergent behaviors.

**Future Directions** The continuous-space approach, as embodied by Proto, thus provides a good foundation for tackling the challenge of autonomy in spatial computers. From this, we have identified the following questions as key to future progress:

- What emergent behaviors can be captured as library routines, and how can they be effectively composed?
- Are there application areas in spatial computing for which the continuous-space approach is unsuited?

- How can the approximation error of a program be predicted from global network properties and the approximation error of its subroutines?
- How can we predict the sensitivity of programs to small perturbations in their input (e.g. from moving devices or changing data)?
- How can faulty or malicious devices be handled, and does the spatial distribution of devices make the problem easier or harder?
- How does high-level programming interact with management of scarce resources (e.g. power, bandwidth)?
- It is difficult to tell the difference between a data source that has moved far away and one that has vanished. How can algorithms cope with this ambiguity?

Finally, although spatial computing is a special case, it is reasonable to expect that some of the principles of engineered emergence discovered will apply to the more general field of autonomic computing as well.

## References

- [1] J. Bachrach, J. Beal, and T. Fujiwara. Continuous space-time semantics allow adaptive program execution. In *IEEE SASO 2007*, July 2007.
- [2] J. Beal and J. Bachrach. Infrastructure for engineered emergence in sensor/actuator networks. *IEEE Intelligent Systems*, pages 10–19, March/April 2006.
- [3] D. Coore. Establishing a coordinate system on an amorphous computer. In *MIT Student Workshop on High Performance Computing*, 1998.
- [4] M. Dorigo and T. Stutzle. *Ant Colony Optimization*. MIT Press, 2004.
- [5] R. Gummadi, O. Gnawali, and R. Govindan. Macroprogramming wireless sensor networks using *kairos*. In *DCOSS*, pages 126–140, 2005.
- [6] M. Mamei and F. Zambonelli. Programming pervasive and mobile computing applications: the tota approach. *ACM TOSEM*, to appear 2008.
- [7] R. Nagpal. *Programmable Self-Assembly: Constructing Global Shape using Biologically-inspired Local Interactions and Origami Mathematics*. PhD thesis, MIT, 2001.
- [8] R. Newton and M. Welsh. Region streams: Functional macroprogramming for sensor networks. In *Workshop on Data Management for Sensor Networks (DMSN)*, Aug. 2004.
- [9] K. Stoy. Controlling self-reconfiguration using cellular automata and gradients. In *8th int. conf. on intelligent autonomous systems (IAS-8)*, 2004.