Chapter 1

ENGINEERING AMORPHOUS COMPUTING SYSTEMS

Radhika Nagpal
Massachusetts Institute of Technology
Cambridge, MA, USA 02139
radhi@ai.mit.edu

Marco Mamei
University of Modena and Reggio Emilia
Reggio Emilia, Italy
mamei.marco@unimo.it

Abstract How does one engineer robust collective behavior from the local interactions of immense numbers of unreliable parts? On the one hand, emerging technologies like MEMS are making it possible to assemble systems that incorporate myriad of information-processing units at almost no cost: smart materials, self-assembling structures, vast sensor networks. On the other hand, the plummeting cost of ad-hoc wireless communication is realizing the idea of pervasive computing: the creation of environments saturated with wireless computing devices collectively providing services anytime and everywhere. We discuss organizing principles and programming methodologies for controlling such *amorphous* systems, by combining robust algorithms inspired by nature with computer science techniques for controlling complexity.

Keywords: multi-agent, self-organization, pervasive computing.

1. Introduction

Over the next few decades, emerging technologies will make it possible to assemble systems that incorporate myriad of information-processing units at almost no cost, provided that all the units need not work per-

fectly and that there is no need to manufacture precise geometrical arrangements or precise interconnections among them. This technology shift requires fundamental changes in methods for constructing and programming computers, and perhaps in our view of computation itself.

Microelectronic mechanical components have become so inexpensive to manufacture that we can anticipate combining logic circuits, microsensors, actuators, and communications devices, integrated on the same tiny chip to produce particles that could be mixed with bulk materials, such as paints, gels, and concrete. Imagine coating bridges or buildings with smart paint that can sense and report on traffic and wind loads and monitor structural integrity of the bridge. A robot, built of millions of tiny programmable modules, could assemble itself into different shapes, perhaps as a cube for storage and then reconfiguring into a shelter or tool as needed. Already many such novel applications are being envisioned and built [3, 5, 7, 13]. Even more striking is the emerging research in biocomputing, that may make it possible to harness the many sensors and actuators in cells and program biological cells to function as drug delivery vehicles or chemical factories for the assembly of nanoscale structures [29]. Pervasive computing and sensor networks are creating massive distributed systems at a different scale, from remote habitat monitoring to smart buildings and smart cars [16, 26].

These novel computational environments pose significant challenges, beyond just the manufacturing of parts. Digital computers have always been constructed to behave as precise arrangements of reliable parts, and almost all techniques for organizing computations depend upon this precision and reliability. We can envision producing and deploying vast quantities of individual computing elements — whether microfabricated particles or engineered cells or wireless sensors — but we have few ideas for programming them effectively.

The opportunity to exploit these new technologies poses a broad conceptual challenge, the challenge of *amorphous computing* [1]:

1 How does one engineer robust collective behavior from the cooperation of immense numbers of unreliable parts that are interconnected in local, irregular, and time-varying ways?

2 How does one translate prespecified global goals into the local interactions of vast numbers of parts?

The critical task is to identify appropriate organizing principles and programming methodologies for controlling such systems. Hints for how to design robust collective behavior may come from natural systems, such as biology. The growth of form in organisms demonstrates that

well-defined shapes and functional structures can develop through the interaction of cells under the control of a genetic program, even though the precise arrangements and numbers of the individual cells are variable. At the same time, as engineers, we must learn to construct systems so that they end up organized to behave as we *a priori* intend, not merely as they happen to evolve. Therefore a critical piece is to develop programming methodologies, and *languages*, that allow us to combine these robust organizational principles to achieve the global goals we want.

In this article we describe work that has been done as part of the amorphous computing effort to address these challenges. Section 2 presents the amorphous computing model, section 3 discusses how developmental biology can provide inspiration for robust algorithms and section 4 presents examples of how we can combine these algorithms into programming languages. In section 5 we discuss the relationship with pervasive computing and show how similar methods have been developed elsewhere to coordinate behavior in mobile networks of agents[16].

2. The Amorphous Computing Model

An amorphous computer consists of massive numbers of identically-programmed and locally-interacting computing agents, embedded in space. We can model this as a collection of "computational particles" sprinkled randomly on a surface or mixed throughout a volume.

The individual agents have limited resources, limited reliability and local information. The agents are all programmed identically, although each agent executes its program autonomously and has means for storing local state and generating random numbers. Each agent can communicate with a only a few nearby neighbors. In amorphous systems of microfabricated components, the agents might communicate via short-distance radio or through the substrate itself; bioengineered cells might communicate by chemical means. For our purposes here, we assume that there is a communication radius r, which is large compared with size of individual agents and small compared with the size of the entire area, and that two agents can communicate if they are within distance r. The agents can also sense and affect the environment locally.

In many ways the massively parallel nature of an amorphous computer resembles, and takes inspiration from, models such as cellular automata. However it presents a significantly different challenge because the mechanisms must be independent of the detailed configuration of the agents. We assume that access to centralized sources of information is limited, whether it be global clocks or globally-accessible beacons for triangulating position. Rather the goal is for the agents to self-organize global

information as necessary. The agents are not synchronized, although we assume that they compute at similar speeds, since they are all fabricated by the same process. The agents have no *a priori* knowledge of global position or orientation; however some agents may be started in a special initial state. The agents are possibly faulty, and are can die or be replaced at any moment. The individual agents may be mobile, but in many of the examples here we assume that they have fixed location and are randomly distributed on a two-dimensional plane.

3. Developmental Biology as an Inspiration

Biological systems regularly achieve coherent, reliable and complex behavior from the cooperation of large numbers of identically programmed agents. One of the most fascinating examples is embryogensis. Cells with identical DNA, cooperate to form incredibly complex structures from a nearly homogeneous egg, with incredible precision and reliability in the face of constantly dying and growing parts[30]. There is a plethora of examples of regulation in different organisms, that can compensate for large variations in cell size, cell numbers, cell division rates and development time [9]. Even after development, organisms such as the starfish, retain incredible abilities for self-repair and regeneration.

These examples hint at powerful underlying mechanisms that can adapt to variation, while maintaining constraints that may be geometric, topological or functional. Studies of developmental biology can form an important source of inspiration — not only for mechanisms, but also for the kind of robustness that is achievable.

Morphogen Gradients. One example of a mechanism common throughout development is the use of gradients of morphogens to determine positional information and polarity. In the *Drosophila* embryo, cells at one end of the embryo emit a morphogen (protein) that diffuses along the length of the embryo. The concentration of this morphogen is used by other undifferentiated cells to determine whether they lie in the head, thorax or abdominal regions [14]. Different morphogens are used for determining the dorsal-ventral axis, wing and limb development, and even leg bristle polarity. Gradients of morphogens are believed to play an important role in providing position and polarity information in many different organisms, and even in regeneration [30].

We can emulate the concept of a morphogen gradient using a simple agent program. An initial "source" agent, chosen by a cue from the environment or by generating a random value, creates a gradient by sending a message to its local neighborhood with the morphogen name and a value of zero. The neighboring agents forward the message to

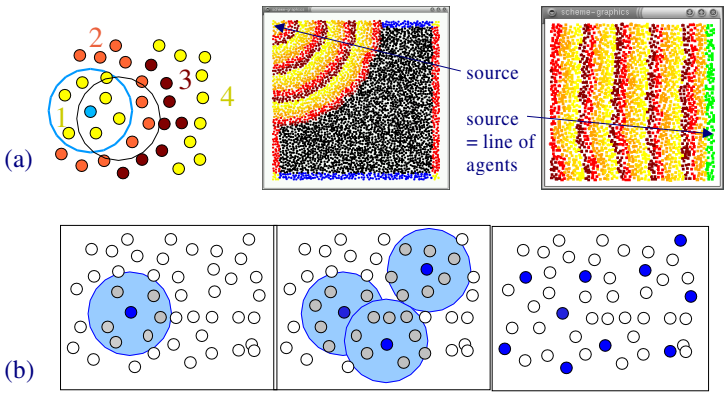


Figure 1.1. (a) Emulating morphogen gradients; gradients created by point and line sources (b) Spacing created by lateral inhibition

their neighbors with the value incremented by one and so on, until the morphogen has propagated through the entire population. Each agent stores and forwards only the minimum value it has heard for a particular morphogen name, thus the morphogen value represents the shortest path from the source. The value provides an estimate of distance from the source: a point reached in n steps will be roughly distance nr away. The quality of this estimate depends on the density of the agents and can be theoretically predicted for random distributions [20].

This very simple program can be used in powerful ways.

1 **Regions and Polarity:** By limiting the maximum value of a morphogen, one can create regions of controlled size. The morphogen can also be used to provide a sense of local orientation; an agent can compare values in its local neighborhood to determine the direction towards or away from the source.

2 **Spatial Patterning:** More than one agent could be the source for the same morphogen, in which case the morphogen value reflects the shortest distance to *any* of the sources. Thus, if a single agent emits a morphogen then the value increases as one moves radially away from the agent, but if a line of agents emits a morphogen then the value increases as one moves perpendicularly away from the line. Complex spatial patterning can be created by positioning the sources, without any change to the agent program.

3 **Selective Propagation:** The agent program can be modified such that agents selectively choose which morphogens to propagate. Thus agents in particular state can act as barriers to specific

morphogens, or as obstacles around which the morphogen must travel. Similarly morphogens can be limited to propagate only with certain spatial regions.

4 **Active Morphogens:** We can allow the source agent to constantly produce a morphogen message, and have the morphogen value stored by any agent lose significance if not constantly reinforced. The result is that the morphogen values adapt as agents, or sources, appear and disappear.

These are just a few of the ways in which the morphogen gradients can be used. This simple agent program is the basis of many different amorphous computing algorithms for self-organizing coordinate systems, distributed storage, and ad-hoc routing. In section 5, we see that the same idea appears in many different forms elsewhere.

Lateral Inhibition. While morphogens produce a sense of distance and orientation, lateral inhibition is believed to produce regularly spaced patterns in many different organisms. For example, in the *Drosophila*, epidermal cells on the leg can produce bristles, however not all cells grow bristles. The bristles are regularly spaced with some minimum distance between them. Lawrence [14] describes the mechanism by which this is achieved: When a cell produces a bristle, it also secretes a short-range morphogen that inhibits nearby cells from growing bristles. An uninhibited cells will eventually attempt to produce a bristle. The result is bristles appear throughout the leg surface but never too close. Lateral inhibition is believed to play a role in creating uniform spacing in many different settings, from the spacing of hair on human skin, to the regular crystal like spacing of ommatidia in the Drosophila eye.

Again this mechanism can be emulated by a simple agent program. An agent picks a random number within a range L, and counts down. If it reaches zero without being interrupted, then it becomes a leader (grows a bristle) and sends out an inhibition message to all its neighbors within the distance r. If an agent hears an inhibition message, then it no longer counts down to become a leader but instead becomes a follower. The process ends after L steps, with all agents as leaders or followers.

The result of this very simple local rule is that *leaders emerge with a spacing of r to $2r$ apart, throughout the surface.* If we extend the inhibition to travel h hops distance, then the spacing between leaders increases to hr to $2hr$. To see why this is true, consider an agent that becomes a leader. It could only have done so because no other leader was within distance r inhibiting it; and once it becomes a leader it inhibits leaders forming within distance r. At the same time, an agent

that is not inhibited continues to count down and eventually becomes a leader. This guarantees that every agent is within distance r from some leader, and with high probability no leaders are closer than r. The spacing is not perfect because two neighboring agents may choose the same random number and reach zero at the same time. Therefore the range L is chosen to make the probability of such collisions low. Nagpal and Coore have shown that the behavior of this algorithm can be analyzed theoretically, even for asynchronous agents and unreliable agents [22]. This is a valuable algorithm in an amorphous computer, and can be used for spontaneously electing leaders, self-configuring hierarchical routing and graph coloring.

Robust Primitives. Morphogen gradients and lateral inhibition are well-matched to the amorphous setting because the gross phenomena of diffusion and spacing are insensitive to the precise arrangement of the individual agents, so long as the distribution is reasonably dense. In addition, if individual agents do not function, or stop broadcasting, the result will not change very much, so long as there are sufficiently many agents. Many phenomena exist in multicellular systems, from quorum sensing to programmed cell death, that can provide inspiration for robust multi-agent algorithms.

At the same time, it is extremely important to be able to analyze the behavior of these algorithms, so that we have a solid ground to build on top of. We can theoretically analyze the behavior of both algorithms, using techniques from distributed graph algorithm analysis and geometric analysis. For example, the morphogen algorithm can be thought of as computing a breath first search tree, while the lateral inhibition algorithm is a computing a maximum independent set of nodes [15]. The spatial locality of communication gives us a relation from tree depth to distance, and maximum independent set to uniform spacing.

A key difference however, is that rather than produce a perfect answer from a perfect graph, as is typical in distributed algorithms, these algorithms aim to provide a *good-enough answer* with high probability — good enough estimates of distance and direction, and good enough spacing. This allows the agent behavior to be simple, scalable, and tolerant to variation. Looking to biology may provide insights for new approaches to fault tolerance. Traditionally, one seeks to obtain correct results despite unreliable parts. However it seems awkward to describe mechanisms such as embryonic development as producing a "right" organism by correcting bad parts and broken communications. In the amorphous regime, getting the right answer may be the wrong idea. In-

stead, the question is how do we structure systems so we get acceptable answers, with high probability, even in the face of unreliability.

4. Towards Programming Languages

While biology may provide a means for thinking about organizing local behavior robustly, computer science can provide tools for managing complexity. One such tool is a *programming language*. The ability to think and describe goals in terms of high-level abstractions, make possible a complexity that is almost inconceivable to generate by manipulating 1s and 0s. Yet the final computation does happen as bits, and the *compiler* translates from a language that is natural for expressing how to do something, to a low-level execution model that a computer can interpret [2].

In the amorphous computing setting the goal is similar — we would like to be able to translate complex global goals into local behavior, but in such a way that the translation is not mysterious and not hand crafted for each goal — in other words, a global-to-local compiler. In this section we describe two programming languages, aimed at pattern formation and self-assembly. The global shape or pattern is described as a program in terms of abstract entities, which is compiled to produce the behavior of an agent, such that the identically-programmed agents organize into the prespecified goal.

Growing Point Language. Coore developed a language for pattern formation on an amorphous computer[8]. The growing point language (GPL) can be used to specify topological patterns consisting of lines of various thickness, such as those specifying the interconnect of an electronic circuit. The specification is compiled into an agent program. Initially the agents start out with identical state except for a few agents. As a result of executing the program, the agents "differentiate" into components of the pattern.

The language represents processes in terms of a botanical metaphor of "growing points" and pheromones. A pheromone is the same as a morphogen with a limited range. A growing point is a locus of activity that modifies the states of agents as it passes through, and it can respond to the gradient of a morphogen by moving towards lower, higher or similar values of morphogens. A pattern is created by writing a program in terms of *abstract entities*: growing points that lay down materials, materials that secrete pheromones, and tropisms that govern the trajectory of the growing point. However, at the level of the agent these high level concepts translate to a set of simple local rules. For example a growing point is simply a piece of state at an agent. The agent collects values of

```
(define-growing-point (make-red-line length)
  (material red-poly)
  (size 1)
  (tropism (and (away-from red-pheromone)
                (keep-constant pheromone-1)
                (keep-constant pheromone-2)))
  (actions
   (secrete 2 red-pheromone)
   (when ((< length 1) (terminate))
         (default
           (propagate (- length 1))))))
```

Figure 1.2. A program in GPL. This procedure generates a line of specified length that attempts to follow constant values of pheromones 1 and 2.

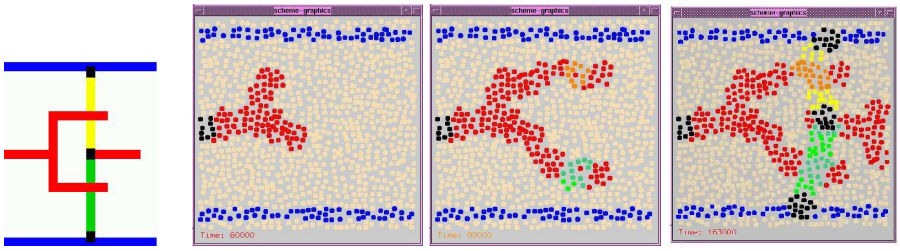


Figure 1.3. The amorphous surface differentiating to create the inverter pattern. All agents execute the same program, which is compiled directly from the GPL specification of the pattern on the left.

morphogens from its neighbors and uses those value to locally compute which neighboring agent to pass the growing point to. The next agent then repeats the same process to determine where to send the growing point next.

Figure 1.2 shows a fragment of a program written in the growing-point language: A growing point process called `make-red-line`, takes one parameter called `length`. This growing point "grows" material called `red-poly` in a band of size 1. This implies that each agent it moves through sets a state bit that will identify the agent as `red-poly`. The growing point moves according to a tropism that directs it away from higher concentrations of `red-pheromone`, in such a way that the concentrations of `pheromone-1` and `pheromone-2` are kept constant. All agents that are `red-poly` secrete `red-pheromone`; consequently, the growing point will tend to move away from the material it has already laid down. The growing point stops when the correct length line has been grown. This procedure is part of a larger GPL program that generates the pattern on the right. This pattern is a caricature of the layout of a

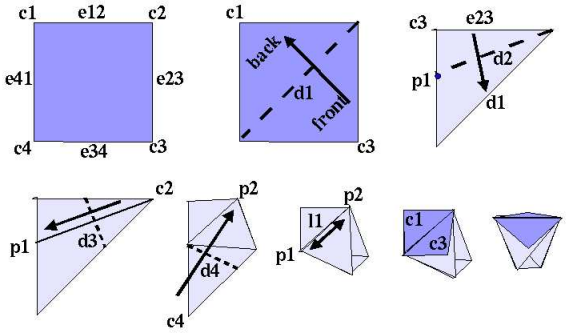
CMOS inverter, where the different colored regions represent structures in the different layers of standard CMOS technology: metal, polysilicon and diffusion. Figure 1.3 shows the agents differentiating to create the inverter pattern. The agents that are part of the top blue rail emit pheromone 1 and the bottom rail emits pheromone 2, thus the code fragment represents the method by which the first red line is drawn parallel to these two rails and away from the edge. The entire program that specifies the shape is only a few paragraphs long, and the resulting state machine for the individual agents requires only about twenty states.

Programmable Self-Assembly. Nagpal developed a language for shape formation on a simulated foldable sheet [20, 21]. In this case the two dimensional surface of agents represents a sheet with a single layer of randomly and densely distributed agents; a set of agents in a line can coordinate to fold the sheet along that line. The folding is modeled abstractly by the simulator, but is inspired by epithelial tissues where a line of epithelial cells can deform to cause the entire sheet to fold along that line; this is believed to be the basis of neural tube formation during embryogenesis [30]. One could imagine building a programmable reconfigurable sheet composed of such flexible agents.

The shape is specified as folding construction on a continuous sheet, using a language called the Origami Shape Language (OSL). The language is based on a set of geometry axioms, described by Huzita to capture the mathematics behind origami paper-folding [10]. A large class of flat folded shapes and line patterns can be constructed using these axioms. OSL builds on these geometry axioms, but also adds concepts such as naming and regions.

The interesting thing about this specification is that it is abstract — there is no notion of morphogens, coordination or even agents! Rather the programmer thinks in terms of a continuous sheet. The agent program is automatically compiled from this description and is composed from a small set of primitives: morphogens, neighborhood query, cell-to-cell contact, polarity inversion and flexible folding.

Figure 1.4 shows a diagram for constructing a cup from a blank square sheet of paper, and the corresponding OSL program. The basic elements of the language are points, lines and regions. Initially, the sheet starts out with four corner points (`c1-c4`) and four edge lines (`e12-e41`). The axioms describe how to generate new lines and points from an existing set of lines and points, purely through folding and unfolding paper. For example, the first operation constructs the diagonal `d1` from the points `c1` and `c2` by using axiom 2; axiom 2 folds the sheet so that `c1` lies on `c2` and then unfolds the sheet to create a line. The sheet can be permanently




```
;; OSL Cup program
;;---------------------
(define d1 (axiom2 c3 c1))
(define front (create-region c3 d1))
(define back  (create-region c1 d1))
(execute-fold d1 apical c3)

(define d2 (axiom3 e23 d1))
(define p1 (intersect d2 e34))
(define d3 (axiom2 c2 p1))
(execute-fold d3 apical c2)

(define p2 (intersect d3 e23))
(define d4 (axiom2 c4 p2))
(execute-fold d4 apical c4)

(define l1 (axiom1 p1 p2))
(within-region front (execute-fold l1 apical c3))
(within-region back  (execute-fold l1 basal c1))
```

Figure 1.4. OSL Program for folding a cup

folded flat along a line, hence the structures created by OSL are flat, but layered. Lines can be used to create regions and regions can be used to restrict folds.

Figure 1.5 shows a programmable sheet differentiating to fold into a cup. Initially the surface is mostly homogeneous, with only the agents on the border having special local state. When the agent program is executed by all the agents in the sheet, the sheet is configured into the desired shape. The overall *global* view of this process is very close to what the diagram of the continuous sheet suggests. This is because each global operation is translated into a local agent behavior. But instead of folding, the geometry is emulated using the biologically-inspired prim-

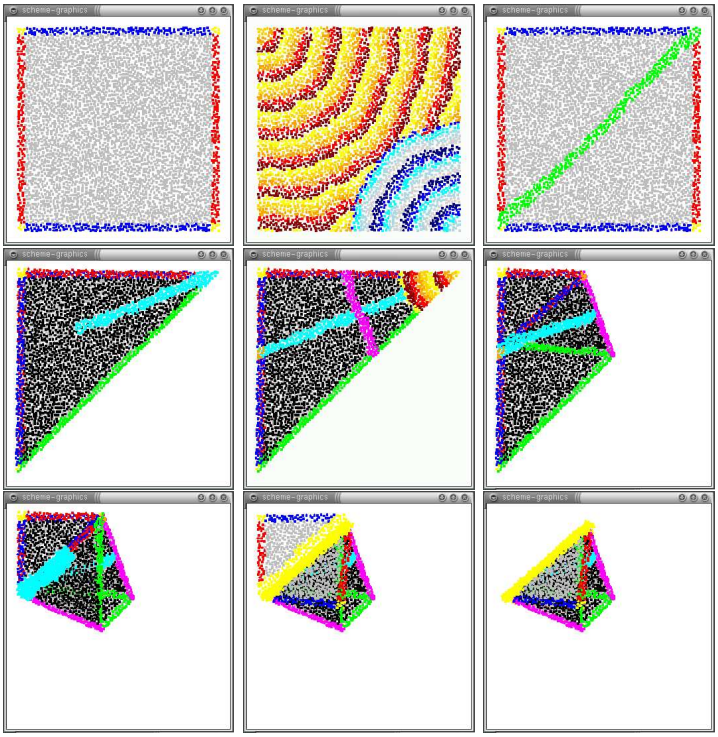


Figure 1.5. Simulation images from folding a cup

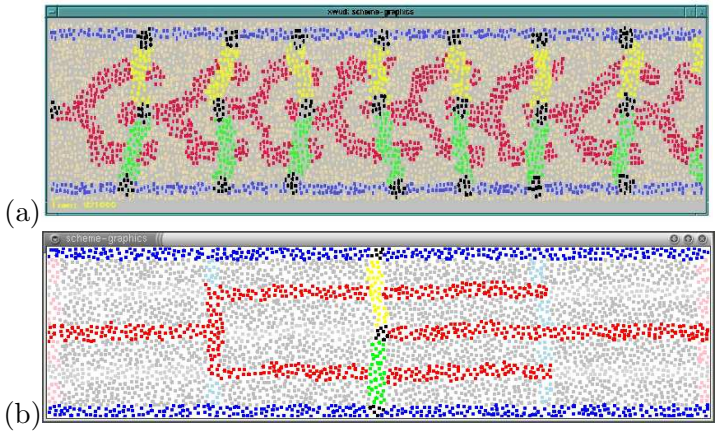


Figure 1.6. Inverter pattern created by (a) GPL (b) OSL when run on a longer sheet

itives. For example in order to implement the first line creation, the agents belonging to `c1` and `c2` create two distinct morphogens. The remaining agents test if the morphogen values are equal; if so then they lie on the new line. This is the local rule corresponding to axiom 2. Morphogens also serve as a form of barrier synchronization, so that agents can determine when it is safe to move on to the next fold operation. Selective propagation of morphogens is used to create regions and confine operations within regions. Each axiom translates into a set of local rules, and the program translates to a sequence of rules.

The Power of Programming Languages. The power of the programming language approach is that it allows us to take advantage of traditional computer science techniques for managing complexity, while relying on biological models for achieving robustness at the local level. The global-to-local compilation confers many advantages: (1) We can reason about the classes of structures that can and cannot be generated by analyzing the expressiveness of the language. (2) The primitives themselves can be made robust by relying on mechanisms inspired by biological systems (3) The analysis of a complex system becomes tractable because it is built in understood ways from smaller parts (4) The high level language makes it possible to *easily* specify complex behavior, without worrying about the millions of parts that are involved.

For example, Coore proved that GPL can generate any prespecified planar graph pattern, up to connection topology, on an amorphous computer. Similarly, OSL can generate any 2D Euclidean construction pattern and all flat folded shapes composed of simple folds. These results are based on results from geometry, that have nothing to do with multiagents or self-organization. At the same time we can separately analyze the robustness of primitives such as morphogens, and how error accumulates when we combine those primitives, so that we can predict what densities and numbers of agents are required to satisfactorily achieve a given high-level goal. Similarly we can analyze time and space complexity. In both languages, the complexity of the agent program is directly proportional to the complexity of the high-level description; by using *procedures* to capture regular patterns and common folding sequences, one can compile more efficient agent programs.

The languages themselves imply certain global properties. For example, the GPL encodes patterns with an *inherent length scale* and can easily describe fractal and space filling structures. The OSL language on the other hand describes structures in a scale-independent manner, by recursively segmenting relative to the original sheet boundary. This results in patterns that scale automatically, and even asymmetrically,

without any change to the program. For example when the GPL program for an inverter is executed on a long sheet, it results in a chain of inverters of the same size. In OSL a longer sheet simply stretches the inverter (figure 1.6). The two languages encode very different properties, that can be derived directly from the choice of high-level language, and result in different local strategies. Insights from these languages can be used to design new languages with similar properties.

So far the work in amorphous computing has focused on languages for pattern and shape formation. However, the desire to achieve global-to-local programming is not unique to amorphous computing. The emerging field of pervasive computing poses the same challenge - vast numbers of computing devices embedded in our everyday environments, need to be programmed so that specific global services result from their coordinated activities. In the rest of this chapter, we discuss how this programming methodology can impact pervasive computing.

5. Pervasive Computing

Consider a scenario a few years hence in which a large city like Boston might have several wireless base stations in every building - a number of nodes in the order of 10^7. If most of the electrical devices in the buildings and those carried on by people are wirelessly networked too, then the total number of nodes could be as high as 10^{10}. If these nodes communicate peer-to-peer with nearby devices, then one could envision the entire city as connected into a mobile ad-hoc network approximately 10^3 hops in diameter. It is clear that this *pervasive computing* scenario strongly resembles the amorphous computing model, and poses similar challenges, with the significant addition of mobility.

For example, consider the problem of programming a group of agents to coordinate their movements through an environment. Such agents may represent humans carrying wireless PDAs, navigator-augmented cars, or autonomous robots. As in the amorphous computing scenario, instead of programming the individual agents behavior directly, we would like to express desired motion patterns in a global way. For example, in a traffic management application, the goal may be to engineer collective behaviors to reduce the traffic. We would like to be able to translate the desired global behavior into the agents' local interactions (e.g. who gives the precedence to whom) [17]. Because of these analogies, we started to look at ideas similar to the ones exploited in amorphous computing to organize the behavior of pervasive computing devices.

Coordinated Movement in Mobile Agents. In our research project at University of Modena and Reggio Emilia, we have used an

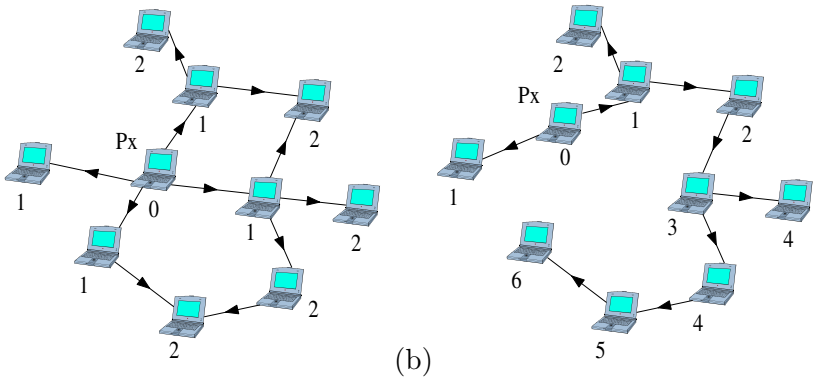


Figure 1.7. TOTA keeps a distributed fields' structure coherent despite dynamic network reconfigurations: (a) A peer Px propagates a field that increases its value by one at every hop. (b) When the field source Px moves, all fields are updated to take into account the new topology.

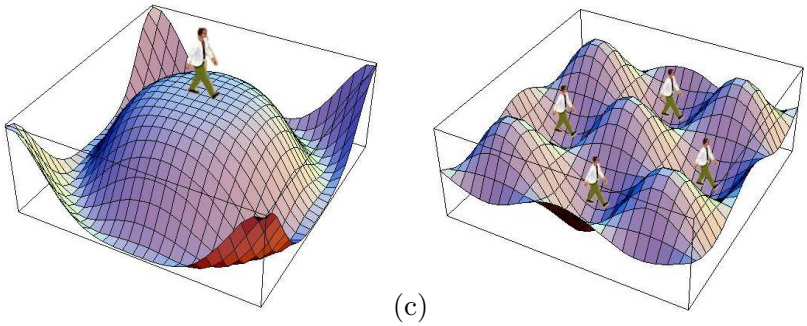
idea similar to that of morphogen gradients, which we call fields, to drive agent motion patterns. Agents are wirelessly connected in a mobile ad-hoc network (e.g. they are humans carrying on Wi-Fi PDAs) and fields have been modeled by means of distributed data structures, created by an agent, and propagated to its neighbors hop-by-hop. Specifically, we have developed the TOTA [16] middleware that provides agents with a high-level interface to define and spread these distributed data structures. Each field is defined by means of a content C and a propagation rule P identifying how the field should distribute in the network and how its value should change during the distribution.

Moreover, to take into account the dynamism of mobile networks the spatial structures resulting from field propagation are kept coherent by the TOTA middleware despite network dynamism (see figure 1.7). From the agents's viewpoint, executing and interacting are basically reduced to injecting fields, perceiving local fields, and acting according to some application-specific policy.

Let us focus on an example, imagine security guards in a museum who move and monitor the museum in a coordinated way; they have to preserve a specified distance, say d, from each other. The security guards can be provided with wireless enabled palm computers, connected through an ad-hoc network and running the TOTA middleware. Each guard's palm (agent) can generate the field in figure 1.8-a that propagates in the surroundings and reaches a minimum value at distance d from the agent. The final shape of this field approaches the distribution shown in figure 1.8-b. Each agent can then sense its immediate neigh-

```
C = (peerName,val)
P = ("val" is initialized at d, propagates to all peers,
     decreasing by 1 in the first d hops,
     then increasing "val" by 1 for further hops)
```

(a)



(b) (c)

Figure 1.8. (a) Field propagation rule; (b) Distribution of a single field; (c) Regular formation of peers

borhood, looking for the fields generated by all the other guards. It can combine the perceived fields, by computing the minimum value at each point. The result can be considered a field in itself, having minimum points at distance d from other agents. At this point, each agent can just follow downhill this computed virtual field. The field is automatically updated as the agent moves. The result is a globally coordinated movement in which agents reach and maintain an almost regular grid formation (see figure 1.8-c). Following similar strategies, it is possible to realize a vast number of coordinated motion activities (e.g. have a group of agents to meet somewhere, let them move avoiding the emergence of crowds or queues, let them cooperatively surround a prey, etc. [17])

It is worth noting that the TOTA approach is adaptive, in that the fields associated to a given motion coordination policy automatically adapt to the environment in which they are propagated. For example, if the application is executed in a building, as long as the rough mobile network topology resembles environmental physical constraints (i.e. no network links through walls), the fields' shape and thus the coordinated motion patterns adapt to the building topology, without any change in the application code. This property resembles the scale-independence property of the OSL language described in section 4.

Moreover, in TOTA - as in GPL and OSL - agents are not designed in isolation. In the case study, for example, agents achieve the goal of maintaining the formation not because of their own internal algorithm

— actually, *they do not even know about any kind of formation* — they just follow downhill the gradient of the fields being propagated, but this allows the system,*as a whole*, to enforce the formation.

Apart form these similarities, there is still a gap between this approach and the GPL and OSL programming languages. In TOTA, in fact, we still have not identified a general methodology to help us identify, given a specific motion pattern to be enforced, which fields have to be defined, how they should be propagated, and how they should be followed by agents. So this must be coded by hand for each specific application. On the contrary, the availability of a programming language like GPL or OSL, would enable us to specify the *global* motion pattern we would like to achieve and have a compiler to automatically derive suitable fields, propagation rules and agents' algorithm to follow fields. We are confident that such a programming language can be found in the future, and would possibly make the model applicable to a wider class of distributed coordination problems, even beyond motion coordination.

Other Examples. Distributed data structures, like morphogen gradients, driving agent activities, are emerging in many disparate scenarios. The research projects Anthill [18] and SwarmLinda [19] both use algorithms based on field-like data structures spread in the network by mobile software agents to enable file sharing in Internet-scale peer-to-peer (P2P) applications. Instead of being propagated in a breadth-first manner, like the morphogen gradients, the agents spread the data structure as they randomly move across the network. As a result paths are created between peers that share similar files, thus enabling a fast content-based navigation in the network of peers.

Similarly, in wireless ad hoc networks, morphogen gradients and fields can be used to design of routing mechanisms. Examples of this can be found in Gradient Routing [25] and in Directed Diffusion [11], where peers spread morphogen gradients across the network, so that other packets can reach their intended destination by following downhill the gradient associated with the destinations. Analogous techniques have been also used in [23] to create a coordinate system over ad-hoc networks. Here nodes evaluate their coordinates by triangulating the distances - expressed by means of morphogen gradients - from elected beacons.

In robotics, the idea of fields driving robots movement is not new. One of the most recent re-issue of this idea, the Electric Field Approach (EFA) [12], has been exploited in the control of a team of Sony Aibo legged robots in the RoboCup domain. Following this approach, each Aibo robot builds a field-based representation of the environment from the images captured by its head mounted camera, and decides its move-

ments by examining the fields' gradients of this representation. Similarly, Pheromone Robots [24] use field like distributed data structures to drive robot vehicles to achieve useful large-scale results in surveillance, reconnaissance, hazard detection, and path finding.

A modular robot is another example; it is a collection of simple autonomous actuators, with few degrees of freedom, connected with each other. A distributed control algorithm is executed by all the actuators to let the robot assume a global coherent shape or a global coherent motion pattern (i.e. gait). An interesting proposed approach adopts an idea similar morphogen gradients to control such a robot [27]. Here, morphogen gradients (called hormones) are created and propagated through the robot. Robots' modules decide how to bend their actuators depending on the locally perceived hormone pattern.

Shifting from physical to virtual movements, the popular videogame "The Sims" [28] exploits sorts of computational fields, called "happiness landscapes" and spread in the virtual city in which characters live, to drive the movements of non-player characters. For instance, if a character is hungry, it perceives and follows a happiness landscape whose peaks correspond to places where food can be found, i.e., a fridge.

The fact that similar notions, such as gradients, are found everywhere, suggests that they are fundamentally suited to these types of environments. However high-level programming languages and global-to-local compilation are rare. In the amorphous computing examples, the programming languages made it possible to easily achieve complex and robust desired behavior. We believe that in these other environments, the invention of appropriate global languages could have a similar far-reaching impact.

Acknowledgements

The Amorphous Computing project was started by Abelson, Sussman, and Knight, and the work discussed here reflects the contributions of many people in the Group. Support for Amorphous Computing research was provided in part by the Advanced Research Project Agency of the Department of Defense, contract number N00014-96-1-1228, and in part by a grant from the National Science Foundation, Division of Experimental and Integrative Activities, contract number EIA-0130391. Further support was provided by the Italian MIUR and CNR in the "Progetto Strategico IS-MANET, Infrastructures for Mobile ad-hoc Networks".

References

[1] Abelson, Allen, Coore, Hanson, Homsy, Knight, Nagpal, Rauch, Sussman, Weiss, "Amorphous Computing", Communications of the ACM, 43(5), May 2000.

[2] Abelson, Sussman, "Structure and Interpretation of Computer Programs", The MIT Press, 1996.

[3] Berlin, "Towards Intelligent Structures: Active Control of Buckling", PhD Thesis, MIT, 1994.

[4] Bonabeau, Dorigo, Theraulaz, "Swarm Intelligence", Oxford University Press, 1999.

[5] Butler, Byrnes, Rus, "Distributed Motion Planning for modular robots with unit-compressible modules", Intl Conf. on Intelligent Robots and Systems, 2001.

[6] Butler, Kotay, Rus, Tomita, "Generic Decentralized Control for a Class of Self-Reconfigurable Robots", IEEE Intl Conf. on Robotics and Automation, 2002.

[7] Cheung, Berlin, Biegelsen, Jackson, "Batch Fabrication of Pneumatic Valve Arrays by Combining MEMS with Printed Circuit Board Technology", Symposium on Micro-Mechanical Systems, ASME Intl. Mech. Engineering Congress and Exhibition, 1997.

[8] Coore, "Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer", PhD Thesis, MIT, 1999.

[9] Day, Lawrence, "Morphogens: Measuring dimensions: the regulation of size and shape", Review Article, Development 127, 2977-2987, 2000.

[10] Huzita, Scimemi, "The Algebra of Paper-folding", 1st Intl Meeting of Origami Science and Technology, 1989.

[11] Intanagonwiwat, Govindan, Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks". MobiCom, 2000.

[12] Johansson, Saffiotti, "Using the Electric Field Approach in the RoboCup Domain", RoboCup, 2001.

[13] Kahn, Katz, Pister, "Mobile Networking for Smart Dust", MobiCom 1999.

[14] Lawrence, "The Making of a Fly: the Genetics of Animal Design", Blackwell Science U.K., 1992.

[15] Lynch, "Distributed Algorithms", Morgan Kaufmann Pub., 1996.

[16] Mamei, Zambonelli, Leonardi, "Tuples On The Air: a Middleware for Context-Aware Computing in Dynamic Networks", Intl. ICDCS Workshop on Mobile Computing Middleware, 2003.

[17] Mamei, Zambonelli, Leonardi, "Distributed Motion Coordination with Co-Fields: A Case Study in Urban Traffic Management", IEEE Symp on Autonomous Decentralized Systems, 2003.

[18] Meling, Montresor, Babaoglu, "Peer-To-Peer Document Sharing Using the Ant Paradigm", Proceedings of the Norsk Informatikkonferanse (NIK), 2001.

[19] Mendez, Tolksdorf, "A New Approach to Scalable Linda-systems Based on Swarms", ACM SAC 2003.

[20] Nagpal, "Programmable Self-Assembly: Constructing Global Shape using Biologically-inspired Local Interactions and Origami Mathematics", PhD Thesis, MIT, 2001.

[21] Nagpal, "Programmable Self-Assembly Using Biologically-Inspired Multiagent Control", Autonomous Agents and Multiagent Systems Conf. (AAMAS), 2002.

[22] Nagpal, Coore, "An Algorithm for Group Formation in an Amorphous Computer", Intl Conf on Parallel and Distributed Computing and Systems (PDCS), 1998.

[23] Nagpal, Shrobe, Bachrach, "Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network", in the 2nd Intl Workshop on Information Processing in Sensor Networks (IPSN), 2003.

[24] Payton, Estkowski, Howard, "Progress in Pheromone Robotics", 7th Intl. Conference on Intelligent Autonomous Systems, 2002.

[25] Poor, "Embedded Networks: Pervasive, Low-Power, Wireless Connectivity", PhD Thesis, MIT, 2001.

[26] Priyantha, Chakraborty, Balakrishnan, "The cricket location-support system", MobiCom 2000.

[27] Shen, Salemi, Will, "Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots", IEEE Transactions on Robotics and Automation 18(5):1-12, 2002.

[28] The Sims, http://thesims.ea.com

[29] Weiss, "Cellular Computation and Communications Using Engineered Genetic Regulatory Networks". PhD Thesis, MIT, 2001.

[30] Wolpert, "Principles of Development", Oxford University Press U.K., 1998.