A Biological Programming Model for Self-Healing

Selvin George David Evans Steven Marchette

Department of Computer Science
University of Virginia
Charlottesville, VA
[selvin, evans, sam7p]@cs.virginia.edu

Abstract

Biological systems exhibit remarkable adaptation and robustness in the face of widely changing environments. By adopting properties of biological systems, we hope to design systems that operate adequately even in the presence of catastrophic failures and large scale attacks. We describe a programming paradigm based on the actions of biological cells and demonstrate the ability of systems built using our model to survive massive failures. Traditional methods of system design require explicit programming for fault tolerance, which adds substantial costs and complexity to the design, implementation and testing phases. Our approach provides implicit fault tolerance by using simple programs constructed following guiding principles derived from observing nature. We illustrate our model with experiments producing simple structures and apply it to design a distributed wireless file service for ad hoc wireless networks.

1. Introduction

Fault tolerant system design has traditionally explored explicit mechanisms for error checking, introducing redundancies to account for failure of components and designing signaling mechanisms to alert other systems or interfaces. This involves analysis and formal verification of processes in the system and explicit programming of recovery mechanisms. In this paper we consider an alternative to traditional fault-tolerant systems design based on a style of programming based on local interactions and responsiveness to surroundings in which robustness is intrinsic to the programming process.

The key contributions of this paper are the development of the cell-based programming paradigm we introduced in [George02], a description of how the paradigm can be used to construct structures that heal themselves, an analysis of the robustness properties and the capacity to withstand catastrophic failures of systems constructed using our cell-based programming paradigm, and a framework and example for applying our approach to system design.

2. Nature's Programs

Nature has evolved programs that exhibit remarkable robustness properties over billions of years through the untimely deaths of trillions of organisms. Programs that don't produce organisms reliably simply don't survive to future generations. Programs that produce organisms that cannot adapt to failures and changes in their environment are not transmitted to future generations.

The hallmark of biological development is that a single cell undergoes successive (often asymmetrical) divisions and all the newly created cells regulate the development process to form a complete organism. Biological programs are able to withstand a large number of individual cell failures (approximately 100 million of your cells died during the time you spent reading this parenthetical clause!) and adapt to many different environments; nevertheless they are remarkably expressive compared to human-engineered programs. The human genome consists of approximately three billion base pairs (which could easily be encoded on a single CD-ROM); the difference between the genome of any two human beings fits on a fraction of a floppy disk.

A cell is the basic unit of life and all living organisms are composed of one or more cells. The capacity of organisms to adapt to changing and often hostile environments, tolerate limited failures and heal damaged organs is not because of the robustness of individual cells, but because of the interactions between large numbers of cells. A cell is able to divide into two daughter cells, emit chemicals to the surrounding environment, and actively deform by applying physical forces across its walls. Different chemicals within and around the cell control these actions. A cell can sense chemicals on its walls, as well as in its environment. The nucleus of the cell contains DNA, which encodes different genes that have been retained through evolution. A gene is activated when a certain condition is true. This condition could be a critical concentration level of a chemical or a set of chemicals that the cell possesses or that the cell has sensed. When a gene is activated it may cause certain cell actions, which could result in the turning other genes on or off.

Morphogenesis is the development of form in an organism and typical multi-cellular organisms develop from a single fertilized egg cell. Morphogenesis is robust to many kinds of local failures and adapts to a wide range of environments. For example, during the development of a sea urchin even when one of two cells dies at the second stage of development, the remaining cell develops into a complete (albeit smaller) organism ([Wolpert02], original experiments by Driesch, 1892). This is in contrast to mosaic development (first proposed by Weismann in the 1880s) where cells are differentiated after the very first division. If one of the cells in a frog embryo is destroyed after the first division, the other cell will not develop into a viable frog, but rather into something resembling half an embryo.

Almost all complex organisms have some sort of mechanism for healing simple wounds. In humans, when a minor injury happens, an inflammatory response occurs and the cells below the dermis (the deepest skin layer) begin to increase collagen (connective tissue) production. Later, the epithelial tissue (the outer skin layer) is regenerated. Apart from the fact that cells around the injury are able to adapt to a different function based on the new circumstances, it is their level of awareness that these cells possess that makes such healing possible [Mazzotta94].

Many organisms can regenerate new heads, limbs, internal organs or other body parts if the originals are lost or damaged. Organisms take two approaches to replacing a lost body part. Some, such as flatworms and the polyp *Hydra*, retain populations of stem cells throughout their lives, which are mobilized when needed. These stem cells retain the ability to re-grow many of the body's tissues. Other organisms, including newts, segmented worms and zebrafish, convert differentiated adult cells that have stopped dividing and form part of the skin, muscle or another tissue back into stem cells. When a newt's leg, tail or eye is amputated or damaged, cells near the stump revert from specialized skin, muscle and nerve cells into blank progenitor cells. These progenitors multiply quickly to about 80,000 cells and then grow into specialized cells to regenerate the missing part [Pearson01].

Nature also exhibits self-organization at the level of societies. Ant colonies are examples where self-organization achieves robustness to large-scale attacks (often by competing colonies). A colony has many different types of ants – workers, warriors, drones and a queen. If all the warrior ants die, then some of the worker ants transform into warrior ants so that activities such as nest patrolling continue. Ants deposit chemicals into the atmosphere, which are then sensed by other ants and appropriately decoded. In the case, where all warriors are killed, the concentration of chemicals deposited by warriors reduces dramatically and hence this absence of chemicals induces many more worker ants to convert into warrior ants [Bonabeau99]. Thus, the ant colony organizes itself automatically to tolerate failures.

In summary, nearly all programs for multi-cellular organisms in nature exhibit several properties, which are essential to their ability to survive failures of large numbers of their cells and thrive in hostile conditions:

- **Environmental Awareness.** Cells behave in different ways depending upon properties (including chemical concentration) they sense about their surroundings. Cells communicate with nearby cells using a shared environment.
- **Localization.** Cells can communicate over short distances using chemical diffusion. For most of the development process there is no global coordination and limited synchronization: an organism needs to know how to make a central nervous system before it has one.
- **Adaptation.** Cells are capable of performing different functions depending upon changes in the environment. All cells contain the same program and can hence respond to aberrant behavior from neighbors by adapting their own behavior.
- **Redundancy.** Typical organisms have many cells devoted to the same function throughout development, so that failures of individual cells are usually inconsequential. Biological systems also exhibit redundancy of function, where several distinct mechanisms evolve for the same purpose in a single organism so that failure of one mechanism will not cause system failure.

Although it may be possible to achieve robustness without these properties, nearly all robust programs found in nature exhibit all of these properties. Hence, studying programming models designed around these properties offers a promising approach to building robust systems.

3. Cell-Based Programming Model

Cellular automata have been studied extensively since von Neumann's early work [vonNeu53] (the related work section summarizes more recent work). Our model of cell-based programming adds to traditional cellular automata the notion of cell division and a rudimentary model of the physical forces involved. In our model, cells live in an environment and can sense properties of that environment and take actions that effect that environment. This enables inter-cell communication based on chemical diffusion through a shared environment. We describe a few key aspects of our model next.

Cell Division. A cell can divide into two daughter cells that may be dissimilar in orientation and chemical composition but have the same program. A cell has an axis called the apical-basal axis. Divisions can be either perpendicular to this cell axis or along the plane containing the axis. In our model, cells can divide in any direction. Cell divisions may be asymmetrical. Differences in chemical composition as well as different chemicals on their cell walls cause the two daughter cells to behave differently. Cell division is modeled by using a transition from one state to two states as shown in Figure 1. Parameters to the division control the locations and orientations of the daughter cells.

Gene Actions. Genes can activate or deactivate depending on the presence or absence of a particular protein or a certain degree of chemical concentration. Activation or deactivation of a gene results in cell actions like production of chemicals. The combination of active genes defines the state of the cell.

Cell Actions. Cells produce different proteins depending on what genes are active. Chemicals produced this way diffuse

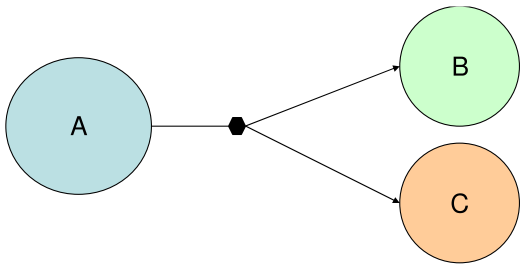

Figure 1. Cell division. The initial cell in state *A* divides into two daughter cells in states *B* and *C*.

into the environment. They may affect nearby cells that can sense the concentration of particular chemicals.

We abstract the complexities of communication in natural systems with two types of communication between cells: *diffusion* and *emission.* Cells can communicate with neighboring cells by diffusing chemicals over a limited range as illustrated in Figure 2. Diffusions are omnidirectional: they spread in all directions, and cells cannot determine from which direction they came. The characteristics of the diffusion process are related to the properties of the environment such as viscosity of the medium and gravity. Emissions model communication directly through cell walls. They are directional: cells can emit chemicals in a particular direction, and sense from which direction emissions came.

Cells can induce nearby cells into performing specific actions by using chemical emission or diffusion. Similarly the death of a cell causes cessation of chemical diffusion and induces nearby cells into actions such are regenerating the dead cell. This awareness is essential for self-healing mechanisms.

Simulating Cell Programs

We built a simulator for cell programs to study their properties and conduct experiments involving simulated random and catastrophic failures. Our simulator is freely available from http://swarm.cs.virginia.edu/cellsim.

A cell program begins with cells in an initial configuration. The cells change state or divide based on sensed chemicals that can be either caused by the environment or by nearby cells. The simulator computes the state of each cell and simulates cell division and cell death to determine the new set of cells for the next step. It also simulates chemical diffusion that results when a cell diffuses chemicals so that neighboring cells can sense them. State transition and cell division by a cell occur in response to the sensed chemicals.

Our simulator supports several different models of cells and environments. Environments control physical constraints on cell placement and how chemicals spread. Environments allow us to experiment with different parameters that control the amount of random variance in the location of a new daughter cell, what happens when a cell divides into space that is already occupied by another cell, and how chemicals diffuse and evaporate, and how modeled forces in the environment affect the rate of diffusion in different directions. For this paper, we use the simplest cell and environment model: cells have a single state and live in a discrete space; diffusions are linear, travel the same distance in all directions, and evaporate completely every time step. In addition, a simple cell model is used: cells divide only in orthogonal directions, and only have orthogonal neighbors. A forthcoming paper will consider how different cell and environment models affect the behavior and robustness of programs.

4. Fault Tolerance and Healing

We illustrate the fault-tolerance and healing properties of structures built using our programming model. The physical environment for the system is based on coordinate geometry. Cells can divide along the three axes (*x*, *y* and *z*) in either direction (+ or −). We use a simple diffusion model in which the chemical concentration decreases linearly with increasing distance (a more physically realistic exponential decay model is not necessary for these experiments). Next, we demonstrate three examples of structures produced using our programming model.

Sphere

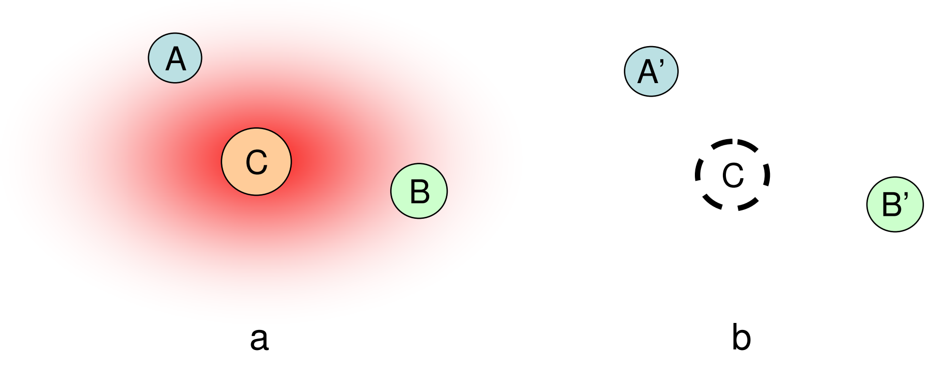

Figure 2. Diffusion. (a) Cell C diffuses chemical; nearby cells sense varying concentrations of the chemical based on their proximity and may change state in response. (b) Cell C dies and ceases diffusing the chemical; nearby cells sense the absence of the chemical and change state.

```
state center { emits (alive, 1)
  diffuses (radius, 10)
  transitions (alive from dir < 1) -> (center, body) in dir; }

state body { emits (alive, 1)
  transitions (alive from dir < 1) & (radius > 1) -> (body, body) in dir; }
```

Figure 3. Sphere program.

The cell program shown in Figure 3 generates a sphere. The radius of the sphere is determined by the amount of chemical *radius* that the center cell produces. The initial configuration is a single cell in the *center* state. That cell will emit one unit of the *alive* chemical in all directions. In this program (and many others), we use the *alive* chemical as an indicator of the presence of a cell. The *center* state also diffuses the *radius* chemical. The diffusions amount indicates the maximum distance from this cell where the diffused chemical will reach in measurable concentrations. Here, the diffusion amount controls the radius of the sphere.

On each simulation step, all cells will evaluate their transition rule conditions in order and select the first rule for which the condition is true. For state *center*, the first transition rule condition is:

```
alive from dir < 1
```

This will be true if there is any direction from which the concentration of *alive* is less than 1. The direction variable *dir* will be bound to a direction which makes the condition true. If more than one direction satisfies the condition, *dir* will be bound to one of the satisfying directions selected at random.

The action part of the transition rule,

```
(center, body) in dir
```

indicates what action to take when the condition is satisfied. Listing two states in an action indicates a division, so the effect of the transition is to divide into two daughter cells one in state *center* and the other in state *body*, in the direction *dir* bound as necessary to satisfy the condition. Following this transition rule, the *center* cell will keep producing *body* cells as long as there is some direction from which alive is not sensed. If *alive* is sense in all directions, no transition condition is satisfied and the cell will remain in its current state. Cells in the *body* state will keep dividing as long as there is some direction from which they do not sense *alive* and they sense at least one unit of the radius chemical. Hence, the body cells will fill in the sphere.

When an initial configuration of one cell in state *center* is given as shown in Figure 4, the program develops a sphere of radius 10 units. This program has simple fault tolerance capabilities. The sphere will regenerate no matter how many of the body cells are killed as long as the center cell continues to function. Notice that this program does not explicitly mention the actions to be taken when a failure occurs. Instead, recovery is intrinsic in the way the program is written in terms of local interactions through sensed chemicals.

The performance of the development and regeneration of the sphere program is shown in Figure 5. In this graph, the

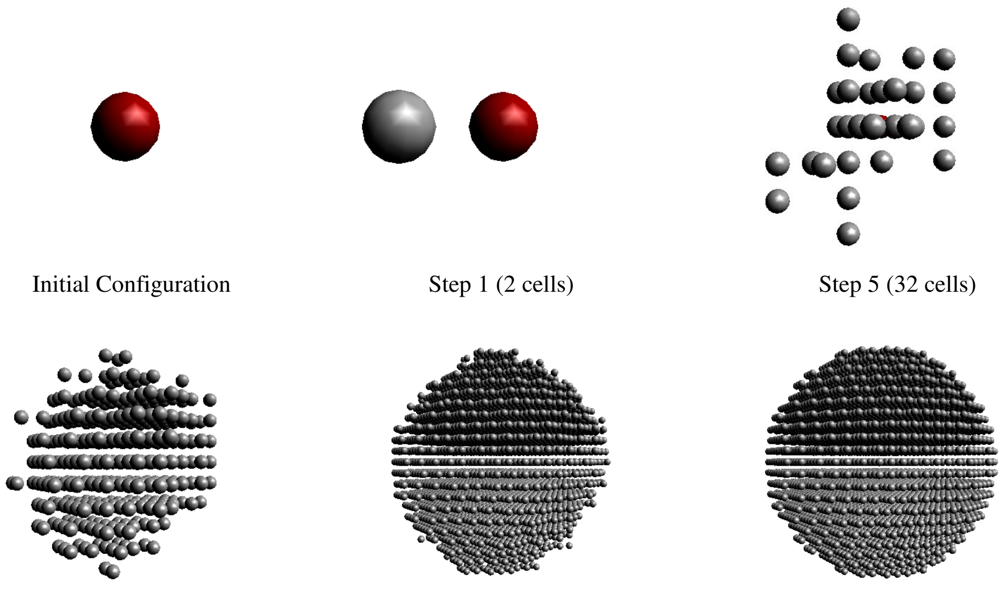

Figure 4. Sphere Growth.

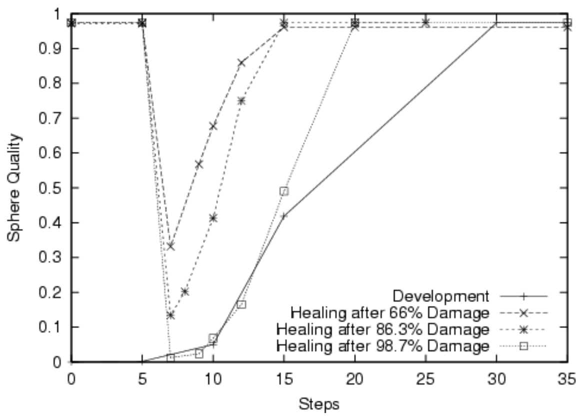

Figure 5. Sphere recovery.

sphere's ability to withstand any degree of damage to its body and recover quite quickly from it is shown by plotting sphere quality against number of steps for different degrees of damage. Sphere quality is measured as $(n_1 - 2n_2)/N$ where

n_1 = # of cells in formed sphere lying within ideal sphere
n_2 = # of cells in formed sphere lying outside ideal sphere
N = # of cells in ideal sphere which is $(4/3)\pi r^3$

A sphere of radius 10 grows within 30 steps. Regeneration of sphere after a damage of 66%-99% of the body cells takes between 9 and 15 steps.

To produce a more robust sphere, we need a mechanism for regenerating the sphere even when the center cell is killed. One approach is to create a core of cells around the center which will regenerate a *center* cell if the *radius* chemical is not sensed. The robust sphere program requires one extra state and three addition transition rules. It will regenerate the sphere as long as the center cell and all core cells are not killed simultaneously.

Cube

Cubes do not appear often in nature, but can be constructed using our model. It is not surprising, however, that the program to generate a robust cube is more complex than that required to produce a sphere.

The program shown in Figure 6 uses four states and nine transition rules to generate a cube from the initial condition of one cell of state *A* located where the front bottom left corner of the cube will be. The initial cell differentiates into *B*, *C*, and *D* cells which divide in the *x*, *y*, and *z* directions respectively. The *B* cells traverse horizontally in a line, dividing into *C* and *D* cells which subsequently divide to fill the vertical and depth directions. As in the sphere programs, the *alive* chemical is emitted so cells can sense the presence or absence of neighbors in each direction. Chemical diffusion is used by the cube cell program as well to restrict the dimensions of the cube: chemicals *X*, *Y*, and *Z* for the respective directions. The diffused chemicals also implicitly locate damaged areas of the cube and instruct the *B*, *C*, or *D* cells to grow in the appropriate direction that will heal it. With our current simulator, cells are oriented in absolute space; taking advantage of relative cellular orientations could reduce the number of states and transitions needed.

The simple cube program is robust to failures of all cells except the initial *A* cell. If that cell stops diffusing the *X* chemical, the *B* horizontal cells will fail to regrow. To improve robustness, we introduce specialized self-healing cells to recover critical cells. The robust cube program generates a cube in the same manner as above, but includes

```
state A { emits (alive, 1)
  diffuses (X, 8), (Y, 8), (Z, 8)
  transitions
    (alive from X+ < 1) -> (A, B) in X+;
    (alive from Y+ < 1) -> (A, C) in Y+;
    (alive from Z+ < 1) -> (A, D) in Z+; }

state B { emits (alive, 1)
  diffuses (Y, 8), (Z, 8)
  transitions
    (alive from X+ < 1) & (X > 0) -> (B, B) in X+;
    (alive from Y+ < 1) -> (B, C) in Y+;
    (alive from Z+ < 1) -> (B, D) in Z+; }

state C { emits (alive, 1)
  diffuses (Z, 8)
  transitions
    (alive from Y+ < 1) & (Y > 0) -> (C, C) in Y+;
    (alive from Z+ < 1) -> (C, D) in Z+; }

state D { emits (alive, 1)
  transitions
    (alive from Z+ < 1) & (Z > 0) -> (D, D) in Z+; }
```

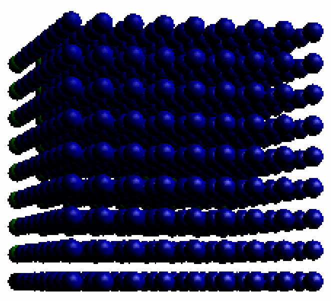

Figure 6. Simple cube program.

```
state corner {
  emits (A, 6), (alive, 1)
  transitions
    (alive from dir < 1) -> (corner, segment) in dir; }

state segment {
  emits (alive, 1)
  forwards (A, A - 1);
  transitions
    (A from dir > 1.5) & (alive from opposite(dir)) > 0
       -> (segment);
    (A from dir > 1.5)
       -> (segment, segment) in opposite(dir);
    (A > 0.1) -> (corner);
    (A < 0.1) -> die; }
```

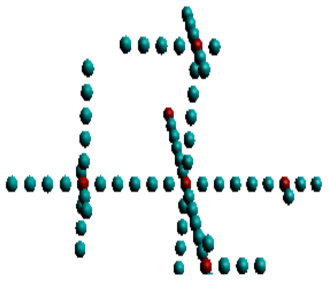

Figure 7. Mesh program.

additional self-healing mechanisms to recover from failures of key cells. Instead of relying on a single corner cell, each of the corners of the cube is differentiated into a set of similar states that diffuse regulator chemicals. Regulation chemicals control division and specialization into the eight corner states as the cube develops. They also mutually inhibit, such that one in the presence of any of the neighboring corner's states has no effect. Should any corner cell fail, its absence will be detected and the nearby cells will recreate the corner. The regulation chemical secreted by adjacent corners controls the regrowth of corner cells. Hence, the cube will recover from any failure that does not kill at least four corners simultaneously.

Mesh
Figure 7 shows a program for producing a three-dimensional regular mesh structure. As long as a corner survives, the mesh will continue to heal and grow. Because of the final segment transition that results in a *die* command, segment cells that are not connected to a corner will commit suicide. The mesh program could be used to construct an overlay network. Hence, killing cells that are not able to communicate with a corner cell conserves resources.

Summary
These experiments show that many unreliable cells running a single program and having no global knowledge of position or identification and without using global communication can organize themselves and create desired structures using chemical diffusion and cell induction. Although the structures we have built may not be intrinsically useful, building simple shapes provides a good experimental platform for exploring the expressiveness, scalability and robustness of our programming model.

5. A Framework for Building Systems

While it is interesting to observe properties of cell programs applied to structure growing tasks, our real interest is in building systems that perform useful functions in a robust and scalable way. By applying this model to building systems, we endeavor to create distributed systems that achieve many of the desirable properties commonly found in biological systems such as reliability, scalability, self-healing and survivability. Our programming paradigm requires programmers to design systems by engineering the local interactions that produce the desired global functionality.

To implement real systems with current technology, we need ways of emulating cell actions (including division) and communication. It is hard to produce physical divisions, but easy to create new processes. We can model division by finding a suitable host and starting a new process on that host. Communication by diffusion and emission corresponds well to networking. In a wireless network, a single transmission diffuses over an area; to achieve longer diffusions, a transmission may be repeated over multiple hops.

To create an application, a programmer must describe the desired behavior in terms of actions in response to current state and received messages. If the programmer can provide a self-awareness and activation mechanism (in the form of status and control variables), define the local neighborhood within which each cell communicates and provide a mechanism for diffusing chemicals (i.e., delivering messages) to cells within that local neighborhood, then applying the cell based programming approach reduces to plugging a piece of software that follows the transitions of a cellular automaton and acts accordingly.

5.1 Distributed Wireless File Service

We illustrate our programming paradigm with DWFS, an application layer peer-to-peer file sharing service. It is designed to run on wireless nodes that may be mobile. All nodes are identical and have low power and low network transmission and reception capacity. The radio transmission and reception associated with the wireless communication is expensive with respect to energy. As with most applications, the DWFS should balance the conflicting goals of reliable functionality and network longevity. New nodes can be added and nodes can enter and leave the network at any time. The design should scale to very large networks.

Our implementation is intended as a proof of concept to show that applications built using our programming model have useful robustness properties, rather than as a fully functional and useful file sharing service. Hence, we limit our implementation to immutable files where once a file is published it cannot be modified. The only two operations are *read* (*file*) and *store* (*file*). Files are identified using a number (which could correspond to a URL). A realistic file sharing service would need to also support other operations such as deleting and updating a file.

5.2 Protocol

The DWFS protocol provides mechanisms for requesting and publishing a file.

File Request. The file request protocol is shown in Figure 8(a). A client sends a file request in the form of a broadcast with the file identifier. This is similar to the cell-programming notion of diffusion. All the servers within the one-hope transmission range receive this message. The server on a node that contains the file responds to this request by transmitting the requested file. Note that requests are not forwarded: a server node that receives a request for a file it does not have simply ignores the request. Our publication protocol is designed to ensure that there is a high probability that at least one node within the transmission distance will be able to respond to a request for a published file.

File Publication. The file publication protocol is shown in Figure 8(b). When a new file is published, the publishing server diffuses two chemicals: one represents the inhibit message and the other represents the replicate message. The inhibit chemical is diffused over a smaller range than the replicate chemical. Nodes that receive the replicate chemical but not the inhibit chemical will replicate the file. Each node that replicates the file in turn diffuses both the chemicals as the original publisher did. This distributes the file throughout the network and builds up the required redundancy necessary for fault-tolerance. Given sufficient density of nodes, we have high confidence that there will be at least one copy of the file within the broadcast range of the inhibit message of any node. File propagation occurs when nodes replicating the file in turn publish it to their neighbors.

5.3 Handling Failures

Failures include movement or death of a node, breaking of a network path (even when a file operation is in progress), failure of a significant percentage of the nodes, non-simultaneous failure of all nodes caching a particular file. Failure handling in DWFS is inherent in the way the system is programmed. Failures are sensed by the absence of chemicals and servers react accordingly. Nodes periodically emit the *inhibit* and *replicate* chemicals for each file they store. All nodes that receive a particular publication's *replicate* chemical but do not receive the corresponding *inhibit* chemical will replicate the files that node holds. If that node fails, it will stop transmitting both the *inhibit* and *replicate* chemicals. The replicating nodes will continue to transmit those chemicals, and now a new node in the region previously covered by the failed nodes inhibit signal is likely to start replicating the file.

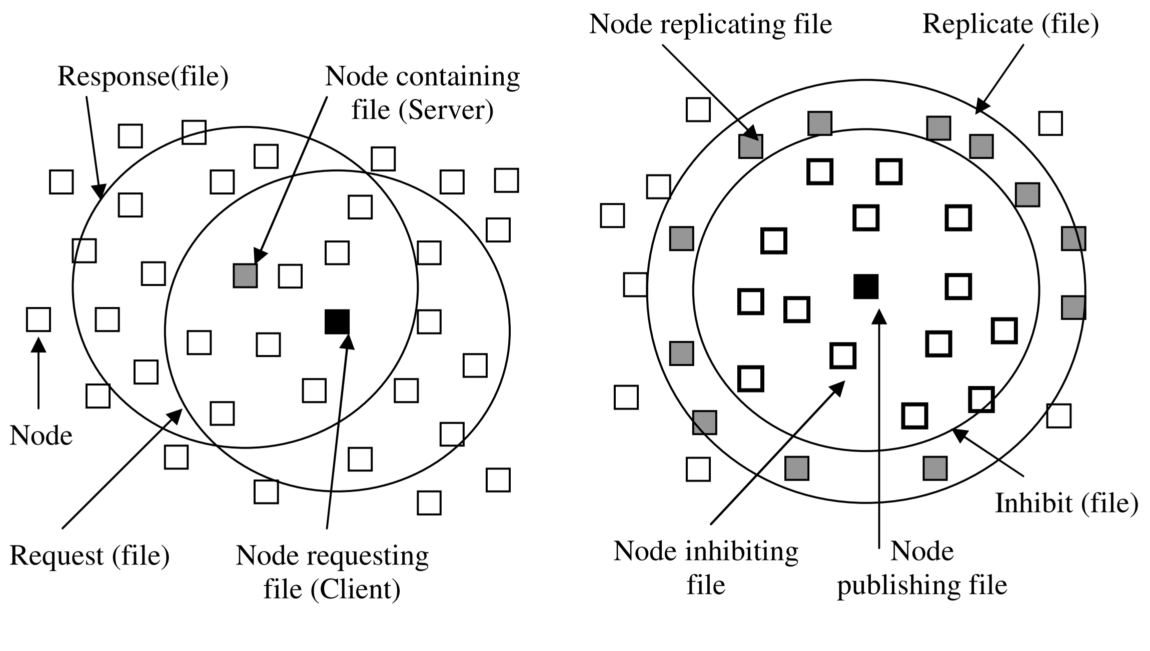

Figure 8. DWFS Protocols. (a) File request and response. (b) File publication.

The other type of failure to consider is an overload. When a node receives more requests to store files and it is running out of storage space, it initiates an asymmetric cell division. The subsequent publication requests now cause transfer of the file onto the new node. A node may periodically rate files based on access frequency and transfer half of the frequently used files onto a different close-by server in order to reduce traffic on itself.

5.4 Evaluation

Our evaluation focuses the robustness of file availability provided by DWFS in the face of individual node failures. File availability is the ratio of the number of successful file requests to the total number of file requests. In our experiments, every request is for a published file, so an ideal system would achieve 100% file availability.

Figure 9 shows the degradation of file availability with increasing node speed with constant initial node energy of 100. If a node does not receive a file after a timeout, it retransmits the request. In the simulation we varied both the timeout interval from 0.5s to 2.5s and the number of retransmissions from 0 to 3. These graphs indicate that with a few retransmissions, we can achieve high file availability even for nodes that move rapidly relative to their broadcast range.

Figure 10 shows the variation of file availability over time for different values of initial energy. File availability degrades only gradually provided the nodes have sufficient energy. For example, with initial energy = 50, even after 75% of nodes have expired the file availability exceeds 90%.

Our results show that by appropriate choice of timeouts and retransmission attempts, we can achieve a high degree of file availability even with nodes that move at high speeds and fail due to loss of battery power. As larger numbers of nodes fail, file availability degrades gradually. Our design incorporates the biological metaphor of chemical diffusion and sensing chemicals (inhibit and replicate) to achieve a robust design with a simple and flexible program.

For our experiments, sending or receiving a request or response uses 5 energy units, so with initial energy = 100, a node will expire after 10 messages. At each time step, each node has a 1% probability of requesting a file.

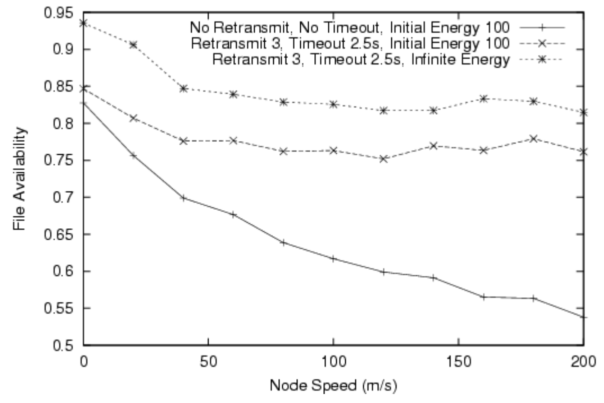

Figure 9. File availability with mobile nodes.

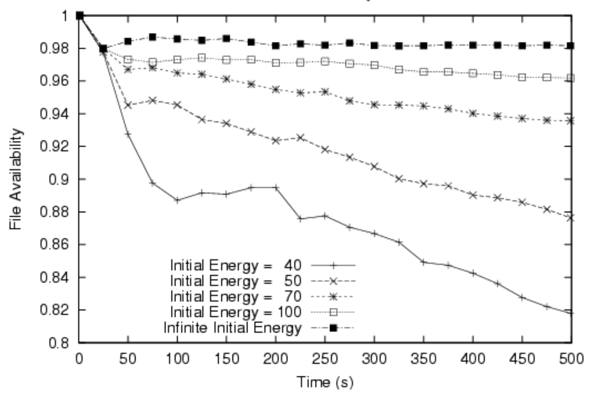

Figure 10. File availability over time.

Experiments were also conducted to study the behavior of DWFS using networks of different densities. Nodes are randomly distributed within an area of size 2km by 2km and the node density was varied from 1 to 20 within each broadcast area. Figure 11 shows the variation of file availability over different values of network density. It was observed that after reaching a critical network density of about 6 nodes per broadcast area, the file availability improves sharply and becomes very close to the maximum at a density of 8. Shivendra et al. showed experimentally that in packet radio networks, the local neighborhood should at least be 5 to ensure connectedness [Philips98]. In this experiment DWFS is able to maintain high file availability at network densities close to this critical value.

6. Related Work

The very first engineers were inspired by biology, as have been thousands of mythical (e.g., Daedalus) and real (e.g., Leonardo DaVinci) engineers since. John von Neumann studied cellular automaton [vonNeu53] and Alan Turing studied morphogenesis [Turing52], although he did not draw analogies between biological programs in genes and computer programs [Saunders93]. More recently, several active research areas in computer science have emerged inspired by biology including evolutionary computation and swarm intelligence.

Evolutionary computation (also known in various forms as

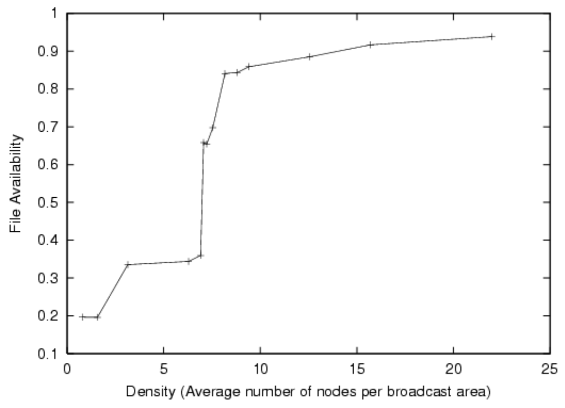

Figure 11. File availability v/s Network density.

genetic algorithms and genetic programming) attempts to apply principles of biological evolution to produce computer programs by devising a fitness function that evaluates how well a program satisfies a goal, breeding successive generations of programs using various combination and mutation strategies, and selecting survivors based on how well programs satisfy the fitness function. Evolutionary computation has been demonstrated to produce solutions to complex problems that improve on the best human developed solutions [Koza99]. Our work differs from this in that we are not using evolutionary approaches to develop solutions, but rather developing solutions by observing the solutions nature has evolved. The key advantage is that since the solutions we produce are designed by humans, we are likely to have designs that are easier to understand, modify and reason about.

Embryonics [Mange96, Mange98] is an architecture for hardware inspired by biological development. A system is implemented as a grid of artificial cells implemented by electronics. Cells compute their location within two-dimensional space and differentiate their behavior based on that position. Embryonics assumes cells continuously conduct a self-checking test and issue a failure signal when the test fails. Systems built using the embryonic architecture exhibit self-healing based on coordinates being automatically reassigned when faults are detected based on inter-cell communications [Stauffer01]. Ortega and Tyrrell analyzed the reliability of embryonic systems [Ortega99] and concluded that simple design combined with automatic reconfiguration provided advantages in enabling a high probability of reliable operation in the presence of failures.

Swarm intelligence looks primarily to social insect behaviors for inspiration [Bonabeau99]. Biologically inspired algorithms have been developed for many problems including network routing [DiCaro98a, DiCaro98b, Scho96, White97], distributed intrusion detection and response [Fenet01], graph exploration [Yano01], terrain coverage [Koenig01] and peer-to-peer applications [Mont01]. Fisher and Lipson proposed using techniques based on social insects to design survivable systems [Fisher98]. As with our work, all of these use independent agents interacting in a common environment to achieve global properties. Our work differs in that by using cells as the inspiration for our computing units instead of complex organisms like insects, our designs assume more limited devices and place more emphasis on the local interactions instead of long distance and time interactions through an environment. Second, instead of focusing on optimization problems, we are interested in controlling and reasoning about behavior produced through local interactions.

A related approach is amorphous computing [Abel00], which considers approaches for programming a medium of randomly distributed computing particles. The Growing Point Language (GPL) [Coore98], Origami Shape Language (OSL) [Nagpal01], and Paintable Programming [Butera02] are examples of programming mechanisms for producing global self-organization using simple local coordination. As with our work, the challenge is to produce programs that generate predictable behavior with a locally unpredictable and non-traditional programming model. Because the underlying execution environment is inherently redundant and decentralized, robustness is practically inevitable if programs are constructed in the right way.

Researchers have also studied more formal models of computation based on nature. The chemical abstract machine [Berry90] is an abstract machine based on the chemical metaphor. States of a machine are chemical solutions where floating molecules can interact according to reaction rules derived from lambda calculus. Solutions can be stratified by encapsulating sub solutions within membranes that force reactions to occur locally. We have not yet developed a formal semantics for our programming model.

7. Conclusions

Nature has evolved clever and robust solutions to challenging engineering problems. By learning from those solutions, we can design distributed systems that have similar robustness properties. In the future, systems with very large number of often-unreliable agents will be deployed. These systems will need to operate robustly without manual intervention and be able to adapt to a wide variety of failures. A biological approach to programming these applications offers the promise of achieving robustness, scalability and survivability from small and simple programs.

Acknowledgements

This work has been funded in part by the National Science Foundation through NSF CAREER (CCR-0092945) and NSF ITR (EIA-0205327) grants. The authors thank Lance Davidson, Salvatore Guarnieri, Qi Wang, and Brad Zhang for their contributions to this work.

References

[Abel00] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman and R. Weiss, *Amorphous Computing*, Communications of the ACM, Volume 43, Number 5, p. 74-83. May 2000.

[Berry90] G. Berry and G. Boudol, *The Chemical Abstract Machine*. ACM Symposium on Principles of Programming Languages, January 1990.

[Bonabeau99] Eric Bonabeau, Marco Dorigo, Guy Theraulaz. *Swarm Intelligence: from Natural to Artificial Systems*, Santa Fe Institute, Oxford University Press, 1999.

[Butera02] William Butera. *Programming a Paintable Computer*. MIT Media Lab, PhD Thesis 2002.

[Coore98] Daniel Coore. *Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer.* MIT PhD Thesis. December 1998.

[DiCaro98a] Gianni Di Caro, *AntNet: Distributed Stigmergic Control for Communications Networks*, Journal of Artificial Intelligence Research 9 (1998): 317-365.

[DiCaro98b] Gianni Di Caro. *Two Ant Colony Algorithms for Best-effort Routing in Datagram Networks.* 10th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'98). IASTED/ACTA Press, 1998.

[Fenet01] Serge Fenet, Salima Hassas. *A distributed Intrusion Detection and Response System Based on mobile autonomous agents using social insects communication paradigm*. First International Workshop on Security of Mobile Multiagent Systems, 2001.

[Fisher98] David A. Fisher and Howard F. Lipson. *Emergent Algorithms - A New Method for Enhancing Survivability in Unbounded Systems*. Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences. 1998.

[George02] Selvin George, David Evans and Lance Davidson. *A Biologically Inspired programming model for self healing systems.* Workshop on Self-Healing Systems November, 2002.

[Koenig01] Sven Koenig, B. Szymanski and Y. Liu. *Efficient and Inefficient Ant Coverage Methods*. Annals of Mathematics and Artificial Intelligence. Vol 31, Issue 1/4. 2001.

[Koza99] Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1999. *Genetic Programming III: Darwinian Invention and Problem Solving*. San Francisco, CA: Morgan Kaufmann Publishers.

[Mange96] D. Mange, M. Goeke, D. Madon, A. Stauffer, G. Tempesti and S. Durand. *Embryonics: A New Family of Coarse-Grained Field-Programmable Gate Arrays with Self-Repair and Self-Reproducing Properties*. In *Toward Evolvable Hardware,* Springer Lecture Notes in Computer Science, Volume 1062, 1996.

[Mange98] D. Mange, A. Stauffer, G. Tempesti. *Embryonics: A Macroscopic View of the Cellular Architecture*. In *Evolvable Systems: From Biology to Hardware*, M. Sipper, D. Mange, A. Pérez-Uribe, editors, Springer Lecture Notes in Computer Science Volume 1478, 1998.

[Mazzotta94] Mary Y. Mazzotta. *Nutrition and wound healing*. Journal of the American PodiatricMedical Association. Volume 84, Number 9, p. 456–62. September 1994.

[Mont01] Alberto Montresor. *Anthill: a Framework for the Design and Analysis of Peer-to-Peer Systems.* 4th European Research Seminar on Advances in Distributed Systems. May 2001.

[Nagpal01] Radhika Nagpal, *Programmable Self-Assembly: Constructing Global Shape using Biologically-inspired Local Interactions and Origami Mathematics*, PhD Thesis, MIT Department of Electrical Engineering and Computer Science, June 2001.

[Ortega99] C. Ortega and A. Tyrrell. *Self-Repairing Multicellular Hardware: A Reliability Analysis.* Proceedings of the 5th European Conference on Artificial Life. September 1999.

[Pearson01] Helen Pearson, *The regeneration gap*, Nature Science Update. 22 November 2001.

[Philips98] Philips, Shivendra, Panwar and Tatami, *Connectivity properties of a packet radio network model*, IEEE Transactions on Information Theory, 35(5), Sept 1998

[Saunders93] P. T. Saunders. *Alan Turing and Biology.* IEEE Annals of the History of Computing, 15(3):33-36, 1993.

[Scho96] R.Schoonderwoerd, O.Holland, J.Bruten and L.Rothkrantz. *Ant-based load balancing in telecommunications networks.* Adapt. Behav. 5 (1996): 169-207.

[Sipper97] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Pérez-Uribe, A. Stauffer. *A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems*. IEEE Transactions on Evolutionary Computation, Vol. 1, No 1, April 1997.

[Stauffer01] A. Stauffer, D. Mange, G. Tempesti and C. Teuscher. *A Self-Repairing and Self-Healing Electronic Watch: The BioWatch.* Springer Lecture Notes in Computer Science Volume 2210, 2001.

[Turing52] Alan Turing. *The Chemical Basis of Morphogenesis*, Philosophical Transactions of the Royal Society B (London). 1952.

[vonNeu53] John von Neumann, *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966 (Originally published in 1953).

[White97] T. White. *Routing with swarm intelligence*. Technical Report SCE97-15, Systems and Computer Engineering Department, Carleton University, September, 1997.

[Wolpert02] Lewis Wolpert, Rosa Beddington, Peter Lawrence, Thomas M. Jessell, *Principles of Development*, Oxford University Press. 2002.

[Yano01] Vladimir Yanovski, Israel A. Wagner, Alfred M. Bruckstein. *Computer Vertex-Ant-Walk – A robust method for efficient exploration of faulty graph.* Annals of Mathematics and Artificial Intelligence. Volume 31, Issue 1/4. 2001.