Biology-inspired self-healing system design

Teemu Kemppainen
University of Helsinki
Department of Computer Science
Seminar on self-healing information systems, spring 2007
Email: teemu.kemppainen@helsinki.fi

***Abstract*—When engineering self-healing systems, inspiration can be sought from nature. Biological organisms present proven self-healing models that have been developed during millions of years of evolution. In this paper common self-healing properties found in nature are described. A programming paradigm inspired by properties of the biological cell is discussed, and a software architecture for distributed systems based on the model of a biological ecosystem is presented.**

I. Introduction

Consider a small scratch or a wound on your fingertip: without any conscious activity from your part, the cells of your body are able to heal in a very efficient and clever way. In response to the wound the deeper skin layer will produce more connectivity tissue, and after a few days or a week, the outer cell layer will be healed [1]. The healing process is fully automatic.

Nature also provides much more radical self-healing patterns. For instance a newt is able to regenerate an entire tail or leg by converting redundant cells into stem cells that replace the damaged body part [2].

The development of multi-cellular organisms from one single egg cell is called morphogenesis. Here, for instance sea urchins show remarkable self-healing behaviour. The sea urchin will survive and develop through morphogenesis even if the other cell dies after the initial division of the egg cell [1].

Self-healing behaviour can also be found on the level of biological societies. There are workers, warriors, drones and one queen in an ant colony. If the colony is attacked, and most of the warrior ants are killed, some workers compensate for the loss by taking over the role of a warrior [1].

Clearly, nature has developed during millions of years of evolution extremely efficient and proven methods for self-healing. When designing self-healing information systems, inspiration and design guidelines can be sought from these natural phenomena that even our own everyday existence depend on.

In this paper we will first present in section II a generalization of some self-healing properties found by observing nature. One recent programming paradigm called the cell based programming paradigm is presented in section III with a sample algorithm of a self-healing sphere programmed using the paradigm in subsection III-B. A programming platform for distributed services with a biological ecosystem as a model, is discussed in section IV. The paper is summarized in section V.

II. Self-healing properties in nature

In biological organisms many properties that contribute to self-healing can be found. By observing how nature achieves this desired self-healing behaviour and extracting general design patterns from these observations, and then implementing these patterns in computer systems, we will achieve self-healing behaviour according to Fig. 1.

Many observations and generalizations of self-healing properties found in nature and implemented into self-healing systems have been described in recent research papers ([1]–[5]). The self-healing properties that the cell based programming model, described in the next chapter, uses, can be summarized as [1], [2]:

- Environmental awareness: Cells in an organism are environmentally aware. The actions of the cells vary depending on their environmental surroundings. If the chemical concentration in the nearby environment changes, the cell may start behaving differently in reaction to the change. This property also enables inter-cell communication, as cells can affect the chemical concentration in their shared environment.
- Adaptation: Cells are adaptive meaning that they can transform into another role when needed. Different programs are encoded into the genome of every cell enabling the cell to perform different functions by executing an alternative program when required. For instance, cells around a wound automatically adapt to a new function in order to compensate for the wounded parts.
- Redundancy: There are multiple cells performing the same function in the organism. This way failure of individual cells doesn't immediately threaten the survival or development process of the organism. Most organisms utilize redundancy. Some organisms even have redundant organs, which are used when the first one fails.
- Decentralization: A very important aspect enabling self-healing behaviour is decentralization. No central coordinator is needed to command the cells into self-healing actions. All necessary functionality is built-in into the DNA found in every individual cell. Cells communicate

with nearby cells using a shared environment, and this way they can to some extent affect each other and direct each other's behaviour, but no large scale central coordination is present.

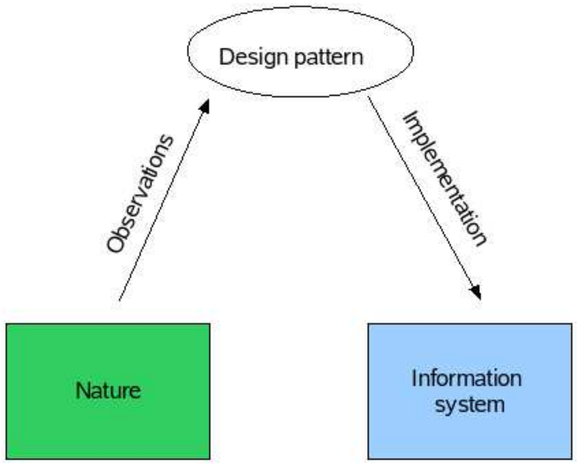

Fig. 1. Extracting general design patterns by observing nature, and achieving self-healing behaviour by implementing these patterns in information systems.

III. The cell as a programming paradigm

When designing self-healing systems, one approach is to look at the function of the cell and design a computer system thereafter. This approach has been investigated by George et al [1], [2]. We will here present their cell-based programming paradigm using a spherical structure as an example, and then explore a distributed file service programmed using the same set of ideas.

A. Modelling the function of a cell

The computerized modelling of a cell is an ongoing research topic with its roots in the works of John von Neumann [6]. The basic self-healing behaviour of a cell is a more recent area of research. Simple self-healing structures of cells can be modelled on a computer as George et al. have shown [1]. In order to achieve this, we first need to decide which properties are needed for the cell.

In essence, the cell in George's model needs to provide three functions in order to be utilizable in self-healing systems. Firstly, a cell needs to be able to replicate by **division**, where one cell transforms into two. This way damaged or died cells can be replaced, and complex structures can be autonomously built starting with just one stem cell.

Secondly, cells need to be able to **communicate** with their surroundings. They need to be able to sense chemical substances around them, and also be able to emit chemicals to their environment. A function where a cell spreads a chemical omnidirectionally out to its surroundings is called **diffusion**. The spreading of a chemical in a particular direction is called **emission**.

Thirdly, cells need to be able to take certain **actions** based upon the chemicals they sense – or do not sense – around them. For instance, if a neighbouring cell dies, it ceases to diffuse chemicals. This way nearby cells notice its death and can take necessary action, like replicate to fill the functionality of the dead cell. These actions are triggered by the DNA encoded program that every cell carry. A certain chemical balance triggers a certain gene and makes the cell act purposefully.

Using these basic functions, it is possible to build structures with self-healing functionality. Multiple unreliable cells, running an identical program, can organize and collaborate without a central coordinator.

B. The sphere program

Fig. 2 presents a program that builds a self-healing sphere based on the three basic properties described earlier. The sphere, depicted in Fig. 3, will regenerate no matter how many body cells are killed as long as the center cell stays alive [1]. The structure is of course simple and might not have too many real-world applications, but still quite well illustrates the concept of biological programming.

We will now describe the program in Fig. 2. To begin with notice that the program contains code for **center** and **body** cells. We will first describe the code for the center cell. In the sphere structure, there is one center cell and many body cells. The cells communicate by emitting and diffusing chemicals called *alive* and *radius*. The *alive* chemical indicates that a cell is living and the *radius* chemical controls the size of the sphere.

The sphere starts with only one center cell. This cell will emit one unit of the *alive* chemical in all directions – (`emits (alive, 1)`) – and diffuse the *radius* chemical (`diffuses (radius, 10)`) so that the chemical can be sensed as far as 10 units away. The line

```
 (transitions (alive from dir < 1) ->
(center, body) in dir;
```

controls the replication of the center cell. The replication is modelled as a transition. If the center cell senses less than 1 unit of the *alive* chemical from any direction *dir* – `alive from dir < 1` – it will transform into two cells, one center and one body. The new cells will be placed in direction *dir*, i.e. in the direction from where the emission of alive was less than one. These divisions will be taking place for as long as there is any direction from which the concentration of the *alive* chemical is less than one.

The *body* cell is even simpler. It will emit one unit of the *alive* chemical in all directions – (`emits (alive, 1)`). Its transformation condition – (`(alive from dir < 1) & (radius > 1)`) – means: if the cell senses less than one unit of the *alive* chemical from direction *dir* **and** more than one unit of the *radius* chemical, it will transform into two daughter cells, both of type body, in direction *dir*. Remember, the center cell diffuses the *radius* chemical in such a way that it can be sensed within 10 units distance. Thus an autonomous, self-healing sphere with radius 10 will be built, according to Fig. 3. The structure will rebuild no matter how many body cells are killed. There is no single coordinator.

Notice, however, that in this simple program used mainly to illustrate the cell based programming principle, there is a

```
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; }
```

Fig. 2. The code for a cell producing a self-healing sphere [1].

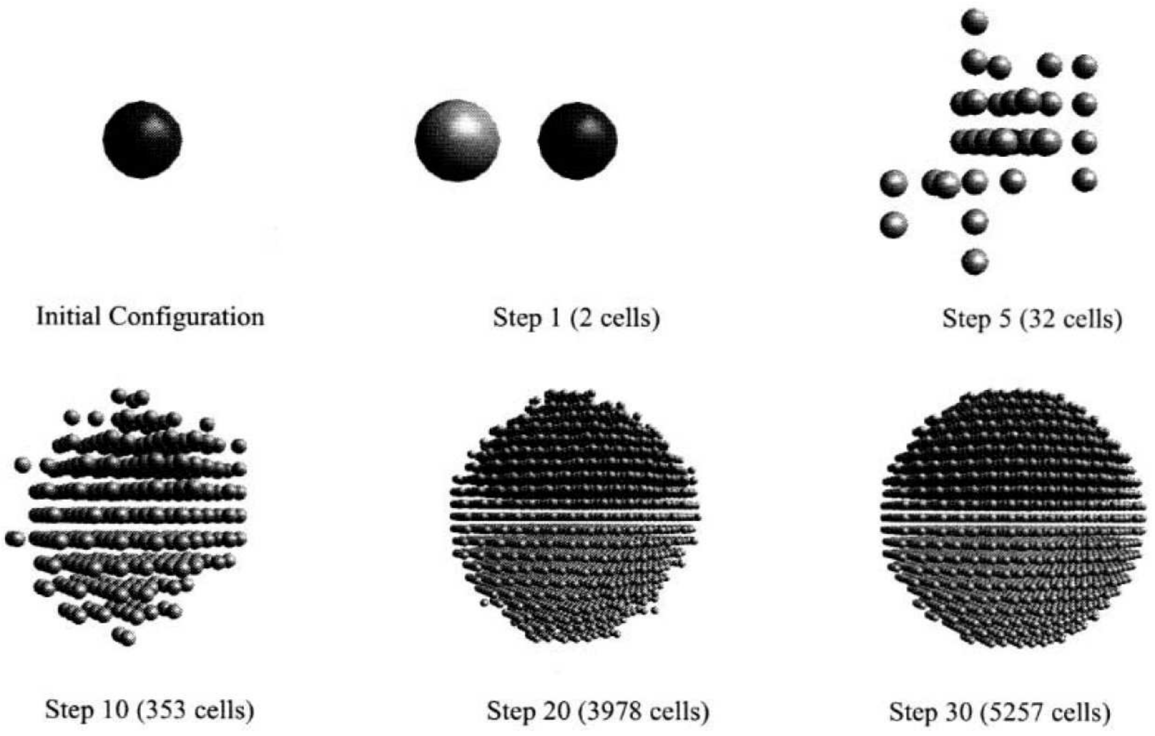

Fig. 3. The growth of the self-healing sphere [1].

single point of failure: the structure won't survive the death of the *center* cell. This can be avoided with a more complex program with one more state and three more transitional rules [1].

C. Case study: a file service

Conceptually, the equivalent of a “cell” in a computer system is a “process”, or a “node” in a distributed system. The ”diffusion” and ”emission” of chemicals corresponds to message passing between nodes. Using these generalizations and the basic ideas from the sphere program, where each cell carry an identical program with certain environmental conditions (messages) activating particular purposeful features, even more useful applications can be built. As a case, let's consider a simple distributed wireless peer-to-peer file service, DWFS [1].

In DWFS, each node carry the same program. The DWFS system supports requesting and publishing files. New nodes can join and existing nodes can leave at any time. The system survives death of nodes and broken connections. There is no coordinator and the system, designed for wireless connections, functions on the application layer.

File request is depicted in Fig. 4 (a). The node requesting a file ”diffuses” a *request* chemical with the file identifier to nodes nearby, and a node containing the requested file will send it. A node that does not carry the requested file simply ignores the request.

File publication is depicted in Fig. 4 (b). Here, two chemicals are used: the *inhibit* and *replicate* chemical. When a file is published, these chemicals are piggybacked with the file. The *replicate* chemical is spread wider than the *inhibit* chemical, as shown in Fig. 4 (b). A node receiving the *replicate* chemical, but not the *inhibit* chemical, will replicate the file further by sending it to neighbouring nodes piggybacked with the two chemicals exactly like the original publisher did. In this way, the files are quickly spread across the network to multiple nodes.

Fault tolerance is achieved as follows. Nodes periodically broadcast the files they hold along with the two chemicals, as was the case with file publication. Nodes receiving the *replicate* but not the *inhibit* chemical will replicate the file. Death of nodes is noticed by the lack of these transmissions. A node that previously was in the area of a dead node's *inhibit* chemical, will now receive the *replicate* chemical from

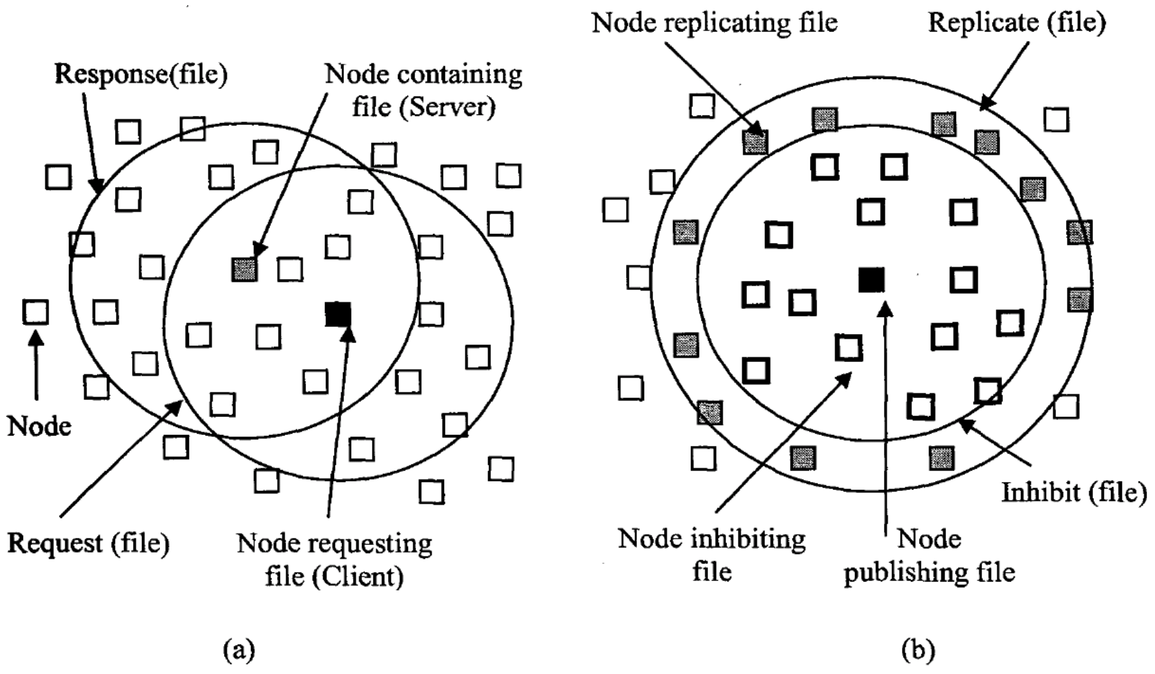

Fig. 4. (a) File request and (b) File publication in DWFS [1].

another node, and will subsequently start replicating the file, consequently replacing the dead node.

Properties from biological organisms have thus been implemented in DWFS to achieve a fault-tolerant and distributed file sharing service.

IV. An ecosystem as an architectural style

In this section, we will discuss a biologically inspired networking application architecture that achieves self-healing behaviour by modelling an ecosystem. In order to make the SymbioticSphere [5] architecture, depicted in Fig. 5 understandable, we will first briefly describe the Bio-Networking Architecture ([3], [4]), upon which SymbioticSphere is built.

The concept of the Bio-Networking Architecture was first introduced in 2001 [3] and has been revised in 2005 [4]. The inspiration to this model has been dwelled from many natural phenomena, including the bee and ant colonies.

The bee colony present remarkable scalability, decentralization and autonomy. The entire hive of the bees is built without a central coordinator. Every bee purposefully does its part in building the hive consisting of hexagonal cells. Later, bees autonomically adapt to low surplus of honey in the hive. If the honey surplus falls too low, many bees adapt the role of a food gatherer and leave the hive to collect food. When the honey level is high, bees remain in the hive to rest.

The function of the bee colony is collectively achieved without a single coordinator, as every bee has a built-in program that guides its actions through different stages of the life of the hive. Also, the colony can easily survive the death of individual bees.

A. The Bio-Networking Architecture: platforms and cyber-entities

The Bio-Networking architecture consists of two components: middleware platforms and cyber-entities (CEs). The middleware platforms run above the operating system of a host computer and provide cyber-entities with an operating environment the CEs need. Cyber-entities, in turn, provide application services to human users or other cyber entities [3].

The modelling of energy flow is central to the Bio-Networking Architecture. Users of the CEs services are required to pay energy in exchange for the service they receive. CEs, in turn, pay energy to the platform for the services they need. The energy flow, thus, reminds of the food chain in biology.

Another important factor gained by the modelling of energy flow is the idea of natural selection. If CEs or platforms don't receive enough energy, i.e. their services are not needed, they will die. On the other hand, if they gather large amounts of energy, i.e their services are used excessively, they will live on and in response to very high energy levels, even replicate.

Thus, the supply of services that are needed a lot will grow in response to large demand, whereas the supply of services that are not that much needed will fall.

The idea here is, that the CEs residing on a certain platform provide some service, for instance a web server service. If the web server is used a lot, the CEs will replicate and even migrate onto other platforms. Thus the capacity of the web server will rise in response to increased demand, resulting in better throughput. If, on the other hand, demand is low, also the supply will shortly adapt thereafter.

Self-healing properties can be attained, too. For instance, CEs can monitor the amount of colleagues they have. If the amount drops too low, for instance when many CEs die or

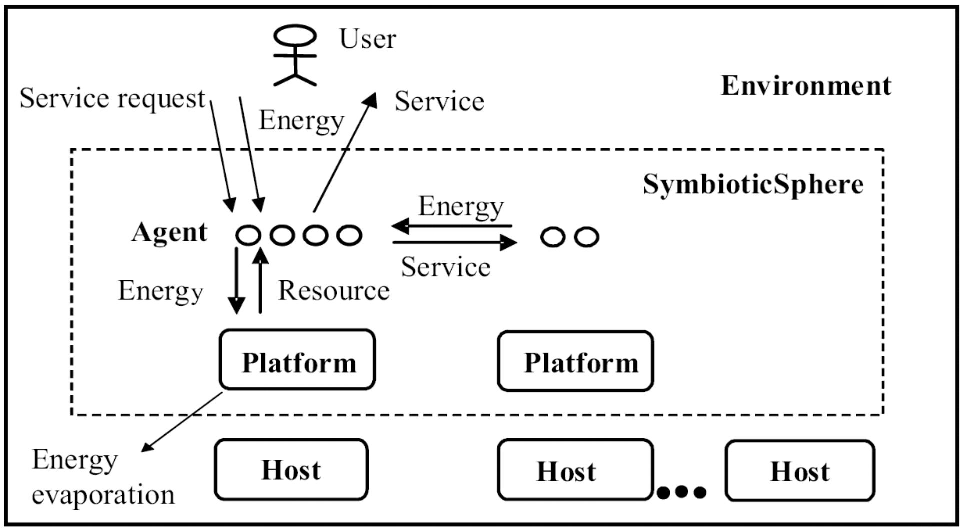

Fig. 5. The ecosystem of SymbioticSphere [5].

crash, the remaining CEs will autonomously replicate in order to guarantee continuity of the service [3].

B. The SymbioticSphere ecosystem

In SymbioticSphere [5], both the Cyber-entity and the middleware platform components are modelled as biological species. They are considered to live in symbiosis in a shared ecosystem. As in the Bio-Networking Architecture, the basis of self-healing functions is achieved by modelling the energy flow.

In SymbioticSphere the cyber-entities are now called *agents*. Users, who can be both human users or other software components, pay agents in exchange for application services. The agent can provide services such as web server services or file services. The agents in turn run on *platforms*, using their services, and pay energy in exchange. In the SymbioticSphere model, 10% of the energy gained from users will in time be given to the platforms.

This will lead to an ecosystem with a symbiotic relationship between two species, where both species will thrive or fail hand-in-hand. If there is a lot of demand for the services the agents provide, the agents will gain a lot of energy, which in turn means that also the platforms will gain a lot of energy, as the platform gains 10% of the share. Now both species will replicate, and the supply of the service is increased as a result of demand. Now the user will experience better throughput and accessibility. If a host, platform or agent fails or crashes, it will quickly be replaced in result of increased traffic to the other agents, who will consequently replicate.

On the other hand, if the services are not needed, the energy level of the agents will fall and, ultimately, as a result of starvation, the agents will die. This leads to reduced energy feed to the platforms, and they, too, will starve and die. Should the need of services begin to rise, the supply would again start to grow and shortly be up to meet the demand.

The function of SymbioticSphere is depicted in Fig. 5. Here, the ultimate source of all energy, the sun, is represented by the User. Users use services that the agents provide, and in return give agents energy. Agents can also use each others services, and hence trade energy with each other. In order to function, agents need the resources of the platform components, and in return for those resources they give platforms energy. Platforms run on host computers on a network. The energy used by platforms is considered to be evaporated.

C. Design principles in SymbioticSphere

Now, let's look at the design principles behind Symbiotic-Sphere [5]. They are all extracted from observing what properties of natural phenomena contribute to nature's effective self-healing.

- Decentralization: firstly, SymbioticSphere is decentralized. This is how a single point of failure, or a performance bottleneck, is avoided. There is no central coordinator controlling or delegating the functions of the agents or platforms. All decisions and actions are taken on the level of individual components.
- Autonomy: secondly, SymbioticSphere is autonomous. The agents and platforms monitor their environment and autonomously make decisions based on what is happening around them. Decentralization and autonomy together are the base properties that make building of large-scale autonomous networked services possible. These two properties are also common to those of the cell based programming paradigm presented in section II.
- Natural selection: the third important principle of SymbioticSphere is that of natural selection. This property is not modelled in the cell based paradigm. Natural selection is modelled using the concept of *energy*. Users give agents energy in exchange for services, and agents give energy to platforms in return for using their operating environment. A high amount of stored energy triggers the *replication* function of an agent, whereas lack of energy leads to starvation and, ultimately, death. Consequently, a process of natural selection is achieved by the modelling

of energy flow. Unused agents and, subsequently, their underlying platforms, will starve due to lack of energy, whereas popular services gain a lot of energy and will replicate.

A property related to natural selection not implemented in SymbioticSphere is that of evolution. Even though it is not implemented, even this function is modelled in the Bio-Networking Architecture [3] upon which SymbioticSphere originally is built. CEs contain functions and bodies. Bodies of a web CE can contain, for instance, a individual web page (HTML file). When two CEs reproduce sexually, their common baby will come to contain the bodies of **both** parents, i.e. two web pages. In the Bio-Networking Architecture, CEs are also able to reproduce asexually. Reproduced this way, the baby CE will contain the web page of its parent. The baby CE will also be given some of the energy of its parent(s).

- Emergence: this is the property of platforms to move towards healthier host computers, and the property of agents to move toward platforms whose resource availability is high. This is how the SymbioticSphere architecture responds to dynamical changes in the environment, for instance to sudden bursts of requests.
- Symbiosis: finally, by symbiosis we mean the cooperation of the components, agents and platform. Recall that they are modelled as different species living in a shared ecosystem in symbiosis. They operate for mutual benefit and they both need each other to survive. This property is used in SymbioticSphere to provide adaptive behaviour.

V. Conclusion

We have now explored two different biology-inspired approaches to designing self-healing systems. The first one was the cell-based programming paradigm. Here, software processes are considered cells. All cells carry identical programs, corresponding to the genome and DNA of biological cells, that contain instructions on how to react to certain changes in the environment and how to affect the environment by diffusing and emitting chemicals.

We considered a spherical self-healing structure, consisting of one center cell and multiple body cells, that was based on a few simple message passing rules and replication conditions. The structure would survive the death of any amount of body cells and completely heal without any intervention or central coordination. This illustrated the cell based programming model.

Our second example was a software architecture inspired by a **biological ecosystem**. By modelling the energy flow, corresponding to the food chain, and two species living in symbiosis, self-healing and adaptive network applications can be built. Large scalable systems, that would self-adapt to increased demand of services by increasing capacity, could be designed using this concept. As an example of an actual implementation of this architectural model we considered the SymbioticSphere.

Notice also some common design principles between the biological programming model and that of SymbioticSphere. Where the former is a cell based programming paradigm and the latter a software architecture, both have in common the concept of decentralization and autonomy. Individual programs – cells or components – contain the necessary programming for different environmental conditions. Purposeful decisions of which program to use in any particular situation is done on the level of individual cells/components. Thus the need of a central coordinator is avoided. This way of thinking naturally suits best the needs of networked and distributed systems.

References

[1] S. George, D. Evans, and S. Marchette, “A biological programming model for self-healing,” in *SSRS ‘03: Proceedings of the 2003 ACM workshop on Survivable and self-regenerative systems*. New York, NY, USA: ACM Press, 2003, pp. 72–81.

[2] S. George, D. Evans, and L. Davidson, “A biologically inspired programming model for self-healing systems,” in *WOSS ‘02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM Press, 2002, pp. 102–104.

[3] M. Wang and T. Suda, “The bio-networking architecture: a biologically inspired approach to the design of scalable, adaptive, and survivable/available network applications,” in *Proceedings of the 2001 Symposium on Applications and the Internet*. IEEE, 2001, pp. 43–53.

[4] J. Suzuki and T. Suda, “A middleware platform for a biologically inspired network architecture supporting autonomous and adaptive applications,” *IEEE J. Sel. Areas Commun.*, vol. 23, no. 2, pp. 249–260, Feb. 2005.

[5] P. Champrasert and J. Suzuki, “Symbioticsphere: A biologically-inspired autonomic architecture for self-adaptive and self-healing server farms,” in *WOWMOM ‘06: Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 469–474.

[6] J. V. Neumann, *Theory of Self-Reproducing Automata*, A. W. Burks, Ed. Champaign, IL, USA: University of Illinois Press, 1966.