Complexity and the Evolution of Computing: Biological Principles for Managing Evolving Systems

Steve Burbeck[*], 2004-2007[**]

Computers collaborate in the Internet/Web much the way cells collaborate in multicellular organisms. But cells do it better! What can we learn from them?

Single cell organisms evolved into multicellular organisms long ago. Today we are seeing a similar transition in computing. Twenty years ago few computers communicated directly with others. Now hundreds of millions of computers exchange information at Internet speeds. The digital world inexorably becomes more complex. Bigger groups of computers collaborate in more complicated and less transparent ways. In doing so, they encounter problems common to all complex systems – problems already solved in the evolution of living systems. This paper explores those problems and some architectural solutions to them.

[*] The author worked at IBM from 1995 to 2005. He is now an independent consultant. His homepage is available at www.evolutionofcomputing.org . A "webified" version of these ideas is accessible from there.

[**] Revision History – Private drafts began in 2000. But this project stayed very much in the background while the author was at IBM. The first public version, in the form of a PowerPoint presentation to the TII/Vanguard Conference on Complexity in Los Angeles, was presented 9/28/2004. The first version in the form of a paper was released 11/14/2004 with minor modifications a month later. This version includes substantive revisions and additions that have accumulated over the last couple of years.

Introduction

Complexity in the digital world is beyond our control. Operating systems, applications, browser plug-ins, clusters and grids, blogs and wikis, peer-to-peer file-sharing networks, and collaborating web-services grow and mutate before our eyes. Computing systems are seldom *designed* these days, they *evolve* and as they do so they become ever more Byzantine and complex. The Internet and the Web are the most obvious example of this evolution. While the underlying protocols were designed, the resulting networks emerged in unexpected ways as people discovered novel ways to exploit those protocols. Precise control and management of these complex systems is no longer possible.

As computing systems evolve and become ever more complex, many people have begun speaking about them in terms of biological analogies. Carver Mead said years ago that, "engineers would be foolish to ignore the lessons of a billion years of evolution."[1] His observation is even more applicable today since our computing systems increasingly have a life of their own.

Coping with the escalating consequences of complexity requires a new perspective. Computing today is more about what happens *between* computers than what happens *inside* them. That is, it is the interactions between them as they collaborate in networks that provide the most value and generate the biggest challenges. In terms of a biological analogy, we are dealing with the transition from single-cell computing to multicellular computing. Therefore, this paper proposes that we focus on the way complex biological systems made the evolutionary transition from single-cell organisms to multi-cell organisms. I will explore a few broad strategies that multicellular living systems use and that satisfy the following guidelines:

- They are rare in single cell organisms;
- They are nearly universal in multicellular organisms;
- They evolved before or coincident with the emergence of multicellular life;
- And they are suggestive of analogs in computing.

Complexity is out of control

Civil engineers who create steel bridges have a saying that "rust never sleeps." The comparable maxim in computing ought to be that "complexity never sleeps." And, once complexity is out of control, it takes control. Computing professionals work tirelessly to reduce complexity but all too often their efforts actually exacerbate it because the already complex systems are far beyond our comprehension.

If there is any doubt that complexity is out of hand, think of the various types of IT specialists who are focused on issues that were almost unheard of a decade ago. IT professionals expend substantial resources detecting and cleaning virus and worm infections, patching machines, modifying firewalls, updating virus detection software, updating spam filters, and the like. There are computing epidemiologists who seek to identify new viral outbreaks before too much damage occurs. And there are computing pathologists who dissect new viruses and worms to see how they work. Is this not like the world of multicellular life, where small viruses and bacteria constantly vie for life's energy and replicative power?

[1] Carver Mead, a professor of Computer Science at Cal Tech was talking then (1993) about constructing a silicon retina and silicon cochlea. The rapid growth of the Internet was just beginning at that time. Its current extent and complexity were not imagined then.

Much of the runaway complexity is inherent in what we are now asking computing systems to do: principally to participate in ever more complex networks of machines. In addition to the increasingly complicated human-controlled client PCs on the net, the Internet now includes PDAs, cell phones, wireless routers, web cams, bar-code readers, RFID sensors, credit-card readers, and microphones for VOIP telephony. Effectors such as pumps and valves, traffic light controllers, music synthesizers, and all manner of household appliances increasingly have web interfaces. If a hacker hijacks your "smart" coffee-maker, you may be irritated, but not threatened. If he hacks into your bedroom baby-cam or the system that controls traffic lights between home and work, it goes beyond irritation. If he hacks into the system that controls the pumps and valves at a large oil refinery, chemical plant, or nuclear power plant it can become a disaster. Yet the complexity of the elements, not to mention the complexity of their interactions, seems to provide an ever-growing smorgasbord of unanticipated and unprotected opportunities for hackers.

Truly, the network ***is*** the computer these days, and that network is under attack. Researchers at the Internet Storm Center estimate that, on average, an unpatched Windows PC connected to the Internet will survive for about 20 minutes before it is compromised by malicious software. We cannot download and install the necessary patches to protect the machine in 20 minutes! And attacks are accelerating. Estimates of the number of new Windows viruses and worms in 2006 ranges between 5,000 and 20,000. The capabilities of viruses and worms continues to evolve as well. For example, many worms try "… to install a 'sniffer,' seeking to use infected computers to capture login and banking information from other computers on the same network."[2]

It is tempting to believe that the only solution to such dangers is to redouble our efforts to control complexity. Certainly we should continue to construct better engineering solutions to each problem: reduce complexity, create more perfect firewalls, and structure the interactions between all computers under our control. But we must also understand that such measures are stopgaps. As Tahar Elgamal points out, "The hard truth of network security is that while many approaches are good, no individual effort makes the network completely safe. Implement enough fixes, and you only succeed at making your network more complex and, hence, more ungovernable, with solutions that wind up acting at cross-purposes."[3] The same can be said for each of the other specialized tasks in managing complex computing systems.

Therefore, while we continue improving our systems in a piecemeal way, we must accept our inability to *control* the computing or networking environment**.** That environment is constantly changing in ways that are beyond the control of any person or organization. Instead, we should look for large-scale architectural approaches that are inherently less susceptible to failure. Where better to look than the biological world where complexity has been evolving for billions of years?

[2] http://news.netcraft.com/archives/2004/09/13/new_worm_installs_network_traffic_sniffer.html
[3] http://news.com.com/Begging+for+trouble+on+security/2010-1009_3-5306242.html

Parallels between biology and computing

Information processing

In the last decade, it has become relatively common for biologists to think of biological systems in information processing terms. As a National Science Foundation workshop report puts it: "A series of discoveries over the past fifty years have illuminated the extraordinary capabilities of living cells to store and process information. We have learned that genes encoded digitally as nucleotide sequences serve as a kind of instruction manual for the chemical processes within the cell and constitute the hereditary information that is passed from parents to their offspring. Information storage and processing within the cell is more efficient by many orders of magnitude than electronic digital computation, with respect to both information density and energy consumption."[4]

The central idea of this paper is that computing professionals would do well to understand the parallels too. All living organisms, from single cells in pond water to humans, survive by constantly processing information about threats and opportunities in the world around them. For example, single-cell E-coli bacteria have a sophisticated chemical sensor patch on one end that processes several different aspects of its environment and biases its movement toward attractant and away from repellent chemicals.[5] At a cellular level, the information processing machinery of life is a complex network of thousands of genes and gene-expression control pathways that dynamically adapt the cell's function to its environment.

Despite the above similarities, we cannot directly compare the information processing "power" of a cell to that of a computer. Size, energy usage, and computational density clearly favor the cell. Memory capability may perhaps be compared somewhat directly. A human cell has about one Giga-byte of program memory (i.e., 3.5 billion bases where each base encodes 2 bits of information) however a modern PC probably uses considerably more working memory than a cell. In terms of computational power, cells use a highly parallel architecture whereas computers use a serial architecture. At this point even the most powerful computer (IBM's Blue Gene) cannot simulate all the processes of a single cell in real-time. In fact, it can't even simulate one process – the folding of a *single* protein molecule – in anything like real-time. Nor can even the most complex cell accomplish what a mundane four-function calculator can do, especially in terms of precision and reproducibility. Cells and computers simply face different tasks and have different capabilities. Still, individual computers and single cells play similar roles in the large-scale sweep of evolution. Just as computers are the initial unit of computation, cells are the initial unit of life. And the challenges of communication and collaboration between networked computers are similar to those between cells in a multicellular organism.

The need for encapsulation

Both biological and computing systems face the issue of encapsulation, i.e., ways of limiting the scale and extent of interactions. If it weren't for the risk – actually virtual certainty – of runaway interactions, a pond, lake, or even an ocean could be one large biochemical "organism" and all computing on the planet could share one vast unprotected address space. However, the probability of unforeseen and unwanted behavior grows rapidly as the number of possible

[4] From an NSF Workshop report by Arthur Delcher, Lee Hood & Richard Karp, see www.nsf.gov/pubs/1998/nsf97168/nsf97168.htm

[5] See Dennis Bray article at www.pnas.org/cgi/content/extract/99/1/7

interactions grows. So, both life and computing have evolved ways to compartmentalize their activity in small, more controllable units. Life evolved the cell, then other protective walls such as skin, bark, shells, and exoskeletons. Computing evolved segmented address spaces that differentiate between code and data spaces, and then abstract virtual memory spaces. Software encapsulation began with separate functions, elimination of GOTO statements, and finally enforced encapsulation in message-sending object-oriented systems. Barriers in multicellular computing include firewalls, DMZs, and protected network domains, i.e., intranets,

Change by evolutionary processes

We tend to talk about computing systems as if they were engineered. And some clearly are, particularly hardware systems. Software systems, however, especially large or old software systems, owe much more to evolution than we sometimes wish to acknowledge. Complex software systems may start with good engineering but all too soon the best intentions of software engineers give way to expediency.

While it may be imprecise to refer to a given system as being either evolved or engineered, we intuitively understand what we mean by engineering. Engineered systems reflect a number of accepted principles of good design. For example, the parts in engineered systems have known functions, irrelevant parts are removed, redundancy is explicit, and designers attempt to maintain separation of concerns, i.e., a part participates in just one functional unit and is designed to do one thing and do it well. Engineered systems do everything possible to prevent the emergence of unforeseen consequences. In contrast, parts in evolved systems may have mysterious interactions with several apparently separate functions and may not seem to be optimized for any of those roles.

Biology seeks to understand the frozen consequences of 3.5 billion years of evolutionary accidents. As evolution progresses, the living systems become more elaborate and develop hierarchical levels of complexity from complex chemistry to complex cells to complex organisms and on to complex ecologies.[6] Biological systems are a triumph of layer after layer of what software professionals would call “clever hacks.” Each new layer exploits the hacks that have come before. So, biological systems contain vestiges of the ancestral history of the organism. Vestigial functional units may or may not be relevant to current circumstances. Some may still be functional in unusual circumstances (e.g., rarely used metabolic functions). Others, such as substantial segments of human DNA may have no function.

To any IT manager, the above should sound very familiar. The history of computing may be relatively short but, as we learned in the “year 2000” experience, it is long enough for legacy computing systems to be full of obscure code that may or may not be relevant to current circumstances. The similarity cannot be pushed much further, though. In computing systems, complexity tends to make the systems more fragile whereas in biological systems , redundancy more often reflects a robustness in the face of unforeseen circumstances far beyond the redundancy found in engineered systems.

Evolved systems, be they biological, social, economic, or computing systems, change over time as a result of the interaction between various sources of novelty and various mechanisms for weeding out “undesirable” or “unfit” novelty. In biological evolution, novelty is presumed to occur by various random processes and weeding-out occurs when an organism does not survive long enough to produce offspring. Novelty in computing usually arises from human creativity. Because computers are general purpose modeling tools, there are always new ways they can be

[6] “The minor transitions in hierarchical evolution and the question of a directional bias,” McShea, D. W., J. Evolutionary Biology, Vol. 14, pp. 502-518, Blackwell Science, Ltd., 2001.

used, new ways for them to interact, and new architectures for their design and construction. Novelty is weeded out when the novel systems simply don't work, or don't scale. But most often novelty fails simply because the marketplace rejects it.

The fitness of new variations in either biological systems or computing systems is determined within a constantly changing "fitness landscape." These systems co-evolve with many other biological or computing systems that may compete or cooperate (or both). Co-evolution in predator/prey or symbiotic relationships tends to drive evolution more rapidly, something that should sound familiar as we cope with today's co-evolutionary spiral between computing virus/worm predators and their Windows prey. The interplay between email spammers and spam filter developers is another obvious example.

Artificial monocultures, i.e., large populations of genetically identical organisms such as corn fields or rubber plantations are big, stationary targets for diverse evolving predators. Their lack of diversity puts them at risk in the co-evolutionary arms race. Once any virus, bacteria, fungus or insect manages by mutation to escape the defenses of one plant in the monoculture, all plants are immediately at risk. The Irish potato famine in 1845-50 is an unfortunate example of what can happen. To our dismay, computing professionals are only recently realizing that this same principle applies equally well to the Windows/IE/Outlook monoculture.[7]

The evolution of multi-cell systems

The transition from single-cell to multicellular life did not happen on one sharp step. We do not know with much precision when and how the strategies that survive today arose nor what alternatives were tried and failed. All we know is that what we see today survived the test of time.

From our current perspective it appears that multicellular life evolved from single cells in two stages. First, single cell organisms evolved the ability to form loose cooperative communities, often called biofilms,[8] that can perhaps be thought of as "training wheels" for multicellular life. The earliest colony bacteria were the cyanobacteria that evolved more than three billion years ago. Their fossil remains are visible today because colonies of cyanobacteria secreted a thick gel as protection from solar radiation unattenuated by an atmosphere that lacked ozone. This gel, in turn, trapped sand and debris from the surf which, together with lime secreted by the bacteria, formed the beautiful patterns of the Stromatolite fossil reefs visible in Australia. These structures are highly variable in size from twig-size to semi-truck size.[9] Biofilms remain common today. Present-day examples of biofilms include slime mold, dental plaque, films on rocks in streams and many more.

About one billion years ago true multicellular organisms formed – plants, animals, and fungi – known generically as Metazoans. All cells in a Metazoan organism share the same DNA. As the organism develops, the cells specialize by sequestering and permanently silencing much of their DNA according to developmental genetic programs. Some organisms have multiple stages of stable forms, e.g., insects that exhibit larval, pupae, and adult forms. But these developmental stages all involve programmed cell differentiation. For most cells, some stem cells being the exception, differentiation is dramatic and irreversible. The full complement of genes and DNA control sequences in the multi-cellular genome is far more complex than that of single cell

[7] See http://www.wired.com/news/privacy/1,62307-0.html and S. Forrest, A. Somayaji, and D. Ackley, "Building Diverse Computer Systems," *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS VI)*, May 5-6, 1997, p. 67.

[8] www.erc.montana.edu/CBEssentials-SW/bf-basics-99/bbasics-bfcharact.htm

[9] From www.fossilmuseum.net/Fossil_Galleries/Stromatolite_fossil_gallery/Stromatolite_fossils.htm

organisms. But any given type of cell in a multicellular organism – and there are about 250 different types in humans – is functionally much simpler than free-living single cell organisms. "Each differentiated cell type only makes the proteins coded for by a few percent of the 50,000 total genes."[10] For example, all cells in the body have the gene for hemoglobin, but only red blood cells express it.

Conventional wisdom asserts that the primary benefit of multicellularity, hence presumably what drove its evolution, is the division of labor, or specialization, provided by differentiated cells.[11] But differentiation evolved slowly. Prior to the emergence of differentiation there was a more temporary form of specialization in cooperating communities of single-cell organisms (biofilms); there was polymorphic messaging between genetically identical cells (e.g., quorum sensing); there was programmed cell death (apoptosis); and there was (stigmergy).[12] And, as we will see later in this paper, the mechanisms that support specialization, messaging, stigmergy and apoptosis are intimately intertwined. So the notion of a "primary" benefit is misleading.

The evolution of multicellular computing shows interesting parallels to that of multicellular organisms. Early large-scale distributed network computing began when personal computers were used as terminals to mainframes in place of dedicated terminals. As more "smarts" migrated from the mainframe to the terminal, the interaction between client and server became richer and more varied. Now we see loosely organized general-purpose computers in web communities, P2P file-sharing, and ad hoc compute grids such as SETI at home. These loosely organized communities are comparable in complexity and organization to biofilms. Some Grid architectures are more formal and specialized and therefore are perhaps analogous to simple Metazoa such as the hydra or perhaps small jellyfish. As computing continues to evolve, I confidently assert that specialization will increase, messaging will become ever more elaborate, stigmergy structures will proliferate and become ever more economically important, and apoptosis-like mechanisms will evolve to shut down or disconnect computers infected by viruses and worms. In other words, computing will adopt architectures more akin to multicellular biological organisms.

[10] See www.bio.unc.edu/courses/2003spring/biol104/lecture8.html Note that since this was written the estimate of the number of human genes has dropped to less than 25,000. The principle remains the same.
[11] See Maynard Smith, J. & Szathmáry, E. *The Major Transitions in Evolution,* 1995. Or Pfeiffer and Bonhoeffer, PNAS, 2003, http://www.pnas.org/cgi/content/full/100/3/1095.
[12] The terms quorum-sensing, apoptosis, and stigmergy are discussed in more detail later in the paper

Know your enemy: characterizing complexity[13]

A crucial similarity between the evolution of living systems and that of computing systems is the way they become ever more complex. But just what does that mean?

There have been attempts by heavyweights such as Andrei Kolmogorov, Gregory Chaitin, Charles Bennett, and Stephen Wolfram[14] to rigorously and formally define complexity. Each of those attempts has captured some properties of complexity but none capture it in a general way. Given that we can't satisfactorily define complexity, it should come as no surprise that we cannot satisfactorily measure complexity in a general way either.[15] So, what could it mean to say that evolving complex dynamic systems have a habit of becoming more and more complex? Without stepping into the deep waters of trying to define complexity, let me say that such systems become less and less predictable without becoming more random.[16]

We tend to think of complexity as more intricacy than we can comprehend. That describes only one sort of complexity, often called *detail*, *structural*, or *static* complexity. Another sort of complexity, typically called *dynamic* complexity, is inherent in the systems themselves, not in the limits of human comprehension. This second sort of complexity emerges naturally in systems that evolve over time as a function of their operation: e.g., meteorological, cosmological, biological, ecological, geological, social, economic, or computing systems.[17]

Both sorts of complexity bedevil us in our computing systems. We struggle with static complexity such as program source code and database schema that define structural relationships between networks of interacting elements. These structural descriptions typically become so intricate that they exceed our cognitive ability to understand them. Dynamic complexity is qualitatively different; it is about what happens at runtime as the behavior of a system, e.g., a computer program, unfolds via interactions between elements.

Emergent behavior in complex dynamic systems

Consider a flock of starlings for example. The birds attempt to stay in the flock while avoiding collisions with other birds. The turns and twists each bird makes while satisfying these goals affect the paths of many nearby birds which, in turn, affect the paths of still more birds. Thus the dynamics of the flock as a whole are complex and inherently unpredictable. Yet anyone who has watched flocks of starlings can see clearly that the behavior of the flock is not random. Note that this sort of complexity has nothing to do with human cognitive limits. It arises from the distributed nature of the task the birds face and the different perspective each bird has.

Unexpected behavior emerges by the continued action of positive feedback in complex systems. Consider an example of positive feedback that is familiar to most of us: a microphone induces a loud squeal from a speaker when the microphone gets too close to the speaker or the amplifier

---

[13] This section on complexity benefits greatly from many conversations over the years with my long-time colleague, collaborator, and co-author Sam Adams. He is a Distinguished Engineer in IBM Research.
[14] See www.complexsystems.org/commentaries/jan98.html or www.nctimes.net/~mark/bibl_science/complexity_ben.htm
[15] See cscs.umich.edu/~crshalizi/notebooks/complexity-measures.html, and "What is complexity," Christoph Adami, BioEssays 24:1085-1094, Wiley Periodicals, 2002.
[16] We lack satisfactory definitions and measures of randomness too, so I use that term somewhat loosely.
[17] There is a brief but useful discussion of the two types of complexity available at http://www.stewardshipmodeling.com/dynamic_complexity.htm

gain is turned up too high. The positive feedback occurs because the sound picked up from the microphone is amplified, sent out through the speaker and returns to the microphone to be picked up louder than before. Once you understand the mechanism, it seems unremarkable and controllable. To kill the squeal, simply move the microphone farther from the speaker or turn down the amplifier gain.

Similar, though less manageable, positive feedback tends to emerge in random networks. As a thought experiment, consider a large open space, say a football field, on which a hundred speakers, amplifiers and microphones are placed randomly. Now, begin adding connections (wires), one at a time, between a randomly chosen microphone and amplifier or randomly chosen amplifier and speaker.[18] Sooner or later, a squeal will arise because a speaker happens to be close enough to the microphone that feeds it. Let us call that trio an *autocatalytic set*, i.e., a self-reinforcing set of elements that produces an emergent property of the system (the squeal). Adding more wires will soon create a second squeal, and a third, and so on. There tends to be a threshold beyond which new feedback loops are created very rapidly and shortly thereafter nearly all the speakers will be emitting squeals. Because the output of any given speaker reaches more than one microphone, some of the autocatalytic sets (i.e., feedback loops) may involve more than one microphone/amplifier/speaker trio and, by the same token, a given speaker or microphone may be participating in more than one autocatalytic set, i.e., emitting multiple squeals at different pitches (the pitch of each squeal is determined by the speed of sound and the distance between the speaker and the microphone that generate the feedback).

In 1965, Stuart Kauffman did a more formal experiment analogous to the speakers on the football field using random Boolean nets in a computer simulation. He discovered, to his surprise, that autocatalytic sets (called *state cycles* in these Boolean nets) inevitably emerge very quickly and the number of autocatalytic sets that emerge is roughly the square root of the number of elements in the Boolean network.[19] More generally, mutually reinforcing feedback loops form in all sorts of complex systems. The probability of the emergence of autocatalytic sets increases as more elements are added, more interconnections are added, or the elements themselves become more complex and therefore can interact with others in more complex ways. Familiar natural examples include:

- Sand dunes form from the simple interactions of very large numbers of grains of sand with each other and with the wind. If there is enough sand being blown by the wind, any obstruction can start the formation of a dune: a bush, a fence post or even an ant hill. If the pile of sand is large and steep enough to create a "wind shadow," more sand will collect in the shadow, enlarging the pile. Given a sufficient supply of sand and wind, the emerging sand dune may eventually grow to more than 300 meters in height and move as much as 20 meters per year. In North China and the Sahara, sand dunes threaten to engulf entire towns.
- Money, in the form of chunks of metal or cowrie shells, emerged as early as 5000 BC. But money became really useful once coins with specific values, i.e., denominations, emerged about 700BC in Lydia (now a part of Turkey). Before coins were minted, commerce was done by barter, which limited commercial exchanges to pairs of people who desired what the other had (or perhaps three-way exchanges that were very difficult to arrange). Coins allowed a new kind of commercial interaction between people, one in which the money

[18] Note: since this is an abstract thought experiment, we allow multiple microphones to feed the same amplifier, a microphone to feed multiple amplifiers, and an amplifier to feed multiple speakers. We also ignore the likelihood that, in real life, amplifiers may blow fuses and speakers may shred themselves.
[19] For an easily read discussion, see Chapter two of *Complexity: Life at the edge of chaos*, by Roger Lewin, 1992l, Macmillan Publishing Co. Kauffman has developed these ideas into a theory, more formally known as NK Systems. See S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*, Oxford University Press, New York, 1993.

provided an agreed upon temporary store of value that was specific neither to a particular commodity or service, nor to a specific place and time. Feedback arose because, as the benefits of using money were made clear, more coins were minted which allowed those benefits to be experienced by more and more traders. Within two hundred years, the idea of using coins with specific denominations had spread as far as China. Commerce was changed forever.

- In computing, TCP/IP and HTTP protocols create new sorts of interactions between computers. The Internet emerged from TCP/IP and the Web emerged from HTTP. P2P protocols are another example. They supported the emergence of Napster and its descendents that now threaten to obsolete the music recording industry. These sorts of systems grow because of positive feedback known as the "network effect." That is, as the network grows it becomes more attractive for others to join the network. Telephones and fax machines are the usual example of prototypical positive feedback network effects.

It is important to note that emergent behavior is *qualitatively* different from that of its elements. Sand dunes are far different from grains of sand, both in scale and in behavior. A marketplace based upon money is qualitatively different from one based on barter because prices emerge that create relationships between all goods and services. More specialized goods and services can participate on an equal footing with everyday commodities. Similarly, the emergent behavior called the Web is dramatically different from communities that swap files by FTP even though the technical differences between FTP and HTTP are relatively minor.

How many elements are required for emergence? The answer depends upon the complexity of the elements and their interactions. Unexpected behavior can emerge from a large number of very simple elements (a sand dune) or a small number of interacting elements if their interactions are rich and they are sufficiently adaptable. At the extreme of complexity -- human beings -- two is enough. Anyone who has been in a long-term relationship can attest that a couple is more than the sum of the behavior of the two individuals. Interactions less complex than a marriage require a few more individuals, e.g., human behavior in crowds. Small flocks of birds or schools of fish can also provide surprising collective behavior. And, as every programmer has discovered, even small software systems all too often demonstrate surprising emergent behavior (bugs).

At a higher metalevel, i.e., where the elements themselves are complex dynamic systems, interactions can be richer. Relatively small sets of more complex elements can create astonishing emergent behavior

- Brains -- Neurons are much smaller than grains of sand yet their interactions are far richer because each neuron is itself a complex dynamic system. Neural "circuits" of only a few hundred to a few thousand neurons can produce amazing behavior. The entire central nervous system of small free-living metazoans such as rotifers and nematodes has less than 300 neurons. The entire nervous system of *Aplysia* (a molusc) contains 18-20 thousand neurons. These animals aren't noted for their intellectual gifts, yet their behavioral repertoire is far greater and more "purposeful" than a hurricane or a sand dune: they seek food, avoid danger, mate, and learn.

- Computers -- The earliest computers had a few thousand gates and programs with a few thousand machine instructions. While their planned emergent behavior was primitive by today's standards, they still could calculate artillery ballistic tables, prime numbers and the like. And unplanned emergent behavior (bugs) popped up even then. In modern computers with upwards of a billions gates and hundreds of millions of lines of code, the behavior, both planned and unplanned, is beyond our comprehension.

Multi-level complexity

When stable autocatalytic sets of interacting elements emerge, the sets themselves inevitably begin to interact with one another. For example, denominated coins supported the abstract notion of placing a numeric value, or "price" not only upon a cow or basket of wheat, but also upon an arbitrary set of assets, goods, and services that are otherwise incomparable. So a set of goods can become an atomic element to be traded. This allowed us to form banks (around 400 BC) and to assign a value to an entire ship and its cargo for the purpose of investment and risk sharing.[20] Similarly, a company (or eventually a corporation) that can own ships or cargoes can also be assigned a price. Valuing companies and other long-lived collective enterprises gave rise in the Middle Ages to stock markets, known as bourses, which treat the value of shares in a whole company as atomic elements to be traded. Various modern derivatives markets, in turn, treat whole market indexes, e.g., the S&P 500, as the atomic elements upon which another level of trading, such as index futures, can be constructed. Each new level is built upon the one before and, in turn, becomes the atomic elements from which the next level can emerge.

This issue of multi-level emergence is especially important to the present paper for at least two reasons: 1) because both life and computing have resulted from layer after layer of emergent phenomena so the similarities should be instructive; and 2) because multi-level emergence poses severe, sometimes insurmountable, challenges for the predictability and manageability of the resulting systems. To understand how those layers, or meta-levels, interact, it is worth going into some detail on their emergence over time.

The many successive levels of emergence that led to the life on the planet Earth today cannot be known precisely. The short story, as best science can say today, begins at the "big bang." A couple of seconds after the big bang, the universe was a dense sea of solitary neutrons, protons, electrons, positrons, photons and neutrinos. As the universe expanded and cooled over a few hundred thousand years, many of these particles joined into the stable autocatalytic sets we call atoms – mostly hydrogen atoms with some helium and a tiny admixture of the next lightest nuclei: deuterium, lithium and beryllium.[21] Perhaps 600 million years pass until gravity interactions generate the third level organization: galaxies and stars. Inside stars gravitational pressure and temperature ignite fusion reactions in which small, light nuclei fuse together to create the larger, heavier nuclei of elements up to iron, including elements needed for life as we know it such as carbon, oxygen, nitrogen, sodium, potassium, calcium, etc. But fusion cannot create elements heavier than iron. More esoteric "neutron capture" reactions create the remaining elements, some of which are crucial to life.[22] The violent death of stars in supernovas spews all these newly created elements out into interstellar space. Time passes, stellar dust from supernovas again condenses by gravitational attraction to form second generation stars that have planetary systems with the full complement of chemical elements needed for rock, water, air and, ultimately, life. Time passes and one such planet – the earth – slowly cools to temperatures that can sustain life.. All sorts of autocatalytic chemical reactions in the earth's oceans and

[20] "At Athens, hundred of ship cargoes were required annually to satisfy Attica's enormous need for food and other items. Virtually all of these cargoes were dependent on loans. These financings, together with the additional loans generated by the Piraeus's dominant position as an entrepôt for the eastern Mediterranean, provided creditors with an opportunity to absorb over many transactions the risk of a total loss from the sinking of a single ship." (See Cohen, Edward E. 1992. Athenian Economy and Society: A Banking Perspective. Princeton University Press, Princeton, pp. 140-141.)

[21] (See: www.brainyencyclopedia.com/encyclopedia/t/ti/timeline_of_the_big_bang.html). Note: this short story ignores the first couple of seconds after the big bang in which matter/anti-matter annihilated each other, and stable sets of quarks become neutrons and protons.

[22] For example, nickel, copper, molybdenum, zinc, tin, iodine. Zinc, for example, is crucial to many DNA binding proteins that control gene expression.

atmosphere create the early organic carbon-based compounds that slowly combine to create successively larger and more complex organic compounds.[23] Natural processes create small bilipid membrane vesicles filled with water and complex sets of organic chemicals.[24] They are reminiscent of simple cells, but aren't alive, i.e., they cannot gather energy from the outside world nor can they replicate. Perhaps 3.8 billion years ago, unexplained "magic"[25] creates mechanisms for replication that allows the emergence of simple single-cell life. For a couple of billion years, single cell organisms evolve dizzying complexity in many steps: absorbing mitochondria and chloroplasts, creating the nucleus, and so forth. Perhaps 3.5 billion years ago, cyanobacteria evolved physically co-located cooperative structures held together by sticky secretions from the cells (e.g., gel or slime). These cooperative cyanobacteria are believed to be responsible for the conversion of the early carbon dioxide atmosphere to the oxygen-rich atmosphere of today. Finally true multicellular (Metazoan) organisms form sometime between a billion and 600 million years ago.[26]

All of the levels described above are evident in *every* living cell or organism today. The biochemistry that emerged in the Earth's oceans clearly operates in every cell. Virtually all of the energy used by every living cell comes from nuclear fusion in the sun via photosynthesis in plants. All the hydrogen in cells was created in the big bang itself. The heavier elements were created in stars and supernovas. And many of the random events that generate novel mutations that evolution can exploit are due to UV radiation from the sun, cosmic rays from distant galaxies, and occasionally even neutrinos from the big bang itself. So, every one of the dozen or so layers of emergent behavior still participate in a great cosmic dance, one small figure of which is Earth's biosphere containing all the species of living organisms.

The evolution of computing is much shorter and better known. Modern computing began during World War II to facilitate code breaking and computation of artillery ballistics. Carefully constructed sets of vacuum tubes, resistors, and capacitors with positive bi-stable feedback became the first logic gates. At first, these gates were wired together by hand to perform higher-level logic and arithmetic functions. Soon, the notion of a machine instruction emerged. Machine instructions represent coordinated sequences of gate changes with predictable results. Early programs were carefully crafted sequences of machine instructions that exploited very clever (and today verboten) kinds of self-modifying code. As we learned more about programming, we created reusable sub-routines or functions, i.e., sets of machine instructions that can be treated as a unit, again, typically with predictable results. The first levels of code abstractions appeared: assembly language and compiled COBOL, FORTRAN and soon thereafter, ALGOL. At that stage a program ran on the bare metal without an operating system, I/O drivers, or other abstraction layers. The program finished when the computer halted, which was made obvious when all the blinking lights on the operator's panel froze although, given the unreliability of the early machines, frozen lights all too often simply signaled a hardware failure or a program bug. It should also be noted that, at that stage, programs were crafted for a single machine. Without any layers of abstraction between the program and the hardware, a program was unique to that machine. So each of the relatively isolated and rare computers attracted a group of computing people – amateurs of course; it would be years before computing became a profession[27] – that explored what the computer could do and shared their techniques and code. As

[23] For example, carbonyl sulfide (COS), a simple volcanic gas, induces the formation of polypeptides from individual amino acids in water solution. Science, vol 306, 8 October, 2004, pp. 283-286.
[24] See www.msu.edu/user/ottova/chap1.html
[25] We don't know the steps that led to the evolution of the fantastic mechanisms of RNA, DNA, protein, etc. that support replication, hence life. I use the term "magic" simply to reflect our ignorance.
[26] The time is very imprecise because the earliest metazoans were probably small, soft, creatures like hydra that did not leave fossils,
[27] The first issue of the Communications of the ACM was published in January, 1958.

computers became more alike and people began using computers for more routine tasks, operating systems emerged as did outboard I/O processors, databases and other kinds of middleware. The previously separated communities of computer people had much to share with one another. The emergence of minicomputers and then "microcomputers" (which morphed into PCs), based on the early microelectronic CPU chips, opened computing up to the rest of us and completely changed the common notions of what a computer was for. The rapid decrease in price/performance due to Moore's Law continued the expansion of computing into new areas and in unheard-of numbers. In the mean time, in the '60s, the early efforts to connect computers arose. Rather than moving rolls of paper tape, boxes of punch cards, or reels of magnetic tape from machine to machine, it would be convenient to send data over a wire. This line of evolution eventually led to the ARPAnet in the early '70s, which led to the modern Internet. And, finally, in 1989-1991, Tim Berners-Lee and collaborators proposed and built a combination of HTTP, HTML, and a crude browser that became the World Wide Web. All of these emergent levels, from individual gates (now made with microscopic transistors on silicon wafers rather than vacuum tubes) to collaborating web services, participate in our everyday experience of computing.

What is amazing is that the evolution of the "virtual" world occurred so rapidly. Computing had become so complex by the '70s (a mere three decades after the first general purpose computer) that different threads of evolution – hardware, software, networking, and cultural/economic – were operating separately but in parallel, dependent upon each other and reinforcing each other in completely unexpected ways. Finally, in the mid 90's, the Internet/Web and the dot.com boom supercharged the evolution of computing by driving it into every nook and cranny of modern culture. We now see amusing consequences such as adults asking their middle-school kids to help them with some computer problem, or business executives with expensive laptops that are considerably less powerful than their kid's three hundred dollar game box. Many of us walk around with one or two computers more powerful than a 1960 mainframe in our pockets. And some of them such as cell phones and pocket email devices are hooked to the Internet at least intermittently. Now human society, arguably the pinnacle of the evolutionary path that evolved step by step from atoms, is influenced as much by bits as by atoms. That is, *the evolution of life and the evolution of computing are merging, bringing the complexity of both realms together in completely unpredictable ways.*

The "three-level rule"

In order to predict and manage the behavior of multi-level complex systems, we must be able to reason about *how cause-and-effect crosses the levels*. Unfortunately, there are cognitive, and even physical, limits to our ability to trace cause-and-effect through multiple levels. We can focus on the elements of one particular level and contemplate the "part/whole" relationships in two directions. For example, we can think about how the elements, say molecules or machine instructions, are made up of atoms or orchestrated gates and how the properties of those atoms or gates affect the properties of the molecules or instructions. With a switch of cognitive repertoire, we can also contemplate how the molecules interact with other molecules to form crystals or participate in various chemical reactions or how machine instructions cooperate to carry out a function or subroutine. A skilled practitioner – a chemist or programmer – may be able to think about interactions in both directions at once. That is, a skilled practitioner may be able to reason simultaneously about three levels of abstraction. But it is quite difficult and seldom even useful to do so. Reasoning about four levels of abstraction at one time is, arguably, beyond the ability of the human mind.

Unpredictability in multi-level complex systems

Cause and effect in complex systems does not necessarily reach across levels. For example, except for exceedingly rare interactions with neutrinos, the quarks that collaborate to form protons and neutrons interact solely with each other; only their collective mass, charge, and strong nuclear force has external effects. Quarks are an extreme example of isolation, but both evolved and designed systems tend to use mechanisms to restrict interactions simply because unrestricted interactions tend to devolve to chaos. Most of the chemicals inside a living cell are isolated from the external environment by cell walls and perhaps also isolated within internal organelles. Modern computers are designed to prevent the inappropriate reading, writing, or execution of memory contents. Firewalls seek to prevent inappropriate interactions between computers over Internet connections. And so forth.

Nonetheless, most systems, certainly most biological and computing systems, are inherently multi-level systems in which some of the complexity within a level *i*s, and often must be, exposed to the level above and/or the level below. Tracking cause and effect through these sorts of multiple levels is exceedingly difficult and often impossible. The details of a hurricane or tornado are *fundamentally* not explainable by invoking the physics of individual air and water molecules.[28] Nor can a computer be understood by pondering the behavior of electrons, or logic gates. Similarly, the behavior of even a single cell cannot yet be predicted by understanding its chemistry at a molecular level. We also intuitively recognize these sorts of limits and their consequences for predictability in business, social, economic and ecological spheres.

Occasionally, multi-level phenomena become explicit and our inability to manage them has profound, and usually unpleasant, consequences. For example, pharmaceutical drug discovery is a biochemical discipline in which we face many levels of complexity between the "cause" and the "effect." We seek to find a small-molecule, i.e., a drug, that modulates an intracellular biochemical pathway in a way that desirably affects the human body (or even the human mind, e.g., anti-depressants) yet does not affect *any other* cellular process in a deleterious manner. The major reason it is so difficult to find drugs and even more difficult to determine that they are safe and effective is that there are so many levels of complexity between intracellular chemical reactions and whole-body effects and side-effects. Similarly, in the computing realm, multi-level problems account for some of the most recalcitrant bugs we face. Perhaps the most egregious example in computing is the prevalence of buffer-overrun exploits in Windows. A buffer-overrun takes place, essentially, at the machine language level, i.e., the necessary code for array-bounds checking translates to perhaps a dozen extra machine instructions. Yet the effects of buffer-overruns can be Internet-wide. The "SQL Slammer" denial of service worm that slowed or blocked 80% of all Internet traffic for a few hours January 25th, 2003, was due to a buffer-overrun in Microsoft SQL Server software caused by a handful of ill-considered machine instructions.[29]

[28] Weather is inherently very sensitive to initial conditions. Since the motions of the underlying molecules are random and obey Heisenberg's Uncertainty Principle, there is a limit to the detailed accuracy with which it can be predicted.

[29] See www.byteandswitch.com/document.asp?doc_id=27509. Programmers allow buffer-overruns in part because of laziness or incompetence, but also out of a misplaced concern for saving the few CPU machine cycles (a few nanoseconds on a 2-GHz processor) needed for bounds-checking array references in C code. That misplaced concern threatens the entire Internet, which is a dozen levels of abstraction above the machine language level. This is the epitome of "penny-wise and pound-foolish."

Constraints on freedom to evolve

Interactions between levels can constrain the possible evolutionary paths of adjacent levels. Every level other than the "highest" is at least somewhat constrained to evolve more slowly because changes that invalidate an implicit "contract" with the higher level tend not to survive long. Thus, single-cell organisms (and viruses) can mutate rapidly because their behavior is relatively free from higher-level constraints. Individual cells within multi-cell organisms are not so free to explore new behavior. Similarly, unconnected digital devices (PDAs or cell phones for example) change at a dizzying pace whereas PCs and servers become more powerful and cheaper but must generally retain the ability to run almost all applications from the prior generation. In multicellular computing, we see most innovations happening at the highest level where new kinds of collaboration are being proposed every month. These innovations seldom require changes to the underlying ecology of computers, operating systems, or Internet/Web protocols. Those that do, tend to fail.

Principles for managing multicellular systems

Our computing systems *are* recapitulating the evolution from single-cell to multicellular life. The above discussion was intended to show that we might profitably look at living systems for strategies that could apply to computing. The thesis of this paper is that multicellular life exploits four broad strategies – see table below – that 1) are rare in single cell organisms, 2) are nearly universal in multicellular organisms, and 3) evolved before or concurrent with the emergence of multicellular life. Subsequent sections examine each of these strategies in some detail for insight into how we might use their analogs to help manage the further evolution of computing.

Table 1. The four broad strategies

	Multicellular Organisms	**Networked Computing**
Specialization supercedes general behavior	Cells in biofilms specialize temporarily according to "quorum" cues from neighbors. Cells in "true" multicellular organisms specialize (differentiate) permanently during development.	Today most computers, especially PCs, retain a large repertoire of unused general behavior susceptible to viral or worm attack. Biology suggests more specialization would be advantageous.
Communication by polymorphic messages	Metazoan cells communicate with each other via messenger molecules, *never* DNA. The "meaning" of cell-to-cell messages is determined by the receiving cell, not the sender.	Executable code is the analog of DNA. Most PCs permit easy, and hidden, download of executable code (Active-X, java, or.exe). Biology suggests this should be taboo.
"Self" defined by a stigmergic structure	Metazoans and biofilms, build extracellular stigmergic structures (bone, shell, or just slime) which define the persistent self. "Selfness" resides as much in the extracellular matrix as in the cells.	Intranets and databases are stigmergy structures in the world of multicellular computing, as are many Web phenomena such as search engines, folksonomy sites, wikis and blogs. But they are ad hoc structures. Stigmergy and definition of "self" needs to be more systematic.
"Self" protected collectively by programmed cell death (PCD), or Apoptosis	Every healthy Metazoan cell is prepared to commit suicide -- a process called apoptosis. Apoptosis reflects a multicellular perspective - sacrificing the individual cell for the good of the multicellular organism.	A familiar example in computing is the Blue Screen of Death which is a *programmed* response to an unrecoverable error. A civilized metazoan computer should sense its own rogue behavior, e.g., download of uncertified code, and disconnect itself from the network.

Specialization and differentiation

Specialization is a *general* characteristic of multi-level biological systems that supports a useful trade-off in complexity; "...in evolution, as higher-level entities arise from associations of lower-level organisms, and as these [higher-level] entities acquire the ability to feed, reproduce, defend themselves, and so on, the lower-level organisms will tend to lose much of their internal complexity."[30] This principle appears to apply to social insects as well. For example, "...individuals of highly social ant species are less complex than individuals from [socially] simple ant species."[31]

Specialization begins at the level of individual cells in multicellular organisms. Bacterial cells capable of participating in biofilms (cooperative communities of single-cell organisms) can usually also live as isolated cells. But when the single-cell organisms participate in a cooperative biofilm, they temporarily specialize. Metazoan cells, i.e., the cells in "true" multicellular organisms, don't just temporarily specialize, they permanently differentiate as the organism develops from a fertilized zygote to an adult. "The different cell types in a multicellular organism differ dramatically in both structure and function. If we compare a mammalian neuron with a lymphocyte, for example, the differences are so extreme that it is difficult to imagine that the two cells contain the same genome."[32] Occasionally, through some mischance, a cell loses some of its differentiation. The cells in which this occurs are called neoplastic (i.e., newly changeable). Neoplastic cells are not just abnormal, they are a forerunner of cancer.

Specialization is *possible* because the environment faced by a cell in a multicellular organism is quite different from that faced by a single-cell organism. Metazoan cells live in a cooperative homeostatic environment protected and nourished by the whole organism. In contrast, a single-cell organism must be prepared to deal with all sorts of unfavorable environments. That flexibility requires a large repertoire of behavior. E coli, a commonly studied bacteria, can use flagella to move toward nutrients and away from toxic chemicals. But they only create these flagella when they need them. If they are in a favorable environment, they dispense with flagella.

Specialization is *necessary* because maintaining the full complement of all possible behavior is costly and/or dangerous. For individual cells, one obvious cost is in energy consumption; the maintenance of all the unnecessary cellular machinery is not free. Since specialized Metazoan cells use only a small fraction of the total genome, their energy costs are dramatically reduced. But also cell specialization induces very different and incompatible cell chemistry, shape, and function. A nerve cell could not function as a reliable communication channel between point A and point B if it also builds bone around itself, fills itself with hemoglobin for carrying oxygen, accumulates fat, and secretes stomach acid. Specialization is also necessary to reduce the number and type of messages to which the cell can respond. There are thousands of different types of molecular messages active in a complex multicellular organism. Each cell responds to only a small subset. It would be worse than meaningless for a cell to retain the ability to respond to all these messages – it would be chaos. A general-purpose cell also would be susceptible to *all*

[30] A Complexity Drain on Cells in the Evolution of Multicellularity, McShea, D. W., *Evolution*, Vol 56, No. 3 Pages: 441-452, 1997. And, "The hierarchical structure of organisms: a scale and documentation of a trend in the maximum." McShea, D.W., *Paleobiology* 27:405-423, 2001.

[31] "Individual *versus* social complexity, with particular reference to ant colonies," Anderson, C & McShea, D. W. *Biol. Rev.*, vol 76, pp. 211-237, 2001. Note: the individual ants of highly social species are not only less complex, but more specialized and more varied in size and shape. Yet, as with specialized metazoan cells, they all have the same genetic complement.

[32] See "An overview of Gene Control" at www.ncbi.nlm.nih.gov/books/bv.fcgi?tool=bookshelf&call=bv.View..ShowSection&searchterm=control&rid=cell.section.1981

viruses. Viruses infect a cell by binding to surface proteins. Since different surface proteins characterize different specialized cells, each kind of virus can infect only certain types of cells. The cold virus infects cells in the nasal passages, the hepatitis virus infects certain liver cells, the HIV virus infects certain cells in the immune system, and so forth. If every cell expressed all cell markers, any virus could bind to and infect all cells. Catching a cold would be fatal.

Finally, specialization is *deeply intertwined* with the other three principles. The orchestration of many kinds of simpler specialized elements requires a polymorphic messaging strategy in which each specialized cell may interpret received messages differently. Moreover, the existence of a higher-level entity with a physically co-located body or social group requires an additional sort of communication strategy, called stigmergy, in which messages are attached to specific locations in the body, or colony. And, because cells that have for some reason lost their specialization are dangerous to the organism as a whole, all cells must be capable of programmed cell death, or apoptosis.

Specialization in computing

Computers too are becoming more specialized both in hardware and in software. Routers are specialized for bandwidth and, increasingly, for different quality of service. Data base servers are specialized for I/O, caching, and query. Web servers are specialized for small transactions and throughput. Computational engines such as Linux clusters and the even more specialized machines such as IBM's BlueGene are specialized for parallel operation and FLOPS. Portable devices such as PDAs, cell phones, and MP3 players are specialized for low power and long battery life. Game boxes are specialized for rapid graphics calculations. And the many embedded devices such as those in cars are further specialized for reliability, durability, precise real-time control, and the like.

Specialization in computing is *possible* for at least two reasons. First, because the various roles computers are asked to play in modern life have become more specialized. The role of a router is nothing like the role of an iPod. The computing industry continues to provide more options in both hardware and software at different costs. CPU chips vary in cost from less than a dollar to several hundred dollars depending upon speed, function, and power usage. Different types of memory, disk, display, and communications are also available at different prices. Software options affect costs too. For example, the PalmOS, which is popular for PDAs and some cell phones, requires considerably less compute power and memory than its Windows CE competitor. Hence a Palm PDA can last longer on a battery charge than a WinCE PocketPC device. Software

royalties are also a direct cost that can be avoided, e.g., by using Linux. The second reason specialization is possible is the growing multicellularity of computing – specialized computers can rely on other specialized computers for services it does not provide itself. No longer does one computer have to do everything. That's the point of Service Oriented Architectures (SOA).

Specialization is *necessary* for at least three reasons. First, many of the specialized requirements are incompatible. A PDA or cell-phone, for example, must run on minimum battery power whereas a computation engine must expend power with abandon in order to maximize FLOPS. Second, excess generality, especially in software, imposes costs on the manufacturer in terms of size of the software engineering team needed for development and maintenance. Software that supports a larger than necessary set of functions also lengthens time-to-market, is almost inevitably more buggy, and thus requires a larger and more expensive customer support staff. Finally, the more function a system supports, the larger the "surface area" exposed to attack by viruses, worms, spyware, sniffers, and other malware. It is no accident that most "malware" enters our systems by first infecting a general-purpose end-user PC.

Factors opposing specialization

> At first blush, there appears to be one glaring exception to increasing specialization in computing. Windows PCs continue to agglomerate all possible function. That is deceptive. Windows is the pinnacle of single-cell computing, not the basis of multicellular computing. Windows is specialized for the goal of maintaining its monopoly by being all things to all people. Microsoft's acknowledged ambition is to have Windows functionality, APIs and proprietary protocols in every server, PC, cell phone, PDA, MP3 player, and game box on the planet. The very opposite of specialization. As the evolution toward multicellular computing gains speed, that ambition is doomed! An API compatible Windows for all these applications is required to support a wide range of general behavior, making it difficult to specialize and making it a large monoculture that is a ripe entry point for Internet viral and worm infections. Windows supports a rich ecology of third-party applications that rely on the large Windows marketplace as well as the Windows API. That ecology is strongly supported by many programmers who have, at great cost in time and effort, learned how to use the Win32 API set. This self-reinforcing ecology guarantees that any transition to more specialized personal computers will be slower than it otherwise might be.

But Microsoft is not the only factor opposing specialization. Computing systems differ from biological systems in important ways. First, the elements of biological systems are self configuring, self protecting, and self optimizing in ways that computing systems are not. IBM's efforts to create *autonomic* computing systems are, in large part, an attempt to redress this deficiency. But those efforts will move slowly. In the mean time, we must configure, provision, and maintain our systems by the mostly manual efforts of IT professionals (or gifted amateurs). The more specialization in computing, the more specialization will be required in already hard-pressed IT staffs. So, until computers are primarily self-configuring there will be a countervailing force against specialization that is not present in biology.

Economic forces and user needs also tend to counter specialization. A few years ago there was a flurry of interest in what then was called "Thin Client." That meant, in effect, a replacement for the ever-present Wintel PC that would be specialized to support the needs, and *only* the needs, of the knowledge worker. That is, a specialized PC. Other somewhat related efforts have come and gone over the last decade. These efforts come up short in part because of the economies of scale of a one-size-fits-all PC, and in part because it isn't as easy as it seems to determine what functionality the employees really need, especially given the rapid innovation in computing that tends to emerge first in the Wintel ecology (e.g., P2P and VOIP). A similar issue has surfaced more than once with respect to Microsoft Office. Most external commentators (and many inside Microsoft as well) admit that Office has grown obscenely bloated. Few users actually use more than a small percent of all that functionality. The problem is that there is little if any consensus about which functions are essential. Nor can IT directors easily agree on exactly what functionality a putative Thin Client should support. The recent emergence of an "office" Linux PC is shaking things up a bit because, by starting from scratch, the question becomes what functions do we need to *add*, not what can we *subtract*.

So, we will see accelerating specialization in most, if not all, areas of computing, especially in small low-power personal computing devices and special-purpose sensor/effector devices. Worldwide shipments of new cellphones, PDAs and iPods already far outstrip those of Wintel boxes. Specialization moves forward whether or not the Wintel world cooperates.

Multicellular communication

Both living and computing systems are information processing systems. Necessarily, then, they must export and import information. Living organisms import information about their environment to find food or avoid danger. And their unavoidable need to export biochemical byproducts of their metabolism provides information to surrounding organisms. Computers likewise must import and export information. Isolated computers used to communicate in various ad hoc ways determined by their programmers without much regard for the needs of other computers – hence the thousands of proprietary binary formats developed over the years. Because multicellular organisms and computers are specialized, hence their behavior requires more orchestration, the way they exchange information has become far more important. Cooperation between specialized elements requires complex and selective messages.

Both biological messages and digital messages are transmitted by linear sequences of interchangeable elements – "alphabets" if you will. Cells use messenger molecules constructed of chains of simple chemical subunits. Computers use messages that are sequences of bytes.

Life has evolved two sorts of chain molecules: proteins, which are chains of amino acids, and DNA/RNA, which are chains of nucleotides[33] that carry the genetic 'program' of the cell. DNA/RNA differ from messenger proteins in their relationship to the cellular CPU (its nucleus in which complex control mechanisms determine which chunks of genetic material will be expressed). DNA and RNA are “executable” by the cellular machinery whereas proteins are not. Transfer of messenger proteins causes the cell to select behavior from its existing repertoire whereas transfer of genetic material changes the repertoire itself.

Digital messages in computing are strings of bytes that range from idiosyncratic binary codes to highly structured XML messages such as SOAP or other Web Services messages. Some strings are executable and some are not depending on the computer’s CPU (or interpreters for scripting languages such as Java or ActiveX). Thus, both life and computing have evolved two forms of complex information media: one executable and the other not.

The distinction between the two kinds of message is central to communication strategies both in biology and in computing. The parallels are instructive for understanding multicellular computing. Whereas single-cells and computers benefit by exchanging executable code, code exchange in multicellular systems is dangerous. Polymorphic non-executable messages are far better suited to multicellular systems. In fact, DNA exchange is shunned by multicellular life. We are learning about the dangers of exchanging code the "hard way" as we cope with increasingly dangerous digital viruses and worms in multicellular computing.

Communication in biology

There is no more stark divide between single-cell and multicellular organisms than the way they pass information from one to another. Single-cell and multicellular organisms use almost completely opposite strategies for communication. Most information communicated directly from one single-cell organism to another is passed by transferring DNA (a process known as conjugation). In contrast, multicellular organisms exchange DNA *only* in the process of sexual reproduction. This rule against multicellular genetic transfer is so universally obeyed that

[33] Protein chains tend to be from a few dozen to a few thousand amino acids in length. Functional RNA chains can be as short as 10-20 units or as long as a million units. Bacterial DNA molecules are a few million bases long. Human chromosomes, each of which is one contiguous chain of paired nucleotides, range in length from 50 million to about 250 million base-pairs.

Loewenstein[34] calls it "…the taboo of intercellular transfer of genetic information." Virtually all cell-to-cell communication in multicellular organisms is transmitted by protein messenger molecules.

That is the state of affairs today. There likely was a lengthy transition period in the evolution from single-cell to multicellular life in which cells continued to use DNA and RNA for communication in addition to a growing dependency upon protein messenger molecules. Over time, though, organisms that did not obey the taboo against exchange of DNA lost out in the contest of survival of the fittest.

Communication by DNA transfer

Bacterial DNA transfer is a normal and powerful means for single-cell organisms to communicate information about new ways to compete and survive in their shared chemical environment. It allows successful mutations to quickly spread to a large population of bacteria. For example, DNA transfer is responsible for the rapid spread of antibiotic resistance among bacteria. DNA transfer does not just occur between cells of the same species. Perhaps 25% of the E. coli genome has been acquired from other species. Such cross-species gene transfer speeds the spread of new traits by a factor of 10,000.[35]

We do not have to look far to understand why multicellular organisms shun DNA transfer. Importation of DNA essentially "reprograms" the cell.[36] If such a transfer turns out to be beneficial to a single-cell organism, so much the better; it survives to pass the new DNA on to its progeny. If not, it dies, is quickly replaced, and is mourned by no one. Multicellular organisms are made up of differentiated cells that all contain the same complement of DNA. Specialized Metazoan cells may have completely different function from nearby cells. These differences are required for the survival of the organism. Transferring active DNA between cells would undermine differentiation in unpredictable ways. For example, a motor neuron touches many muscle cells in order to direct their contraction. Imagine the chaos if a muscle cell could inject its active DNA, which makes contractile proteins, into the nerve cell. Then the nerve, rather than telling the muscle to contract, would itself contract detaching it from the muscle. Moreover, even if such DNA transfers provided potential evolutionary advantage, they would be of no long-term value unless they could somehow make their way into the germ line and be passed on to progeny.

Communication by protein messenger molecules

Metazoan cells, instead of exporting DNA, export messenger molecules, primarily proteins, which bind to receptor proteins on the surface of other cells. These messenger molecules cannot reprogram the receiving cell. In fact, they cannot even guarantee a given behavioral response. A particular messenger molecule, in general, elicits different behavior in different receiving cells. Insulin, for example, triggers very different responses in skeletal muscle cells, two kinds of fat cells, and cells in the liver. In computing terms, therefore, these messages are *polymorphic*, i.e., their meaning is determined by the receiver. Because single-cell organisms function for the benefit of only one kind of cell (themselves), they do not need polymorphism whereas multicellular organisms cannot do without it.

Cooperative communities of single-cell organisms, e.g., biofilms, also exploit molecular messengers, if only in a simplified manner. For example, "quorum sensing"[37] bacteria sense the concentration of a kind of messenger molecule that they, themselves, export. When enough

[34] See "The Touchstone of Life, Werner Loewenstein, Oxford University Press, New York, 1999 [p. 277]
[35] Science, vol. 305 no. 5682, pp. 334-335, July, 2004
[36] In the extreme case of a viral infection, it completely hijacks the cell and often kills it.
[37] See www.nottingham.ac.uk/quorum/what.htm

similar bacteria are present, the concentration rises and all of them sense a "quorum." They then change their gene expression behavior, for example to turn on the production of virulence factors. When acting cooperatively in real-time, their most important communication is via messenger molecules whereas, when acting as separate cells, they exploit DNA transfer via conjugation. When a virulent biofilm, e.g., of Staphylococcus or Salmonella is treated by antibiotics, any bacteria that happen to survive because they are more resistant to the drug are left as free single-cells that can then pass on their resistance by conjugation.

Interestingly enough, social insects echo the communication strategies of cells much the way they echo the consequences of specialization. Ants and termites, for example, use polymorphic molecular messages to organize their specialized behavior. Differently specialized ants, e.g., workers and defenders, respond differently to chemical markers (*pheromones*). Pheromones, are simply intercellular molecular messengers that happen to be exported into the environment. They play important roles in the behavior of many species, including mammals and probably humans[38].

Communications in multicellular computing

Computers send messages via strings of bytes rather than strings of chemical subunits. The bytes may represent either executable code or data. These days, executable code may be in a variety of formats. Of course, .exe files are directly interpreted by the Wintel hardware/OS. On other platforms there are comparable executable formats. Various scripting languages become executable too if the right interpreter is available. Data can be in many forms including binary formats agreed upon by the two communicating computers. Graphics, video and audio formats abound. And newly emerging Web-Services exchange messages in the form of structured XML dialects.

The need for a Taboo against transmitting code

Multicellular computing is beginning to evolve a taboo against code transfer. Computers have "owners" that set the agenda of the machine and generally frown on having their computer hijacked by whoever happened to sneak in rogue code. Many organizations, most especially sensitive government or financial organizations, already do their best to enforce such a taboo. Enforcement is complicated by the fact that it is not always possible to separate code from data, especially when end users can still be so easily tricked into executing an email attachment. Code sneaks in as supposedly innocuous files like .jpg images and gets executed as a result of a buffer-overrun exploit.[39] For the foreseeable future, the computing industry will continue to improve its enforcement efforts and attackers will continue to sneak code past the defenses by ever more clever tricks. The only strategy likely to settle this arms race is a basic change in system architecture that presupposes a taboo.

Specialization tends to work against code transfer. As in the case with multicellular organisms, each machine in a community of specialized machines may function in a very different context. Code for a router is meaningless in a PDA or a parallel computation engine. The Windows monoculture, with its common APIs, permits meaningful, if dangerous, code transfers. But without such a monoculture to provide a common context across machines, it simply makes little sense to base multicellular computing on transmitting code.

[38] See http://www.cf.ac.uk/biosi/staff/jacob/teaching/sensory/pherom.html for a review of the evidence of human pheromones.

[39] Note that new hardware capabilities from Intel and AMD allow memory to be marked "no execute" which will eventually make buffer-overruns less of a problem. See for example: msdn.microsoft.com/security/productinfo/XPSP2/memoryprotection/execprotection.aspx

What, then, can we say about computing paradigms that fundamentally depend upon the transmission of code? For example, robots, whether on the surface of Mars, in orbit around Jupiter, or in a battlefield situation often require code updates on the fly. But such robots are simply not going to be connected to the open Internet (except perhaps as toy examples for the enjoyment of netizens). Communication with battlefield robots is no doubt encrypted and secured in multiple ways. Still, history shows that such efforts are never perfect. More generally, computer scientists have spent a decade or more exploring mobile agents that move code from one machine to another. Mobile agent systems can probably be sufficiently guarded if they are used with specialized hardware and operating systems rather than general-purpose Windows clients. Finally, a small industry is growing around the notion of cycle-scavenging, light-weight peer-to-peer (P2P) grids for distributed computing that require the transmission of code. These P2P scavenging systems may be protected sufficiently given that they tend to be aimed only at strictly computational problems to be run in the background. As long as they are carefully designed to suspend themselves as soon as the user needs the CPU, they *may* not represent a risk. Nonetheless, the whole point of such systems is to scavenge cycles from millions of Windows machines. That is, they depend upon a large monoculture which is an inherently dangerous climate. Moreover, CPU cycles are getting increasingly cheap and light-weight grids are useful only for a narrow range of "embarrassingly parallel" problems. So one wonders why taking the risk is worthwhile.

Communication by polymorphic non-executable messages

As code transfer loses popularity, a different and rapidly growing trend, Service Oriented Architectures (SOA) and Web Services, is gaining popularity. It mimics living system's use of polymorphic message sending. The meaning of all SOA and Web Services messages is determined by the receiving computer, not the sender. So the multicellular computing world seems already to be evolving the same basic architecture that evolved in biology. Today, it looks like future orchestrated collaborations between computers on the Internet will be based upon one variety or another of SOA. Heavyweight SOA, based upon SOAP, WSDL, and a host of other standards, is gaining favor in corporate IT systems. Lighter weight SOA mashups, e.g., those based on AJAX are growing rapidly in the Internet at large. In either case, useful Web Services are emerging on the net and multicellular combinations of Web Services are becoming much more common. Still, legacies live on. We must coexist for many years with legacy systems that use proprietary communication protocols, old EDI protocols, HTML protocols, and many less common formats. If biology is any guide, many of these will never fully disappear. They will become frozen historical accidents to puzzle future students of the history of computer science.

The evolution of semantics

The *form* of future collaborations between computers seems on its way to being settled, i.e., by moving towards SOA. But the substance is not. Polymorphic messages encoded in XML are *syntactically* but not *semantically* self-describing.[40] If polymorphic messages are to be the basis of communication, there has to be some agreement on the *meaning* of the messages. SOA messaging semantics will be sorted out by various standards bodies, conventions, and much evolutionary trial and error. None of these are completely satisfactory – standards are slow, conventions conflict, and evolution is messy. But, if biology is any guide, evolution will dominate.

[40] XML does *not* encode semantics. Only the syntax is self describing. When people read XML, they perceive the human-readable tags as carrying semantics. But those words mean nothing to the receiving computers. Semantics remains in the minds of the reader and writer of the text, not in the XML itself.

Evolution of messaging semantics in multicellular organisms operates upon all aspects of the message process at once. An individual organism begins with a fertilized egg that divides. Its daughter cells all share the same DNA. These daughter cells differentiate, staying physically co-located. Hence, the organism's DNA codes for ***all*** aspects of the messaging behavior: the behavior of the "sending" cell, i.e., how and when it synthesizes and exports a given messenger molecule, the shape and lifetime of the messenger molecules,[41] and the behavior of the "receiving" cell(s), i.e., which ones have binding sites for a given molecule and what pathways are triggered by receipt of the message. If the semantics of the resulting message transfer are not beneficial or at least neutral to the health of the organism, the whole organism is at higher risk of death before it can pass the DNA on to the next generation. Thus survival of the fittest, which operates at the whole organism level, simultaneously punishes poor "syntax" and muddled "semantics" by culling mistakes at either end of the communication.

A single corporate network infrastructure may play an evolutionary role similar to that of a single multicellular organism. That is, it is a unitary, logically contiguous, dedicated collection of computers that supports the whole organization, for good or ill. Its routers, firewalls, VPN links, LDAP servers, DMZs, and specialized application servers must share an agreed upon and internally compatible architecture and implementation. And the semantics of pairwise collaborations must be sensible. The corporate IT staff work to ensure this. If the system is not up to the job, disaster may well ensue for the whole organization. A corporation with a seriously flawed infrastructure may well go out of business, thus failing to pass on its ineffective infrastructure architecture and semantics. Bank mergers are a classic case. A bank's competitiveness often depends in large part upon its IT infrastructure, which is a carefully (or not so carefully) crafted multicellular computing organism. The weaker bank, when acquired by a stronger one, typically remakes its IT infrastructure to be compatible with the winning bank's architecture. This sort of system evolves its messaging architecture in a manner similar to that of a multicellular organism, by a remorseless process of survival of the fittest.

The evolution of message semantics in the open Internet is more complicated. A person's computer may play a different role in many different sets of multi-computer collaborations, some for private uses and some in their role as an employee, customer, or supplier in business relationships. This is similar to the more fluid and ambiguous semantics in ecologies of organisms where a single organism plays many different roles in different ecological collaborations. Predators recognize prey, and vice versa, by all sorts of chemical signals. So, chemical signals mean one thing between individuals of the same species and another to their predators or prey. For example, the musky smells of animals travel on the wind. Thus predators attack from downwind so as not to warn the prey. The "come hither" scent of a desert mouse is intended to attract a mate. That it also attracts a hungry snake is not just an unfortunate accident. Snakes coevolve with the mice. In a similar manner, especially valuable services (e.g., Google, eBay, Amazon or SalesForce.com) support an API with defined semantics that attracts third party services. Or, services with plentiful and demographically valuable users attract competing services that will offer the same API with the same semantics to attract those users. Successful poachers of either sort can then add to the API and the semantics in an attempt to freeze out competitors (this is the familiar Microsoft "Embrace, Extend, and Extinguish"[42] strategy). Such coevolution can result in richer and more useful semantics.

[41] Intracellular mechanisms degrade almost all proteins, some quite rapidly. The rate at which degradation occurs depends in large part upon the precise sequence of the amino acids. The resulting half-life determines their range and the duration of their effect – that is, messenger half-life is an explicit aspect of control that evolves along with the sending and receiving cells.

[42] http://en.wikipedia.org/wiki/Embrace,_extend_and_extinguish

Some efforts, such as UDDI[43] have attempted to provide a semantic "yellow pages" service as a rendezvous point for finding services with the "right" semantics and to organize such semantic information into useful taxonomies. So far, these efforts have been premature, over-engineered and overly complex. It has been like attempting to start a telephone Yellow Pages before there are enough phones to matter. So, the semantics problem in the Internet remains difficult.

Alan Kay [44§] proposes that the semantics travel with the message by using objects as the message carrier rather than just data. An object carries both data and the code that provides some of its meaning. Security issues would be handled by assuming that the machines are specialized as object engines that provide separate address spaces for all objects so that one cannot interfere with another. However, object collaboration still requires some shared understanding of the semantics of the polymorphic messages. Reflexive systems, those in which metadata is available with which objects can "reason" about message interfaces of other objects, might be agreed upon and such reasoning might support the bootstrapping of enough semantics to dramatically speed the evolution of useful ways of collaboration. Or, SOA brokers could match up "compatible" parties. This object approach might offer real advantages, however it is a radical departure from the way our current machines work and would require substantial evangelism and investment to succeed. Nonetheless, one day perhaps some classes of computer applications might work very well in general object engines.

The power and accessibility of interpreters

Computers march to the beat of more than one type of code. The CPU defines the machine language. But today most computers also run interpreters that execute different sorts of script or byte-codes. In addition to Java and C# byte codes, end-user machines may well also support Javascript, Perl, PHP, Python, Ruby and many others. Even Postscript and some image formats are interpreted. All of these transform data to executable code if the right interpreter is available. And all can be dangerous. For example, Javascript recently has been shown to open cross-site scripting vulnerabilities.[45] Javascript is vital to SOA or Web 2.0 implementations using the increasingly popular AJAX techniques.[46]

Interpreters are dangerous in large part because they may allow unfettered access to the OS's API set, ActiveX controls, disk storage, the TCP/IP stack, the keyboard, the screen or even the PC's microphone. All of these can provide exposure to misuse by malware. Access to the TCP/IP stack enables the machine to replicate worms and be hijacked to implement zombie DDoS and other denial of service attacks. Access to the keyboard enables spyware to sniff passwords and credit card numbers. Ability to write to the screen can enable phishing or spoofing. Access to the microphone can enable a browser application to capture sounds in the room. For example, Google has been experimenting with determining what TV shows the user is watching from snippets of sound captured by the PC's microphone.[47]

---

[43] See www.uddi.org/. This approach is part of a more general strategy for harnessing human abilities (as the final arbiter of true semantics) to electronic brokering of that information. See "The Tao of e-Business Services," S. L. Burbeck, 2000, www.ibm.com/developerworks/webservices/library/ws-tao/

[44] This paragraph is based on a personal communication with Alan Kay. Multicellular computing was the context for that discussion. He has often voiced the view that "dumb" data should be avoided.

[45] See http://news.com.com/JavaScript+opens+doors+to+browser-based+attacks/2100-7349_3-6099891.html?part=rss&tag=6099891&subj=news

[46] See http://en.wikipedia.org/wiki/AJAX

[47] See http://www.technologyreview.com/read_article.aspx?id=17354&ch=infotech

In the single-cell computing world, the machine's hardware and software facilities were assumed to be under the control of the local user of the machine. In a multicellular world, they can too easily become extensions of a remote multicellular system without our knowledge or consent. We must therefore improve our ability to control (or neuter) interpreters so that their behavior cannot compromise the computer. One approach might be to provide better low-level access protection or permission facilities. The familiar Unix "permission" structure, in which applications and users have predefined permission to read, write or execute files, was designed to protect a single-cell world where the disk was the central, key resource. From a multicellular computing perspective, communication traffic between computers and between the computer and its user, not management of its file system, will usually be the most important task the computer does. Hence communication between machines (e.g., TCP/IP) and with the user (keyboard, mouse, screen, microphone and speakers) should be controlled with a revamped permission structure appropriate to modern attacks on networked computers.

The issue of loading code

The historical notion that what one does with a computer is load it with code early and often is outdated in this brave new world swarming with viruses and worms. That attitude assumed that the person loading new code was an expert that knew what the code did and where it came from. Those assumptions still apply today in IT organizations that maintain corporate servers and perhaps in the case of skilled amateurs maintaining their own PCs. They emphatically do not apply to the average computer user let alone the average cell phone or PDA user.

What is needed is a new perspective based upon the principle that loading code, while essential for the operation of a single computer, is fundamentally threatening to the multicellular computing system of which it is a part. Until very recently, virtually all our practices for installing and maintaining software were legacies of the single-cell computing world. The programming priesthood would determine what code should be installed on important machines and the rest of the users were largely on their own. Then corporate networks found themselves compromised by end-user machines and we began to recognize the risks of loading code onto *any* machine within the firewall. Lots of stopgap measures have been tried including individual firewalls for each machine, encrypted wireless nets, anti-virus software, rules and more rules. Yet little has been done in the way of fundamental rethinking and each month brings discoveries of new security holes not covered by existing band-aids.[48]

Biological systems do not provide much insight into this issue since all cells in all multicellular organisms begin with a full complement of DNA. Loading code is a problem unique to computing systems. Other than the few embedded computers for which all code comes in ROM, code must be installed before a computer can do anything. With current practices and economics, we must also maintain and enhance the code, e.g., distribute and install patches, new versions, or new applications. In the long run, many computers may be so cheap and ubiquitous that maintenance, patching, and enhancing their code is neither practical nor worthwhile – the user simply throws away the computer (e.g., a cell phone) and buys a new one. However, at present and for the foreseeable future, we must continue to deal with code distribution, patching and versioning.

[48] Microsoft made claims that Vista would make major contributions to this issue. So far, however, Vista has not fulfilled such claims. As of April, 2007, several serious security holes already have been found. Instead of fundamental rethinking, it appears that Microsoft has merely redoubled their efforts to round up the usual suspects and offloaded responsibility for permitting code installation onto woefully unprepared end-users.

Stigmergy and "self" in multicellular systems

As multicellular computing becomes more and more complex, we face an increasingly difficult problem of protecting large diverse systems, such as corporate IT systems, from intruders, viruses, worms, denial-of-service attacks, etc. One idea that comes up periodically is to somehow mimic an immune system. Yet that's not quite as straightforward as people often assume. The underlying assumption in the immune system metaphor is that a biological "self" is a collection of cells with the right genetic IDs. Therefore, determining self is just a matter of "checking the IDs" of all the cells, perhaps via certificates or other nominally incorruptible tokens. However, that is a misreading of what "self" versus "other" means in the biological world. While it is indeed crucial for a Metazoan to distinguish its own cells from *predatory* bacteria or virus infected cells, that is only part of the story. It turns out that most multicellular organisms include cells other than those directly related to the metazoan's cells, e.g., many species of symbiotic bacteria and fungi.

For a single-cell organism, the boundary of "self" is straightforward. It includes the outer cell wall and all the structures, e.g., organelles, that exist within the cell wall. Inside is "self" and outside is "other." For multicellular organisms, the answer is more complicated but it comes down to one simple fact: the organisms share a *physically co-located structure* – their "body."

A multicellular "self" begins with a fertilized egg. As this single cell divides repeatedly according to its developmental program, the organism grows. In the process, the living cells create or assemble nonliving structures that provide form, cohesion, containment, stiffness, moving parts and protection. Examples include bone, sinew, connective tissue, fur, shell, skin, scales, chitin, bark, wood, and all manner of other non-living extracellular material. That is, Metazoan cells construct and continually maintain the very bones, sinews, etc. that help to protect, organize and provide the physical structure of the multicellular body. This body, together with all the cells that live within it, defines the self. Therefore, a multicellular self is a *unit of benefit* in the competition for survival of the fittest. Although only the germ line is passed on when an individual survives long enough to reproduce, all the cells in a multicellular organism, together with their non-living constructs, compete as a unit and therefore live or die as a unit. The body of such a multicellular "self" is a *stigmergy* structure that is created by the organism's cells. That means that the cells build the body and the body, in turn, helps to coordinate the actions of the cells. We will explore the notion of stigmergy in considerable detail below. The important point here is that evolutionary processes select for the fitness of the whole organism, i.e., the whole stigmergy structure.[49]

Defending the "self"

Multicellular organisms are under constant attack from viruses and various single-cell organisms. Clearly, multicellular life evolved surrounded by these predators. To this day the cells of all the Metazoan organisms on the planet are far outnumbered by single-cell organisms that are, in turn, far outnumbered by viruses. Without powerful defenses, the multi-cell organisms would be overwhelmed very quickly. So multicellular organisms evolved defenses based upon distinguishing self from "other."[50]

[49] Note that this is a mechanistic definition of self that does not, and is not intended to, deal with philosophical, psychological or theological notions of a human *conscious* self. We are talking about the body, not the mind or the soul.

[50] The issue of maintenance of "self" is called morphostasis. One useful discussion of morphostasis can be found in www.morphostasis.org.uk/Papers/Evolving.htm

The common belief is that "self" is determined by the immune system. That is only partly true. While the immune system "identity" of individual cells is helpful for defense, multicellular organisms did not evolve to support a more perfect *defense*. They evolved to support a more perfect *offense*, i.e., to exploit the evolutionary advantages of added complexity and specialization.

Genetic identity of all cells in the body is *not necessary* nor even desirable within a multicellular organism because most multicellular organisms benefit from symbiotic relationships with a wide variety of single cell organisms that live within them. Complex organisms are typically ecologies in which many various species of single-cell organisms play vital cooperative roles. In humans, at least a thousand species of bacteria and yeasts cohabit with us and play beneficial roles.

> "It has been estimated that there are more bacterial cells associated with the average human body than there are human ones and one of the most important functions of our normal flora is to protect us from highly pathogenic organisms. … A few of our normal flora produce essential nutrients (e.g., vitamin K is produced by gut flora) and our normal flora may prevent pathogenic microorganisms from getting a foothold on our body surfaces. ... Like it or not, we have large amounts of many types of bacteria growing almost everywhere in and on our bodies. About 10 percent of human body weight and 50 percent of the content of the human colon is made up of bacteria, primarily the species known as Escherichia coli, or E. coli."[51]

These single-cell protectors are often the first line of defense against infection. Our formal immune system only comes into action when the first line fails. An immune system that insisted upon destroying these symbiotic organisms would destroy itself.

Nor is the genetic identity of cells in Metazoans *sufficient* to determine self. Starfish generate new selves from pieces of themselves when dismembered. Many plants generate new selves from cuttings. And human identical twins have identical genetic makeup. In all these cases, multiple distinct selves share the same DNA. That is, each of these genetically identical selves benefits separately from surviving long enough to pass on its genes to the next generation.

Stigmergy: the organizing principle in multicellularity

Much of the communication between cooperating entities (cells, social insects or computers) is indirect. The entities deposit cues, i.e., persistent messages, in external structures – connective tissue, termite mounds, or databases as the case may be – that help to organize the behavior of other entities that later encounter these cues. The information embedded in external structures is created or organized by the collective action of the cells/computers/insects whose behavior is then, in turn, organized according to the persistent embedded information. The term stigmergy was coined in the 1950's by Pierre-Paul Grasse[52] to refer to the reciprocal relationship between social insects and the structures they build, e.g., termite mounds, ant hills, beehives and even the pheromone-marked ant trails of nomadic ant species such as army ants.

Although the term was coined to describe social insect behavior, the "active" elements and the "passive" structures they build arose during the evolution of multicellularity. The very bodies of all multicellular organisms are stigmergy structures. In the earliest and simplest multicellular organisms the extracellular matrix may be nothing more than a "slime" excreted by the cells that forms a clump or thin film in which the cells live and through which messenger molecules diffuse from one cell to its neighbors. More complex multicellular organisms have much more complex extracellular matrix structures that support more subtle complex communication. The cells

[51] From bioweb.wku.edu/courses/BIOL115/Wyatt/Micro/Micro2.htm

[52] See "Self-organization in social insects." Bonabeau, E., Theraulaz, G., Deneubourg, J.L., Aron, S. & Camazine, S., *Trends in Ecology and Evolution*, vol 12, pp. 188-193, 1997.

deposit all sorts of molecular cues in the extracellular matrix that influence the behavior of other cells. Plants create rigid stigmergy structures made largely of cellulose and other complex sugars. Complex animals create connective tissue that gives structure to their various organs and generally holds their bodies together and protects it: molluscs create shells, insects create chitinous exoskeletons, and vertebrates create bone.

Social insects, cooperating cells, and cooperating computers communicate both with *signals* and *cues*. The distinction is that signals are active short-lived communication events whereas cues are information embedded in the stigmergy structure to be read and reread many times. Both are specialized messages in the sense that they mean different things to different specialized receiving cells (or insects or computers). However cues are further specialized by their location. In addition to the information intrinsic to their molecular form or digital content, there is also information inherent in their location in the stigmergy structure. Cues support more complex kinds of communication than do signals, hence cues play the larger role in complex ant societies whereas signals play the larger role in simple ant societies.[53]

As is the case with social insects, cells in multicellular organisms communicate both by signals (polymorphic messenger molecules moving indiscriminately through blood, lymph or other liquids) and by cues (polymorphic messenger molecules attached to the extracellular matrix). For example, bone, when stressed, provides cues to osteocytes and other bone cells for its own reshaping to better handle the forces placed upon it. And smooth muscle cells in the walls of blood vessels modulate their contractility according to cues from the extracellular matrix.[54] Not surprisingly, as with social insects, simple multicellular organisms communicate primarily by signals whereas complex multicellular organisms communicate more by cues. Here again, multicellular computing recapitulates biology.

Stigmergy in computing systems

Digital stigmergy structures are models populated with digital data by a distributed community of "clients" that also interrogate the current state of the data. That shared data plays an emergent organizing role in otherwise independent entities. Examples in single-cell computing include:

- The Windows Registry -- its model is a hierarchical key-value dictionary. That is, values can contain more keys. It is populated by and queried by various applications and by the operating system itself. Unix has similar structures that support cooperative behavior by cues left in structures accessible to the participants.

- Cut and paste buffers -- the model is a single (or sometimes a time-ordered set) of structured and typed data recognizable to one or more applications. The applications deposit this data when the user "cuts" or "copies" data from within an application. Any other application that recognizes the data type may "paste" it into its document.

- Blackboards in AI systems -- blackboards were first developed in the mid '80s for emergent communication and control of various expert-system inferencing subsystems. A blackboard is shared read-write memory in which assertions and/or goals are posted by the various inferencing "agents" and read by others. These days blackboards are finding application in first-person shooter game software.

[53] "Individual *versus* social complexity, with particular reference to ant colonies," Anderson, C & McShea, D. W. *Biol. Rev.*, vol 76, pp. 211-237, 2001. p. 228

[54] Extracellular matrix controls myosin light chain phosphorylation and cell contractility through modulation of cell shape and cytoskeletal prestress." Polte, TR, Eichler, GS, Wang, N, & Ingber, DE. Am J Physiol Cell Physiol 286: C518-C528, 2004.

Familiar examples of stigmergy in corporate intranets include databases, file servers, DNS servers, email servers, and all manner of routers and other communications structures. Consider for example a customer database for a corporation. Its model is typically a relational data model of customers, their orders, and their accounts. It is populated and read by sales personnel, the shipping dept., accounts receivable personnel, and perhaps to some extent by the customers themselves if there is a Web interface for orders.

Most complex computing systems support persistent organization such as corporations, universities, governmental agencies or large web services such as Google, Amazon, eBay, Yahoo, and the like. Corporations are not only a type of collective "self," recognized in law, but also are a "self" based upon their stigmergic structures. The people in the company create external structures that serve, in turn, to help organize their activity. These typically include buildings (offices, factories, warehouses, etc.), equipment (ships, trucks, milling machines, desks, copiers, and computers), records (the "books," the contracts and other documents), and persistent financial structures such as bank accounts and shares of stock and bonds issued by the corporation. Increasingly today the most important structure in a corporation is not the bricks and mortar, but the IT infrastructure – the physical and logical networks (VPNs), databases, and applications that manage and transform business-critical information.

As with the temptation to think that the biological self is defined by the immune system, "selfness" in computing systems is often misconstrued to be about the identity of the connected leaf devices and the identity of authorized users. Yet the identity of a given machine can't be the determining factor since machines can be lost, stolen or compromised by a virus or worm. Nor can the identity of the person be the determining factor since people move from company to company, they forget their passwords, they leave their passwords around on sticky-notes, or they choose trivial easily guessed passwords.

For these and other reasons, we are beginning to recognize that the perimeter of a corporate network, i.e., the collection of PCs and other personal devices used by employees, is inherently indefensible. Moreover, the perimeter is also becoming undefined because it now intersects with supplier and customer systems and the corporate employees themselves work remotely, sometimes in a disconnected mode. Yet just inside the indefensible perimeter lies the core of the corporate infrastructure, i.e., the definable and defensible network, databases, and corporate application servers. That core stigmergy structure is the corporate IT "self." The fundamental job of the staff in an IT organization is to maintain this vital corporate computing infrastructure. They play a role similar to ants maintaining the nest, or bees maintaining the hive.

That does not mean that IT staff can abandon the perimeter – certainly not today. While the IT infrastructure can, in theory, be defended, its defenses are not yet sufficiently strong to ignore the perimeter. The network is susceptible to denial-of-service attacks if nothing else. All too many databases and corporate applications are protected primarily by passwords, with all their known weaknesses. All possible efforts must still be bent toward minimizing the danger represented by the perimeter devices, especially Windows machines that are the target of almost every attack. One day, however, the IT infrastructure will protect itself even when the perimeter devices are compromised. To that end, IBM and Cisco are collaborating on a new approach based on a smart network defense, i.e., a self-defending network. Defense is enforced at first level routers, hubs, and access points – that is, at the edge of the IT stigmergy structure.

Novel stigmergy structures in the Internet

Stigmergy in computing is about creating a persistent self. However "self" in computing isn't about the persistence of a physical body, it's about the persistence of a reliable means of communication between the individual machines and a reliable way to store the persistent data

required for the organization to survive and grow. That is, stigmergy in computing is about persistent structures in a world of bits not atoms. Since the world of bits changed dramatically with the birth of the Internet and then the World Wide Web, notions of stigmergy in computing are changing too.

Linux, for example, can be thought of as a software termite mound. The stigmergic structure is provided by the "blessed" Concurrent Versions System (CVS) code repository servers that are under the control of the Linux "inner circle" (Linus Torvalds, Alan Cox, etc.). Clearly, the Linux CVS code tree is built by and helps to organize the efforts of a worldwide community of Linux programmers. While Linux is perhaps the best known example of an open-source stigmergy structure, Apache web server software, MySQL database software, Mozilla browser software (in its new and very popular Firefox incarnation), and OpenOffice (a Microsoft Office replacement) are having a similar impact. Before the success of Linux and Apache, it was pretty much taken for granted that large-scale, robust software had to be built by a corporation dedicated to its success that provides a co-located stigmergy structure of servers, programmers, buildings, marketing arms, PR flacks, and large financial resources. However, Linux, Apache, MySQL, Mozilla and OpenOffice all directly compete with key portions of the Windows monopoly that is supported by Microsoft's well funded organization. They are doing well enough to strongly suggest that an open, distributed development structure has strengths that were previously underestimated.

The open Internet supports many other public stigmergy structures that are collectively managed and used by humanity as a whole[55]. The most obvious examples are, of course, web sites which are persistent and which organize human browsing behavior via links. Other examples include:

- Web search (e.g., Google) where the stigmergy structure is their huge infrastructure of crawler data plus the crawlers and servers that create and use it. The search engines react as humans change the Web and human browsing is, in turn, organized by the results. Many people these days go to Google rather than manage bookmarks trusting that the site they want will come up near the top of the search results and enjoying the serendipity of exploring other top links.
- MMORPGs -- Massively Multiplayer Online Role Playing Games such as World of Warcraft, EverQuest, Second Life, and many others are based on servers that maintain "models" of their virtual worlds (in the Model-View-Controller sense) acting as the stigmergy structures around which tens of thousands of online gamers organize their play. The game model is altered in "real-time" by the behavior of the players and the player's behavior is altered by the constantly changing model. WoW has some 6.5 million users and claim that they have supported over 250 thousand simultaneous users. EverQuest has at least 430,000 players and has supported over 100,000 simultaneous players. Second Life supports about 250,000 users on 2,579 dual-core Opteron servers organized in a semi-decentralized architecture
- Instant messaging communities (AOL, MSN and Yahoo) where the stigmergy structure is the "presence registry" that tracks who is online at any given moment. People who use instant messaging recognize how their behavior is modified by being aware of which of their friends and colleagues are accessible online.
- Free VOIP services (Skype) are similar to the instant messaging communities except that they communicate with higher bandwidth voice rather than just small text messages. The ability to set up free phone conferences via Skype with up to five people anywhere in the

[55] "As a whole" is an exaggeration. Only the digitally literate can participate. Moreover, more or less successful attempts have been made to balkanize the Internet into censored and controlled subnets, e.g., in oppressive countries such as China, North Korea, Singapore, Saudi Arabia and Iran.

world has the potential to change global business behavior, not to mention family communication.

- Free and open file sharing networks (e.g., Gnutella, eDonkey, Freenet, Grokster and BitTorrent), construct a virtual stigmergy structure comprised of a constantly shifting collection of client machines that support the appropriate set of P2P protocols. Each participating machine makes available some files. Music file sharing communities, together with the new end user iPod devices, are reshaping the way people relate to music and thus reshaping the entire music industry. They threaten to do the same for the movie industry.
- Public databases. For example, most medical and post-genomic bioinformatics research world-wide is completely dependent upon public databases such as GenBank, PDB, Pub Med, and BIND[56]. Most academic and/or public affairs disciplines, where data is usually assumed to be a public good, have similar sorts of communities organized around major databases. These databases, which are clearly stigmergy structures, often support easy-to-use, yet sophisticated domain-specific search functions. Social incentives, such as mandates by granting agencies are encouraging researchers to put their new data into these databases and most researchers focus much of their activity around use of that public data.
- Blogs and Wikis are other examples of public stigmergy "selves." Wikipedia, for example, is an open encyclopedia with about 400,000 entries that is created and maintained by users all over the world.
- Social networking sites such as MySpace and Facebook help form and organize human digital social groups.
- Folksonomy sites such as Flikr, del.icio.us, and YouTube collect and organize tags for unstructured data. Folksonomy stigmergy structures consists of both the data (e.g., photos, bookmarks, or videos) and the database of tags that support search. Thus, while the data and the tags are each stigmergy structures, their interdependence strengthens both.

Note that all of these novel communities are organized around new stigmergy structures, i.e., new selves, of a sort that didn't exist previously. Their worldwide scope, speed, and enablement of new kinds of interaction emerge out of the scope, speed, and protean flexibility of the Internet.

The rapidity with which open Internet-based systems mutate provides an interesting challenge for corporate IT infrastructures that inevitably mutate slowly. One of the goals of "On Demand Computing" is to make corporate infrastructures more flexible so that they can evolve to exploit changing conditions more quickly. Yet, ad hoc P2P networks are already "On Demand" in the sense that their infrastructure grows at the same rate as their "customer" base since each user contributes some resources to the network in exchange for participating in the network. Thus Skype, a free VOIP network is growing very rapidly with almost no expenditure of its own resources while the old-fashioned telephone companies struggle to adapt. Similarly, public databases, once accepted in their relevant communities, can grow much more rapidly than can a competing proprietary database. Many a bioinformatics startup company has found it impossible to provide proprietary for-fee databases in competition with free public databases.

The open question is: what other models of "smarter" interactions might emerge. Friend-of-a-friend networks have been tried by startups such as Friendster, Tribe, Tickle, Ryze, LinkedIn and others. Apparently they haven't worked or haven't been done quite right. Yet similar sites such as MySpace and FaceBook are growing rapidly. Clearly, something along those lines will succeed when it combines the right stigmergy structures that generate some sort of persistent larger "self" with ongoing value. Anticipating and/or understanding how to create new types of stigmergy structures in the open Internet will be a powerful force for innovation.

[56] GenBank manages DNA sequence data, PDB manages protein structure data, Pub Med manages biological and medical publications, and BIND manages protein-protein interaction data. There are hundreds of other useful bioinformatics databases freely available on the Web.

Maintaining the multicellular "self" by cell suicide

Perhaps the least obvious of the four principles of multicellularity is apoptosis (sometimes spelled aptosis or apotosis and also known as Programmed Cell Death). The basic principle is that, for the good of the organism, all of its cells must be prepared to suicide. They do it in a very carefully programmed way so as to minimize damage to the larger organism. And they don't do it just when things go wrong! The apoptosis mechanism is a normal and a *creative* aspect of multicellular life. Every year the average human loses half of his/her body weight in cells via apoptosis. And orchestrated apoptosis helps the growing embryo to sculpt many aspects of its final form. Because apoptosis is so crucial to multicellular organisms, it is very carefully intertwined with the other three multicellular principles.

Self-organization of a Metazoan organism, aided by stigmergy, is not the end of the story. No sooner has a Metazoan begun to organize itself, than it is subject to the slings and arrows of fate. Constituent cells can become infected with viruses or bacterial predators, their DNA can be damaged by replication errors, radiation or mutagenic chemicals, or they can lose their differentiation and thereby become neoplastic and eventually cancerous. Infected cells will in turn infect their neighbors. DNA replication errors or mutations will be passed on to the cell's progeny. Cancer cells, of course, grow rapidly without bound and disrupt vital bodily functions.

The solution multicellular life evolved is a set of mechanisms that trigger apoptosis, or cell suicide. Metazoan cells have evolved mechanisms for detecting internal errors that might threaten the whole organism and react to such errors by killing themselves. For many types of cells, loss of contact to the extracellular matrix (the body's stigmergy structure) triggers apoptosis. That is, if they somehow become detached from their proper place in the body, they suicide. In general, each cell in advanced Metazoans has two specialized sorts of receptors on its surface that connect to the apoptosis mechanism. One type receives messages that suppress apoptosis and the other type receives messages that encourage apoptosis. Cell survival is determined by the balance between these two types of message.

Needless to say, the semantics of the "live or die" messaging mechanisms have evolved with care – all the more so because apoptosis is used not only to kill dangerous cells, but also to sculpt many body structures during development from an egg to an adult. For example, apoptosis causes the disappearance of a tadpole's tail and apoptosis removes the webbing between embryonic human fingers so that the fingers can separate[57].

Apoptosis is a highly ordered process that removes the cell with a minimum of risk or damage to nearby cells. In an orderly manner, apoptosis shrinks the cell, degrades the DNA in their nucleus, breaks down their mitochondria, and then breaks the cell into small, neat, membrane-wrapped, fragments. Finally, phagocytic cells like macrophages and dendritic cells engulf the cell fragments. The phagocytic cells also secrete cytokines that inhibit the inflammation[58] that otherwise would be a danger to the surrounding cells. Apoptosis removes cells in such a careful and well controlled manner because removing cells is just as important to the health of the multicellular organism as growing new cells.

[57] See www.knowledgerush.com/kr/jsp/db/viewWiki.jsp?title=Apoptosis for more discussion of apoptosis.
[58] From http://users.rcn.com/jkimball.ma.ultranet/BiologyPages/A/Apoptosis.html

Apoptosis evolved coincident with the first types of multicellular life: colonies of bacteria.[59] "...the invention of apoptosis was an essential feature of the evolution of multicellular animals."[60] Today all multicellular organisms, both plant and animal, rely on some form of PCD. It clearly evolved to deal with the sorts of issues that face multicellular organisms but not single cell organisms: initial development of the body, and after development, the maintenance of the organism against threats of DNA damage, viral infection, and loss of differentiation. Most importantly, it solves those issues from a *multicellular* perspective: sacrificing the individual cell for the good of the multicellular organism.

Computing could benefit from the two central lessons of apoptosis: first, the system is architected so that no cell is indispensable, and second, it is not a top-down system. The task of protecting the system is given to every part of the system, including the very cells that may represent the threat.

Apoptosis in computing

Of the four architectural principles discussed in this paper, apoptosis seems most foreign to the world of computing. But the principle is used in some mission critical military or avionics systems (e.g., the space shuttle or fly-by-wire controls for jets). Redundant computers, working in parallel, monitor each other. If one computer goes astray, it is shut down. In critical corporate applications and databases we have hot backup systems that detect an imminent breakdown and switch to a backup. Those examples are characterized by a strong awareness that the larger system *must* survive. In less mission-critical situations, our old "single-cell" attitudes tend to dominate. We strive to make each individual computer as robust as possible and assume that the robustness of the system is somehow the sum of the robustness of its parts. Of course, when put that way, most of us will recognize that a better metaphor is probably that of a chain, which is no more robust than its weakest link; the more links the more likely one will fail. Still, the idea of cutting out the weakest link seems foreign to most computing professionals. We would rather redouble our efforts to strengthen it. That reflects a fundamental misconception about our ability to *control* complexity. In multicellular computing systems, we seldom have control of every participating computer, let alone have the ability to strengthen every one of them.

Apoptosis mechanisms evolved along with multicellular life and multicellular life could not have evolved without apoptosis. Multicellular organisms assumed, and exploited, the fact that all cells are expendable. Multicellular computing systems, in contrast, evolved without assuming cell suicide. Instead, our default assumption was, and still is, that each and every computer is valuable and must be protected. That was acceptable in early multicellular computing systems, e.g., client-server systems, only because they did not have to face today's onslaught of viruses and worms.

As the value of each and every computer diminishes and the threat of viruses and worms increases, our attitudes must adjust to the new reality. If we are to make use of the example set by biological evolution, we should architect our computing systems so that we can sacrifice any individual machine in order to protect the larger organism.

[59] For example, modern cyanobacteria, the descendants of the earliest bacterial colonies, have a PCD pathway. "The demise of the marine cyanobacteria, Trichodesmium spp., via an autocatalyzed cell death pathway, Berman-Frank, I., Bidle, K.D., & Falkowski, P.G. Limnology and Oceanography, 49(4), pp. 997-1005, 2004

[60] See: "Identification of caspases and apoptosis in the simple metazoan Hydra," Cikala M, Wilm B, Hobmayer E, Bottger A, David CN, *Curr Biol.*, Sep 9;9(17):959-62, 1999.

Digital analogs of cell suicide

As we contemplate the adoption of a computing equivalent of cell suicide we must first recognize that the digital world differs in several important ways from the biological world:

- Since multicellular computing is about virtual interaction rather than physical interaction, a computer that is isolated from communication with any other computer is effectively dead. Thus, in computing, we may only be talking about ostracizing an errant computer (cutting it off from the net) and keeping it ostracized until it is cleaned or rebuilt rather than shutting it down completely.
- In biological systems, all cells share the same apoptosis mechanism hence cell suicide is essentially the same process no matter what kind of cell it is. Computers, however, play very different roles in the IT structure that will require different ways of dealing with the need to sacrifice a particular computer for the good of the system as a whole. Dealing with an infection in a key database server is far more delicate than dealing with an infected perimeter machine, especially an employee's PC.
- Cells are replaceable. Kill one and another soon takes its place. Employees, at least knowledge workers who depend almost totally on their computers, are not only far less replaceable, but have their own opinions about being cut off from the corporate system. IT administrators would face a revolt if they routinely ostracized employee's machines first and asked questions later. Moreover, the detection of wayward behavior may be fallible – it could be spoofed, perhaps by an unscrupulous competitor or disgruntled customer or ex-employee. That opens the way to a new kind of denial-of-service attack: spoof the corporate system into cutting off all the employees. This would be a computing version of the Ebola virus. Note that the machines in the Internet backbone and peer-to-peer networks are more similar to the biological systems in that removing any machine, or even many machines, may slow the whole system a bit, but no machine is irreplaceable. That is because these systems were designed from the beginning to be multicellular.
- Finally, sacrificing a cell to save the whole organism begs the question of just what we are trying to save in the IT system. While we want to save the stigmergy structure, saving it at the expense of cutting off large numbers of individual employee's machines may kill the company. The IT infrastructure is like the skeleton of the organism. It makes no sense to kill all the perimeter cells just to save the skeleton. We need a better understanding of what we can sacrifice and what we are protecting by doing so. Are we seeking to save the majority of machines, save the "most important" first, save the CEO's PC at all costs, or perhaps save customer facing machines?

Approaches to programmed suicide in computing

The lesson from apoptosis tells us that the first level defense should be the individual computer, especially those attached at the perimeter of the network. They should be able to recognize their own anti-social behavior and detach themselves from the network. The second stage should be based on specialized computers designed to recognize infected computers that, for one reason or another, have not triggered their own suicide mechanism. The second stage system then tells the infected or otherwise wayward machine to ostracize itself. Modern anti-virus detection could serve in both roles. But we must also consider the tradeoffs in the amount of CPU and/or disk access power we want to expend detecting infections. As with biological systems, multicellular computing systems exist to provide a better offence, not the perfect defense.

Ideally, the systems on the periphery could change the degree of defensiveness as they run according to "threat-level" messages from a more global detection mechanism. Think of this

central system as a special immune-system organ. It could be based upon honeypots[61], or on sniffing and analyzing network packets for signs of unusual activity, or any of the many other approaches to network security. The value, however, comes from each computer being able to devote more of its time to detecting possible infections if warned about an elevated threat level. That is, the central system would, in effect, warn every machine in the network of an ongoing infection, perhaps in a way analogous to the biological inflammation response or fever.

All the detection approaches ultimately depend upon how effectively and reliably the peripheral machines can ostracize themselves. A router can "kill" a misbehaving computer by disconnecting it. However, the computer may have other connections, e.g., by WiFi, BlueTooth or other wireless means. The router managing the Ethernet connection may well be different from ones managing wireless connections. The most reliable mechanism is for the computer itself to disconnect completely from any network access. But it has to do so in a way that cannot be defeated by the infecting virus. It may be sufficient to enforce network suicide at the low-level OS, perhaps in the drivers so that a virus cannot circumvent the cut off. Yet even that may not work if buffer overflow exploits can modify a driver.

It may turn out that we cannot trust any software-only solution. We may need one-way hardware shut offs, i.e., the software can shut down all networking devices in a way that cannot be reversed without active human intervention. This might be comparable to the level at which CTRL/ALT/DEL is wired into the keyboard of a Wintel machine.

In the final analysis, however, we must remember that cell suicide can be at least as much a threat as the viral infection itself if the multi-cellular system cannot tolerate the loss of infected computers. So, above all, we need to begin by architecting the system so that it can treat all computers as expendable.

[61] Systems set up specifically to attract attacks. See www.securityfocus.com/infocus/1659

How the four principles are intertwined

In multicellular organisms each differentiated cell functions in a specialized way as part of a coherent larger system. The ways they specialize and collaborate with other specialized cells in the same organism evolved together. That is, the specializations coevolved – one specialization supports and depends upon another. Similarly, the four principles coevolved during this process so that virtually all cells participate in *all four* architectural realms at once.

Specialization –All healthy Metazoan cells are specialized. Even adult stem cells are partially specialized. What is perhaps the most specialized aspect of the cells, other than their unique shape and function, is their unique repertoire of functional (polymorphic) message receptors. Yet they all share common behaviors too. Included in the common behavior are participation in the cues and signals of their stigmergy relationship with the rest of the body, and obedience to apoptosis messages. That is, as multicellular organisms evolved specialized behaviors, they had to also evolve appropriate messaging, stigmergy and apoptosis behaviors.

Polymorphic Messaging – Complex messenger proteins often act as "bundles" of messages. That is, one messenger protein may have separate domains, each with a different messaging function.[62] And often, the different message domains address separate architectural principles. For example, one domain initiates signal cascades specific to the unique function of that type of cell (i.e., specialized messaging), another facilitates or verifies physical attachment to the extracellular matrix (i.e., deals explicitly with the stigmergy structure), and yet another provides signals that either suppress or encourage apoptosis! The existence of these multi-part messages shows not only that the organisms evolved along with the four principles, but also how fundamental these principles are. A single multi-part message speaks to the functional relationship of the cell *to the whole organism/tissue/organ* rather than to just a single cell function.

Stigmergy –Virtually all cells other than some simple floating cells such as red blood cells are affected by stigmergy cues and/or signals. Cells that are attached to the Extracellular Matrix (ECM), i.e., the stigmergy structure, leave long-lasting cues (persistent messages) in those structures that affect other cells. In turn, the cells respond to such cues in ways that cause them to modify the physical structures; that's how the structures are built in the first place. Cells that are normally attached or in direct contact with the ECM require constant feedback from the ECM. Absent the appropriate attachment cues, they suicide (undergo apoptosis).

Apoptosis –almost all cells except cancerous cells participate in apoptosis signaling all the time. Most cells must receive "you're OK" messages sufficiently often to prevent their suicide. Even very simple interchangeable and disposable cells such as red blood cells undergo apoptosis.

Not only do cells reflect all four principles at once, the principles themselves are interdependent in the sense that each one relies on the others.

- Specialization requires a stigmergy structure (a body) to nurture and protect the specialized cells. They would not survive long in isolation. In turn, the stigmergy structure, i.e., the whole organism, benefits from their specialized activity. A stigmergic body makes no sense without specialization; Even cells in biofilms specialize. So, too, do ants and termites. The more complex their stigmergic social interactions, the more

[62] See "Exploring and Engineering the Cell Surface Interface" abstract (Stevens, M. M. & George, J. H., Science, vol 310, Nov. 18, 2005, pp. 1135-1138)

specialized are their roles in the insect colony.[63] Thus specialization and stigmergy are interdependent at a fundamental biological level.

- Once cells specialize, they must interpret messages accordingly. That's how a collaborating group with various specializations can respond in an orchestrated way to some common stimulus. There is no orchestra conductor telling each one what to do. A common message signals the situation and each specialized cell with a receptor for that message responds in accordance with its specialized role. Similarly, specialized ants and termites must interpret messages from the perspective of their special abilities. So specialization and polymorphic messaging are strongly interdependent.
- Since apoptosis exists to sculpt and protect a (stigmergic) body, clearly stigmergy and apoptosis are interdependent. But also, the apoptotic messaging pathways depend upon polymorphic messaging and the cellular response to apoptosis messages differs according to the specialized function of the cell. Thus apoptosis and specialization are interdependent.

Implications for multicellular computing

As multicellular computing architectures evolve, especially the emerging Service Oriented Architectures (SOA) and the less formal Web Service mashups, we would be wise to adopt and carefully interleave all four principles. Similarly, the architecture of multicellular computing messages should be fully multicellular. Those who "design" multicellular computing systems (or better put, attempt to grow such systems), should not only give thought to the various kinds of specialized "cells" that are needed, but also to how those cells implement the four principles and fit into a *multicellular* message architecture.

In summary, as we develop architectural patterns – both hardware and software – for the use of each of the four principles in computing, we must develop them so that the four patterns can be interwoven. And there must be a meta-pattern that lays out how the interweaving is to happen.

[63] See http://www2.isye.gatech.edu/%7Ecarl/papers/AndersonMcShea.pdf

Conclusions

The dynamic complex interactions that drive the evolution of computing toward multicellularity have a life of their own. Yet they are amenable to some steering. The question is: what roadmap should we steer by?

I argue that we should look to biological metaphors such as the four I have proposed to help us make multicellular computing systems more manageable. The fact that we are already seeing evolution toward at least three of the four principles (apoptosis is used occasionally but it is not yet common) suggests that they are indeed general enough and useful enough to provide us with good insights. That should come as no surprise since, without them, multicellular biological life would not be possible and the complexities faced by multicellular life are not unlike those faced by multicellular computing.

If we are more explicit and disciplined about making the transition toward multicellular computing and we learn from the lessons of multicellular life, we stand to make the transition smoother and perhaps even quicker. To that end, we need to foster, in our education and in our cultural transmission of received wisdom within the computing community, multicellular attitudes such as:

- What goes on within any given computer is far less important than what goes on between computers.
- Code that is inside a computer may be valuable, but when it is moving between computers it must be assumed to be dangerous.
- General purpose computers, especially large monocultures of them, will become ever more troublesome as virus writers continue to seek out their weaknesses. Specialize them.
- The meaning of messages should be determined by the receiver of the messages, not the sender.
- Individual CPUs are, or should be, expendable and be prepared to commit suicide (or disconnect) neatly and safely when compromised.
- The multicellular computing organism is paramount. When a single computer offends, cut it off. Better yet, empower the individual computers to sense problems within themselves that cause them to cut themselves off.
- If you want to create a new multicellular computing organism, think about what its stigmergy structures might be. Conversely, if you see the emergence of a new kind of stigmergy structure, e.g., a public data base wrapped in a Web Service interface, think about what multicellular organism(s) will emerge around it.

The stakes are high. We are at the *beginning* of the computing equivalent of the Cambrian explosion[64] – the period of roughly forty million years ending about 500 million years ago in which there was a fantastic flourishing of new types of multicellular biological organism. Most of these experiments exist today, if at all, only as fossils that happened to be preserved in the Cambrian mud. Given how rapidly we have recapitulated a couple billion years of single cell evolution, perhaps we can recapitulate the forty million year Cambrian period in the next couple of decades. If so, it behooves us to pay close attention to how multicellular computing is progressing. Forward-looking businesses and technologists will have much to do as multicellular computing evolves. Great opportunities will be available for those who anticipate (or lead) the successful architectures and great peril for those who back the wrong horse. They face a choice between winning and being a fossil in the mud.

[64] See, for example, www.palaeos.com/Ecology/Radiations/CambrianExplosion.html

It is already clear that there are competing organizational strategies, e.g., completely decentralized P2P communities at one extreme versus centrally controlled hierarchical IT structures at the other extreme, with some compromise ideas, e.g., Grid architectures, in between. And there are competing notions of how Internet messaging should work, e.g., heavyweight SOA (which may sink under its load of complexity) versus lighter weight AJAX (which may fail to scale well). We also have competing vendors, each pushing the idea that what they sell is just what is needed for the future. Don't believe them!

Most of all, we have competing visions of how evolving computing infrastructures interface with evolving business and cultural systems that use the computing systems. That is, different visions of what happens where the silicon meets the flesh. The digerati imagine cyborg-like symbiosis in which people are festooned with wearable or implantable computers that aid them in all manner of normal social interactions (or, perhaps, insulate them from normal human interaction). There are neo-Luddites who imagine that we can "keep computers in their place" which, presumably, is somewhere far from those who hold such views. And there are neo-fascists (or those who fear them) who imagine that every move we make and thought we voice will be surveiled by ever-present computer-controlled cameras, microphones, and RFID tracking devices. What all of these ideas share is a lack of understanding that the evolution of computing is already beyond our control. The coevolution of complex computing systems interacting with human social and economic systems already has a life of its own.

It is also worth remembering that the processes that give rise to emergent complex systems do not stop, or even pause, at one level. Before one level is fleshed out, another emergent metalevel will be forming based upon collaborations between the units at the lower level. And so it is today. Multicellular computing "selves" made most visible by their stigmergy structures are collaborating to form even larger and more complex systems. We can think of some of the more public multicellular systems as organs in some larger "body" we have not quite envisioned yet. That is, some system may already be evolving that combines a Google plus a BitTorrent, plus a Skype, plus who knows what else acting as "organs" to achieve some larger goal. All that is certain is that each stage of multi-level emergence will surprise us.