Journal of the Washington Academy of Sciences,
Volume 85, Number 4, 281-289, December 1998

Gratuity in Nature and Technology

by Paul T. Arveson

Abstract

Frequently in science, we encounter patterns or structures in one field that have parallels in a different, apparently unrelated field. Because of the inherently cross-disciplinary nature of this subject, sometimes the parallels can go unrecognized for a long time. One of these patterns is often referred to as "levels of reality", or "levels of explanation". This "laminar" structure of our knowledge implies that there is some general feature or characteristic that separates systems into distinct and irreducible levels or dualities. Here this feature is identified as what Jacques Monod called gratuity and it brings together in one brief summary a variety of ways in which we observe gratuity in both natural and man-made systems.

Monod's insight was that a living cell is a very special kind of machine, in which its information content is free and unconstrained by the cell itself. In DNA, there is no energetic preference for one set of nucleic acids over any other; the sequence is an arbitrary code, as far as the cell is concerned. Rather, the cell's processes are analogous to the relationship between a computer's software and its material substrate. In a computer, the sequence of numbers and letters in the software are not constrained but are free from the structure of the computer's hardware. So we are faced in these cases with systems that cannot be described in simple mechanistic terms of cause-and-effect. Their behavior is governed not from "below," but rather from "above." In the following sections, several examples of gratuity and its application in widely diverse fields are described.

Introduction

This article draws parallels between the manifestations of "levels" in biochemistry and computer science (computer networks and computers themselves). A common idea found in these fields is Monod's concept of 'gratuity.' Many theories in other fields are not considered here but are recognized to be relevant to this concept, including for example quantum physics (wavelike behavior of atomic interference reported by Berman, 1997), the brain (theory of neuronal group selection, Edelman, 1994), social animals: Holland (1995), ecology: Gell-Mann (1994), and economics: Anderson (1987); Arthur et al. (1997). Technology has drawn many fruitful lessons from biology; it is now suggested that organizational management could also benefit from such cross-disciplinary learning.

Gratuity in Biochemistry

Jacques Monod won the Nobel prize for his elucidation of the process by which gene expression is regulated in biosynthesis, a process called the operon model. I will not describe the whole model here, but quote Monod's own analysis of the concept from Chance and Necessity (1971):

> "There is no chemically necessary relationship between the fact that beta-galactosidase hydrolyzes beta-galactosides, and the fact that its biosynthesis is induced by the same compounds. Physiologically useful or rational, this relationship is chemically arbitrary — gratuitous, one may say."

We are in the habit of describing chemical processes in terms of cause and effect. That won't work for this process. It is not deterministic in a simple, mechanical way. Monod identifies the relationship as one of "gratuity":

> "The fundamental concept of gratuity — i.e. the independence, chemically speaking, between the function itself and the nature of the chemical signals controlling it — applies to allosteric enzymes. ... Between the substrate of an allosteric enzyme and the ligands prompting or inhibiting its activity there exists no chemically necessary relationship of structure or of reactivity. The specificity of the interactions, in short, has nothing to do with the structure of the ligands; it is entirely due, instead, to that of the protein in the various states it is able to adopt, a structure in its turn freely, arbitrarily dictated by the structure of a gene.
>
> "From this it results — and we come to our essential point — that so far as regulation through allosteric interaction is concerned, everything is possible. An allosteric progein should be seen as a specialized product of molecular engineering, enabling an interaction, positive or negative, to come about between compounds that are chemically foreign and indifferent to this reaction.
>
> "In a word, the very gratuitousness of these systems, giving molecular evolution a practically limitless field for exploration and experiment, enabled it to elaborate the huge network of cybernetic interconnections which makes each organism an autonomous functional unit, whose performances appear to transcend the laws of chemistry if not to ignore them altogether."

So the essential aspect of gratuity is a kind of independence or transcendence; the freedom of a system's behavior from being constrained by its parts. Such a system can act to control a signal without altering it, or as a transparent channel for a signal. In biochemical 'signals' of the operon model, the DNA fully determines the products formed, but when they are formed is under the control of a process at a 'higher level' in the system.

"Microscopic cybernetics" is Monod's own phrase by which he suggests analogies between biology and computer circuits. Features of the operon model include positive and negative feedback regulation (enzyme induction and coordinate repression). He even

showed how two interacting operons can form a stable oscillator (just as two dual-input NAND gates can be connected to form a flip-flop circuit):

> "... both interations of the repressor are noncovalent and reversible, and that, in particular, the inducer is not modified through its binding to the repressor.... Thus the logic of this system is simple in the extreme: the repressor inactivates transcription; it is inactivated in its turn by the inducer. From this double negation results a positive effect, an 'affirmation'.... The logic of biological regulatory systems abides not by Hegelian laws but, like the workings of computers, by the propositional algebra of George Boole."

This was written in 1970; Monod was already aware of the relevance of cell biology to computer operations via Boolean algebra. Cross-disciplinary research of this kind continues; some of the most interesting work is being conducted by workers at the Santa Fe Institute, including Murray Gell-Mann (1994), Stuart Kaufmann (1996), and John Holland (1996). See also the excellent survey by Waldrop (1992).

To illustrate, the following table shows a scheme that identifies the "levels of organization" of living things.

Table 1. Independent Layers in Biological Systems

9. ecosystem with numerous species
8. species
7. individual organism or phenotype
6. organs
5. cell and organelles
4. nucleus *
3. operons and RNA mediated synthesis
2. DNA code
1. molecules and bonds

* A very recent addition to examples of gratuity is the discovery that a cow egg can support the development of another mammalian species, if its nucleus is substituted for the cow's (Dominko and First, 1998).

Gratuity in Telecommunication Networks

Gratuity is established in computer network protocols by distributing network functions among a series of layers; wrapping the data in packets with headers containing layer-related information; and limiting interaction between the layers to that involving the headers. In transmission, starting from the top layer, each lower layer adds its own

header to the data sent to it, and on reception each higher layer strips off a header and sends the data to the next layer. These processes are described by various technical terms, but all are related to the general principle of gratuity, where the data are independent of the frame or header. Here are some examples:

- Frame relay, the X.25 standard, creates frames around packets of data.
- Cell relay (ATM, Asynchronous Transfer Mode) adds headers to small packets.
- The Internet Protocol (IP) establishes tunneling of IP packets through foreign-networks.
- Sometimes large data packets have to be broken up to fit through a network, and then reassembled; this is called transparent fragmentation in IP or segmentation in ATM.

All these are equivalent to wrapping a message in a sealed envelope. This is done at the expense of lost bandwidth, but has many advantages: service over a connectionless network; error detection; receipt acknowledgment; standardized packet handling; guaranteed quality of service; security, etc.

Layering also has two distinct advantages for the design of networks: it facilitates development of the protocols, and it simplifies their functional components (hardware and software). Advanced protocols like ATM take thousands of pages to document; such complexity would be impossible for humans to manage without being able to limit the information required at any given layer.

The OSI (Open Systems Interconnection) model is traditionally used to illustrate the arrangement of layers in networks. The identification of functions in each layer differs depending on the particular set of protocols used, because the original OSI model is somewhat idealized. Below is shown a practical example of layer designations, applicable to an advanced network with TCP/IP running over an ATM subnet:

Table 2. Example of Layers in a Computer Network (TCP/IP over ATM).

6. Application layer - HTTP, FTP, Telnet, email, etc.
5. Transport Control Protocol (TCP) Layer - manages the network
4. Internet Protocol (IP) Layer - network addressing and routing
3. ATM Layer - several sublayers for switching and routing
2. Data Link Layer - error detection, framing, flow control
1. Physical Layer - cables, switches, voltages etc.

Gratuity in Computer Technology

Maurice Wilkes, one of the early pioneers of computer science, has recently (1995) written a concise retrospective on its early history. This little book is helpful in separating

significant changes from all the hype and clutter. Wilkes notes three crucial turning points in computer development; all had something to do with gratuity.

Unlike other kinds of machines, a computer's function is to process data; it doesn't matter what physical form the hardware takes, as long as its logic is correct. In describing the conversion of computers from vacuum tubes to transistors, Wilkes remarks on the smoothness of this transition with a beautiful simile: "It was as though the foundations of a cathedral were being wholly renewed, while services were going on in the choir and the organ was playing."

Another achievement led to what Wilkes called "the software avalanche": the appropriate segregation of hardware tasks from software tasks. Some early computers used parallel arithmetic processors in order to save calculation time, at the expense of more complex programming. Later, to simplify programming the change was made to a single CPU to allow a single instruction stream for all operations. Operations were slower, but this saved programmer time. It also "made possible the development of modern software with its layer upon layer of complexity."

A key to this development was the realization that computers could be used to prepare and edit their own programs. The use of index registers allowed subroutines to be called to simplify programming efforts. Then a team led by Grace Hopper at UNIVAC developed a "compiler" that expanded program pseudocode into machine code. Wilkes (1952) independently suggested a similar concept. At that point, languages more readable by humans could be developed, including FORTRAN, COBOL, ALGOL, BASIC, and PL/1 during the 1960s. The syntax of these languages was based on human needs, not the machine's; this is a third historical example of gratuity.

Wilkes' "software avalanche" led to higher-level computer languages, including complex compilers and hardware simulators used to design more advanced computers. Whenever systems support their own growth in this way, the growth rate is exponential. This is why we have observations such as Moore's law, which describes the 'doubling time' of computer processing power every 18 months.

Another significant application of gratuity is the concept of 'object oriented languages' such as Smalltalk, invented at the Palo Alto Research Center in the early 1970s. An object "is a self-contained software package consisting of its own private information (data), its own private procedures (private methods) that manipulate the object's private data, and a public interface (public methods) for communicating with other objects" (Fingar, 1996). In this methodology, a programmer's task is not so much with writing lines of code as with assembling a set of pre-defined objects for a task.

The hiding of internal information within objects is called encapsulation. To use an object, you only need to know what messages the object can accept. The advantage of encapsulation is that objects can be combined, changed, and reused without having to rewrite any internal code. This is clearly an application of gratuity, where the 'insulation' of internal source code from the high-level object language is strictly enforced.

Further layers continue to be developed in computer technology, as the systems become more complex and powerful. The need to sequester and limit information seen by the designer or user continues to be the only feasible way to develop systems of this complexity. Hence, we see system descriptions with increasing numbers of layers. Computer technology using objects is now expanding to create higher levels of sophisticated system "architecture" in organizations. Norman Simenson, a veteran software engineer from the early days at IBM, has recently written a working definition of "architecture" (1997):

"An architecture is a partitioning scheme, or algorithm, that partitions all of the system's present and foreseeable specifications into a workable set of cleanly bounded 'buckets' with nothing left over. That is, it is a partitioning scheme that is exclusive, inclusive, and exhaustive. Ideally, each bucket will be a standalone partition. A major purpose of the partitioning is to arrange the elements in the buckets so that there is a minimum of message exchanges needed among buckets. In both software and hardware, a good bucket can be seen to be a meaningful 'object'."

This definition clearly entails gratuity, albeit in different language. Below is a table describing one scheme suggested scheme for layers of information technology that might be deployed in a future business enterprise (upper layers from Fingar, 1996).

Table 3: A Scheme for Layers in an Enterprise Information System

14. Agents: sequencing of workflows, events.
13. Events: representations of actions that initiate or influence business scenarios.
12. Scenarios: Assemblage of entities representing a business function or process (transactions, orders).
11. Entities: tangible and conceptual problem domain subjects (people, trucks), associations (marriage, employment) and roles (manager, clerk).
10. Intrinsics: Technology and problem domain for independent base classes (money, size, etc.)
9. Technology encapsulation: CORBA, DCOM, (persistence, lifecycle, etc.)
8. Object-oriented languages, models, and development environments
7. Graphical user interfaces (computer-human interfaces), input/output data
6. Source code layer (text)
5. Assembler language layer
4. Machine language layer (binary codes)
3. Boolean algebra, binary logic
2. Circuit layer: gate arrays, registers, read-only memory, buses etc.
1. Physical layer: physical laws and materials parameters

The Emerging Language of Gratuity

It is noteworthy that the word 'levels' is a metaphor. Although the word is commonly used to refer to separated functions in complex systems, in most cases the 'levels' do not have any physical existence. In a computer, only the bottom one or two layers are physical; the rest are in software. In the case of computers, it may be more appropriate to use another metaphor, such as a "chain" of functions, or even better, a nested series of "shells," as in a set of babushka dolls. (In fact, in the UNIX context the word "shell" is often used.) In more complex systems, the relationships are more "weblike"; ultimately in our technological systems we are reconstructing ourselves, i.e., our biological bodies and ecosystems.

In network terms we may describe biological gratuity by saying that the DNA is the data packet to be sent; the chromosomes (headers) encapsulate the data, and other top-layer functions control when and how often the packet is sent. This is a "top-down" process. On the other hand, the structure and function of all the products are ultimately determined by the DNA, i.e., "bottom-up." There is an asymmetrical interdependence, or "metadependence" of information coming from above and below the synthesis layer. In computer terms, we could say that the user or the top-level program determines what actions occur, even though we "know" that the immediate cause is electric signals in the physical layer. Among software engineers, this low-level reality tends to be repressed into the unconscious, and in place of the user, new kinds of top-level gratuitous objects continue to be developed, such as "agents," "avatars," "daemons," etc. Perhaps the endpoint of "computer system" development is Data, the android in Star Trek — he knows everything, but (unlike HAL of 2001) he has no ambition.

Management Applications: How to Get Organized

There are many other ways in which the concept of gratuity can be observed in nature and technology. Rather than continue with technical examples, I wish to discuss the potential relevance of gratuity to the practical problems of managing complex organizations. In modern enterprises such as the federal government or large corporations, managers often are overwhelmed at the complexity they are expected to comprehend and lead. The common phrase for this is 'information overload' and this may lead to frustration, stress and cynicism. However, nature demonstrates that although such tasks are extremely difficult, they are not fundamentally impossible. They just have to be divided up and delegated appropriately. In the operon model and many other biochemical processes that science has elucidated, gratuity limits information by isolating (or encapsulating) functions from each other, allowing only a few carefully defined inputs and outputs to pass, thus creating "layers." Some kinds of data will pass transparently through a layer; some kinds will originate in each layer. But the net effect is to simplify the entire

system's behavior and its presentation at the top layer, where strategic decisions affecting the entire system can be made — whether that system be a jaguar (Gell-Mann's example) or a corporation's growth strategy.

Taking this lesson from biochemistry and technology, we conclude that managers should not try to 'micromanage' operations of low-level business units on a continuing basis. Rather, they should establish metrics to monitor the efficiency (cycle time, productivity) and effectiveness (relevance to mission or strategy) of each unit. Then they should monitor these metrics, and leave the internal functions to adapt and improve at will (freedom or 'gratuity' is established within each unit). In order to limit the data flows to prevent information overload, measurements should be aggregated at each level of organization. In other words, strategic planning and reengineering studies should find ways to limit data, as well as to deliver it.

Bureaucratic organizations may have over a dozen levels in their chain of command. If a level does not add value (i.e., create new information), it is only a transparent or pass-through channel at best, or an obstruction at worst. This realization is leading many agencies to 'flatten' their organizational structure by removing layers of middle management. In addition to reducing costs, a useful goal of this reengineering process is the appropriate allocation and isolation of functions and simplification of their linkages. By applying gratuity, it is possible at least in principle to build systems or organizations of ever-increasing complexity without getting bogged down in information overload or inefficiency.

References

Anderson, P. W., Arrow, K. J., and Pines, D., eds. (1988). *The Economy as an Evolving Complex System.* Santa Fe Institute Studies in the Sciences of Complexity, 5. Redwood City, CA: Addison-Wesley.

Arthur, B., Durlauf, S. N., and Lane, D., eds. (1997). *The Economy as an Evolving Complex System II, Proceedings.* Santa Fe Institute Studies in the Sciences of Complexity. Santa Fe, NM: Santa Fe Institute.

Berman , P. R. (1997). Illustration of "the wave nature of matter objects independent of the complexity of their internal structure." *Atom Interferometry*, ed. Paul R. Berman, Academic, reviewed in Physics Today, Dec 1997, pp. 68-70.

Dominko, T. and First, N. (1998). "Cow Eggs Play Crucial Role in Cloning Effort", *The Washington Post*, Jan. 19, 1998, p. A1. (University of Wisconsin at Madison).

Edelman, G. M. (1987). *Neural Darwinism: The Theory of Neuronal Group Selection.* New York: Basic Books.

Fingar, P. ed. (1996). *Next Generation Computing: Distributed Objects for Business.* New York: SIGS Books & Multimedia.

Gell-Mann, M. (1994). *The Quark and the Jaguar.: Adventures in the Simple and the Complex.* New York: W. H. Freeman.

Holland, J. H. (1995). *Hidden Order: How Adaptation Builds Complexity.* New York: Helix Books (Addison-Wesley).

Jacob, F. et al. (1961). J. Mol. Biol. 3, 318.

Jacob, F. et al. (1964). Comptes Rendus Acad. Sci. Paris 258, 3125.

Kaufmann, S. A. (1996). *At Home in the Universe: The Search for the Laws of Self-Organization and Complexity.* New York: Oxford University Press.

Monod, J. (1972). *Chance & Necessity.* New York: Vintage Books (Random House).

Simenson, N. F. (1997). *The Architect's Roles and Responsibilities*. American Programmer, July 1997.

Waldrop, M. (1992). *Complexity: The Emerging Science at the Edge of Order and Chaos*. New York: Touchstone Books (Simon & Schuster).

Wilkes, Maurice V. (1995). *Computing Perspectives*. London: Morgan Kaufman Publishers.