Cell Operating System and its Compilation Processes

Nevena Ackovska[1], Stevo Bozinovski[2], and Gjorgji Jovancevski[1]

[1] Institute of Informatics, FNSM, "St. Cyril and Methodius" University
P.O. Box 162, 1000 Skopje, Macedonia
[2] Department of Mathematics and Computer Science, South Carolina State University
Orangeburg, SC 29117, USA
{nevena, gjorgji}@ii.edu.mk; sbozinovski@scsu.edu

Abstract: Several biologically inspired approaches have been successful in improving the understanding and implementation of new information technologies; remarkable examples being neural networks and genetic algorithms. This paper takes a biologically inspired approach towards the parallel systems. It views the DNA through a "system software microscope" and discusses related issues, examples being file system, program preparation, and it's compilation processes among others. Our work explores the analogy between the computer operating systems and the molecular biology control systems, concerning the issues of improving the computer operating systems and its capabilities.

1 Introduction

Computer viruses and other types of invasive software made a paradigm shift in operating system (OS) design: from performance only oriented approach to security and reliability oriented approaches [1]. Since Linux kernel has 2.5 ML (millions lines of code) and Windows XP kernel has 5 ML, they are obviously huge and potentially unreliable and insecure. Several approaches have been taken into consideration in order to protect the OS kernel, including armored operating systems, paravirtual machines, multiserver operating systems, and language based protection [1]. Other issue addressing the reliability of operating systems is the threads and their nondeterministic nature. Approaches toward resolving the issue of blocked processes and deadlocks have also been discussed [2].

This paper addresses the above mentioned issues using bionic approach. The idea is that a biological cell has some kind of operating system which resides in the chromosomes [3, 4]. We will use this cell operating system as new metaphor for DNA. By describing the DNA based processes using such a metaphor we could learn lessons from biology about building new operating systems.

A metaphor is understood as a paradigm transformation; we use knowledge from a familiar system in order to understand and develop solutions in another system. Remarkable examples of such trans-disciplinary benefits are neural networks and genetic algorithms, today standard approaches in understanding and building

intelligence [5]. In understanding genetic systems we also use the robotized flexible manufacturing metaphor [6, 7, 8, 9].

2 A Need for Understanding the Cell Control System

There are several levels of information and material processing in a cell. As Figure 1 emphasizes, the cell communicates with two environments: one is the behavioral environment, which it faces during its life, and the other is the genetic environment, through which it communicates its genetic material.

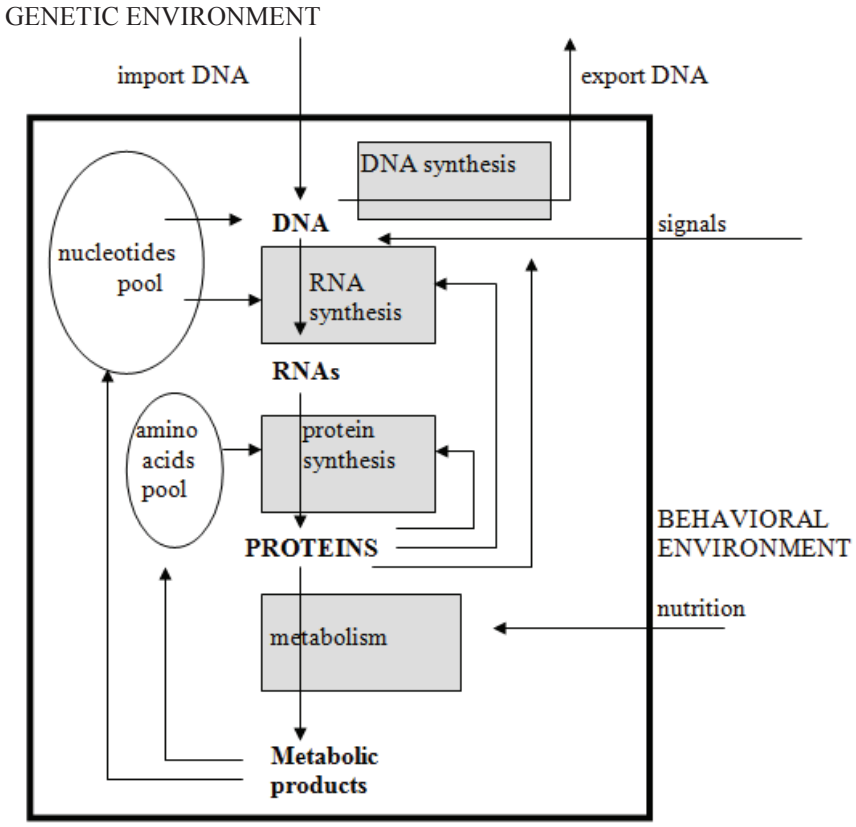

Fig. 1. Several levels of processing in a biological cell

The cell is an active autonomous agent. Example of a cell is the bacterium Escherichia coli; another example is a specialized cell in a multi-cellular system such as the human skin. In both cases the cell receives signals from the environment, including signals from other cellular agents, and responds to them. The cell has special sensors for various signals, and some of the cells have motors which allow them to move in the environment. As an example, E. coli has about 50-70 sensors for

various signals from the environment, and about 6-8 motors that are actuators for its flagella. Using its sensors, a cell would move towards an attracting spot in the environment, following a chemical gradient [10, 11]. The cell actively responds to environment changes. The response can be a behavioral one or a product manufacturing one [3, 12]. Another type of response is either reproduction or termination of its life, as a specific reaction to an environment change [13].

Figure 1 also emphasizes several levels of processing, relevant for the cell control hierarchy, such as DNA level, RNA level, protein level, and metabolism level. The cell has a hierarchical control structure. At the lowest level of control in prokaryotes (cells without a nucleus) are the operons [14], structures that control simultaneous activation of a group of genes. Regulons are at a higher level, controlling simultaneous activation of various genes and operons. Modulons are at a level higher, controlling regulons, operons and genes. It is not known whether this is the highest level of control in prokaryotes. In eukaryotes (cells with a nucleus containing DNA), the control structure is even more complex.

3 The Cell Operating System

As previously mentioned, the cell is governed by an event recognition and control system, at several hierarchical levels, such as operon, regulon, and modulon levels. All these structures are somehow coordinated. For example, when a cell is undergoing replication, all the other processes are oriented towards the support of this extremely complex process. Taking this into account, our observation is that there should be a general control structure that orchestrates the priorities of the cell activities. This observation leads us to consider the hypothesis of the existence of a Cell Operating System (COS). Moreover, the COS should be considered a database operating system, which takes care of a large database, the DNA itself [4]. This concept is shown in Figure 2.

Figure 2 points out that the DNA is a database operating system, which has a rather large gene database to take care of. Database operating systems have been of interest in systems software for some time [15, 16]. It also points out that the lower level of control systems consists of various feed-forward and feedback regulatory loops controlled by operons or some other control structures. The genosome in Figure 2 denotes a transcription-translation machinery that, given a gene, produces a protein or RNA. Thus, the Cell Operating System is a real-time database operating system, comprising the hierarchy of control levels of the cell control system and taking care of the gene database.

When discussing computer operating systems, there are several issues that should be addressed: system complexity that an OS should handle, program preparation, management of processing resources, management of storage resources, and security among others. Here we will address some of these issues for DNA based operating systems.

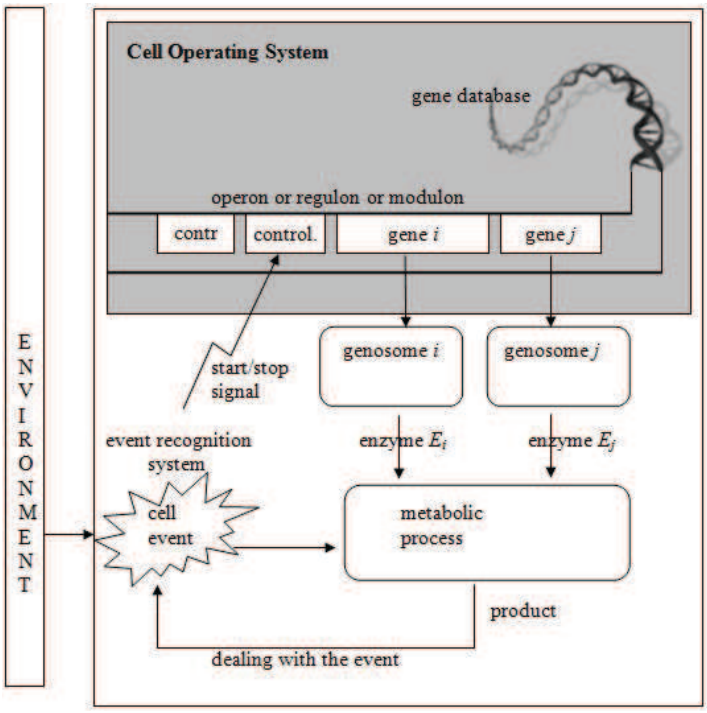

Fig. 2. The Cell Operating System Concept

3.1 Control System Complexity

To get a feeling of the potential information and control complexity of the cell activities, we take into consideration its three main components: resources, processors, and processes. Let us look at the potential complexity of the best-known model organism, Escherichia coli. It has a genome consisting of 4800 genes, a proteome of about 2500 proteins, and a metabolome (set of all metabolic reactions) of an unknown number [13]. Not all proteins are processors, but we can estimate that it is a parallel distributed processing system of up to thousand processors executing their programs written in the genome, working over about 5000 resource segments (genes and control segments), and carrying out an unknown number of processes.

3.2 Medium of Residence

The Cell Operating System is tape based. In prokaryotes, COS is located on a single tape. In eukaryotes, there are several tapes (chromosomes) on which the COS is resident. The tapes are multi-head accessible, and their reading machines, e.g. RNA polymerases, are able to access a needed piece of information directly, without unwinding the whole tape. Therefore, it could be considered a random access tape,

which in terms of functionality is equivalent to a random access disk. We can presume that COS actually resides on random access devices, and further in the text we will use the notion of a disk when considering the residence of the COS.

Having a concept of a disk in mind, we can imagine a disk cylinder of COS. It is a string of information that resides physically on various disks, but is read by several reading heads simultaneously; in the cell it can be achieved by folding the tapes in an appropriate fashion. Figure 3 shows the concept of a DNA disk cylinder.

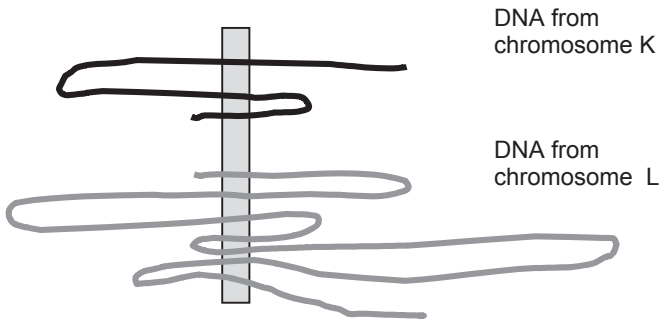

Fig. 3. The concept of a DNA disk cylinder

In a sense, the concept of a cylinder enables spatially distant information to be processed in parallel. And indeed, the files on different chromosomes could be processed in parallel. For example, in order to produce hemoglobin, protein genes from chromosome 11 and chromosome 16 should be processed using some kind of synchronized mechanism.

In eukaryotes, COS should be presumed to reside on two distributed logical disks, one inherited from the mother and one from the father. The two-sex organisms have the same, diploid, OS residence. Each logical genome disk contains several logical chromosome sub-disks, or minidisks (in IBM terminology). In humans, there are 22 diploid autosome minidisks, plus two haploid sex minidisks, known as X and Y, in two combinations, XX (female) or XY (male). Some organisms (e.g. yeast, Saccharomyces cerevisiae) can also live in their haploid form, having one genome disk per cell. A haploid, one-sex organism, with only one resident genome disk, has no defense against mutations. On the other hand, a diploid, two-sex organism has a backup disk, which protects from recessive mutation. As a consequence, a possible three-sex organism would prevent against more complex mutations, but this would add complexity to the system and restrict evolution.

3.3 Cell Files

A file is a logical unit that can be operated on with standard OS procedures. Files reside on disks or tapes. They can be physically compact, as well as fragmented (physically distributed on several disk locations or event disks). Files can consist of data and/or programs.

Looking for a concept of a file in the genetic system, we found that the transcription units (or scriptons [17]) are analogous to cell files. A transcription unit is a segment of DNA that eventually becomes transcribed to RNA. In prokaryotes, a transcription unit often produces a transcript with several genes (so-called polycistronic RNA). It is a transcript with several genes, obtained from an operon structure. In eukaryotes it produces a precursor RNA, which contains the information about a single gene, but in order to obtain it, additional processing needs to be performed.

The eukaryotic files are rather complex and contain segments of a gene, interleaved with segments that do not belong to that gene. Those segments are known as introns (interleaving segments), as opposite to exons (gene expressing segments). To the people involved with genetics, there is a standard question considering this problem: how did it happen that eukaryotic genes became segmented? However, using the cell operating system metaphor, the answer is straightforward – busy files are fragmented. This leads to the concept of distributed file systems [18, 19]. Using the COS metaphor, the appearance of introns is due to file fragmentation processes. Simply, after years of evolution, one should expect fragmented files in the eukaryotic file system. Therefore, this file-centered approach, defined in the COS metaphor, offers simple answers to nontrivial problems in genetics.

The intron-splicing process, that is observed in a cell during the RNA-processing, could be observed as a process of defragmentation (garbage collection), a process well known in OS terminology. Some introns could be complete files, in-betweens of segments of another file; an example being the tetrahymena ribozyme file [20]. Some of these files, including the Tetrahymena ribozyme, are able to catalyze their own cleavage [e.g. 21]. Similarly, in some programming systems, example being LISP and Java, garbage collection is performed by the programming system itself.

The need for fragmented files can also be understood by the concept of modular reusable subroutines, which are used by various program files. Let us consider the exons, the expressing gene segments. Using this approach, it is easy to see that exons could be functional units, such as subroutines (or methods in OOP) of a more complex program file. The subroutines could be reentrant, meaning that the same exon could be used in different RNA's. And indeed, examples are antibodies exons, which are used to produce various combinations of programs to cope with ever-changing agents attacking an organism [22]. In fact, this view towards exons, although in different terminology, was previously postulated by Gilbert [23, 24].

We believe that while the genes are the proper concept when talking about heredity and translation processes, the concept of a file is very useful in describing the transcription process. This makes the first step in the analogy between the computer operating systems and genetic systems. We believe that the whole transcription process could be better explained in the terms of file processing instead of linguistic terms. The cell, especially the eukaryotic cell, undergoes extensive file processing: from copying the pre-RNA file until obtaining the RNA message. This process includes operations like: cut (introns), join (exons), right append (trailer string), left append (header string), letter replacement and so on, which are standard file processing operations in every modern computer operating system.

3.4 Program Preparation – From Source Code to Embedded Systems

The cell files could be data files and executable program files. Data files are the ones that are used in its linear, source code form. For example, mRNA is used in its source code form, as a recipe how to design the linear protein structure.

The executable files are kept as genes in their linear form, but are used in their tertiary forms, after the folding process. The folding, and otherwise prepared 3D forms, can be understood as a compilation process of an executable program. Some proteins, for example the chaperone proteins (fig. 4), can be considered parts of the cell compilers. Example of an executable file is the rRNA file. It is a program for performing operations in the ribosomes, both a pattern recognition operation and an assembly operation. By folding, information-carrying molecules could be treated as executable space programs.

Executable programs may use some of their embedded data structures. Examples are the mRNA binding data structures during the translation process, such as the anti-codon sequence used by tRNA, and the anti-Shine-Dalgarno sequence used by rRNA.

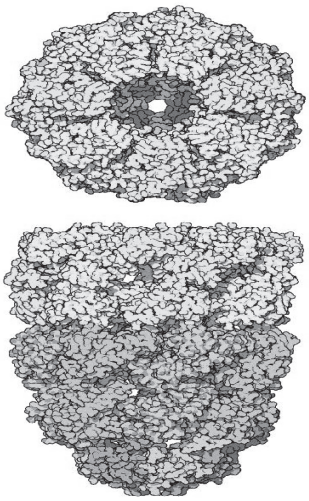

Fig. 4. Chaperone proteins – considered to be part of cell compilers [27]

3.5 Robot compilation in cells

There are two types of compilation that are taking place in the biological cell. The first one is pRNA to protein, and the second pRNA to tRNA or rRNA. An interesting feature of a cell is that through the compilation process a whole machine could be obtained directly from a source code. It consists of either RNA folding of protein synthesis + protein folding

The executable software is represented in the space representation of the cell processors as enzymes and ribosomes. Through the 3D folding of the cell program files, the compilation process could be observed as an embedding process. This

process directly produces cell robots and other cell machines. For example, the tRNA molecule is a cell robot; it is a mobile robot carrying building components, i.e. amino-acids, to the ribosomes, where rRNA is involved in the assembly of proteins. Other remarkable examples of cell robots are the lac-repressor (Fig. 5), that uses its “arms” to fold DNA and disable its access; the dynein, that transports various cellular cargos by "walking" along cytoskeletal microtubules towards the minus-end of the microtuble; the aminoacyl-tRNA synthetase, that loads the amino acids to the appropriate tRNAs; and many more. Let us note that the first relation between molecular genetics and robotics was made in 1985 [6], pointing out that tRNA is a mobile robot.

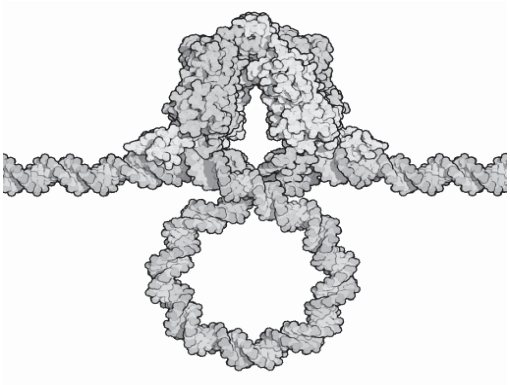

Fig. 5. Lac repressor [27]

Our present technology in building robots assumes that the program development line is considered to be separate from the robot machinery line. The compilation process in the currently existing operating systems takes a file and produces an executable module, which then could end up into an EPROM, which in turn could end up into a robot. Biologically inspired, we should learn from nature to build robots in a single line, from source code to embedded systems. The comparison of the compilation processes in a cell and in a modern computer operating system is given in the figure 6.

4 Conclusions

The lesson that we could learn from the molecular biology is about the specificity of the compilation process. We discussed that the cell robots and other OS embedded systems, such as lac-repressor or tRNA, are developed in a single program compilation line, not separately, as in today’s technologies.

This feature of the genetic systems, the integration of the material and information processing is already of great interest in the scientific community. Today scientists make an effort in bringing information and material processing closer. Nanotechnology science makes it possible to produce nanostructures that do some

kind of processing, while having the means for production of other nanorobots that do specialized work in cells (and possibly self replication) [25]. These robots have their software and their hardware interleaved in the same building-coding blocks. Perhaps, just like the evolution of natural life, the evolution of artificial life will also continue with big steps from the nano level. In human-made systems this feature is not implemented, although there is some effort in this direction [26]. So we could take into consideration that, in order to produce better, more flexible, yet more robust operating systems, perhaps the integration between the hardware and the software, including systems software, is inevitable.

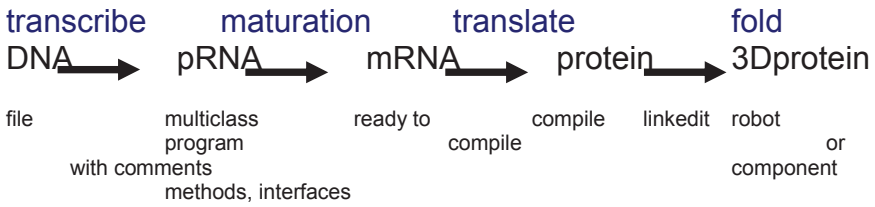

Fig. 6. Comparison – compilation process in a cell and in OS

References

1. Tanenbaum A., Herder J., Bos H., "Can we make Operating systems reliable and secure", *Computer* , may 2006, pp. 44-51.
2. Lee E. "The problem with threads". *Computer*, May 2006, pp.33-42
3. Bozinovski S., Mueller B., diPrimio F. Biomimetic autonomous factories: Flexible manufacturing and cell systems software. GMD Report 115. German National Research Center for Information Technology, Sankt Augustin, 2000
4. Bozinovski S., Jovancevski G., Bozinovska N. "DNA as a real time, database operating system". *Proc SCI 2001*, Orlando, 2001, pp. 65-70
5. Pfeifer R., Scheier C. *Understanding Intelligence.* The MIT Press, 1999
6. Bozinovski S., Hristofi A., Koco I., Sestakov M. Flexible manufacturing systems: basic conceptual model and a method of internal transport. (In Macedonian) Proc. Conf ETAI, Ohrid, Macedonia, 1985 p. 390-398
7. Bozinovski S. Flexible manufacturing systems: A biocybernetic approach. In E. Popov and M. Vukobratovic (eds.) Proc Symp Robotics and Flexible Manufacturing Systems, Moscow, Russia, 1986, p. 192-197
8. Bozinovski S. Flexible manufacturing systems: A biocybernetic approach, (In Russian) Prolemji mashinistroenya i avtomatizacii 16: 31-34, 1987
9. Bozinovski S., Bozinovska L. Flexible production lines in genetics: A model of protein biosynthesis process. Proc. Int. Conf. on Robotics, Dubrovnik, 1987, p. 1-4
10. Bolsover S., Hyams J., Jones S., Shephard E., White H., *From Genes to Cells*. Willey-Liss, 1997
11. Berg J., Tymoczko J., Stryer L., *Biochemistry*, Freeman and Company, 2002
12. Kilian A., Müller B. Life-like learning in technical artifacts: Biochemical vs. neuronal mechanisms. Proc. 9th. International Conference on Neural Information Processing, Singapore, 2002, pp 296–300.

13. Lengeler J., Mueller B., di Primio F. Cognitive abilities of unicellular mechanisms (In German), GMD Report 57, German National Research Center for Information Technology, Sankt Augustin, 1999
14. Monod J., Pardee A., Jacob F. The genetic control of cytoplasmic expression of 'inducibility' in the synthesis of b-galactosidase by Escherichia coli, *Journal of Molecular Biology 1:* 165-178, 1959
15. Gray J. Notes on database operating systems. In *Operating Systems: An Advanced Course*, 393-481, Springer Verlag, 1978
16. Singhal M., Shivaratri N. Advanced Concepts in Operating Systems. McGraw-Hill, 1994
17. Ratner V. *Control Systems in Molecular Genetics.* (In Russian) Nauka, Novosibirsk, 1975.
18. Nutt G. *Centralized and Distributed Operating Systems*. Prentice Hall, 1992
19. Tanenbaum A. Distributed Operating Systems. Prentice Hall, 1995.
20. K. Kruger, P. J. Grabowski, A. J. Zaug, J. Sands, D. E. Gottschling, T. R. Cech, "Self-splicing RNA: Autoexcision and autocyclization of the ribosomal RNA intervening sequence of tetrahymena" Cell, Vol 31, 1982, pp.147-157
21. M. D. Been, "Versatility of Self-Cleaving Ribozymes", Science, Vol. 313, 2006, pp. 1745-1747
22. Lodish H., Berk A., Zipursky L., Matsudaira D., Baltimore D., Darnell J., *Molecular Cell Biology.* Freeman and Company, 2000
23. Gilbert W. Why genes in pieces? Nature, 271, 501, 1978
24. Brown T. *Genetics: A Molecular Approach.* Chapman and Hall, 1992
25. Liao S., Seeman N. C. "Translation of DNA Signals into Polymer Assembly Instructions", Science, Vol. 306, 2004, pp. 2072-2074
26. L. Demeester, K. Eichler, C. H. Loch "Organic Production Systems: What the Biological Cell Can Teach Us About Manufacturing", Manufacturing and Service Operation Management, Vol. 6, No. 2, INFORMS, 2004, pp. 115-132
27. www.rcsb.org/pdb Molecule of the month by David S. Goodsell