Journal of Biological Physics and Chemistry, Vol.10(1), 2010 p.39-43

03BA10P

Biological programming

Gergely Bándi[†] and Jeremy J. Ramsden[‡]

Cranfield University, Bedfordshire, MK43 0AL, England

Biology offers a tremendous set of concepts that are potentially very powerfully usable for the software engineer, but they have been barely exploited hitherto. In this position paper we propose a fresh attempt to create the building blocks of a programming technology that could be as successful as life. A key guiding principle is to develop and make use of unambiguous definitions of the essential features of life.

1. INTRODUCTION

A number of algorithms used in software technology today were originally inspired by or borrowed from biology in some way, and their success suggests that further development in this area could be very fruitful. Some of these algorithms, such as artificial neural networks, cellular automata and the so-called evolutionary algorithms, just to name a few of the better known ones, have indeed no significant competitors. While these algorithms emulate or at least get their main idea from some part of biology as we know it, it is apparent that there are still many other resources existing in nature that are not yet used in software technology [1]. Furthermore, sometimes the basic biological idea was misunderstood or distorted in the borrowing process. Evolutionary algorithms represent an example of this. They are almost always used to optimize something and the process has a clear fixed goal, whereas real biological evolution is open-ended, making it a qualitatively different process.

One possible reason for the great underuse of biological resources could be the great distance between the concepts of biology and software technology. As an example of this distance, consider cancer: it would be hard to perceive the reason for cancer's biological success translated into algorithmic terms without having in place the software equivalent of a working immune system (which confers a selective advantage upon a tumour compared with alien parasites), which in turn implies having an essentially almost entire biological system in place. Furthermore, the persistent paucity of clearly defined biological concepts and structures makes it hard to emulate biological phenomena in computer science.

Hence, responding to the challenge of exploiting biological knowledge requires the creation of a much more extensive common ground between the two fields than exists at present. This in itself poses a challenge, because there is a vast amount of domain-specific knowledge in biology that is not used in any other science. Even in materials science, where there is strong interest in mimicking some of the extraordinary materials found in nature, such as spider silk or the feet of the gecko, there has been little attempt to establish the inner workings that biology uses to create these materials; the mimicry might only go as far as some possibly rather superficial structural features, and functionally the result is likely to be very inferior. For example, "artificial muscle" is typically merely a responsive gel.

An important landmark was the appearance of Gerd Sommerhoff's book *Analytical Biology* [2], in which he described the basics of adaptation in biology in mathematical terms and through that definition was able to go on to describe more complex concepts like learning and evolution. While this was still only a part (albeit a very important and quintessential one) of biology, it well demonstrated how a few basic conceptual building blocks can go a long way toward describing much more advanced features in great detail. This demonstration is of great value to us: since mathematics is one of the bases of software technology, Sommerhoff's results can be directly used to describe biology in software terms. This is the starting point of the present paper.

2. BASIC BIOLOGICAL CONCEPTS

The first task in creating software-based biological life (i.e., the emulation of life) is to identify the most important qualities that life possesses and its main building blocks. Note that the granularity of the emulation is of crucial importance. It needs to be fast enough to work as a software method and to be able to emulate complex dynamics, but it needs to be accurate enough to capture the important internal processes of biology. Of course, were it as accurate as possible it would be a direct simulation and would contain far more detail than necessary to be able to clearly perceive the important concepts.[1]

[†] g.bandi@cranfield.ac.uk
[‡] j.ramsden@cranfield.ac.uk

[1] This is the approach adopted by existing projects such as E-Cell (http://www.e-cell.org/ecell/ and http://dev.e-cell.org/ [3,4])

Journal of Biological Physics and Chemistry **10** (2010) 39–43
Received 24 March 2010; *accepted* 31 March 2010

Moreover it would be almost impossible to make meaningful deductions from the emulation runs. Hence, the aim in this paper is to place the complexity of the proposed software somewhere between the complexity of ordinary contemporary algorithms and the complexity of a real biological entity.

The most important concepts chosen to define the virtual organism are:

- life/living
- organism
- organ
- tissue
- cell.

Note that these concepts are organized into a hierarchy. They will be elaborated upon in §4.

3. THE CREATION OF A VIRTUAL LIVING ORGANISM

Our aim is to create a *cell-based* virtual living organism (VLO) that would accomplish what we expect from software today. All the work is to be done on a cellular level and, as in nature, the cells will work concurrently. The cells and the virtual organisms made from them should communicate similarly to their organic counterparts and a congeries of them would create a living ecosystem to achieve defined goals, just like today's software systems.

To realize this aim, we define a programming paradigm, and shall realize and demonstrate it with a corresponding programming tool such as a software library or even (in outline) a programming language.

The result could be used for many purposes. It could be a tool for programmers to create products that are radically different in their philosophy than today's software products. It could be also a research platform to create new biologically inspired algorithms by modelling and emulating real biological processes. Theoretical biological research could also use this tool to model and emulate cutting edge theories of the mechanisms underlying observable phenomena much faster than a traditional simulation, and to a far greater extent, just like the prototyping tools frequently used in software engineering.

4. IDENTIFICATION AND DEFINITIONS OF THE BASIC CONCEPTS

Our starting point is the definitions found in dictionaries. They outline the required properties of the entities. Of course not all the properties can be applied in every case; in other words these are sufficient requirements, not necessary ones. We propose that the properties can be captured by the following:

Life/Living:

- has metabolism
- has growth (self-extension)
- has reproduction (self-replication)
- responds to stimuli (implying sensorial functions)
- can adapt to the environment (implying control functions)
- has self-preserving ability (self-regulation)
- has organs performing functions[2]
- has organization at many levels

Organism:

- is a structure of interdependent and subordinate elements (organs) whose relations and properties are largely determined by their function in the whole
- is able to carry on the activities of life by means of organs separate in function but mutually dependent
- is an individual form of life
- is a body made up of organs (and/or other parts) that work together for a common cause

Organ:

- is a differentiated structure consisting of cells and tissues
- is a biological unit of an organism
- performs a function or group of functions or coöperates in an activity
- usually comprises a main tissue unique to the organ together with sporadic tissues

Tissue:

- is an aggregate of interconnected morphologically similar cells that have similar structure and function, together with the intercellular substance

Cell:

- is the smallest structural unit of living matter
- is an autonomous self-replicating unit
- is either a functional, independent unit of life or a subunit of a multicellular organism
- is typically specialized into carrying out a restricted set of functions.

5. THE BASIC STRUCTURE AND DETAILS OF ITS LAYERS

The basic structure of the organism has several levels. To repeat, all the work will be done at the cellular level, which is the lowest level in our hierarchy. All the other

and "virtual cell" or VCell (http://vcell.org/bionetgen/ and http://www.nrcam.uchc.edu/vcell_software/login.html), which seek to explicitly include every chemical reaction in the cell. It is, however, doubtful whether this approach would be practicable for a multicellular organism. Even the existence of internal structure in a single cell poses some problems in simulating the biological processes.

[2] An individual cell has organelles or transient molecular structures for this purpose.

levels have two purposes: (i) structural; and (ii) providing services. Every level has several requirements for properties and services (with the exception of the organ systems, which are strictly a tool for theoretical structuring). All the services each level has to provide are enabled via levels beneath them, ending with the cells doing the actual work. Figs 1 and 2 depict the main structure.

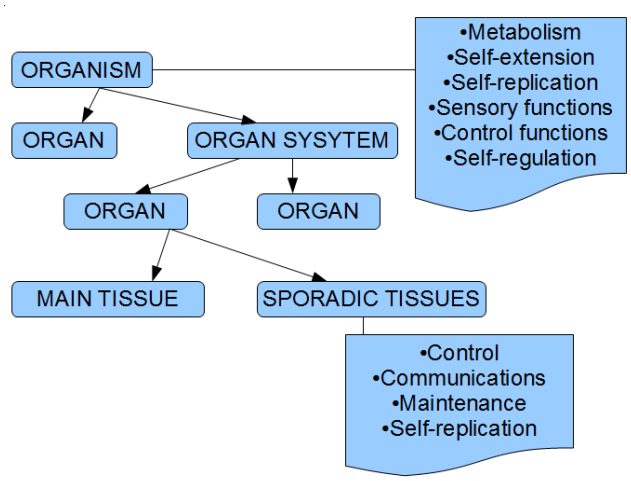

Figure 1. The basic structure of the virtual living organism (VLO) down to tissue level with a set of properties and functional requirements. An organ system is a group of organs working together to carry out a bigger task.

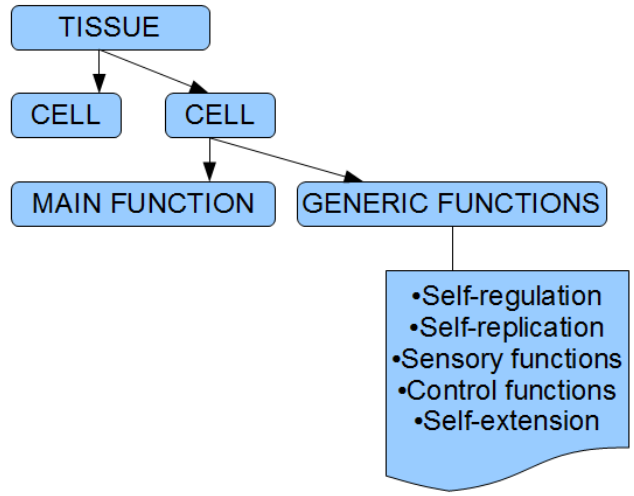

Figure 2. The basic structure of the virtual living organism (VLO) from tissue level down to cellular level with examples of generic cell functions.

5.1 Cells in more detail

Cells are essentially biological machines. They have inputs and outputs and their work connects the two. In biology most cells' inputs are (bio)chemical compounds, but sometimes they can also be some kind of energy, like light or electricity. These inputs can easily be translated into software terms as data, regardless of content (e.g., a record of an event or the previous result of another cell's work). Outputs in biology are, again, typically (bio)chemicals, but can also be energy or physical work. They can be mostly translated as data, but could also be any kind of interaction with the internal or external environment (e.g., a property change), cf. input–output (IO) operations.

Cells can be considered to be only doing a fairly simple job within the process they belong to, but by working together they can achieve almost any kind of complex task necessary for the organism. When a chain of cells is carrying out some complicated task, sometimes specialized transport cells are responsible for transmitting the interim results of the process.

In the VLO, cells give out jobs to other cell types, and wait for the job results whenever they are needed. This is similar to the current implementation of functions today, except that the results are not spontaneously incorporated, but need to be awaited when the next step depends on them. For example, if a cell of type A needs the services of a cell of type B, the A cell will emit a job request to any cell of type B, after which it will continue with its work until the results are required for the next step, upon which it will sleep until the results arrive. When a cell of type B decides to process the received job request, it computes the result and sends it back to the specific cell that issued the job. One can think about these job requests as chemicals: they stack up around the cell as part of the intercellular substance. This substance is part of the tissue by definition.

5.2 Tissues in more detail

Tissues are collections of similar cells together with their intercellular substance. This means that the job queueing is done inside the tissue. This is the place where cells can be directly addressed if needed for any technical purpose, which implies that in the VLO a tissue can be seen as a collection of job queues and sets of cells, with a few standard services that all cells use. Services are included in the tissue to emulate the laws that cells are not able to circumvent. For example, if cells could control addressing and how jobs are handled, they could accomplish many unnatural manoeuvres (i.e., ones that cannot be found in biology). Of course, if required some exceptions can be allowed.

The types of cells found inside a tissue are those that work closely together to achieve a higher goal. In programming terms tissues can be thought of as a bigger functional unit incorporating all the smaller functions execution relies on (within reason).

5.3 Organs in more detail

Organs consist of a main tissue and sporadic tissues. The main tissue carries out the main functions of an organ while the sporadic tissues provide standard services to the organ and to the cells of the main tissue. Every cell can more or less easily access anything within its own organ. To reach anything outside, they need to rely on, for example, the communication service of the organ, which is realized implemented by the cells belonging to the communication sporadic tissue of any organ.

5.4 Organisms in more detail

Organisms can consist of organ systems or organs. An organ system might be a user interface containing several organs controlling the input and output in multiple available media. An organism has to have a few default organs that will ensure that the organism fits the definition given earlier (§4).

6. ADVANTAGES AND POSSIBILITIES OF THE VIRTUAL ORGANISM STRUCTURE

From a software engineering point of view, having the structure described in this paper brings several interesting possibilities. By making smaller parts of the program independent threads working from a job queue, it becomes possible to achieve a dynamic load balancing by causing cells with a high workload to divide and thus making their number of threads relatively higher than in other parts of the program (e.g., haematopoiesis).

Another advantageous possibility is upgrading cells with newer versions without generally slowing down the system. By uncoupling the old version of a cell from its job queues and replacing it with the newer version, the new one can start working while the old ones finish what they have started before the upgrade was initiated. If the messaging system is complex enough, the new type of cells could even be inside another identical organism, or maybe even on another computer, hence instead of upgrading the process would be a seamless relocation. Of course for this to work, other technical difficulties would have to be solved as well.

7. COMPARISON WITH EXISTING TECHNOLOGIES AND TOOLS

Several existing technologies and tools may appear to have similar main concepts to the ones outlined in this paper. Let us compare those that are the most similar and see the conceptual differences.

An obviously different technology but one still worth mentioning is the cellular automaton. These machines have independent cells doing the work somewhat along the lines of what is outlined here, but these cells are usually simple ones and spacial orientation is very important. The goal of cellular automata is to do a very specific, usually modelling-related task: they are used to model decentralized behaviour and are far less structured than the model proposed here.

Multiagent systems represent a more complex version of independently working cell-like behaviour. These systems have a strictly decentralized topology with only local views allowed for agents. While this is fairly close to reality for most biological cells, in fact the cells constituting an organism are ordered according to a very well defined topology, and in some sense can have a global view of the whole system (e.g., in the embryonic phase).

Hardware engineering uses the concept of embryonics as a biological idea for designing hardware elements. This technology is conceptually restricted to a few parts of biology only and does not imitate whole structures like a complex organism, but rather resembles a multicellular system. Still, within its domain it has some goals and tools similar to what the present work aims to realize in software.

Among existing software tools there are also a few similar examples. One of these is the language called *little b*.[3] The creation of this language was based on a starting idea almost identical to that of the present work in order to create a model that can generally describe biology; the difference is that this language has goals that require it to be much more granular. It focuses on systems biology, but is rooted at the molecular level, turning it into something more like a simulation from our viewpoint. Aiming to emulate the inner workings of a complex organism makes it necessary for us to make our model much more abstract than *little b*.

Another tool that is formally somewhat similar is the multiagent-based Artificial Life Framework.[4] This is a Java framework that implements multithreaded agents with their own internal mechanics (like messaging). While this project is not based on biology, its details bear some resemblance to an initial goal of ours, namely to create a C++ library to simulate agent-like cells running in their own threads.

8. CONCLUSIONS

A new programming paradigm is outlined, based on the essential features of biology. In the absence of a

[3] http://www.littleb.org/
[4] http://www.artificiallife.org/

generally agreed concept system for biology, these essential features are captured from current literature, most usefully from dictionary definitions, which seek to reflect current usage.

The paradigm should be considered as emulation rather than simulation, hence quite different from the fine-grain representations of life such as E-Cell. Particular emphasis is laid on emulating the quintessential features of life rather than what might satisfice current needs, as is often done in biologically inspired software. This should make it feasible to continue to expand the paradigm *pari passu* with increasing knowledge and understanding of biology.

One practical way in which this might be useful would be in the emulation of cancer, where it could yield a prediction of the probability of creating a viable aneuploidic cell (assuming that aneuploidy is the most characteristic feature associated with cancer [5]), and of functions such as the dependence of cancer incidence on the age of an organism, and even suggest hitherto unsuspected ways of preventing the further growth of existing tumours.

REFERENCES

1. Banzhaf, W., Beslon, G., Christensen, S., Foster, J.A., Képès, F., Lefort, V., Miller, J.F., Radman, M. and Ramsden, J.J. From artificial evolution to computational evolution: a research agenda. *Nature Rev. Genetics* **7** (2006) 729–735.
2. Sommerhoff, G. *Analytical Biology*. Oxford: University Press (1950).
3. Saito, Y., Hashimoto, K., Sakurada, T., Fujita, Y., Takahashi, K. and Tomita, M. Design and development of software environment for whole-cell simulation. *Genome Informatics* **12** (2001) 316–317.
4. Takahashi, K., Yugi, K., Hashimoto, K., Yamada, Y., Pickett, C.J.F. and Tomita, M. Computational challenges in cell simulation: a software engineering approach. *IEEE Intelligent Systems*, September/October 2002, pp. 64–71 and references therein.
5. Duesberg, P., Li, R., Fabarius, A. and Hehlmann, R. The chromosomal basis of cancer. *Cell. Oncol.* **27** (2005) 293–318.

Biological programming

Ramsden, Jeremy J.

2010-03

Jeremy J. Ramsden and Gergely Bandi, Biological programming, Journal of Biological Physics and Chemistry, 2010, Vol. 10, No. 1, pg 39

http://dspace.lib.cranfield.ac.uk/handle/1826/4725