Abstract Machines of Systems Biology (Extended Abstract)

Luca Cardelli

Microsoft Research Cambridge
7, J.J. Thomson Avenue, Cambridge, CB3 0FB, UK
E-mail: luca@microsoft.com

Summary. Living cells are extremely well-organized autonomous systems, consisting of discrete interacting components. Key to understanding and modelling their behavior is modelling their system organization, which can be described as a collection of distinct but interconnected abstract machines. Biologists have invented a number of notations attempting to describe, abstractly, these abstract machines and the processes that they implement. *Systems biology* aims to understand how these abstract machines work, separately and together.

1 Introduction

Following the discovery of the structure of DNA, just over 50 years ago, molecular biologists have been unravelling the functioning of cellular components and networks. The amount of molecular-level knowledge accumulated so far is absolutely amazing. And yet we cannot say that we understand *how a cell works*, at least not to the extent of being able to easily modify or repair a cell. The process of understanding cellular components is far from finished, but it is becoming clear that simply obtaining a full part list will not tell us how a cell works. Rather, even for substructures that have been well characterized, there are significant difficulties in understanding how components interact as a *system* to produce the observed behavior. Moreover, there are just too many components, and too few biologists, to analyze each component in depth in reasonable time. Similar arguments apply also to each level of biological organization above the cellular level.

Enter *systems biology*, which has two aims. The first is to obtain massive amounts of information about whole biological systems, via high-throughput experiments that provide relatively shallow and noisy data. The Human Genome Project is a prototypical example: the knowledge it accumulated is highly valuable, and was obtained in an automated and relatively efficient way, but is just the beginning of understanding the human genome. Similar effort are now underway in *genomics* (finding the collection of all genes, for many genomes), in *transcriptomics* (the collection of all actively transcribed genes), in *proteomics* (the collection of all proteins), and in *metabolomics* (the collection of all metabolites). *Bioinformatics* is the rapidly growing discipline tasked with collecting and analyzing such *omics* data.

The other aim of systems biology is to build, with such data, a science of the *principles of operation* of biological systems, based on the *interactions between components*. Biological systems are obviously well-engineered: they are very complex and yet highly structured and robust. They have only one major engineering defect: they have not been designed, in any standard sense, and so are not laid out as to be easily understood. It is not clear that any of the engineering principles of operations we are currently familiar with are fully applicable. Understanding such principles will require an interdisciplinary effort, using ideas from physics, mathematics, and computing. Here, then, are the promises of systems biology: it will teach us new principles of operation, likely applicable to other sciences, and it will leverage other sciences to teach us *how cells work* in an actionable way.

In this paper, we look at the organization of biological systems from an information science point of view. The main reason is quite pragmatic: as we increasingly map out and understand the complex interactions of biological components, how can we *write down* such knowledge, in such a way that we can inspect it, animate it, and understand its principles? For genes, we can write down long but structurally simple strings of nucleotides in a 4-letter alphabet, that can be stored and queried. For proteins we can write down strings of amino acids in a 20-letter alphabet, plus three-dimensional information, which can be stored a queried with a little more difficulty. But how shall we write down *biological processes*, so that they can be stored and queried? It turns out that biologists have already developed a number of informal notation, which will be our starting points. These notations are abstractions over chemistry or, more precisely, are abstractions over a number of biologically relevant chemical toolkits.

2 Abstract Machines

An *abstract machine* is a fictional information-processing device that can, in principle, have a number of different physical realizations (mechanical, electronic, biological, or even software). An abstract machine is characterized by:

- a collection of discrete states,
- a collection of operations (or events) that cause discrete transitions between states.

The evolution of states through transitions can in general happen concurrently. The adequacy of this generic model for describing complex systems is argued, e.g., in [22].

Different chemical toolkits studied in biochemistry, can be seen as a separate abstract machines with appropriate sets of states and operations. The abstract machines we consider here are each grounded in a different chemical toolkit (nucleotides, amino acids, and phospholipids), and hence have some grounding in reality. Moreover, each abstract machine corresponds to a different kind of informal *algorithmic notation* that biologists have developed (Figure 1, dotted bubbles): this is further evidence that abstract principles of organization are at work.

The *Gene Machine* (better known as Gene Regulatory Networks) performs information processing tasks within the cell. It regulates all other activities, including assembly and maintenance of the other machines, and the copying of itself. The *Protein Machine* (better known as Biochemical Networks) performs all mechanical and metabolic tasks, and also some signal processing. The *Membrane Machine* (better known as Transport Networks) separates different biochemical environments, and also operates dynamically to transport substances via complex, discrete, multi-step processes.

These three machines operate in concert and are highly interdependent. Genes instruct the production of proteins and membranes, and direct the embedding of proteins within membranes. Some proteins act as messengers between genes, and others perform various gating and signaling tasks when embedded in a membrane. Membranes confine cellular materials and bear proteins on their surfaces. In eukaryotes, membranes confine the genome, so that local conditions are suitable for regulation, and confine other reactions carried out by proteins in specialized vesicles.

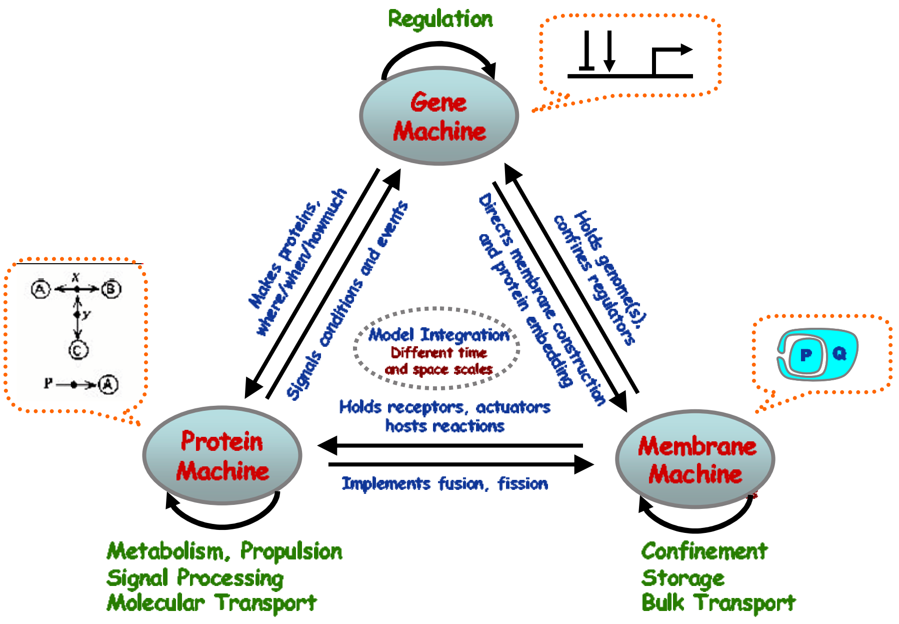
Regulation
Gene Machine
Makes proteins, where/when/howmuch
Signals conditions and events
Directs membrane construction and protein embedding
Holds genome(s), confines regulators
Model Integration
Different time and space scales
Holds receptors, actuators hosts reactions
Implements fusion, fission
Protein Machine
Membrane Machine
Metabolism, Propulsion
Signal Processing
Molecular Transport
Confinement
Storage
Bulk Transport
A
B
C
x
y
P
Q

Figure 1 Abstract Machines, Molecular Basis, and Notations

Therefore, to understand the functioning of a cell, one must understand also how the various machines interact. This involves considerable difficulties (e.g., in simulations) because of the drastic difference in time and size scales: proteins interacts in tiny fractions of a second, while gene interactions take minutes; proteins are large molecules, but are dwarfed by chromosomes, and membranes are larger still. Before looking at the interactions among the different machine in more detail, we start by discussing each machine separately.

3 The Protein Machine (Biochemical Networks)

Proteins are long folded-up strings of amino acids with precisely determined, but often mechanically flexible, three-dimensional shapes. If two proteins have surface regions that are complementary (both in shape and in charge), they may stick to each other like Velcro, forming a protein **complex** where a multitude of small atomic forces crates a strong bond between individual proteins. They can similarly stick highly selectively to other substances. During a **complexation** event, a protein may be bent or opened, thereby revealing new interaction surfaces. Through complexation many proteins act as enzymes: they bring together compounds, including other proteins, and greatly facilitate chemical reactions between them without being themselves affected.

Proteins may also chemically modify each other by attaching or removing small phosphate groups at specific sites. Each such site acts as a boolean switch: over a dozen of them can be present on a single protein. Addition of a phosphate group (**phosphorilation**) is performed by an enzyme that is then called a **kinase**. Removal of a phosphate group (**dephosphorilation**) is performed by an enzyme that is then called a **phosphatase**. For example, a *protein phosphatase kinase* is a protein that phosphorilates a protein that phosphorilates a protein that dephosphorilates a protein. Each (de-)phosphorilation may reveal new interaction surfaces, and each surface interaction may expose new phosphorilation sites.

It turns out that a large number of protein interactions work at the level of abstraction just described. That is, we can largely ignore chemistry and the protein folding process, and think of each protein as a collection of features (binding sites and phosphorilation sites) whose availability is affected by (de-)complexation and (de-)phosphorilation interactions.

Finding a suitable language in which to cast such an abstraction is a non-trivial task. Kohn designed a graphical notation for networks of interacting proteins [28]. This was a tremendous achievement, summarizing hundreds of technical papers in page-sized pictures, while providing a sophisticated and expressive notation that could be translated back into chemical equations (according to semi-formal guidelines). Because of this intended chemical semantics, the dynamics of a systems is implied in Kohn's notation, but only by translation to chemical (and hence kinetic) equations. The notation itself has no dynamics, and this is one of its main limitation. The other major limitation is that, although a graphical notation is very appealing, it tends to stop being useful when it overflows the borders of a page or of a whiteboard (the original Kohn maps span several pages).

Other notations for the protein machine can be devised. Kitano, for example, improves on the conciseness, expressiveness, and precision of Kohn's notation [27]; further sophistication in graphical notation will certainly be required along the general principles of [18]. A different approach is to devise a textual notation, which inherently has no "page-size" limit and can better capture dynamics; examples are Bio-calculus [37], and most notably κ-caculus [14], [15], whose dynamics is fully formalized. But one may not need to invent completely new formalisms. Regev and Shapiro, in pioneering work [47], [45], described how to represent chemical and biochemical interactions within existing process calculi (π-calculus). Since process calculi have a well understood dynamics (better understood, in fact, than most textual notations that one may devise just for the purpose), that approach also provides a solid basis for studying systems expressed in such a notation. Finally, some notations incorporate both continuous and discrete aspects, as in Charon [3].

In summary, the fundamental flavor of the Protein Machine is: *fast synchronous binary interactions*. Binary because interactions occur between two complementary surfaces, and because the likelihood of three-party instantaneous chemical interactions can be ignored. Synchronous because both parties potentially feel the effect of the interaction, when it happens. Fast because individual chemical reactions happen at almost immeasurable speeds. The parameters affecting reaction speed, in a well-stirred solution, are just a reaction-specific rate constant having to do with surface affinity, plus the concentrations of the reagents (and the temperature of the solution, which is usually assumed constant). Concentration affects the likelihood of molecules randomly finding each other by Brownian motion. Note that Brownian motion is surprisingly effective at a cellular scale: a molecule can "scan" the equivalent volume of a bacteria for a match in 1/10 of a second, and it will in fact scan such a bounded volume because random paths in 3D do not return to the origin.

4 The Gene Machine (Gene Regulatory Networks)

The *central dogma of molecular biology* states that DNA is transcribed to RNA, and RNA is translated to proteins (and then proteins do all the work). This dogma no longer paints the full picture, which has become considerably more detailed in recent years. Without entering into a very complex topic [32], let us just note that some proteins go back and bind to DNA. Those proteins are called **transcription factors** (either **activators** or **repressors**); they are produced for the purpose of allowing one gene (or signaling pathway) to communicate with other genes. Transcription factors are not simple messages: they are proteins, which means they are subject to complexation, phosphorilation, and programmed degradation, which all have a role in gene regulation.

A **gene** is a stretch of DNA consisting of two (not necessarily contiguous or unbroken) regions: an *input* (**regulatory**) *region*, containing **protein binding**

sites (for transcription factors) and an *output* (**coding**) *region, coding for one or more prot*eins that the gene produces. Sometimes there are two coding regions, in opposite directions [44], on count of DNA being a doubly-linked list. Sometimes two genes overlap on the same stretch of DNA.

The output region functions according to the **genetic code**: a well understood and almost universal table mapping triplets of nucleotides to one of about 20 amino acids, plus start and stop triplets. The input region functions according to a much more complex code that is still poorly understood: transcription factors, by their specific 3D shapes, bind to specific nucleotide sequences in the input region, with varying binding strength depending of the precision of the match.

Thus, the gene machine, although entirely determined by the digital information coded in DNA, is not entirely digital in functioning: a digitally encoded protein, translated and folded-up, uses its "analog" shape to recognize another digital string and promote the next step of translation. Nonetheless, it is customary to ignore the details of this process, and simply measure the effectiveness with which (the product of) a gene affects another gene. This point of view is reflected in standard notation for gene regulatory networks (e.g., see [16]).

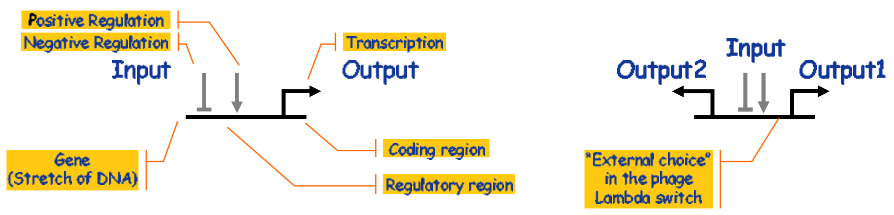

Figure 2 The Gene Machine Instruction Set

In Figure 2, a gene is seen as a hardware gate, and the genome can be seen as a vast circuit composed of such gates. Once the performance characteristics of each gate is understood, one can understand or design circuits by combining gates, almost as one would design digital or analog hardware circuits. The performance characteristics of each gene in a genome is probably pretty unique. Hence, as in the protein machine, we are going to have thousands of "primitive instructions": one for each gene.

The state of a gene machine is the concentrations of the transcription factors produced by each gene (or arriving from the environment). The operations, again, are the input-output functions of each gene. But what is the "execution" of a gene machine? It is not as simple as saying that one gene stimulates or inhibits another gene by a certain factor. It is known that certain genes perform complex computations on their inputs that are a mixture of boolean, analog, and multi-stage operators [51]. Therefore, the input region of each gene can itself be a sophisticated machine.

Whether the execution of a gene machine should be seen as a continuous or discrete process, both in time and in concentration levels, is already a major question. Qualitative models (e.g.: asynchronous automata [49], network motifs [35]) can provide more insights that quantitative models, whose parameters are hard to come by and are possibly not critical. On the other hand, it is understood that pure boolean models are inadequate in virtually all real situations. Continuous, stochastic, and decay aspect of transcription factor concentrations are all critical in certain situations [33], [50].

Despite all these difficulties and uncertainties, a single notation for the gene machine is in common use [16]. There, the gates are connected by either "excitatory" (pointed arrow) or "inhibitory" (blunt arrow) links. What that might mean exactly is often left unspecified, except that, in a common model, a single constant weight is attached to each link.

Any serious publication would actually start from a set of ordinary differential equations relating concentrations of transcription factors, but this is only feasible for small networks. The best way to formalize the *notation* of gene regulatory networks is still subject to debate and many variations, but there is little doubt that formalizing such a notation will be essential to get a grasp on gene machines the size of genomes (the smallest of which, M.Genitalium, is on the order of 150 Kilobytes, and one closer to human cellular organization, Yeast, is 3 Megabytes).

In summary, the fundamental flavor of the Gene Machine is: *slow asynchronous stochastic broadcast*. The interaction model is really quite strange, by computing standards. Each gene has a fixed output, which is not quite an address for another gene: it may bind to a large number of other genes, and to multiple locations on each gene. The transcription factor is produced in great quantities, usually with a well-specified time-to-live, and needs to reach a certain threshold to have an effect. On the other hand, various mechanisms can guarantee Boolean-like switching when the threshold is crossed, or, very importantly, when a message is not received. Activation of one gene by another gene is slow by any standard: typically one to five minutes, to build up the necessary concentration[1]. However, the genome can slowly direct the assembly-on-need of protein machines that then act fast: this "swap time" is seen in experiments that switch available nutrients. The stochastic aspect is fundamental because, e.g., with the same parameters, a circuit may oscillate under stochastic/discrete semantics, but not under deterministic/continuous semantics [50]. One reason is that a stochastic system may decay to zero molecules of a certain kind at a given time, and this can cause switching behavior, while a continuous system may asymptotically decay only to a non-zero level.

[1] Consider that bacteria replicate in only 20 minute while cyclically activating hundreds of genes. It seems that, at least for bacteria, the gene machine can make "wide" but not very "deep" computations, [35].

5 The Membrane Machine (Transport Networks)

A cellular membrane is an oriented closed surface that performs various molecular functions. Membranes are not just containers: they are coordinators and sites of major activity[2]. Large functional molecules (proteins) are embedded in membranes with consistent orientation, and can act on both sides of the membrane simultaneously. Freely floating molecules interact with membrane proteins, and can be sensed, manipulated, and pushed across by active molecular channels. Membranes come in different kinds, distinguished mostly by the proteins embedded in them, and typically consume energy to perform their functions. The consistent orientation of membrane proteins induces an orientation on the membrane.

One of the most remarkable properties of biological membranes is that they form a two-dimensional fluid (a lipid bilayer) embedded in a three-dimensional fluid (water). That is, both the structural components and the embedded proteins freely diffuse on the two-dimensional plane of the membrane (unless they are held together by specific mechanisms). Moreover, membranes float in water, which may contain other molecules that freely diffuse in that three-dimensional fluid. Membrane themselves are impermeable to most substances, such as water and protons, so that they partition the three-dimensional fluid. This organization provides a remarkable combination of freedom and structure.

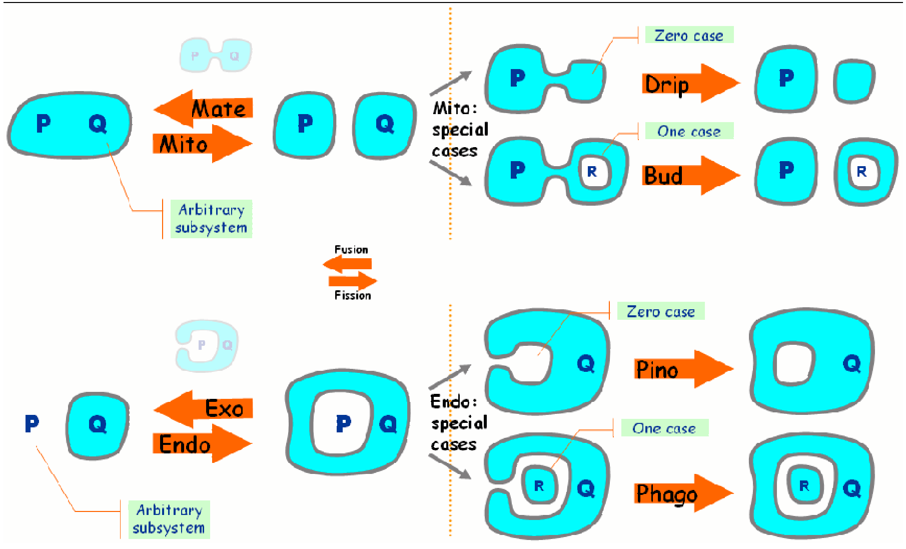

Figure 3 The Membrane Machine Instruction Set (2D)

[2] "For a cell to function properly, each of its numerous proteins must be localized to the correct cellular membrane or aqueous compartment." [31], p.675.

Many membranes are highly dynamic: they constantly shift, merge, break apart, and are replenished. But the transformations that they support are naturally limited, partially because membranes must preserve their proper orientation, and partially because membrane transformations need to be locally–initiated and continuous. For example, it is possible for a membrane to gradually buckle and create a bubble that then detaches, or for such a bubble to merge back with a membrane. But it is not possible for a bubble to "jump across" a membrane (only small molecules can do that), of for a membrane to turn itself inside-out.

The basic operations on membranes, implemented by a variety of molecular mechanisms, are *local fusion* (two patches merging) and *local fission* (one patch splitting in two) [8]. We discuss first the 2D case (which is instructive, and for which there are some formal notations) and then the 3D case (the real one, for which there are no formal notations).

In two dimensions (Figure 3), at the local scale of membrane patches, fusion and fission become indistinguishable as a single operation, *switch*, that takes two membrane patches, i.e., to segments A-B and C-D, and switches their connecting segments into A-C and B-D (crossing is not allowed). We may say that, in 2D, a switch is a fusion when it decreases the number of whole membranes, and is a fission when it increases such number.

When seen on the global scale of whole 2D membranes, switch induces four operations: in addition to the obvious merging (Mate) and splitting (Mito) of membranes, there are also operation, quite common in reality, that cause a membrane to "eat" (Endo) or "spit" (Exo) another subsystem. There are common special cases of Mito and Endo, when that subsystem consists of zero (Drip, Pino) or one (Bud, Phago) membrane. All these operations ensure that what is or was *outside* the cell never gets mixed with what is *inside*. The main reactions that violate this invariant are destructive and non-local ones (such a digestion, not shown). Note that Mito/Mate preserve the nesting depth of subsystems, and hence they cannot encode Endo/Exo; instead, Endo/Exo can encode Mito/Mate [12].

In three dimensions, the situation is more complex (Figure 4). There are 2 distinct local operations on surface patches, inducing 8 distinct global operations that change surface topology. *Fusion* joins two Positively curved patches (in the shapes of domes) into one Negatively curved patch (in the shape of a hyperbolic cooling tower) by allowing the P-patches to kiss and merge. *Fission* instead splits one N-patch into two P-patches by pinching the N-patch. Fusion does not necessarily decrease the number of membranes in 3D (it may turn a sphere into a torus in two different ways: T-Endo T-Mito), and Fission does not necessarily increase the number of membranes (it may turn a torus into a sphere in two different ways: T-Exo, T-Mate). In addition, Fusion may merge two spheres into one sphere in two different ways (S-Exo, S-Mate), and Fission may split one sphere into two spheres in two different ways (S-Endo, S-Mito). Note that S-Endo and T-Endo have a common 2D cross section (Endo), and similarly for the other three pairs.

Cellular structures have very interesting dynamic topologies: the eukaryotic nuclear membrane, for example, is two nested spheres connected by multiple toroidal

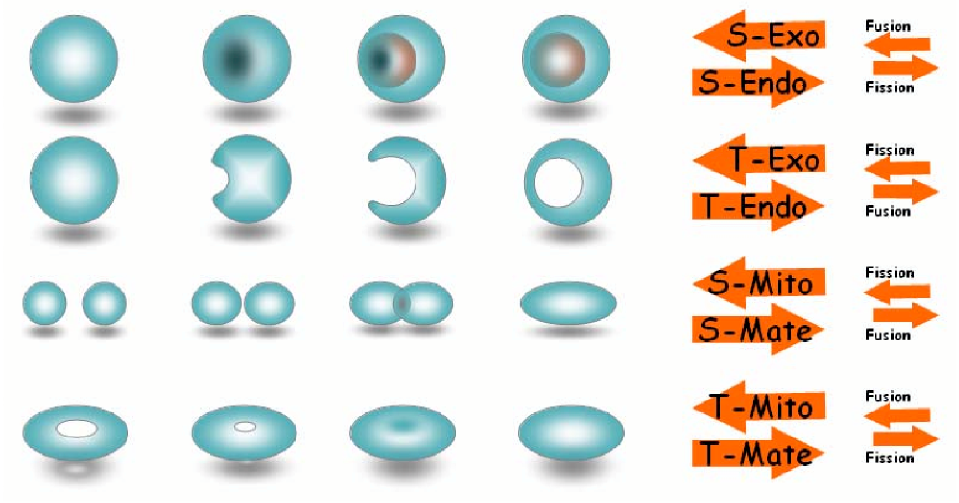

Figure 4 The Membrane Machine Instruction Set (3D)

holes (and also connected externally to the Endoplasmic Reticulum). This whole structure is disassembled, duplicated, and reassembled during cellular mitosis. Developmental processes based on cellular differentiation are also within the realm of the Membrane Machine, although geometry, in addition to topology, is an important factor there.

The informal notation used to describe executions of the Membrane Machine does not really have a name, but it can be seen in countless illustrations (e.g., [31], p.730). All the stages of a whole process can be seen in a single snapshot, with arrows denoting operations (Endo/Exo etc.) that cause transitions between states. This kind of depiction is natural because often all the stages of a process *are* seen at once, in photographs, and much of the investigation has to do with determining their proper sequence and underlying mechanisms.

Some membrane-driven processes are semi-regular, and tend to return to something resembling a previous configuration, but they are also stochastic, so no static picture or finite-state-automata notation can tell the real story. Spectacular membrane dynamics can be found in the protein secretion pathway, through the Golgi system, and in many developmental processes. Here too there is a need for a precise dynamic notation that goes beyond static pictures; there are only a few of those, currently [41], [46], [12].

In summary, the fundamental flavor of the Membrane Machine is: fluid-in-fluid architecture, membranes with embedded active elements, and fusion and fission of compartments. Although dynamic compartments are common in computing, operations such as endocytosis and exocytosis have never explicitly been suggested as fundamental. They embody important invariants that help segregate cellular materials from environmental materials. The distinction between active elements *embedded* on the surface of a compartment, vs. active elements *contained* in the compartment, becomes crucial with operations such as Exo. In the former case, the active elements are retained, while in the latter case they are lost to the environment.

6 Three Machines, One System

Three classes of chemicals, among others, are fundamental to cellular functioning: nucleotides (nucleic acids), amino acids (proteins), and phospholipids (membranes). Each of our abstract machines is based primarily on one of these classes of chemicals: amino acids for the protein machine, nucleotides for the gene machine, and phospholipids for the membrane machine.

These three classes of chemicals are however heavily interlinked and interdependent. Some enzyme are actually made of nucleotides (RNA) instead of amino acids, or by a combination of both. The gene machine "executes" DNA to produce proteins (through RNA intermediaries), but some of those proteins (and some RNA), which have their own dynamics, are then used as control elements of DNA transcription. Membranes are fundamentally sheets of pure phospholipids, but in living cells they are heavily doped with embedded proteins which modulate membrane shape and function. Some RNA translation happens only through membranes, with the RNA input on one side, and the protein output on the other side or threaded into the membrane.

Therefore, the abstract machines are interlinked as well, as illustrated in Figure 1. Ultimately, we will need a single notation in which to describe all three machines (and more), so that a whole organism can be described.

What could a single notation for all three machines (and more) look like? All formal notations known to computing, from Petri Nets to term-rewriting systems, have already been used to represent aspects of biological systems; we shall not even attempt a review here. But none, we claim, has shown the breadth of applicability and scalability of process calculi [34], partially because they are not a single notation, but a coherent conceptual framework in which one can derive suitable notations. There is also a general theory and notation for such calculi [36] which can be seen as the formal umbrella under which to unify different abstract machines.

Major progress in using process calculi for describinh biological systems was achieved in [45], where it is argued that one of the standard existing process calculi, π-calulus, enriched with a stochastic semantics [24], [42], [43], is extraordinarily suitable for describing both molecular-level interactions and higher levels of organization. The same stochastic calculus is now being used to describe genetic networks [29]. For membrane interactions, though, we need something beyond ordinary process calculi, which have no notion of compartments. Already [45], [46] adapted Ambient Calculus [13] (which extends π-calculus) to represent biological compartments and complexes. A more recent attempt, Brane Calculus [12], embeds the biological invariants and 2D operations from Section 5.

These experiences point at process calculi as, at least, one of the most promising notational frameworks for unifying different aspects of biological representation. In addition, the process calculus framework is generally suitable for relating different levels of abstractions, which is going to be essential for feasibly representing biological systems of high architectural complexity.

In summary, the fundamental flavor of cellular machinery is: *chemistry in the service of materials, energy, and information processing.* The processing of energy and materials (e.g., in metabolic pathways) need not be emphasized here, rather we emphasize the processing of information, which is equally vital for survival and evolution [1]. Information processing tasks are distributed through a number of interacting abstract machines with wildly different architectures and principles of operation.

7 Conclusions

Many aspects of biological organization are more akin to discrete hardware and software systems than to continuous systems, both in hierarchical complexity and in algorithmic-like information-driven behavior. These aspects need to be reflected in the modeling approaches and in the notations used to describe such systems, in order to make sense of the rapidly accumulating experimental data.

References

1. C. Adami: What is complexity? *BioEssays*, 24 (2002), 1085-1094.
2. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter: *Molecular Biology of the Cell.* 4th edition, Garland Science, New York, 2002.
3. R. Alur, C. Belta, F. Ivancic, V. Kumar, M. Mintz, G.J. Pappas, H. Rubin, J. Schug: Hybrid modeling of biomolecular networks. In *Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control*, Rome, Italy, March 28-30, 2001, LNCS 2034.
4. M. Antoniotti, B. Mishra, F. Park, A. Policriti, N. Ugel: Foundations of a query and simulation system for the modeling of biochemical and biological processes. In *The Pacific Symposium on Biocomputing* (L. Hunter, T.A. Jung, R.B. Altman, A.K. Dunker, T.E. Klein, eds.), World Scientific, 2003, 116–127.
5. M. Antoniotti, C. Piazza, A. Policriti, M. Simeoni, B. Mishra: Modeling cellular behavior with hybrid automata: bisimulation and collapsing. In *Int. Workshop on Computational Methods in Systems Biology*, LNCS, Springer-Verlag, Berlin, 2003, to appear.
6. M. Antoniotti, A. Policriti, N. Ugel, B. Mishra: Model building and model checking for biochemical processes. In *Cell Biochemistry and Biophysics*, 2003, in press.
7. C. Bodei, P. Degano, F. Nielson, H.R. Nielson: Control flow analysis for the pi-calculus. In *Proc. 9th International Conference on Concurrency Theory*, LNCS 1466, Springer-Verlag, Berlin, 1998, 84–98.

8. K.N.J. Burger: Greasing membrane fusion and fission machineries. *Traffic*, 1 (2000), 605–613.
9. G. Ciobanu, V. Ciubotaru, B. Tanasă: A π-calculus model of the Na pump. *Genome Informatics*, 13 (2002), 469–471.
10. G. Ciobanu: Software verification of biomolecular systems. In *Modelling in Molecular Biology* (G. Ciobanu, G. Rozenberg, eds.), Natural Computing Series, Springer-Verlag, Berlin, 2004, 40–59.
11. M. Curti, P. Degano, C. Priami, C.T. Baldari: Modelling biochemical pathways through enhanced pi-calculus. *Theoretical Computer Science*, 325, 1 (2003), 111–140.
12. L. Cardelli: Brane calculi - Interactions of biological membranes. In *Proc. Computational Methods in Systems Biology*, 2004, Springer-Verlag, Berlin, to appear.
13. L. Cardelli, A.D. Gordon: Mobile ambients. *Theoretical Computer Science*, Special Issue on Coordination (D. Le Métayer, ed.), 240, 1 (2000), 177–213.
14. V. Danos, M. Chiaverini: A core modeling language for the working molecular biologist. 2002.
15. V. Danos, C. Laneve: Formal molecular biology. *Theoretical Computer Science*, to appear.
16. E.H. Davidson, D.R. McClay, L. Hood: Regulatory gene networks and the properties of the developmental process. *PNAS*, 100, 4 (2003), 1475-1480.
17. A. Di Pierro, H. Wiklicky: Probabilistic abstract interpretation and statistical testing. In *Proc. Second Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, LNCS 2399, Springer-Verlag, Berlin, 2002, 211–212.
18. S. Efroni, D. Harel, I.R. Cohen: Reactive animation: realistic modeling of complex dynamic systems. *IEEE Computer*, to appear, 2005.
19. M.B. Elowitz, S. Leibler: A synthetic oscillatory network of transcriptional regulators. *Nature*, 403 (2000), 335–338.
20. F. Fages, S. Soliman, N. Chabrier-Rivier: Modelling and querying interaction networks in the biochemical abstract machine BIOCHAM. *J. Biological Physics and Chemistry*, 4, 2 (2004), 64–73.
21. D.T. Gillespie: Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81 (1997), 2340-2361.
22. D. Harel: Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8 (1987), 231–274.
23. L.H. Hartwell, J.J. Hopfield, S. Leibler, A.W. Murray: From molecular to modular cell biology. *Nature*, 402 (Dec. 1999), C47–52.
24. J. Hillston: *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
25. C-Y.F. Huang, J.E. Ferrell Jr.: Ultrasensitivity in the mitogen-activated protein kinase cascade. *PNAS*, 93 (1996), 10078-10083.
26. H. Kitano: The standard graphical notation for biological networks. *The Sixth Workshop on Software Platforms for Systems Biology*, 2002.
27. H. Kitano: A graphical notation for biochemical networks. *BIOSILICO*, 1 (2003), 169–176.
28. K.W. Kohn: Molecular interaction map of the mammalian cell cycle control and DNA repair systems. *Molecular Biology of the Cell*, 10, 8 (1999), 2703–2734.
29. C. Kuttler, J. Niehren, R. Blossey: Gene regulation in the pi calculus: simulating cooperativity at the Lambda switch. *BioConcur* 2004, ENTCS.

30. M. Kwiatkowska, G. Norman, D. Parker: Probabilistic symbolic model checking with PRISM: a hybrid approach. *J. Software Tools for Technology Transfer*, 6, 2 (2004), 128–142.
31. H. Lodish, A. Berk, S.L. Zipursky, P. Matsudaira, D. Baltimore, J. Darnell: *Molecular Cell Biology*. Fourth Edition, Freeman, 2002.
32. J.S. Mattick: The hidden genetic program of complex organisms. *Scientific American*, October 2004, 31–37.
33. H.H. McAdams, A. Arkin: It's a noisy business! Genetic regulation at the nanomolar scale. *Trends Genet.*, 15, 2 (1999), 65–69.
34. R. Milner: *Communicating and Mobile Systems: The π-calculus*. Cambridge University Press, 1999.
35. R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, U. Alon: Network motifs: simple building blocks of complex networks. *Science*, 298 (2002), 824–827.
36. R. Milner: Bigraphical reactive systems. *CONCUR* 2001, *Proc. 12th International Conference in Concurrency Theory*, LNCS 2154, Springer-Verlag, Berlin, 2001, 16–35.
37. M. Nagasaki, S. Onami, S. Miyano, H. Kitano: Bio-calculus: its concept and molecular interaction. *Genome Informatics*, 10 (1999), 133–143.
38. F. Nielson, R.R. Hansen, H.R. Nielson: Abstract interpretation of mobile ambients. *Science of Computer Programming*, 47, 2-3 (2003), 145–175.
39. F. Nielson, H.R. Nielson, C. Priami, D. Rosa: Static analysis for systems biology. In *Proc. ACM Winter International Symposium on Information and Communication Technologies*, Cancun 2004.
40. F. Nielson, H.R. Nielson, C. Priami, D. Rosa: Control flow analysis for BioAmbients. *Proc. BioCONCUR* 2003, to appear.
41. Gh. Păun: *Membrane Computing. An Introduction.* Springer-Verlag, Berlin, 2002.
42. C. Priami: The stochastic pi-calculus. *The Computer Journal*, 38 (1995), 578–589.
43. C. Priami, A. Regev, E. Shapiro, W. Silverman: Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80 (2001), 25–31001.
44. M. Ptashne: *Genetic Switch: Phage Lambda Revisited.* Cold Spring Harbor Laboratory Press, 3rd edition, 2004.
45. A. Regev: *Computational Systems Biology: A Calculus for Biomolecular Knowledge*. Ph.D. Thesis, Tel Aviv University, 2002.
46. A. Regev, E.M. Panina, W. Silverman, L. Cardelli, E. Shapiro: BioAmbients: an abstraction for biological compartments. *Theoretical Computer Science*, to appear.
47. A. Regev, E. Shapiro: Cells as computation. *Nature*, 419 (2002), 343.
48. Systems biology markup language: http://www.sbml.org
49. D. Thieffry, R. Thomas: Qualitative analysis of gene networks. *Pacific Symposium of Biocomputing* 1998, 77–88.
50. J.M. Vilar, H.Y. Kueh, N. Barkai, S. Leibler: Mechanisms of noise-resistence in genetic oscillators. *PNAS*, 99, 9 (2002), 5988–5992.
51. C.-H. Yuh, H. Bolouri, E.H. Davidson: Genomic cis-regulatory logic: experimental and computational analysis of a sea urchin gene. *Science*, 279 (1998), 1896–1902.