The Modeling and the Simulation of the Fluid Machines of Synthetic Biology

Jean-Louis Giavitto

Ircam, UMR STMS 9912 Ircam – CNRS – UPMC
1 place Igor Stravinsky, 75004 Paris, France

jean-louis.giavitto@ircam.fr

Abstract. In the past century, several conceptual and technological breakthroughs produced the digital computers and open the digital information age. At the very same time, the Watson – Crick model of the digital coding of the genetic information was developed. Despite this parallel development, biology as long focused in the understanding of existing systems shaped by natural evolution whilst computer science has built its own (hardware and software) objects from scratch.

This situation is no longer true: the emergence of *synthetic biology* opens the doors to the systematic design and construction of biological (fluid) machines. However, even if fluid machines can be based on a kind of digital information processing, they differ from the discrete dynamical systems we are used in computer science: they have a *dynamical structure.*

In this paper, we stress the parallel between the development of digital information processing and genetic information processing. We sketch some tools developed or appropriated in computer science that can be used to model and specify such fluid machines. We show through an example the use of MGS, a domain specific language, in the proof of concept of a "multicellular bacterium" designed at the 2007 iGEM competition.

Keywords: fluid machines, synthetic biology, computer modeling and simulation, $(\mathsf{DS})^2$: dynamical systems with a dynamical structure, spatial computing, topological rewriting, domain specific language (DSL), MGS.

1 Introduction

In the preface of [10], Tibor Gánti highlights the divergence of information processing researches as they apply to biological systems (cells) or to artificial systems (computers). As early as 1944, Erwin Schrödinger speculates about "programs" and "genetic code" [34]. Since, information processing has been a constant source of fruitful analogies for genetics [25] and biology has provided many motivations and algorithms to computer science (from evolutionary algorithms to artificial neurons). But, despite the parallel developments of computer science and genetics, cf. Fig. 1, biology as focused in the understanding of existing

Computer Science		*Genetics*
	1928	DNA as the support of the genetic information
The Turing Machine	1936	
The first transistor	1947	
Self-reproducing automata	1951	
	1953	Structure of the DNA: the digital coding of the genetic information
Arpanet	1967	
electronic mail	1972	ARN sequencing
	1975	DNA sequencing
	1983	PCR
Commercial development of the Internet	1990	
	2000 ~ 03	The Human Genome Project

Fig. 1. A parallel history: In the two domains, the storage, the replication, the communication and the modification of (digital or genetic) information is studied.

systems shaped by natural evolution whilst computer science has designed and studied its own hardware and software objects from scratch.

Considering biological entities, like cells or organisms, as "machine" has a long history. For instance, in the 17th century, when René Descartes tried to convince Queen Christina of Sweden that animals were just another form of machine, she is said to have replied: show me a clock that reproduces [1]. Three centuries were to pass before her question received an answer in 1951 with the publication of an article by John Von Neumann [40]. A machine, in the abstract and ideal form of a computation, could effectively build a copy of itself. Therefore, the reproduction argument cannot be used to distinguish in principle biological systems from machines.

The idea that living matter, and specifically cells, can be used as computers is appealing for several reasons: nanoscale devices, massive parallelism, low energy consumption, possible change in computability and complexity classes... and also the hope that computing with biological devices may give to the corresponding software properties usually attributed to living matter: autonomy, adaptability, self-repair, robustness, self-organization.

Tibor Gánti uses the term *fluid machineries* to describe machines based on chemical processes utilized in the living world. In [10] he introduces the *chemoton* (chemical automaton), a minimal cell model composed of three stochiometrically coupled autocatalytic subsystems: a metabolism, a template replication process, and a membrane enclosing the other two. The qualifier "fluid" stresses the fact that, in contrary to electrical machines, real geometrical directions cannot be assigned to the energy exchanged between the components of the fluid automaton. Nevertheless, they can be described as *dynamical systems* with a (chemical) state that evolves in time. This is also the case for *genetic process engineering* [41,42], another example of chemical machines harnessing the cellular machinery. In this engineering discipline, existing genetic elements are modified to implement into cells biochemical logic circuits and programmed intercellular communication.

The objective is to achieve complex, predictable and reliable input/output cell behaviors.

However, we advocate that the term "fluid" also outlines another important property of living systems: they have a *dynamic structure*. Biological processes form highly structured and hierarchically organized dynamical systems, the spatial structure of which varies over time and must be calculated in conjunction with the state of the system (this is specially obvious in developmental biology). We call this type of system a *dynamical system with dynamical structure* [12,13], which we shall abbreviate to $(\mathsf{DS})^2$.

The fact that the very structure of a biological system is dynamical[1] has been highlighted by several authors; we can cite: the concept of *hyper-cycle* introduced by Manfred Eigen and Peter Schuster in the study of autocatalytic networks [6], the theory of *autopoietic systems* formulated by Humberto Maturana and Francisco Varela [39] or the concept of *biological organization* introduced by Walter Fontana and Leo Buss to formalize and study the emergence of self-maintaining functional structures in a set of chemical reactions [9]. The objective of all of these works has been to grasp and formalize the idea of change in the structure of a system, change that is coupled with the evolution of the state of the system.

Coming back to the question of harnessing biological processes to compute, it is interesting to follow the metaphor: if we want to use cells as computing devices, what makes a population of idealized cell intrinsically different from a Turing machine? It may be that, from a technical point of view, there is no difference, meaning that Turing computation and "cell computation" coincide in term of computability. However, the computing devices differ definitively, in the same way that lambda expressions differs from Turing machines.

A Turing machine has a fixed structure. The tape is unbounded and only a finite part of the tape is used during the computation; however, the structure of the tape is fixed *a priori*: a sequence of symbols. This means that the control of the machine is also fixed: the head can move only to the left or to the right. The writing of a symbol on the tape is fully determined by the state of the control automaton of the Turing machine and a symbol in its (left or right) neighborhood. We can say that the control part of the Turing machine (control automaton *and* neighborhood) is predefined. At first sight, the situation does not seem too much different for a population of cells. Obviously the cells in a tissue share the same genetic program and two cells interact and change accordingly their state[2] because they are neighbors. And the dynamic organization of a set of cells can be coded in someway into a linear tape. However this coding is not straightforward at all. The structure of a living system is not predefined: cells growth, divide and their emerging organization as tissue or organisms exhibits a great variety that must be computed with the system evolution and cannot be predicted by the

[1] Biological systems are not the only ones that may exhibit a dynamic structure. For example, in control theory, these questions have also been addressed for instance with *systems of variable structure* [38].

[2] The state of a cell is a complex value that changes due to internal processes. But the state of a cell can also change because of the interactions with the cells in the neighborhood through diffusion and active transport, by endo- and exocytosis, by mechano-transduction, by propagation of an electric field, etc.

simple inspection of the genetic program. Moreover, this neighborhood is dynamic. It means that the control part of a biological system (genetic program and neighborhood) *cannot be fixed* a priori.

The modeling, "the programming", the simulation and the analysis of such $(\mathsf{DS})^2$ raise a difficult problem: how to define an evolution function when its set of arguments (the state variables) is not completely known at the specification time?

In the rest of this paper we sketch a formalism and an experimental programing language called MGS, which address this problem. MGS relies on a notion of "spatial structures rewriting" to specify local interactions between the system components. This approach enables the specification of the components evolution as well as their dynamic organization.

2 A Brief Introduction to MGS

2.1 Lessons from L Systems and P Systems

Computer Science has developed many languages and tools to help model and simulate dynamical systems. *P systems* (membrane computing) [30] and *L systems* [33] are examples of bio-inspired formalisms that have been successfully used in the modeling of biological $(\mathsf{DS})^2$. We will motivate MGS concepts based on relevant features of these two formalisms.

A Common Computational Mechanism. First, one can note that P and L systems share the following three characteristics.

Discrete Space and Time. The structure of the state (the membranes hierarchy in a P system, the parenthesized string in a L system) consists of a discrete *collection* of values. This discrete collection of values evolves by discrete steps. We call "spatial" the organization of the elements in the collection because this structure does not unfold in time.

Temporally Local Transformation. The computation of a new value in the new state depends only on values for a fixed number of preceding steps (and as a matter of fact, only one step).

Spatially Local Transformation. The computation of a new collection is done by a "structural combination" of the results of more elementary computations involving only a "small and static subset" of the initial collection. "Structural combination", means that the elementary results are combined into a new collection, irrespectively of their precise value. "Small and static subset" makes explicit that only a fixed subset of the initial elements are used to compute a new element value (this is measured for instance by the diameter of an evolution rule in a P systems, or the context of a rule in a L system).

A Rewriting Mechanism. Considering these shared characteristics, the abstract computational mechanism is always the same: (1) a subcollection A is selected in a collection C; (2) a new subcollection B is computed from the collection A; (3)

the collection B is substituted for A in C. These three basic steps are reminiscent of the notion of *rewriting* and, indeed, P systems can be described as multiset rewriting and L systems as a kind of string rewriting. However, we prefer to call it a *transformation* because the notion of rewriting is mainly developed on terms (strings are terms in an associative monoid and multisets are terms in an associative and commutative monoid) and we want to be more general.

In addition to transformation specification, there is a need to account for the various constraints in the selection of the subcollection A (the pattern language) and the replacement B, and in the rule application strategy. For example, for L systems and P systems, the basic rule application strategy is the maximal parallel one.

Locality and Interaction. From the perspective of the simulation of $(\mathsf{DS})^2$, several features are appealing in the L systems and in the P systems formalisms.

First, summing up the local evolutions triggered by the rules specifies the global evolution of the state. So there is no need to have a global picture of the state to specify its evolution.

Secondly, elements referred in a rule are referred implicitly through pattern matching. For instance, a pattern a refers to some occurrence of a in the collection: there is no need to use a global reference to a. A global reference (like a coordinate) may be difficult to maintain as the collection evolves (for instance when new symbols are inserted elsewhere).

Third, element in a pattern are related through the relationships induced by the organization of the collection. For example, in a string, the pattern ab denote two symbols a and b that must be consecutive in the string. For multisets, each element is neighbor to all the other elements of the multiset.

2.2 The Topological Organization of the State

After this presentation, the main difference between the two formalisms, for the purpose of simulation, appears to be the organization of the collection: an imbrication of multisets of symbols or a parenthesized string of symbols. Thus, our idea is to generalize the approach of P and L systems by developing a framework where multiple organizations can be considered uniformly. This is the notion of *topological collections* introduced in [19] to describe arbitrary complex spatial structures that appear in biological systems [15] and other dynamical systems with a time varying structure [16,20].

The definition of topological collection is based on a mathematical device developed in algebraic topology: the notion of *chain* [23] that extends the notion of labeled graph.

Incidence Structures. An *abstract combinatorial complex* $K = (\mathsf{C}, \prec, \dim)$ is a set C of abstract elements, called *topological cells*, provided with a partial order $\prec$, called the *boundary relation*, and with a *dimension* function $\dim : \mathsf{C} \to \mathbb{N}$ such that for each c and c' in C, $c \prec c' \Rightarrow \dim(c) < \dim(c')$. The reader must pay attention not to confuse biological and topological cells. We write $c \in K$ when a cell c is a cell of C. A cell of dimension p is called a p-cell. We say that a cell c'

is a *border* of a cell c if $c' \prec c$. The boundary of a cell c is the set of borders of c. The *faces* of a p-cell c are the $(p-1)$-cells c' such that $c' \prec c$ and we write $c > c'$ or $c' < c$; c' is called a *coface* of c. Two cells c and c' are *q-neighbors* either if they share a common border of dimension q or if they are in the boundary of a q-cell (of higher dimension).

A cell of dimension 0 corresponds to a point, a 1-dimensional cell corresponds to a line (an edge), a cell of dimension 2 is a surface (*e.g.* a polygon), etc. For example, a graph is an abstract combinatorial complex (ACC) built only with 0- and 1-cells. Another example is pictured in Fig. 2.

Topological Collections. The next step is to attach a value to the cells of a complex. Algebraic topology takes this value in a commutative group since it gives a natural group structure to the set of chains [23]. We relax this assumption for topological collections: a topological collection C is a function that associates a value from an arbitrary set V with cells in an ACC, see Fig. 2. Thus the notation $C(c)$ refers to the value of C at cell c; $C(c)$ is called the *label* of the cell c. Labels can be used to capture geometric properties or to represent the arbitrary state of a subsystem (a mass, a concentration of chemicals, or a force acting on certain cells).

We write $|C|$ for the set of cells for which C is defined. The collection C can be written as a formal sum $\sum_{c \in |C|} v_c \cdot c$ where $v_c \overset{\text{df}}{=} C(c)$. With this notation, the underlying ACC is left implicit but can usually be recovered from the context. By convention, when we write a collection C as a sum

$$C = v_1 \cdot c_1 + \cdots + v_p \cdot c_p$$

we insist that all c_i are distinct. This notation is directly used in MGS to build new topological collections on arbitrary ACC of any dimension. Notice that this addition is associative and commutative: the order of operations used to build a topological collection is irrelevant.

In the MGS programing language, topological collections correspond to aggregate data types. These data types differ by the specification of their underlying

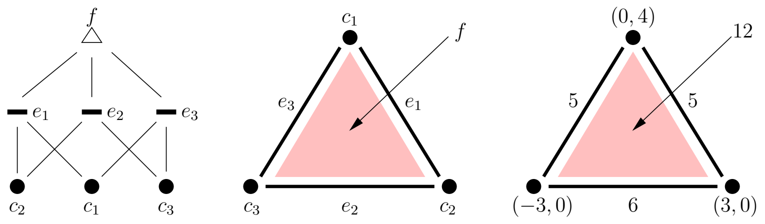

Fig. 2. On the left, the Hasse diagram of the boundary relationship of the ACC given in the middle: it is composed of three 0-cells (c_1, c_2, c_3), of three 1-cells (e_1, e_2, e_3) and of a single 2-cells (f). The three edges are the faces of f, and therefore f is a common coface of e_1, e_2 and e_3. On the right, a topological collection associates data with the cells: positions with vertexes, lengths with edges and area with f.

cellular complex. In the current implementation of the MGS language, usual data structures (records, sets, sequences, trees, arrays, etc.) are represented by special kinds of one-dimensional topological collection, namely vertex-labeled graphs: elements of the data structure are attached to the vertexes and the edges represent the relative accessibility from one element to another in the data structure. MGS also handles more sophisticated spatial structures corresponding to arbitrary ACC of any dimension.

2.3 Topological Rewriting

The next move is to define a suitable notion of topological collection transformation. As mentioned in the introduction, the transformation of a topological collection must be able to express changes in the labels as well as changes in the underlying spatial structure.

A dedicated definition has been developed in [19]. However, thanks to the term structure of a topological collection, transformations can be defined in the framework of set rewriting, following an approach similar to that taken in [31] for hyper-graphs: using the additive representation of topological collections, topological rewriting can be simply defined as an adapted version of conditional first-order associative-commutative term rewriting, see [36] for the details.

A transformation T is a function specified by a set of rewriting rules $\{p_1 \Rightarrow e_1, \ldots, p_n \Rightarrow e_n\}$ where each p_i is a pattern and each e_i is an expression. An application of a rule matches a sub-collection with p_k that is then substituted by the result of expression e_k. In rewriting rules, patterns match sub-expressions, that is, partial sums of the whole sum representing the topological collection that the rule is applied on. It is in this sense that the additive structure of topological collections is preserved (but a transformation is not necessarily an homomorphism).

Patterns. The formal definition of topological rewriting is less interesting than the actual syntax of the pattern language used to specify the left hand side (lhs) of a rewriting rule: as a matter of fact, the lhs of a rule must match a sub-collection, that is a subset of C and a sub-relation of the incidence relation $\prec$ of the complex K. This information can be difficult to specify without the help of a dedicated language. We have studied several pattern languages. We use here a very small fragment of the MGS *path pattern language*. Path patterns can be composed with respect to the neighborhood relationships but we don't use this feature in the example developed in this paper.

Pattern Variables. A pattern variable x matches a cell and its label. Patterns are *linear*: two distinct pattern variables always refer to two distinct cells. The identifier x can be used elsewhere in the rule to refer to the label of the matched cell; the cell itself can be referred through the special identifier $\hat{}x$. This convention avoids the introduction of two identifiers to match a cell and its associated value. Using the additive notation for topological collections, and without the previous convention, this pattern is translated to $x \cdot \hat{}x$ where the variable x ranges over the labels, and where the variable $\hat{}x$ ranges over the cells.

Conditional rules. A *guard* can be used to specify a condition that must be satisfied by the matching. For instance, expression $x/x > 5$ matches a cell $\hat{}x$ labeled by an integer x greater than 5.

Strategies. Rule applications are controlled through a *rule application strategy.* Several strategies are available in MGS like the maximal parallel application and the Gillespie stochastic simulation algorithm used in the simulation of chemical reactions [35]. These strategies control the advancement of time in the simulation (synchronous, asynchronous, stochastic, etc.). They are often non-deterministic, *i.e.*, applied on a collection C, only one of the possible outcomes (randomly chosen) is returned by the transformation.

Multisets as "Free" Collections. Let an ACC $K = \{\bot, c_1, c_2, \ldots\}$ where the c_i are incomparable and $\bot < c_i$. The corresponding topological collection is a multiset and the associated notion of transformation corresponds to classical multiset rewriting. In this sense, any topological collection can be obtained from a multiset by putting additional constraints on $\prec$.

Note that the previous definitions of topological collections and transformations are useful for developing a unified simulation framework but have less interest if one is concerned by the derivation of properties specific to an underlying topology.

3 The modeling of a "multicellular bacteria"

In this section, we illustrate the use of the MGS framework for the modeling and the simulation of a fluid machine.

3.1 The international Genetically Engineered Machine (iGEM) Competition

The emergence of *synthetic biology* [8,22,4,24] opens the doors to the systematic design and construction of biological (fluid) machines. After the construction of the first artificial genetic regulatory networks in E.coli around 2000 [11,7], this domain mainly develops around the engineering of synthetic gene network [21,27,5]. It has been largely popularized through the iGEM competition, a yearly competition launched by the MIT in 2003 [3]. The competition is aimed at undergraduate students that are given the opportunity to design, model and assemble BioBricks [26] to produce *new biological functions* integrated into living systems. More than 160 teams coming from all around the world participate in the 2011 issue.

3.2 Objectives of the Synthetic Multicellular Bacterium Project

The 2007 French team supervised by Ariel Lindner and Samuel Bottani participated in the competition and was ranked first in the "foundational research" category for their *Synthetic Multicellular Bacterium* project. Unlike most projects involving a regulatory network functioning in a single cell following a straightforward sensing/transduction/actuation loop, the functioning of the Synthetic Multicellular Bacterium is implemented at the population level. MGS was used to produce most of the simulations needed to validate the design (one simulation was done in MATLAB). To save room, we present here only the design idea of the Synthetic Multicellular Bacterium and one of the supporting MGS models. The interested reader may found additional information in [37] where several simulations that are inspired or that extend the initial Synthetic Multicellular Bacterium simulations are presented.

The project is aimed at the design of a synthetic multicellular bacterium. This organism would allow the expression of a lethal or dangerous transgenic gene in the *Escherichia coli* bacterium without disturbing the development of its biomass. The main difficulty was to install a mechanism of irreversible bacterial differentiation that makes possible to express the transgene only in a part of the population unable to reproduce. The two lines, *germinal* (not differentiated) and *somatic* (differentiated and unable to reproduce), are interdependent and then constitute a multicellular organization (hence the name "*multicellular* bacterium"). To ensure that the ratio between the two populations makes it possible for the system to grow, the sterile somatic cells are designed to provide to the germinal cells a molecule essential to their reproduction: DAP (diaminopimelate). Fig. 3 sketches the general principle of the project. This design asked for the development of two distinct biological functionalities, one for the cellular differentiation and the other for the feeding of DAP to the germinal cells.

This design is an example of a fluid machine: it is the dynamic organization of the whole population (into two differentiated subpopulations) that is viable and functional. A single cell cannot survive: it is either sterile or unable to reproduce alone. The Paris team provided experimental evidences and theoretical proofs that the Synthetic Multicellular Bacterium organism is viable.

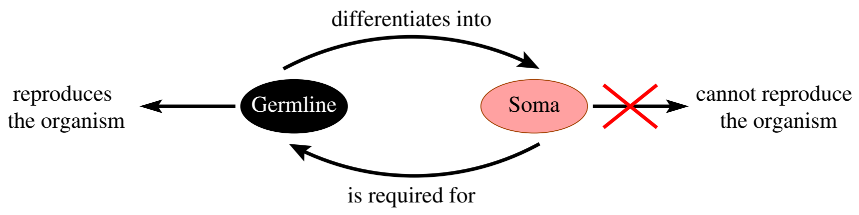

Fig. 3. The Synthetic Multicellular Bacterium is composed of two cell types: germ cells (G) and somatic (S) cells. G cells are able to live by producing two different types of cells: G cells and S cells. S cells are derived from G cells by an irreversible differentiation step, exhibiting a new function required for the survival of the G cells. S cells cannot reproduce. This dependency between G and S cells defines the organism. Additional informations is available at [2,37].

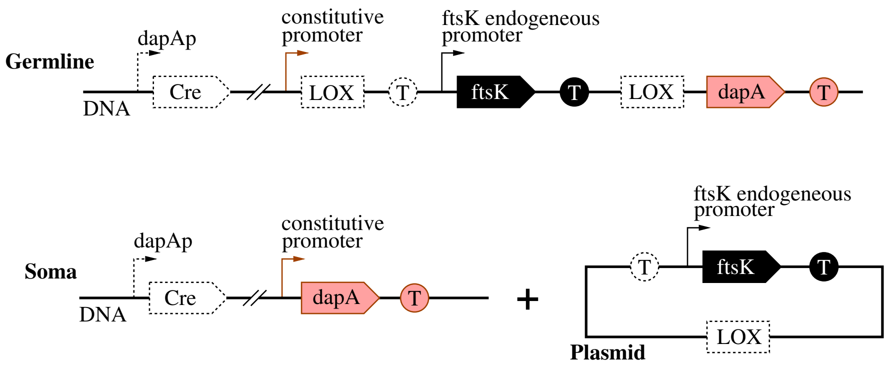

Fig. 4. Gene regulatory networks of the germinal and somatic cells describing the feeding device (light gray) and differentiation device (dotted box). Cre, dapA and ftsK are genes, LOX is a recombination site and T are terminators. Figure reprinted from [37] and adapted from [2].

3.3 The Paris Team Proposal

The Paris team has proposed an original construction to implement this functionality into the E.coli bacterium. The gene regulatory network of the proposal is described in Fig. 4. Two functions are described: a *feeding device* based on the production of DAP molecules (light gray) and a *differentiation device* based on a classical Cre/LOX recombination scheme (dotted box).

In the germline G, the natural promoter controls the expression of fstK, a gene essential for replication. On the contrary, the dapA gene is not active since it lacks a promoter to initiate its transcription and G is auxotrophic in DAP (the corresponding protein product of the gene dapA). DAP diffuses in the environment and is rapidly degraded.

A dapAp promoter is sensitive to DAP concentration and it is located before the gene Cre. The production of Cre, in presence of DAP in the environment, initiate the recombination/differentiation process.

After recombination, the genomic reassembly leads, by the excision of the parts between the two LOX genes, to the soma cell type S (and a plasmid that is rapidly degraded). In the *feeding device* S, DAP is under the control of its constitutive promoter and can be expressed. The synthesized DAP diffuses in the environment allowing to reach G cells. Lacking ftsK genes, S cells are sterile and eventually die.

3.4 One MGS model of the Synthetic Multicellular Bacterium

Several models of the Synthetic Multicellular Bacterium have been developed to study, through simulation, various questions. We present here a model that integrates a simple mechanical and a biological behavior.

In this example, we abstract individual biological cells by disks localized in a 2D Euclidean space (the third dimension is not considered as the Synthetic Multicellular Bacterium is supposed to grow in the plane of a petri dish). Cells push away each other and consequently change their positions in space and their immediate neighborhood. Thus, this neighborhood must be dynamically computed according to the position of the disks in the plane. The state of a cell is described by a record:

```
record bact = { x, y, vx, vy, fx, fy, size, radius, dap, soma }
```

which includes the position, velocity and force exercised on the cell, its radius, the local DAP concentration, and the differentiation state (germinal or somatic).

Our approach is based on the specification of cell-cell dynamical interactions and the computation of the neighborhood of the cells using an implicit Delaunay triangulation. This approach has already been used in systems biology for the modeling of cell population [32]. The MGS declaration

```
delaunay D(c:bact) = [c.x, c.y]
```

defines Delaunay collection called `D`: a graph where nodes are cells and edges are computed implicitly by the run-time using the vector of coordinates extracted from the cells using the function `D`.

Description of the Model. The modeling of Synthetic Multicellular Bacterium is organized into two coupled models: a mechanical model and a biological model.

The Mechanical Model. The mechanical model consists of a mass/spring system. Bacteria are considered as punctual masses localized at the center of their associated circle; the presence of a spring between two masses depends on the neighborhood computed by the Delaunay triangulation. The elongation of the springs rest lengths captures the mechanical effect of the growth of the bacteria.

Each cell computes its acceleration by summing all mechanical forces induced by its incident springs, and consequently moves in space. This is done by the transformation `Meca`. `Meca` sums the forces applied on each cell using a `neighborsfold` expression. A naive Euler schema is used twice to integrate during the time step, acceleration into velocity and velocity into new positions.

```
fun interaction(ref, src) =
    let x = ref.x - src.x and y = ref.y - src.y in
    let dist = sqrt(x*x+y*y) in
    let spring = 0.0-K*(dist-(ref.radius+src.radius))/dist
    in fx=x*spring - ref.vx*MU, fy = y*spring - ref.vy*MU

fun addVect(u, v) =  fx = u.fx + v.fx, fy = u.fy + v.fy
fun sum(x, u, acc) = addVect(acc, interaction(x,u))
```

```
trans Meca[Dt] = {
    e => let f = neighborsfold(sum(e), fx=0,fy=0, ^e)
         in e + { x = e.x + Dt*e.vx,  y = e.y + Dt*e.vy,
                  vx = e.vx + Dt*f.fx,  vy = e.vy + Dt*f.fy,
                  fx = f.fx,  fy = f.fy } * ^e
}
```

The overall form of the unique rule of the Meca transformation is `e =>` v `* ^e` which means that this rule only update the value associated to the cell matched by `e`.

The computation of the value v requires some explanation. Variables in capital represent parameters of the model (constant defined elsewhere). The definitions `interaction`, `addVect` and `sum` specify auxiliary functions. All functions are curried in MGS: so, `sum(e)` is a function awaiting the remaining arguments `u` and `acc`. The `+` operator between records denotes the asymmetric merge. The expression $\texttt{r}_1$ `+` $\texttt{r}_2$ computes a new record `r` having the fields of both $\texttt{r}_1$ and $\texttt{r}_2$: `r.a` has the value of $\texttt{r}_2$`.a` if the field `a` is present in $\texttt{r}_2$, otherwise it has the value of $\texttt{r}_1$`.a`. The expression `neighborsfold`$(f, init, c)$ iterates a binary reduction function f over the labels of the neighbors of c to build up a return value. The argument *init* is used to initialize the accumulator. Note that the `neighborsfold` operator rely on the ACC structure underlying the topological collection to determine the neighbors of c.

The Biological Model. The DAP diffusion `dap_diff` is modeled by a classical continuous model (numerical integration of the diffusion equation is done using a simple Euler explicit schema). New rules are added to deal with cellular growth, division and death: in presence of DAP, G cells grow by increasing their radius. When the G cell radius reaches a threshold, the cell divides. S cells keep on growing then die when another threshold is reached. The corresponding transformation is called `Cell` and computes the evolution during a time step `Dt`:

```
fun divide(b) =
    (b + { size = (b.size / 2.0), ...}) * newcell(D),
    (b + { ..., x = noise(b.x), y = noise(b.y) }) * newcell(D)

trans Cell[Dt] = {
      x / x.soma & x.size > 4
      => if random(1.0) < 0.01 then <undef> else x * ^x

      x / x.soma
      => (x + { size = x.size + Dt*GRate }) * ^x

      x / x.size < 2
      => let dap_diff =
             neighborsfold(\y,acc.acc + Dt*DIFF*(y.dap-x.dap), 0, x) in
         let dap = dap_diff.dap / neighborcount(x) - Dt*CONS
         in if (dap <= Mdap) then if (random(1.0) < DProb)
                                  then (x + { soma = true }) * ^x
                                  else (x + { dap = dap }) * ^x
            else (x + { dap = x.dap + dap, size = ...}) * ^x
```

```
        x / x.size >= 2
        => let dap_diff = ...
           in if (dap >= Mdap) then divide(x) else ...
}
```

The first rule gives the fate of a somatic cell of size greater than 4: it goes to apoptosis (cell death) with a probability of 1% per `Dt` period. The second rule gives the fate of a somatic cell with a size less or equal to 4 (rules are tried in the order of their specification). Such cells increase in size. Third rule apply to germinal cell of size less than 2. Such cells acquire DAP from the environment. If the amount of DAP in the environment is below a given threshold `Mdap`, the cell can spontaneously turn to a somatic cell, with some probability.

The fourth rule is interesting: it specifies the division of a bacterium if some conditions are meet. The function `divide` returns two new labeled cells (new cells for collection of type `D` are build by the expression `newcell(D)`). The first has the same coordinates as the argument. The second cell has the same coordinates perturbed by some small noise. The mechanical evolution will separate quickly the two cells and will reorganize the whole structure. This reorganization will impact the diffusion of DAP. So there is a feedback loop between the spatial organization of the system and the process inhabiting this organization.

Integration of the Two Models. As classical functions, transformations can be arbitrarily composed. This is the key to the coupling of the two models. The iteration of a function can be specified by the MGS option `iter`. It allows to deal with different time scales: assuming that the mechanical process is faster than the cellular process the whole model is captured by the following evolution function:

```
fun SMB(state) = Cell[Dt=Δ₂t](Meca[Dt=Δ₁t, iter=Δ₂t/Δ₁t](state))
```

where the named argument `Dt` corresponds to the time step used in transformations `Meca` and `Cell`. Here transformation `Meca` is applied $\frac{\Delta_2 t}{\Delta_1 t}$ times for only one application of `Cell`.

4 Conclusions

The example above remains simple: there is no need for sophisticated pattern-matching, yet it is a $(\mathsf{DS})^2$ and it exhibits clearly a feedback loop between the spatial structure and the processes inhabiting the structure. The conclusions that are drawn below are also supported by other applications.

The MGS description is concise thanks to the notion of collection and the notion of transformation, which unify (for the purpose of the simulation) the handling of a wide family of dedicated data structures [14]. The MGS specification follows the natural structure of the model: there is generally one evolution rule for each type of evolution. Evolution rules can express simultaneously structural as well as state changes. They can be grouped into transformation associated to some kind of physical laws (mechanics, chemistry, etc.) making manifest multiphysics simulation (*i.e.*, involving multiple physical domains). The coupling between transformations is easily controlled because the functional nature of a transformation. This enables multiscale models. Although transformations are associated to discrete step evolution, the example shows that numerical integration of continuous processes can be integrated smoothly with discrete evolutions (such as cell division in the rule 4 of the transformation `Cell`). The alternative models developed for the Synthetic Multicellular Bacterium (each model focuses on a specific time scale using a dedicated theoretical framework) outline the versatility of the approach. For instance, stochastic model can be expressed simply through the choice of a dedicated rule application strategy [35].

Past work on MGS have focused on the development of a framework relevant for simulation and the validation of MGS concepts through numerous examples, including application in system and synthetic biology; see the MGS home page: `http://mgs.spatial-computing.org`. The theoretical investigation of the topological framework (*e.g.*, can we develop a natural complexity notion on patterns and on topological operations? Can we develop a dedicated static analysis framework for MGS programs? Is there a relevant notion of model-checking? Etc.) is underway.

Acknowledgements

First thanks must go to Antoine Spicher and Olivier Michel for our long standing collaboration within the MGS project and to the 2007 iGEM Paris team. Thanks also go to Hanna Klaudel and Franck Delaplace at IBISC and the members of the SynBioTIC project for their advices and fruitful discussions. This presentation is based on work partially presented in [18,19,12,28,17,37,29]. Parts of this work have been funded by the CNRS and the University of Evry, the IBISC and IRCAM labs, and by the AutoChem and SynBioTIC ANR projects.

References

1. Amos, M.: Cellular computing. Series in systems biology, Oxford University Press (2004)
2. Bikard, D., Caffin, F., Chiaruttini, N., Clozel, T., Guegan, D., Landrain, T., Puyraimond, D., Rizk, A., Shotar, E., Vieira, G.: The SMB: Synthetic Multicellular Bacterium (iGEM'07 Paris team web site). `http://parts.mit.edu/igem07/index.php/Paris` visited in Jully 2011 (2007)
3. Brown, J.: The iGEM competition: building with biology. Synthetic Biology, IET 1(1.2), 3–6 (2007)
4. Canton, B., Labno, A., Endy, D.: Refinement and standardization of synthetic biological parts and devices. Nature biotechnology 26(7), 787–793 (2008)
5. Chin, J.: Programming and engineering biological networks. Current opinion in structural biology 16(4), 551–556 (2006)
6. Eigen, M., Schuster, P.: The Hypercycle: A Principle of Natural Self-Organization. Springer (1979)

7. Elowitz, M., Leibler, S.: A synthetic oscillatory network of transcriptional regulators. J. Biol. Chem 274, 6074–6079 (1999)
8. Endy, D.: Foundations for engineering biology. Nature 438(7067), 449–453 (2005)
9. Fontana, W., Buss, L.W.: "the arrival of the fittest": Toward a theory of biological organization. Bulletin of Mathematical Biology (1994)
10. Gánti, T.: Chemoton theory. Vol. 1: Theoretical Foundations of Fluid Machineries. Vol. 2: Theory of Living Systems. Kluwer Academic/Plenum (2003)
11. Gardner, T., Cantor, C., Collins, J.: Construction of a genetic toggle switch inescherichia coli. Nature 403, 339–342 (2000)
12. Giavitto, J.L.: Topological collections, transformations and their application to the modeling and the simulation of dynamical systems. In: 14th International Conference on Rewriting Technics and Applications (RTA'03). LNCS, vol. 2706, pp. 208–233. Springer, Valencia (Jun 2003)
13. Giavitto, J.L., Godin, C., Michel, O., Prusinkiewicz, P.: Modeling and Simulation of biological processes in the context of genomics, chap. "Computational Models for Integrative and Developmental Biology". Hermes (Jul 2002), also republished as an high-level course in the proceedings of the Dieppe spring school on "Modeling and simulation of biological processes in the context of genomics", 12-17 may 2003, Dieppes, France.
14. Giavitto, J.L., Michel, O.: Data structure as topological spaces. In: Proceedings of the 3nd International Conference on Unconventional Models of Computation UMC02. vol. 2509, pp. 137–150. Himeji, Japan (Oct 2002), lecture Notes in Computer Science
15. Giavitto, J.L., Michel, O.: Modeling the topological organization of cellular processes. BioSystems 70(2), 149–163 (2003)
16. Giavitto, J.L., Michel, O., Delaplace, F.: Declarative simulation of dynamicals systems : the 81/2 programming language and its application to the simulation of genetic networks. BioSystems 68(2–3), 155–170 (feb/march 2003)
17. Giavitto, J.L.: Simulation de systèmes à structure dynamique : modélisation en morphogenèse et application à la conception de machines fluide. In: Colloque National des systèmes complexes : Vers une science et ingénierie des systèmes complexes (SISC'09) (27–29 may 2010), invited Speaker. Video published at address `http://iscpif.fr/SISC09`
18. Giavitto, J.L., Michel, O.: Mgs: a rule-based programming language for complex objects and collections. In: van den Brand, M., Verma, R. (eds.) Electronic Notes in Theoretical Computer Science. vol. 59. Elsevier Science Publishers (2001)
19. Giavitto, J.L., Michel, O.: The topological structures of membrane computing. Fundamenta Informaticae 49, 107–129 (2002)
20. Giavitto, J.L., Spicher, A.: Topological rewriting and the geometrization of programming. Physica D 237(9), 1302–1314 (jully 2008)
21. Guet, C., Elowitz, M., Hsing, W., Leibler, S.: Combinatorial synthesis of genetic networks. Science 296(5572), 1466 (2002)
22. Heinemann, M., Panke, S.: Synthetic biology – putting engineering into biology. Bioinformatics 22(22), 2790 (2006)
23. Henle, M.: A combinatorial introduction to topology. Dover publications, New-York (1994)
24. TESSY EU-NEST PathFinder initiative: Towards a european strategy for synthetic biology. See especially Deliverable 2.6: "Final Roadmap towards Synthetic Biology in Europe" (2008). `http://www.tessy-europe.eu/` visited in jully 2011
25. Keller, E.F.: Refiguring Life: Metaphors of Twentieth-century Biology. Columbia University Press (1995)

26. Knight, T.: Idempotent vector design for standard assembly of biobricks. Tech. rep., DTIC Document (2003), `http://handle.dtic.mil/100.2/ADA457791`
27. Kobayashi, H., Kærn, M., Araki, M., Chung, K., Gardner, T., Cantor, C., Collins, J.: Programmable cells: interfacing natural and engineered gene networks. Proceedings of the National Academy of Sciences of the United States of America 101(22), 8414 (2004)
28. Michel, O., Spicher, A., Giavitto, J.L.: Rule-based programming for integrative biological modeling – application to the modeling of the λ phage genetic switch. Natural Computing 8(4), 865–889 (Dec 2009),
29. Norris, V., Zemirline, A., Amar, P., Audinot, J., Ballet, P., Ben-Jacob, E., Bernot, G., Beslon, G., Cabin, A., Fanchon, E., Giavitto, J.L., Glade, N., Greussay, P., Grondin, Y., Foster, J., Hutzler, G., Jost, J., Kepes, F., Michel, O., Molina, F., Signorini, J., Stano, P., Thierry, A.: Computing with bacterial constituents, cells and populations: from bioputing to bactoputing. Theory in Biosciences pp. 1–18 (2011),
30. Păun, G.: From cells to computers: computing with membranes (P systems). Biosystems 59(3), 139–158 (March 2001)
31. Raoult, J.C., Voisin, F.: Set-theoretic graph rewriting. In: Proceedings of the International Workshop on Graph Transformations in Computer Science. pp. 312–325. Springer-Verlag, London, UK (1994),
32. Barbier de Reuille, P., Bohn-Courseau, I., Ljung, K., Morin, H., Carraro, N., Godin, C., Traas, J.: Computer simulations reveal properties of the cell-cell signaling network at the shoot apex in Arabidopsis. PNAS 103(5), 1627–1632 (2006), `http://www.pnas.org/cgi/content/abstract/103/5/1627`
33. Ronzenberg, G., Salomaa, A. (eds.): L systems: from formalism to programming languages. Springer Verlag (Feb 1992)
34. Schrödinger, E.: What is Life? Cambridge University Press, Cambridge (1944)
35. Spicher, A., Michel, O., Cieslak, M., Giavitto, J.L., Prusinkiewicz, P.: Stochastic p systems and the simulation of biochemical processes with dynamic compartments. BioSystems 91(3), 458–472 (March 2008),
36. Spicher, A., Michel, O., Giavitto, J.L.: Declarative mesh subdivision using topological rewriting in mgs. In: Int. Conf. on Graph Transformations (ICGT) 2010. LNCS, vol. 6372, pp. 298–313 (Sep 2010)
37. Spicher, A., Michel, O., Giavitto, J.L.: Understanding the Dynamics of Biological Systems: Lessons Learned from Integrative Systems Biology, chap. Interaction-Based Simulations for Integrative Spatial Systems Biology. Springer Verlag (Feb 2011)
38. Utkin, V.: Variable structure systems with sliding modes. Automatic Control, IEEE Transactions on 22(2), 212–222 (1977)
39. Varela, F.J.: Principle of Biological Autonomy. McGraw-Hill/Appleton & Lange (1979)
40. Von Neumann, J.: Theory of Self-Reproducing Automata. Univ. of Illinois Press (1966)
41. Weiss, R., Basu, S., Hooshangi, S., Kalmbach, A., Karig, D., Mehreja, R., Netravali, I.: Genetic circuit building blocks for cellular computation, communications, and signal processing. Natural Computing 2(1), 47–84 (2003)
42. Weiss, R., Knight, T., Sussman, G.: Cellular computing, chap. Genetic process engineering, pp. 43–73. Series in Systems Biology, Orford university Press (2004)