DOMAIN SPECIFIC LANGUAGES FOR CELLULAR INTERACTIONS

William L. Harrison[1], *Robert W. Harrison*[2]

[1]Department of Computer Science, University of Missouri-Columbia, Columbia, Missouri, USA
[2]Departments of Computer Science & Biology, Georgia State University, Atlanta, Georgia, USA

ABSTRACT

Bioinformatics is the application of Computer Science techniques to problems in Biology, and this paper explores one such application with great potential: the modeling of life cycles of autonomous, intercommunicating cellular systems using domain-specific programming languages (DSLs). We illustrate this approach for the simple photo-synthetic bacterium *R. Sphaeroides* with a DSL called CellSys embedded in the programming language Haskell.

1. INTRODUCTION

Biological systems are often understood better at the component level than as complete systems. What is frequently missing from the picture is a precise characterization of the concurrent interaction of these components. The thesis of this paper is that the integration of these component-level details into a high-level description of a biological system is an ideal problem for programming language design.

Domain-specific languages are small programming languages specially tailored to particular problem domains. Because DSLs include domain abstractions (and little else), DSL programs are easily written and comprehended by domain experts. Because they are small, DSLs may be given rigorous mathematical specifications allowing reasoning about DSL programs themselves. In the setting of developmental biology, the approach advocated here views each organism or biological phenomena as a particular domain.

The promise of DSLs for system biology is that they allow the biologist to: (1) write formal models describing complex interactions of tissues and organisms with a high-level of abstraction and accuracy, (2) execute these formal models to obtain visualizations of an organism's development, (3) suggest refinements to the language implementers reflecting improvements in the understanding of an organism's structure and behavior, and (4) compare models of cellular systems between different organisms or developmental stages of an organism.

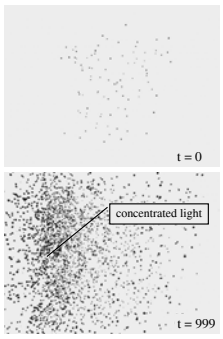
Computer Science has a number of formalisms for concurrency, and a growing body of work in bioinformatics [1, 2, 3, 4, 5] attempts to apply them to capture system-level biological behavior accurately. The current work fits into this school. We apply techniques from the denotational semantics of concurrency [6] to define a language that can represent the complexity of a biological system, while possessing the formal rigor to ensure that the results are valid. This approach is very general—as long as the cell system can be represented as a set of concurrent Markov chains (a form of non-deterministic finite state machines), then the language will be able to simulate its behavior. Because our DSL definition may be directly transcribed into the Haskell language [7], the faithfulness of our formal models of cell systems may be tested automatically. Furthermore, we are able to generate animations[1] simulating life cycles as displayed in Fig. 1.

Fig. 1 illustrates one of the principal advantages of the DSL approach to system biology. To simulate an organism, the biologist writes the program in Fig. 1 (middle) and little else. The meaning of that code is based on the commonly used modeling technique of Markov chains [8]. Details of the language design and implementation (e.g., its concurrency model, etc.) are left entirely up to the computer scientist. This division of labor is extremely valuable in a truly interdisciplinary endeavor such as bioinformatics, because the expert knowledge required (in this case, cell biology and language design) rarely, if ever, occur in the same scientist.

As part of the process of developing CellSys, an object-oriented simulation of *R. Sphaeroides* was written in C++. The program required approximately 1000 lines of code. To understand that simulation program, it is necessary to both be fluent in C++ and to understand how critical details of the system model (e.g., its concurrency model, etc.) may be encoded within C++. While the C++ implementation results in an efficient program, it does not directly reflect the underlying biological model or structure; the biological information contained in the program is spread out over many language constructions. Reusing or modifying such implementations is difficult; anecdotal evidence suggests that new simulations are frequently rewritten from scratch.

This work was partially supported by NIH1 R0l GM62920-04A1, NIH1 P20 GM065762-01A1, the Georgia Research Alliance and the Georgia Cancer Coalition.

[1]The full animation from Fig. 1, created with the Persistence of Vision (`www.povray.org`) package, may be downloaded from (`www.cs.gsu.edu/~cscrwh/cellmovies.html`).



fuzzycase

(*if* $(1.0 < light)$ *then* 1.0 *else* 0.1)	$\rightarrow$	*Tumble*
(1.0)	$\rightarrow$	*Laze*
(0.1)	$\rightarrow$	*AdjSpeed*
(1.0)	$\rightarrow$	*Grow*
(*if* $(size/0.14 < 1.0)$ *then* 0.0 *else* 0.5)	$\rightarrow$	*Divide*
$(0.01/(size + 0.00001))$	$\rightarrow$	*Die*

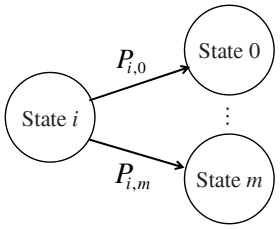


Fig. 1. Simulated evolution (left) for a system of *Rhodobacter Sphaeroides* : An initial population of 100 individuals were placed at random in the middle of a box where the left third was illuminated, and the population was allowed to evolve over time. CellSys programs (middle) consist of code for individual cells; the animation was made by the parallel execution of one hundred such cell programs. The actions *Laze*, *AdjSpeed*, etc., correspond to the typical actions of a *Rhodobacter Sphaeroides* bacterium. The *fuzzycase* operator non-deterministically chooses an action based on the normalized values of the likelyhood functions (i.e., those to the left of the "$\rightarrow$"). Markov chains (right) are state machines with probabilistic transitions and the principal domain abstraction in CellSys (i.e., *fuzzycase* represents a Markov chain's probabilistic transition function).

2. RELATED WORK

Domain-specific languages have been shown to be a valuable tool for specifying rapid prototyping languages, where the domains of interest tend to be drawn from Computer Science. Example domains are compilers [9], web-scripting tools [10], and parser generators [11]. That DSLs have tended to be designed for Computer Science applications is not surprising; the language designers and the domain experts are the same people (namely, computer scientists). DSLs for system biology can serve another purpose—they are the *lingua franca* of the interdisciplinary research in bioinformatics. DSLs for system biology promote a division of labor within a bioinformatics team. Biologists, freed from technical details of constructing simulations, may now concentrate on the faithfulness of their simulations to biological phenomena. Computer scientists, well-versed in language design, may refine or extend the language as needed without the necessity of in-depth understanding of the biological phenomenona.

The use of programming language semantics and automata theory in bioinformatics is not new. A number of researchers have applied techniques from process algebras to this domain ; a far from comprehensive list is [1, 4, 5]. Harel, et al., [2] have developed a state charts tool that categorizes a biological network in terms of a set of state objects. The objects communicate via a user-specified connectivity. Bernaschi, et al., explore the use of cellular automata in the simulation of biological systems [3]. One disadvantage of these approaches is that biologists tend not to be educated in tools that come largely from theoretical computer science (e.g., process algebras such as π-calculus) and typically do not have a background in operational semantics. The DSL approach accommodates biologists with language abstractions they will recognize (e.g., Markov chains in CellSys). Also, programs in embedded DSLs (i.e., DSLs defined within an existing language like Haskell) are directly executable. Executable models and visualization are extremely valuable for checking the faithfulness of biological models, and the approach advocated here offers both.

3. DOMAIN ABSTRACTIONS FOR CellSys

In order to make CellSys useful, it is necessary to implement support for general models of biological systems, and, for this purpose, we propose to use a set of *Markov chains*. Choosing Markov chains as a fundamental abstraction for CellSys means that our approach may be applied to a wide variety of biological systems, most of which can always be reduced to a concurrent set of non-homogeneous Markov chains.

Using systems of Markov chains is a general approach to modeling complicated biological systems [8]. Without loss of generality, we assume that the system is described by a set of arbitrary differential equations (potentially of high order). Under certain conditions, each of these equations can be converted into a system of simpler equations that can be modeled with a Markov chain. It is always possible to convert an autonomous (i.e. time independent) differential equation of high order into a system of first order differential equations with substitutions of the form $z = \frac{dy}{dt}$. Many, but not all, non-autonomous equations can also be converted by this approach.

A Markov chain [12] is a representation of a set of states and the probabilities of moving from one state to another at a given instant (as illustrated in Fig. 1 (right)). As such, Markov chains are a representation of the behavior of a non-deterministic finite state machine. For example, flip-

ping a fair coin would have a Markov chain of order two (heads/tails) and could be written as a 2×2 matrix with 0.5 in each element. When the elements of the matrix are constant, the chain is referred to as a homogeneous Markov chain. The general problem of describing a system of cells requires that the coefficients of the chain change with time and cell state so that non-homogeneous chains will be used.

Defining the differential equation for the probability of being in state n by the transition probabilities from being in other states results in a *Chapman-Kolmogorov equation* of the form shown in Equation 1.

$$\begin{pmatrix} \frac{dP_0}{dt} \\ \vdots \\ \frac{dP_n}{dt} \\ \vdots \\ \frac{dP_m}{dt} \end{pmatrix} = \begin{pmatrix} P_{0,0} & \cdots & P_{n,0} \\ & \vdots & \\ P_{n,0} & \cdots & P_{n,m} \\ & \vdots & \\ P_{m,0} & \cdots & P_{m,m} \end{pmatrix} \cdot \begin{pmatrix} P_0 \\ \vdots \\ P_m \end{pmatrix} \quad (1)$$

In Equation 1, $P_{n,m}$ is the (non-negative real) transition probability of moving from state n to state m. $P_{n,m}$ need not be a constant, and can be a function of time, internal state, or environmental factors. The matrix of all of these probabilities[2] is the transition matrix of a Markov chain. This transition matrix gives rise to a non-deterministic state transition function, pictured in Figure 1 (right). For real $r \in [0, 1]$, the j^{th} transition is taken if $\sum_{l=0}^{j} P_{i,j} < r \leq \sum_{l=0}^{j+1} P_{i,j}$.

3.1. Modeling Bacterial Life Cycles

Bacteria actively seek out sources of food, energy, oxygen and general chemoattractants by swimming towards them. The ability to move and search for ideal conditions is nearly universal in bacteria [13, 14]. This phenomenon has been extensively studied as a model of sensory behavior. Extensive mutational data has been used to characterize the components of the flagella motor and sensory systems [15] and it is possible to individually track bacteria and observe detailed responses to environmental clues, including signals from other individuals [16].

One of the major challenges in modeling bacterial chemotaxis is understanding how a very small organism can sense a gradient of a chemical or light. For example, photosynthetic bacteria swim into bright areas, but are roughly the same size as the wavelength of light and so cannot sense local differences in brightness directly. The bacteria apply a random walk algorithm to find optimal conditions. At the bacterial scale, water is highly viscous and the bacterial flagellum acts like a corkscrew to push the bacterium through the water. Rotating the flagella counterclockwise propels the bacterium forward and, when the direction of rotation is clockwise, the bacterium *tumbles* or randomly reorients. The motion of the bacterium is governed by the frequency at which the bacterium switches from forward motion to tumbling. Bacteria possess a limited memory, and the rate of tumbling is higher when conditions get worse over time.

[2]For fixed i, $\sum_{j=0}^{m} P_{i,j} = 1$.

3.2. The Rhodobacter Sphaeroides Bacterium.

The photosynthetic bacterium *Rhodobacter Sphaeroides* was chosen as a model system. *R. Sphaeroides* will actively swim towards regions of greater light using their flagellar motors to alternately swim and tumble. *R. Sphaeroides* is modeled here as one Markov chain representing the actions individuals may choose such as running, tumbling, dividing, or staying in the same state. These choices are functions of the individual's environment and history. The probability of dividing, for example, is a function of the size of the bacterium; individuals that were in optimal conditions grew faster and thus had more progeny. The exact choice of likelihood functions (i.e., the branch probability functions in Fig. 1 (middle)) is not meant to conform precisely to *R. Sphaeroides*; we expect to only see qualitative agreement between this simulation and experiment.

4. THE CellSys LANGUAGE

Certain aspects of the model are controlled by individual organisms, some are completely beyond their control, and some involve interaction of external aspects with the organism's internal state. Some of these internal aspects are the individual cell's position, size, and velocity, while external aspects include the concentration of light and the loci of other cells in the system. Organisms (i.e., both real organisms and CellSys programs) interact with their environment and one another in much the same way as threads in an operating system: both cells and threads have internal state, operate concurrently, consume resources, fork/divide, and send messages. The run-time system of CellSys maintains the integrity of the global environment and mediates the interactions of cells with the environment and one another.

Organisms may choose some limited number of *actions* which depend on, react to, or alter these aspects (both internal and external). These actions include *tumbling* (i.e., choosing a new orientation), *lazing* (doing nothing), *adjusting speed* (i.e., increasing/decreasing the magnitude of its velocity), *growth* (i.e., trying to consume energy and grow), and cell *division* and *death*.

Taking the discussion of *R. Sphaeroides* as inspiration, we formulate a language, called CellSys, for describing evolving cellular systems. The abstract syntax for CellSys is:

⟨*Action*⟩	::=	*Tumble* | *Laze* | *AdjSpeed* | *Grow* | *Divide* | *Die*
⟨*Likely*⟩	::=	*light* | *size* | $r \in$ *Real* | "other arithmetic"
⟨*Branch*⟩	::=	⟨*Likely*⟩ → ⟨*Action*⟩
⟨*Cell*⟩	::=	*fuzzycase* ⟨*Branch*⟩, . . . , ⟨*Branch*⟩
⟨*CellSys*⟩	::=	⟨*Cell*⟩ ‖ . . . ‖ ⟨*Cell*⟩

There are individual actions (written ⟨*Action*⟩), each corresponding to the actions described in Section 3.2. The syntactic class ⟨*Likely*⟩ are real-valued expressions correspond-

ing to the probabilities in the branches of a Markov chain. The *fuzzycase* expression implements a Markov chain. For the purposes of the current work, expressions in ⟨*Likely*⟩ are arithmetic combinations of constants and factors internal and external to the individual cell (e.g., the size of the cell, concentrations of light, etc.) included in the model. Executing the program *fuzzycase* {...}) chooses an action from among the branches of the (*fuzzycase* {...}) expression. Individual cell programs can be combined together into systems of interacting cell threads with the concurrent execution operator ‖. The code for an initial cell system is a non-empty list of ⟨*Cell*⟩-programs.

The run-time system (RTS) for the CellSys language manages the scheduling of cell threads and defines a interface between the local state of an individual cell and the global environment. The RTS also maintains the physical integrity of the cell system: updating cell positions according to their velocities, not allowing two cells to occupy the same location, etc. Both the RTS and the semantics of CellSys[3] are defined in terms of structures from the denotational semantics of programming languages called *monads* [17, 6]. All of the well-known software-engineering benefits of monadic structuring (i.e., modularity, extensibility, and reusability) apply to our DSL definitions, thereby providing the flexibility to accommodate refinements as needed by the cell biologist. Because Haskell supports monads [7], the semantics and RTS of CellSys may represented directly as a Haskell program.

5. CONCLUSION

DSLs for system biology provide a common vocabulary for the biologist and computer scientist alike. The biologist may construct precise, formal models of cellular phenomena *without* being exposed to details of language definition. The computer scientist may tune the DSL definition with respect to its performance, concurrency model, language features, etc., to accommodate new discoveries or insights by the biologist *without* complete understanding of the exact biological phenomena.

One challenge in bioinformatics research generally is that the work requires sophisticated understanding of both biology *and* computer science. However, expertise in both these fields rarely occurs in the same scientist. We view our work as a means of bridging this gap. One principal advantage of the approach advocated here is the well-known modularity and extensibility provided by monadic approach to language design. This is particularly appropriate for the description of biological systems: as biologists refine their understanding of an organism behavior, that refinement can be relatively easily reflected in the CellSys definition because of its monadic foundation. And biologists may more easily experiment with simulations written in a language with appropriate domain-specific abstractions.

[3] Although a detailed description is beyond the scope of this work, readers interested in more detail about the formal specification of CellSys are encouraged to contact the authors.

6. REFERENCES

[1] A. Regev, W. Silverman, and E. Shapiro, "Representation and simulation of biochemical processes using the π-calculus process algebra," in *Pacific Symposium on Biocomputing*, 2001, vol. 6, pp. 459–470.

[2] N. Kam, I. Cohen, and D. Harel, "Modeling biological reactivity: Statecharts vs. boolean logic," in *Second International Conference on Systems Biology*, 2001.

[3] M. Bernaschi and F. Castiglione, "Design and implementation of an immune system simulator," *Computers in Biology and Medicine*, vol. 31, pp. 303–331, 2001.

[4] D. J. T. Sumpter, G. B. Blanchard, and D. S. Broomhead, "Ants and agents: a process algebra approach to modelling ant colony behaviour," *Bulletin of Mathematical Biology*, vol. 63, pp. 951–980, Sept. 2001.

[5] A. Regev, E. Panina, W. Silverman, L. Cardelli, and E. Shapiro, "Bioambients: An abstraction for biological compartments," *Submitted to Elsevier Science*, 2003.

[6] N. S. Papaspyrou, "A resumption monad transformer and its applications in the semantics of concurrency," in *3rd Panhellenic Logic Symposium*, 2001.

[7] S. Peyton Jones and et al., *Haskell 98 Language and Libraries, the Revised Report.*, Cambridge Univ. Press, 2003.

[8] B. Ermentrout, *Computational Modeling of Genetic and Biochemical Networks*, MIT Press, 2001.

[9] Tim Sheard, Zine el-abidine Benaissa, and Emir Pasalic, "DSL implementation using staging and monads," in *2nd Conf. on Domain-Specific Languages*, 1999, pp. 81–94.

[10] Peter Thiemann, "WASH/CGI: Server-side Web scripting with sessions and typed, compositional forms," *LNCS*, vol. 2257, pp. 192–209, 2001.

[11] D. Leijen and E. Meijer, "Direct style monadic parser combinators for the real world," Tech. Rep. UU-CS-2001-35, Dept. of Comp. Science, Univ. Utrecht.

[12] Pierre Bremaud, *Markov Chains: Gibbs fields, Monte Carlo simulation, and Queues*, Springer-Verlag, 1999.

[13] B. Taylor and I. Zhulin, "In search of higher energy: metabolism-dependent behaviour in bacteria," *Molecular Microbiology*, vol. 28, pp. 683–690, 1998.

[14] D. Shah, S. Porter, A. Martin, P. Hamblin, and J. Armitage, "Fine tuning bacterial chemotaxis: analysis of rhodobacter sphaeroides behavior under aerobic and anaerobic conditions by mutation of the major chemotaxis operons and chey genes," *EMBO Journal*, vol. 19, pp. 4601–4613, 2000.

[15] F. Dailey and R. McNab, "Effects of lipoprotein biogenesis mutations on flagellar assembly in salmonella," *Journal of Bacteriology*, vol. 184, pp. 771–776, 2002.

[16] H. Mao, P. Cremer, and M. Manson, "A sensitive, versatile microfluidic assay for bacterial chemotaxis," *Proc. of the Nat'l Acad. of Science*, vol. 100, pp. 5449–5454, 2002.

[17] Eugenio Moggi, "An abstract view of programming languages," Tech. Rep. ECS-LFCS-90-113, Dept. of Computer Science, Edinburgh Univ., 1990.