Computational Challenges in Cell Simulation:

A Software Engineering Approach

Kouichi Takahashi, Katsuyuki Yugi, Kenta Hashimoto, Yohei Yamada, Christopher J.F. Pickett, and Masaru Tomita, *Keio University*

The simulation of cellular processes involves diverse components and complex interactions. To successfully model such processes, simulation systems must meet a number of computational requirements, including an object-oriented design.

Molecular biology's advent in the 20th century has exponentially increased our knowledge about the inner workings of life. We have dozens of completed genomes and an array of high-throughput methods to characterize gene encodings and gene product operation. The question now is how we will assemble the various pieces. In other words, given sufficient information about a living cell's molecular components, can we predict its behavior?

As with any network of interacting elements, the overall behavior becomes nonintuitive as soon as the number of elements exceeds three. Computers have proven to be invaluable in analyzing these systems, and many biologists are turning to the keyboard. Analysis of biochemical systems differs from sequence analysis in that it seeks to reconstruct experimental phenomena from the known properties of individual molecules, and more importantly, the interactions between them. Such a model, if sufficiently detailed and accurate, serves as a reference, a guide for interpreting experimental results, and a powerful means of suggesting new hypotheses.

Modeling, simulation, and analysis are therefore perfectly positioned for integration into the experimental cycle of cell biology. In addition to demystifying nonintuitive phenomena, simulation lets us test experimentally unfeasible scenarios and can potentially reduce experimental costs. Although we will always need "real" experiments to advance our understanding of biological processes, conducting *in silico*, or computer-simulated, experiments can help guide the wet-lab process, effectively narrowing the experimental search space. Support of intelligent systems, such as knowledge processing of biological literature and knowledge discovery from simulation results, can also promote biosimulation. Recently, we have overcome computational obstacles to simulation, thanks to the tenfold increase in computer power in the last five years and the advent of parallel computing. Many simulation tools now exist; we briefly review some of these in the sidebar, "Current Software Platforms and Projects."

In this article, we introduce the major classes of cellular processes relevant to modeling, discuss software engineering's role in cell simulation, and identify cell simulation requirements. Our E-Cell project aims to develop the theories, techniques, and software platforms necessary for whole-cell-scale modeling, simulation, and analysis.[1] Since the project's launch in 1996, we have built a variety of cell models, and we are currently developing new models that vary with respect to species, target subsystem, and overall scale. Details on these models and the E-Cell project in general are available at www.e-cell.org.

Cellular process simulation

Cellular functions include metabolism, signal transduction, gene expression, motility, vesicular transport, cell division, and differentiation. Table 1 lists some of the cellular processes that we describe here and corresponding computational approaches. For more on these processes, see John Tyson and his colleagues' excellent review of computational cell biology with an emphasis on cell-cycle control,[2] and Robert Phair and Tom Misteli's review of the application of kinetic modeling methods to biophysical problems.[3]

Metabolic pathway models

Scientists have better characterized energy metabolism than any other part of cellular behavior, particularly in human red blood cells, or *erythrocytes*. Given that an erythrocyte is devoid of nuclei and other related features,

1094-7167/02/$17.00 © 2002 IEEE

Current Software Platforms and Projects

Many simulation systems exist. Here we briefly review some of them.

Generic biochemical simulation software

Kinsim is one of the oldest publicly available software packages specialized for rate-equation-based numerical simulation of cellular processes.[1] Kinsim is a DOS application still actively used by biochemists for kinetic analysis of enzyme reaction systems. Another rate-equation–based simulator, Gepasi, has an integrated and easy-to-use interactive Windows GUI and is widely used by biochemists for both research and education.[2] Development of Gepasi's successor, Copasi, focuses on large scalability and distributed parallel computing.

DBSolve is another ordinary differential equation- (ODE-) based simulator.[3] The Process Modeling Tool (Promot), a Lisp-based object-oriented modeling environment, uses the Diva numerics solver as a simulation back end.[4] Scientists use Simulation Control Program (Scop), a commercial tool for block-oriented generic simulation of complex systems, to run differential- and difference-equation–based cell models.[5] A-Cell is a GUI-based software for constructing biochemical reactions and electrophysiological models of neurons and other cell types.[6] Initially intended for genetic circuit simulation, the Simulation Program for Intra-Cell Evaluation (Bio-Spice) is now being developed as a generic modeling and simulation environment linked to object-relational databases.[7] Jarnac, or Scamp II, is a successor of the Scamp simulator and equipped with a powerful and flexible scripting language that lets users program a dynamic object-oriented cell model.[8]

Virtual Cell provides an intuitive Java-applet–based modeling environment that lets modelers construct spatial and biochemical models in both biological and mathematical semantic planes, and has support for empirical 3D data from microscopy.[9] In fact, the Virtual Cell group recently obtained a seminal patent on the simulation of cellular processes that are heterogeneous in space.[10]

Mathematical tools

Power Law Analysis and Simulation (PLAS, http://correio.cc.fc.ul.pt/ aenf/plas.html) is a simple yet powerful tool for modeling, simulation, and analysis of S-Systems. Many groups still use generic modeling packages such as Mathematica or MatLab, although these tools are not customized to support biosimulation per se.

Specialized biochemical simulators

StochSim is a stochastic biochemical simulator that represents individual molecules and molecular complexes as individual software objects.[11] Its unique algorithm effectively simulates biochemical systems such as the bacterial chemotaxis-signaling pathway, where only a few molecules are involved and some of them are multistate. MCell, another simulator, treats molecules individually rather than statistically, with a Monte Carlo-type random-walk algorithm for Brownian dynamics.[12] MCell simulates interactions between ligands and binding sites on receptors, enzymes, and transporters (among other molecules). The high computational costs of both StochSim and MCell discourage their use for simulation of subcellular dynamics, yet their algorithms are likely to become indispensable if given appropriate roles in a mixed-mode whole-cell model.

Infrastructural platform and model description languages

The Caltech Erato Kitano Systems Biology Project is developing an XML-based Systems Biology Markup Language (SBML, www.cds.caltech.edu/erato) as a means of interchanging biosimulation models. The group has set out to encompass all the features of Bio/Spice, DBSolve, E-Cell, Gepasi, Jarnac, StochSim, and Virtual Cell. The University of Auckland and Physiome Sciences are developing a similar modeling language called CellML (www.cellml.org).

The Caltech Erato team is also developing the Systems Biology Workbench (www.cds.caltech.edu/erato/sbw/docs/index.html). SBW is a distributed-object computing environment for biological modeling and simulation. It provides an infrastructure that unifies various software components such as model editors, simulators, and data analyzer–visualizers. Although it is incompatible with standard protocols such as Corba and Soap, it might well become the standard platform of model exchange, data exchange, and interoperation because it uses SBML as its standard language.

References

1. B.A. Barshop, R.F. Wrenn, and C. Frieden, "Analysis of Numerical Methods for Computer Simulation of Kinetic Processes: Development of Kinsim—a Flexible, Portable System," *Analytic Biochemistry*, vol. 130, no. 1, 1 Apr. 1983, pp. 134–145.
2. P. Mendes, "Gepasi: A Software Package for Modeling the Dynamics, Steady States, and Control of Biochemical and Other Systems," *Computer Applications in Biosciences* (CABIOS), vol. 9, no. 5, Oct. 1993, pp. 563–571.
3. I. Goryanin, T.C. Hodgman, and E. Selkov, "Mathematical Simulation and Analysis of Cellular Metabolism and Regulation," *Bioinformatics*, vol. 15, no. 9, Sept. 1999, pp. 749–758.
4. M. Ginkel et al., "Application of the Process Modeling Tool Promot to the Modeling of Metabolic Networks," *Proc. 3rd MathMod*, I. Troch and F. Breitenecker, eds., vol. 2, ARGESIM report no. 15, vol. 2, Vienna Univ. of Technology, Vienna, 2000, pp. 525–528.
5. J.M. Kootsey et al., "Scop: An Interactive Simulation Control Program for Micro- and Minicomputers," *Bull. Math. Biology*, vol. 48, nos. 3–4, 1986, pp. 427–441.
6. K. Ichikawa, "A-Cell: Graphical User Interface for the Construction of Biochemical Reaction Models," *Bioinformatics*, vol. 17, no. 5, May 2001, pp. 483–484.
7. H.H. McAdams and A. Arkin, "Simulation of Prokaryotic Genetic Circuits," *Ann. Rev. of Biophysics and Biomolecular Structure,* vol. 27, 1998, pp. 199–224.
8. H.M. Sauro, "Scamp: A General-Purpose Simulator and Metabolic Control Analysis Program," *Computer Applications in Bioscience*, vol. 9, no. 4, Aug. 1993, pp. 441–450.
9. L.M. Loew and J.C. Schaff, "The Virtual Cell: A Software Environment for Computational Cell Biology," *Trends in Biotechnology*, vol. 19, no. 10, Oct. 2001, pp. 401–406.
10. J.C. Schaff, J. Carson, and L. Loew, *Method and Apparatus for Modeling Cellular Structure and Function*, US patent 6,219,440 B1, Patent and Trademark Office, Washington, D.C., 17 Apr. 2001.
11. N. Le Novere and T.S. Shimizu, "Stochsim: Modeling of Stochastic Biomolecular Processes," *Bioinformatics*, vol. 6, no. 6, June 2001, pp. 575–576.
12. T.M. Bartol, Jr., et al., "MCell: Generalized Monte Carlo Computer Simulation of Synaptic Transmission and Chemical Signaling," *Society for Neuroscience Abstracts*, vol. 22, Nov. 1996, pp. 1742.

Table 1. Cellular processes and typical computational approaches.

Process type	Dominant phenomena	Typical computation schemes
Metabolism	Enzymatic reaction	DAE, S-Systems, FBA
Signal transduction	Molecular binding	DAE, stochastic algorithms (StochSim and Gillespie, for example), diffusion-reaction
Gene expression	Molecular binding, polymerization, degradation	OOM, S-Systems, DAE, Boolean networks, stochastic algorithms
DNA replication	Molecular binding, polymerization	OOM, DAE
Cytoskeletal	Polymerization, depolymerization	DAE, particle dynamics
Cytoplasmic streaming	Streaming	Rheology, finite-element method
Membrane transport	Osmotic pressure, membrane potential	DAE, electrophysiology

DAE—differential-algebraic equations (rate equation-based systems), FBA—flux balance analysis, and OOM—object-oriented modeling (includes E-Cell's substance-reactor model, or SRM).

it is ideal for studying metabolism in isolation—it is, essentially, a "bag of metabolism." Biochemists have successfully collected enough quantitative data to construct kinetic models of an entire cell. These metabolic models, as well as many others, typically constitute a set of ordinary differential equations (ODEs) that describe the enzymatic reaction rate. We can solve these equations by numerical integration, for which several well-established algorithms exist.

Modern metabolic pathway models typically consist of primary state variables for molecular species concentrations, one ODE for each enzyme reaction, and a stochiometric matrix. Researchers derive the rate equations of most modern enzyme kinetics models using the King-Altman method,[4] which is a generalized version of Michaelis and Menten's classical formulation.[5] Additional algebraic equations commonly serve as constraints on the system. Thus, we describe most metabolism models as differential-algebraic equations (DAEs).

The theoretical and practical bases of simulating metabolic pathways are well grounded.[6] However, the design and implementation of simulation software and model-construction methods, which this article attempts to highlight, are still under active discussion.

Signal transduction pathways

Although quantitative data on cellular processes other than metabolism are still relatively sparse, scientists have modeled some systems with considerable success. An example is the signal transduction pathway controlling bacterial chemotaxis.[7]

Signal transduction pathways ordinarily have much fewer reactant molecules than metabolic systems, and applying ODE-based modeling reveals the underlying stochastic behavior of the molecular interactions. Accordingly, researchers have attempted to model signal transduction pathways with stochastic computation instead of deterministic methods, such as ODEs.[8]

Gene expression systems

Gene expression systems, such as the simulation of gene expression in phage-λ,[9] have also been successfully modeled. Like signal pathways, they tend to contain a small number of molecular entities, including transcription factors, polymerases, and genes. These low-copy-number molecules orchestrate gene expression in a highly stochastic fashion. For example, the stochastic behavior of gene expression's initial phase has binary consequences: binding a rare transcription factor and a single gene in a cell can determine whether the gene is turned on or off. In many cases, we can best model gene expression systems with stochastic equations. There are many other ways to model this phenomenon, however, depending on the modeler's interests. For example, we can use ODE models (linear and mass action models), S-System models,[10] or binary and multireaction models.[11]

Gene expression systems have rich interactions with other cellular processes. These systems can control metabolic flux by changing the enzyme concentration while being regulated by signaling proteins. DNA binding factors, which are derived from other genes, dynamically regulate chromosomal structure. When we model a whole cell, elements in the gene expression system sometimes need information about elements in other systems to allow cross-system interaction. Integrating gene expression with other systems at the whole-cell level might best be accommodated by object-oriented data structures, as we describe in earlier work.[12]

Cytoskeletal movement and cytoplasmic streaming

All of these simulation examples treat the properties of protein binding and enzyme kinetics reactions. Certain cellular processes such as cytoskeletal movement and cytoplasmic streaming, however, must be modeled at the biophysical level. Cytoplasmic streaming involves diffusion of relatively heavy proteins, whereas cytoskeletal movement causes structural changes, including cell division and differentiation. Simulations, which have been ongoing since the 1970s,[13] have become more precise with time, concomitant with our growing understanding of cellular processes at the molecular level.[14]

Software engineering in biosimulation technology

We envision a cell biology research cycle that incorporates biosimulation technology, as Figure 1 shows. Every step in the cycle completed outside the wet lab depends on sound software engineering methods. Consider the storage, processing, and utilization of massive amounts of biological knowledge: Only by integrating an intelligent modeling environment with sophisticated data and knowledge bases can we model such a large and complex system. Although this article emphasizes the first half of the cycle (from "Qualitative modeling" to "Run" in the figure), it is important to note that a technological stagnancy in any one of the steps might form a bottleneck, which could threaten the evolution of computational cell biology.

Software development for nontrivial cell simulation projects is notably expensive. For research projects in the traditional computational sciences, where brute force computation remains operative, it is reasonable to develop new—and sometimes disposable—software for each project. Numerous uniform components and a limited number of simple principles characterize most traditional computational science fields, such as computa-

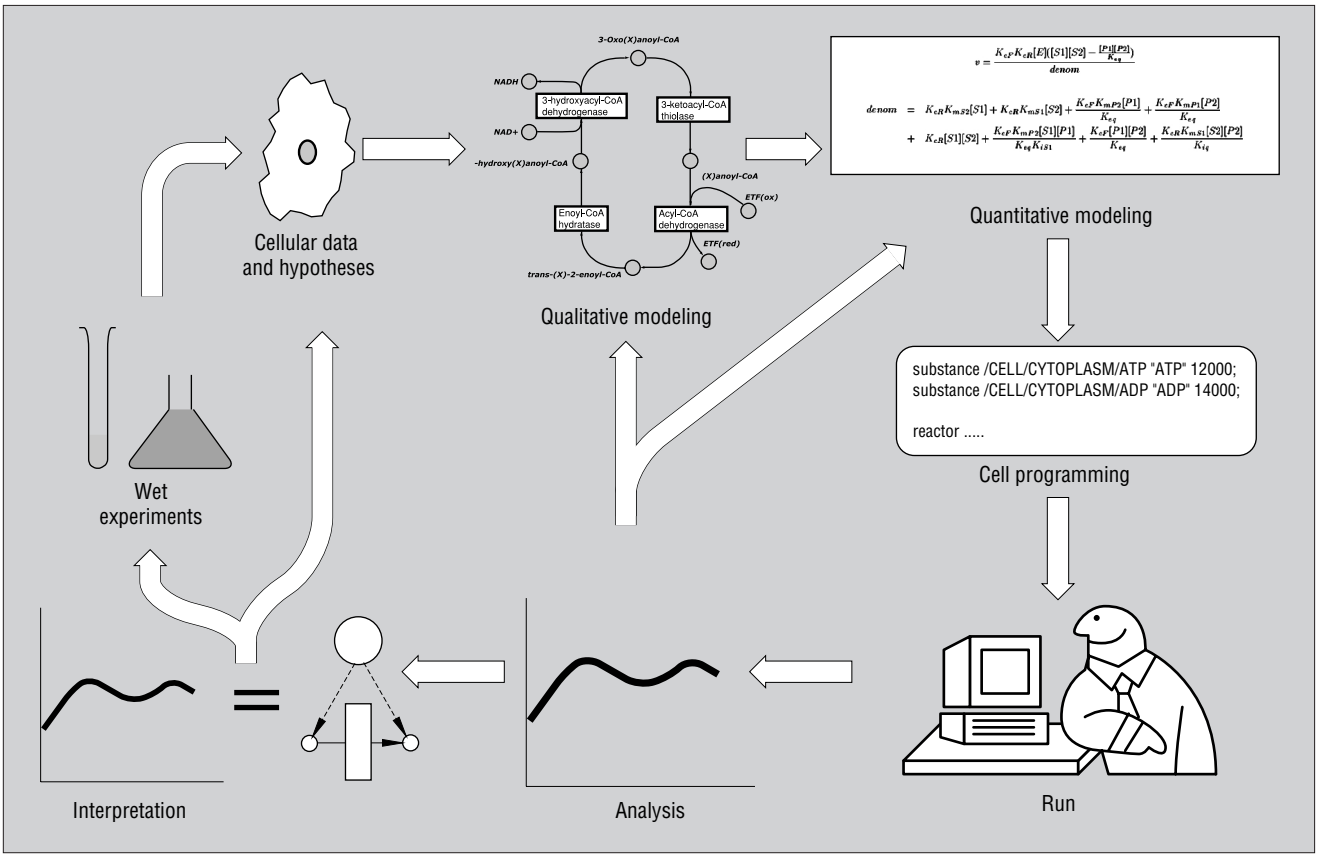


Figure 1. The computational cell biology research cycle. We extend the wet-lab process to include simulation software for a computational cell biology research project. First, we build qualitative models (such as pathway maps) from in vivo and in vitro data and hypotheses, or a reference model (qualitative modeling). Then, quantitative characterization of cellular properties facilitates the transition to a mathematical system model (quantitative modeling). We then translate the numerical and discrete properties of the quantitative model into a modeling language (cell programming), and predict the systemic behavior (run). Analysis of the results suggests new hypotheses (analysis and interpretation), which we subsequently test by wet experiments, and the cycle begins anew.

tional physics. Cell simulation, in contrast, involves many different components interacting in diverse and complicated ways. Table 2 summarizes typical characteristics of several simulation targets in cell biology and physics. The design and implementation of simulation software inevitably reflects the problem's complexity. We count component types that require different data structures or object classes. A **membrane** object, for example, needs a different object class from protein molecules. We do not count different molecular species, however. The number of interaction modes depends on whether we count different enzyme kinetics equations, which are roughly proportional to the number of enzyme-encoding genes, as different interaction modes. Interaction modes other than enzymatic reactions include molecular bindings (complex formations), molecular collisions, DNA replication, cytoplasmic streaming, cytokinesis, and vesicular trafficking.

Requirements analysis

There are several necessary features of a flexible and multipurpose biosimulation software platform. We ignore critical features common to the vast majority of modern platforms, such as model scalability, focusing instead on features not present in all software. While we

Table 2. Rough comparison of typical numbers characterizing various simulation targets in cell biology and physics.

Target	Compartments	Components	Component types	Interaction modes
Prokaryote s *(E.coli)*	~10^{1}	~$10^{13–14}$ molecules ~$10^{3–4}$ species	~10^{1}	~$10^{1–3}$
Eukaryotes *(H. sapiens)*	~$10^{3–4}$	~$10^{17–18}$ molecules ~$10^{4–5}$ species	~10^{1}	~$10^{1–4}$
LSI (electronic circuit)	Usually 1	~$10^{6–7}$	A few	1
CFD (fluid dynamics)	Usually $10^{0–1}$	~$10^{5–6}$	1	1
MD (molecular dynamics)	1	~$10^{2–6}$	A few	A few

recognize that many factors—including simulation granularity, scale, and purpose—can alter these requirements, our goal is a generic description of a cellular platform.

Mixed-mode simulation

In the cell, various components interact in diverse manners. All cellular subsystems are highly nonlinear, and subsystem couplings are often nonlinear as well. This nonlinearity indicates that the whole system is not equivalent to the sum of its subsystems. Although we can to some extent investigate a subsystem in isolation by assuming steady and simplified boundary conditions, we cannot elucidate its real behavior and role unless we consider it a part of the whole.

Cell simulators must therefore allow simulation of cell subsystems in both isolated and coupled forms. To simulate coupled subsystems, we must perform computations on mutually interacting subsystems with different computational properties on a single platform. There is, however, no universal algorithm that can efficiently simulate all subsystems at once, so simulators must allow multiple computation algorithms to coexist in a single model.

To support mixed-mode computation, the software must therefore provide a single abstract programming interface that both allows indiscriminate interaction among modules and gives front-end programs a standard means of visualizing and manipulating these modules. Implementations of the algorithm modules must also be isolated from the system-provided common interface.

Object-oriented modeling

Cell modeling involves the integration of a diverse array of processes. As Table 2 outlines, cell modeling differs significantly from typical modeling problems in physics, where computer simulation has been widely applied. Fluid mechanics, gravitational *N*-body problems, and quantum field simulations have many uniform components (atoms and particles, for instance) that are governed by one or more equations (such as the Navier-Stokes equation, Newton's equation, or quantum wave functions). Preparing structural information and initial conditions for such simulation models is relatively easy. We can assume that uniform components have similar values for some or most constants and parameters. If there are only a few interaction modes, we do not need to conduct surveys and derive equations for each reaction. Moreover, initial conditions are often automatically generated. The major challenge for software designers is to maximize computational efficiency in existing hardware while keeping the simulation software flexible.

In cell simulation or modeling, scientists must either dig through the literature or conduct wet experiments to identify reaction mechanisms, derive appropriate rate equations, and obtain values of rate constants. Having participated in more than 10 cell-modeling projects, we know that this research, review, data acquisition, and model construction phase often takes more than 50 percent of the time available to the project, with most of the remaining time spent debugging models. Proportionately, more time is distributed among these activities as the model gets larger, and intuition and productivity prove more important than computational efficiency. Most simulation engines will have similar runtime efficiencies if they use the same kind of algorithms, and we can achieve the greatest performance increase in presimulation phases. Thus, the modeling front end should directly reflect the biologist's understanding of the cellular system, and models must be modularized for reuse whole or in part as submodels in different contexts.

Ontological diversity is inherent to biology. Although it is somewhat obvious that no existing universal modeling scheme can model all subsystems elegantly and intuitively, cell simulators must attempt such a subsystem, or biological models will never meet life's staggering complexity. Biology is a science of exceptions, and biologists make new, nonintuitive discoveries all the time. For example, we might ask how anyone could predict with any confidence that noncoding complementary RNA might inhibit translation of the coding mRNA. Such considerations lead the software designer to provide a cell-modeling framework that comes packaged with as few presumptions as possible.

The object-oriented paradigm provides an appropriate solution. This programming model is characterized by a data-centric framework in which procedures are tied to the data. The system's actors are described as *objects*; object *properties* are data and methods (operations on the data); and objects are grouped into *classes*, interacting with each other by exchanging *messages*. Inheritance (is-a), aggregation (has-a, consists-of), and dependence (uses-a) relationships can exist between classes. We can describe these relationships in Unified Modeling Language (UML), and code them with object-oriented computer languages such as C++, Java, Smalltalk, and SIMULA.

The object-oriented paradigm was in fact originally devised to program discrete simulations. Also relevant to the biosimulation problem is the fact that many object-oriented data structures, owing to their data-centric nature, have strong affinity to databases. The object model of cell simulators must

- Let users define their own classes. As the RNA example suggests, a cell model might require new data structures currently not provided by the simulation engine.
- Provide abstraction support (for example, via base classes) for common use by algorithms. This allows interactions between different algorithms and subsystems and across scales of time and space, which are critical features of an integrated simulation model.

Internal data structure

Traditionally, dynamic system simulation models are either described in equation-based form and then compiled into computer programs, described in an object-oriented fashion before being translated to an equation-based form, or written directly as computer programs. The second of these could provide the best of both worlds: clean design in the modeling phase and efficient numerical routines in the simulation phase.

Although computationally efficient, all three of these approaches have several inherent disadvantages:

- None of the approaches preserve the model's semantics. Modeling knowledge about system structural or behavioral features are abstracted out and only implicitly represented in the mathematical description.
- Model and program reuse is limited. If the model and its internal representation do not directly reflect each other, changes on one side require a change on the other, making it difficult to reuse models in modified contexts.
- The classical flat description hides critical features and assumptions, such as computation order and the model's causal structure.

Hitz and Werthner showed how object-orientation can help overcome these drawbacks in dynamic system simulation. They also showed how researchers can extend the paradigm, which is well established in discrete event-driven simulations, to dynamic system

simulations in which they normally use differential or difference equations.[15]

Cells are dynamic environments in which the components are constantly synthesized, processed, and degraded. The system's structural properties can change throughout the cell cycle, such as when a cell grows and divides. With a differential-equation–based approach, representation of structural change and dynamic component synthesis and degradation requires convoluted mathematical techniques. Even if the model is object-oriented in the modeling phase, exploring the system's dynamic nature during simulation becomes extremely difficult if we use a compiled representation. Such dynamic structures can be intuitively represented and efficiently implemented in object-oriented simulators.

Even if the model itself (as opposed to its components) cannot change structure during simulation, a one-to-one relationship between the model and simulation data structures is advantageous. We can facilitate saving, loading, and resuming simulation sessions by implementing support for object persistence technology. Keeping the simulation in the same semantic plane as the model helps users understand the ontologically complicated system via simulation results and runtime interaction.

Runtime user interaction

One purpose of a simulation project is to characterize a target system's dynamic behavior. Mathematical analysis methods such as metabolic control analysis, bifurcation analysis, and parameter estimation are usually automated, making user interaction unnecessary.

As the model grows and gets more complicated, however, the advantages of runtime interaction with the simulation engine become significant. Even for prototype models in early developmental stages, modelers would like to observe the effects of stimulations at arbitrary time points. In a modeling project's debug phase, runtime interactions constitute a powerful means to check for the existence and location of "bugs." Cell simulation software often lacks debugging support, but experience has shown that it can eliminate some severe bottlenecks, especially in large-scale, multiauthor projects. After model completion, scientists can intuitively investigate dynamic properties through runtime interactions, and the insight gained is educational to say the least; indeed, such interaction with a model could help bring cell simulation to the classroom.

It goes without saying that the same interaction functionality is required for predefined scripts that dynamically affect the models' variables and structure, and so simulators must seamlessly provide users with both batch-mode and runtime graphical interaction mode user interfaces.

Spatial information

Even within a single cell compartment, there exist many systems where material distributions are not uniform in space and molecule localization plays an important role. Such systems include vesicle trafficking in the secretory process, diffusion of ions and molecules after crossing a membrane, and propagation of an action potential along a nerve fiber's axon.

In whole-cell or multicell-scale modeling, it is reasonable to assume that chemical concentrations are homogenous throughout a given compartment if the reactions are confined to a small space, and when the diffusion rate is faster than the localized reaction rate. However, if neither of these conditions is met, which is often the case for whole-cell or multicell-scale modeling, spatial information support becomes essential. Thus, simulation engines require native support for factors such as compartment structure and location, spatial concentration gradients for all species, and interactions between moving molecules.

One way to incorporate spatial information into the cell simulation is to use a finite-element mesh to divide the subcellular space and describe each element's location with some coordinate system. A common framework would define geometric coordinates' syntax, whereas each computation module would define its semantics. Such spatial support requires that all algorithm modules share an interface, which must describe not only molecular concentrations for systems with many molecules, but also individual molecules for systems with few particles and for purely structural phenomena.

If we use multiple computation modules with different simulation algorithms and spatial–time-step sizes to let diverse simulation schemes with a wide range of temporal and spatial scales coexist, spatial information support might become extremely complicated. The spatial interface therefore not only must be flexible enough to allow support for every module, but also simple enough to prevent confusion. Enabling multiple pluggable modules with contrasting semantics, dimensions, and coordinates to share a common syntax or description method via a common framework will meet both requirements. Thus, a generic, flexible, and simple spatial information interface is a key feature of next-generation simulation engines.

Varying timescales

Cell simulators must be able to handle the coexistence of many timescales in a single cell model. Typical timescales of cellular phenomena vary from femtoseconds (10^{-15} s) to hours (10^{3} s). One example of the relatively high resolution necessary to realistically simulate cellular behavior is found in signal transduction systems dependent on molecular bindings, which need step sizes on the order of (10^{-6} s). Metabolic pathways, on the other hand, are characterized by enzymatic reactions and have close to millisecond (10^{-3} s) timescales. Gene-expression systems are microscopically driven by macromolecular interactions (below 10^{-6} s), yet resulting phenomena are often reported and analyzed in minutes, hours, and days (10^{1} s – 10^{4} s).

In differential-equation–based simulators, adopting semi- and fully implicit stiffness-proof integration algorithms such as Gear's method usually overcomes stiffness. Although we can use these techniques in part for ODE-based subsystems, we cannot fully apply them to mixed-mode simulators. Implementing a mixed-mode stiffness-proof simulator requires a new scheduling algorithm that can orchestrate many subsystems running with different algorithms and variable temporal step sizes.

Types of state variables

A cell model's primary state variables are the quantities of species, and we can use one of two approaches to model them. In the first, each state variable is a positive real number, and the fractional parts are not discarded—this number format is suitable for working with the empirical rate equations of biochemistry. The second approach keeps molecular quantities as natural numbers, and modelers can use a variety of stochastic or deterministic methods to maintain the fractional part's semantics. In addition, realistic cell models usually require a mixture of continuous state variables such as temperature, electric potential, and free energy, and discrete variables such as the state flags of multistate molecules (for example, transiently modified proteins). Therefore, the simulator must handle positive, nonzero molecular quantities, real negative or positive numbers, and discrete states of molecules.

The E-Cell simulation environment

The E-Cell simulation environment, version

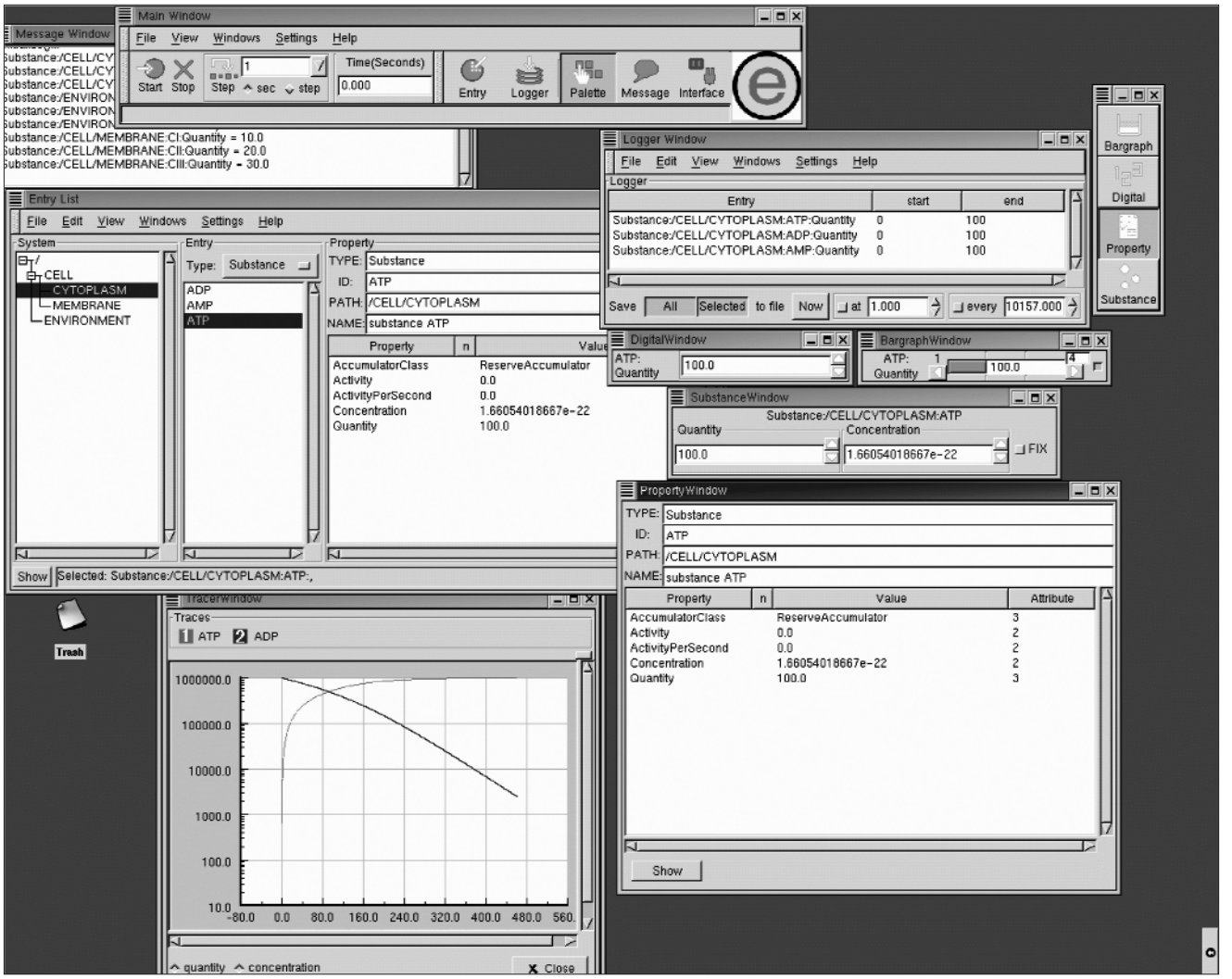


Figure 2. Screenshot of the Osogo user interface. E-Cell 3 provides Python and C++ APIs to support front-end software, including GUI session monitors, model editors, and mathematical analysis sessions. This image is a development snapshot of the Osogo GUI session monitor written in Python, which lets users run the simulation, visualize and save data, and interact with the model. All components of Osogo are Python plug-ins, and extended functionality can easily be provided by additional plug-ins. Osogo lets users seamlessly switch between batch-mode and GUI-mode simulation sessions.

1 (E-Cell 1) is a generic object-oriented whole-cell and multicell-scale simulator that we have been developing since 1996. Its design and implementation, which meet most of the cell simulation-specific requirements we outlined earlier, allow construction and simulation of various cell models using a single piece of software. E-Cell 1 is now a stable platform, and version 3 (E-Cell 3) is currently under active development. E-Cell 2 is the Windows version of E-Cell 1.

We have completely redesigned and reconstructed the E-Cell 3 architecture. Unlike E-Cell 1, E-Cell 3 lets us create *wrappers*, or interface layers, of its C++ API. We have also developed a wrapper for the Python language, allowing development of front-end software in the productive scripting language concurrently with C++. We are currently developing a GUI session monitor, a batch-mode simulation session framework for mathematical analysis, and a model editor, as Figure 2 shows. In the future, additional interface layers will provide seamless integration with distributed computing middleware such as Corba and Soap.

E-Cell 3's simulation engine is essentially a continuation of E-Cell 1's. The most notable improvements are support for parallel computing and enhanced support for spatial information. Simulation engine scalability is an important feature of whole-cell simulation, and in the E-Cell 3 development phase, we focus on improved scalability over ease of use, because we can always improve ease of use through carefully designed and implemented front-end software. Although E-Cell 1 is actually a single-scheme simulator with the SRM object-model, the definition of new object classes has let us develop diffusion-reaction, S-System, and flux distribution analysis modeling toolkits. However, E-Cell 3's simulation engine truly supports mixed-mode computation by integrating native computation modules and object model-based modules.

We need a new computational approach for whole-cell simulation, and sophisticated software engineering plays a vital role in cell simulation projects. Although biochemical process simulation has a considerable history, integrating partial simulation models into a whole or multiple cell model is still an emerging field. Intelligent systems are expected to make it more fertile through knowledge processing, discovery of biological literature, and simulation outputs. The requirements described in this article are

neither complete nor universal, and we hope to evoke active discussion on the importance of software design in cell simulation. ■

Acknowledgments

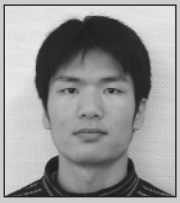
The authors thank Thomas Shimizu at Cambridge University for critical readings and editorial work. Yuri Matsuzaki and Takeshi Sakurada at Keio University kindly helped us prepare surveys and figures. This work is supported in part by a grant from the Ministry of Agriculture, Forestry, and Fisheries of Japan (Rice Genome Project 5Y-2103), a grant from the New Energy and Industrial Technology Development and Organization (NEDO) of the Ministry of Economy, Trade, and Industry of Japan (Development of a Technological Infrastructure for Industrial Bioprocess Project), a grant from the Japan Science and Technology Agency (JST), and the Natural Science and Engineering Research Council of Canada.

References

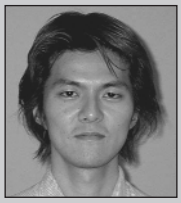
1. M. Tomita et al., "E-Cell: Software Environment for Whole Cell Simulation," *Bioinformatics,* vol. 15, no. 1, Jan. 1999, pp. 72–84.
2. J.J. Tyson, K. Chen, and B. Novak, "Network Dynamics and Cell Physiology," *Nature Reviews Molecular Cell Biology,* vol. 2, no. 12, Dec. 2001, pp. 908–916.
3. R.D. Phair and T. Misteli, "Kinetic Modeling Approaches to *in vivo* Imaging," *Nature Reviews Molecular Cell Biology,* vol. 2, no. 12, Dec. 2001, pp. 898–907.
4. E.L. King and C. Altman, "A Schematic Method of Deriving the Rate Laws for Enzyme Catalysed Reactions," *J. Physical Chemistry*, vol. 60, 1956, pp. 1375–1378.
5. L. Michaelis and M.L. Menten, "Die Kinetik der Invertinwirkung" (Kinetics of the Action of Invertine), *Biochemische Zeitschrift,* vol. 49, 1913, pp. 333–369 (in German).
6. C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Upper Saddle River, N.J., 1971.
7. D. Bray, R.B. Bourret, and M.I. Simon, "Computer Simulation of the Phosphorylation Cascade Controlling Bacterial Chemotaxis," *Molecular Biology of the Cell*, vol. 4, no. 5, May 1993, pp. 469–482.
8. C.J. Morton-Firth, T.S. Shimizu, and D. Bray, "A Free-Energy-Based Stochastic Simulation of the Tar Receptor Complex," *J. Molecular Biology,* vol. 286, no. 4, Mar. 1999, pp. 1059–1074.
9. H.H. McAdams and L. Shapiro, "Circuit Simulation of Genetic Networks" *Science*, vol. 269, no. 5,224, Aug. 1995, pp. 650–656.
10. M.A. Savageau, *Biochemical Systems Analysis: A Study of Function and Design in Molecular Biology*, Addison-Wesley, Boston, 1976.
11. J. Hasty et al., "Computational Studies of Gene Regulatory Networks: *In numero* Molecular Biology," *Nature Reviews Genetics,* vol. 2, no. 4, Apr. 2001, pp. 268–279.
12. K. Hashimoto et al., "Generic Gene Expression System for Modeling Complex Gene Regulation Network Using E-Cell System," *Genome Informatics 1999*, Universal Academy Press, Tokyo, pp. 356–357.
13. C.E. Novitski and A.S. Bajer, "Interaction of Microtubules and the Mechanism of Chromosome Movement (Zipper Hypothesis). 3 Theoretical Analysis of Energy Requirements and Computer Simulation of Chromosome Movement," *Cytobios*, vol. 18, 1978, pp. 173–182.
14. F. Gibbons et al., "A Dynamical Model of Kinesin-Microtubule Motility Assays," *Biophysics J.*, vol. 80, no. 6, June 2001, pp. 2515–2526.
15. M. Hitz and H. Werthner, "Earning Benefits of the Object-Oriented Paradigm in Dynamic System," *Proc. IEEE Hawaii Int'l Conf. System Sciences* (HICSS), IEEE Computer Soc. Press, Los Alamitos, Calif.,1997.

The Authors

Kouichi Takahashi is a PhD student at the Institute for Advanced Biosciences, Keio University, and the E-Cell consortium's system architect of the E-Cell simulation environment. His research interests are modeling-theory development, simulation techniques, and software systems for cell modeling and simulation, digital signal processing, dynamical system analysis, autopoiesis system theory, and in vivo information coding in DNA sequences. He is a member of the International Society for Computational Biology and the Japanese Society of Molecular Biology. Contact him at Keio Univ., 5322 Endo Fujisawa, Kanagawa 252-8520, Japan; shafi@e-cell.org.



Katsuyuki Yugi is a PhD student at the Institute for Advanced Biosciences, Keio University, where he also received an MS in bioinformatics. His research interests are simulation and analysis of mitochondrial energy metabolism and development of large-scale modeling methods. He is a member of the Japanese Society of Molecular Biology. Contact him at Keio Univ., 5322 Endo Fujisawa, Kanagawa 252-8520, Japan; chaos@e-cell.org.



Kenta Hashimoto is a PhD student at the Institute for Advanced Biosciences, Keio University, where he also received an MS in bioinformatics. His research interests are simulation techniques, knowledge processing and software engineering for large-scale cell-modeling projects, and modeling and simulation of gene expression systems. Contact him at Keio Univ., 5322 Endo Fujisawa, Kanagawa 252-8520, Japan; kem@e-cell.org.

Yohei Yamada is an MS student at the Institute for Advanced Biosciences, Keio University, where he also received a BS in bioinformatics. His research interests are software development for multidimensional cell simulation and cell division modeling and simulation. Contact him at Keio Univ., 5322 Endo Fujisawa, Kanagawa 252-8520, Japan; yoyo@e-cell.org.

Christopher J.F. Pickett is an MSc student in computer science at McGill University, where he also received a BSc in biochemistry. After receiving his BSc, he studied at the Institute for Advanced Biosciences, Keio University, for six months. His research interests are cell modeling and simulation, metabolic thermodynamics, programming language theory, and structural biology. Contact him at McGill Univ., School of Computer Science, McConnell Eng. Bldg., Rm. 318, 3480 University St., Montreal, QC, H3A 2A7, Canada; chris.pickett@mail.mcgill.ca.

Masaru Tomita is a professor and the director of the Institute for Advanced Biosciences, Keio University. His research interests are bioinformatics, genome informatics, theoretical molecular biology, and biological simulation. He received a BS in mathematics from Keio University, an MS and PhD in computer science from Carnegie Mellon University, and a PhD in molecular biology from Keio University. Contact him at Keio Univ., 5322 Endo Fujisawa, Kanagawa 252-8520, Japan; mt@sfc.keio.ac.jp.