Sigmoid: A Software Infrastructure for Pathway Bioinformatics and Systems Biology

Jianlin Cheng, Lucas Scharenbroich, Pierre Baldi, and Eric Mjolsness,
University of California, Irvine

Integrating Sigmoid's distributed modules in a three-tier architecture supports a generative, scalable software infrastructure for systems biology modeling projects.

Biological systems encompass everything from immune and nervous systems to ecosystems. *Systems biology*, however, focuses primarily on unraveling molecular systems at the level of pathways and groups of pathways in a cell and its neighboring cells. Here we address the challenge of creating an expert assistance system for modeling biological pathways, using current software technology to decrease development costs and complexity. Our goal is to provide computational support to biologists and computational scientists who need to create and explore predictive dynamical models of complex biological systems such as metabolic, gene-regulation, or signal-transduction pathways in living cells.

A basic methodological roadmap for systems biology outlines four idealized steps:

1. Identify all the system's players or components, such as genes, proteins, and compartments.
2. Perturb the components through a series of genetic or environmental manipulations and record the global response using high-throughput technologies (such as microarrays).
3. Build a global model of the system by describing it biologically, then mathematically, then computationally.
4. Analyze the model to generate a new testable hypothesis. Return to step 2 to test the hypothesis, and in some cases to step 1 to discover missing components.[1–3]

We developed Sigmoid, a generative, scalable software infrastructure for systems biology, to facilitate the third and fourth steps. By integrating the hypothesis and discovery phases in the research process, Sigmoid supports the process of cycling between model building, hypothesis generation, and biological experimentation and data gathering (see figure 1).

This inference cycle demands a scalable software architecture that can capture biological systems' underlying complexity and effectively model their many components and modules that operate at multiple spatial and temporal scales. We thus had to ensure that each infrastructure component is intrinsically scalable both in the number of biological objects and processes, and in their logical complexity. While the infrastructure's primary task is reverse-engineering biological circuits, in the long run we expect it to apply also to bioengineering projects by supporting the design and synthesis of complex sets of molecular interactions with a particular computational, biomedical, or biosynthetic purpose.

Sigmoid: An overview

The Sigmoid modeling system consists of distributed modules that implement the following components (see figure 2):

- pathway and cell model generation and simulation (via Cellerator, our cell model generator[4]),
- a pathway modeling database,
- Web services-oriented middleware,
- a biologist-friendly GUI, and
- parameter optimization and other data mining technologies (slated for future development).

Key to the infrastructure's design is its scalability, ensured by leveraging symbolic computer algebra and self-generation of database and other code from high-level representations such as Unified Modeling Language schema. UML offers an innovative approach to coordinating development of Sigmoid's various soft-

1541-1672/05/$20.00 © 2005 IEEE
Published by the IEEE Computer Society

ware modules.[5] Consulting with bioinformaticians and biologists, we used UML to diagram the most important biological objects—reactions and molecular reactants—and their relationships. This UML diagram then actually became source code from which several parts of our system were automatically generated: specifically, the Sigmoid pathway-modeling database (in Structured Query Language) and the corresponding Java object hierarchy, along with support files for facilitating the object-relational mapping and end-user documentation. The GUI uses Java "reflection" to automatically discover much of what it needs to know about the Sigmoid schema. This guarantees that the software implements something very close to the agreed-upon biological objects.

At the implementation level, this approach creates a classical three-tier architecture with a back end, middle layer, and front end (see figure 3). The back end includes the database, simulator, and other model manipulators. The GUI front end doesn't access these modules directly but rather through a Web services middleware module, which also brokers communications between the back-end components themselves. Developing the middleware adds to development overhead but also provides immense advantages in distributed computing, performance, flexibility, and scalability. The sidebar "Sigmoid Building Blocks" describes the technologies used in each layer.

To keep the infrastructure flexible and manageable as it grows, we took a generative approach that partially automates the production of both executable code and mathematical models. This lets us start from high-level inputs such as UML diagrams and reaction notations that non-computer scientists could understand. During implementation, we made visible the design of many essential software objects and their relationships, which made it easy to gain biologists' insights on the infrastructure.

Although we're developing Sigmoid components simultaneously, some, such as the database and simulator, are more mature and self-sufficient than others and can be used independently of the GUI or middleware.

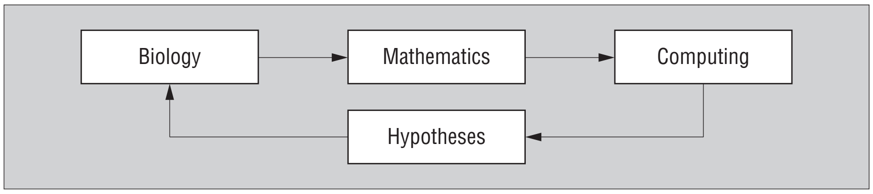


Figure 1. Basic systems biology inference cycle.

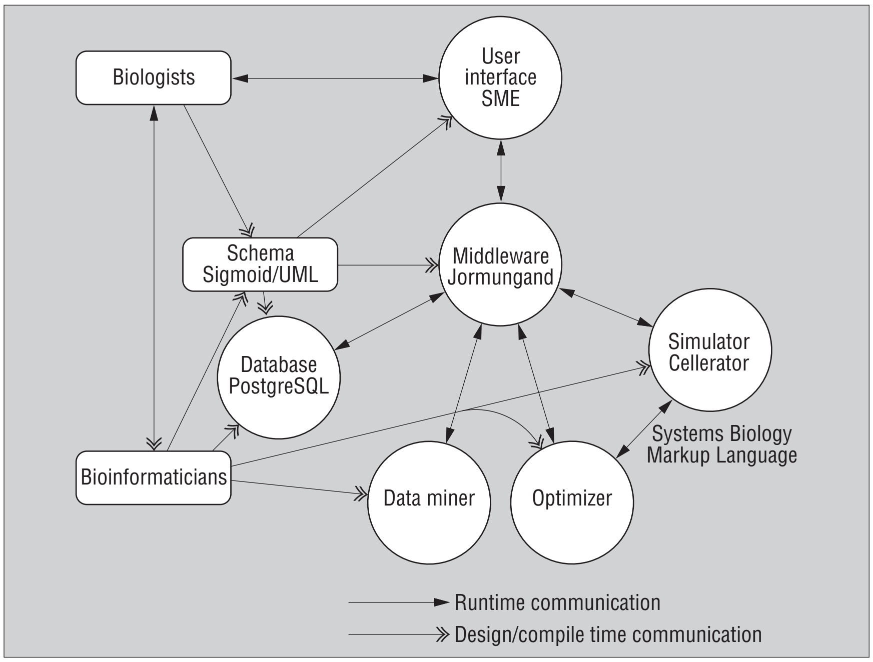


Figure 2. A top-level view of the Sigmoid architecture. Separating modules into a communicating distributed system increases the architecture's scalability. Our simulator is the Cellerator model generator and simulator, the database is Sigmoid (autogenerated from UML schema not shown), and the user interface is the Sigmoid Model Explorer (SME).

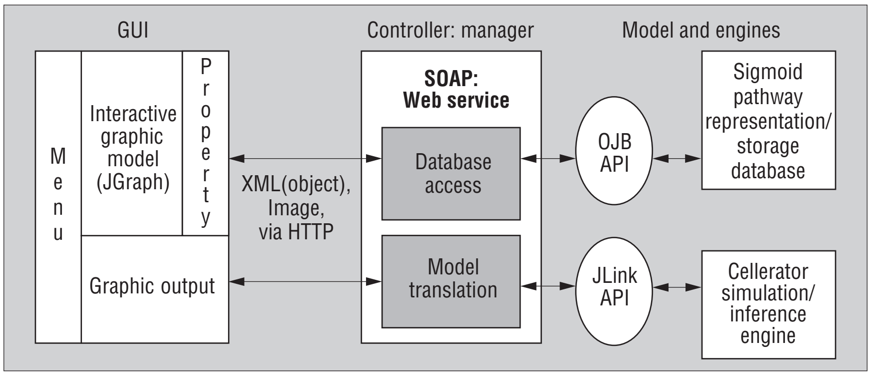

GUI
Controller: manager
Model and engines
Menu
Interactive graphic model (JGraph)
Property
Graphic output
XML(object), Image, via HTTP
SOAP: Web service
Database access
Model translation
OJB API
JLink API
Sigmoid pathway representation/ storage database
Cellerator simulation/ inference engine

Figure 3. Sigmoid's three-tier architecture.

Model generation and simulation: Cellerator

Modeling a biological pathway involves simulating dozens if not hundreds or even thousands of elementary chemical reactions. Whatever the details of the equations (typically differential equations) used to model an individual reaction, manually building a model containing many reactions is a tedious and error-prone process. Unlike electronic circuits, which consist of relatively few elementary building blocks, biochemical reactions can take numerous elementary forms.

Our architecture therefore requires a library of reusable reaction models expressible in a simple, higher-level language to specify molecular species and reaction type. For example, syntax such as "$A + B \rightarrow C$; mass action with rate k" specifies that molecular species A inter-

Sigmoid Building Blocks

The Sigmoid architecture uses a wide variety of software, languages, and tools, and we use publicly available open-source tools as much as possible.

The front-end GUI uses

- Java,
- Java reflection,
- JGraph (www.jgraph.com),
- JavaScript,
- Java Web Start (http://java.sun.com/products/javawebstart),
- HTML,
- XML, and
- a Web browser.

The middleware uses

- Java,
- Java Servlet,
- Java Server Pages,
- XML,
- Simple Object Access Protocol (SOAP, www.w3.org/TR/soap12-part1),
- Apache Jakarta Tomcat Web server (http://jakarta.apache.org/tomcat/), and
- Object Relational Bridge (OJB, http://db.apache.org/ojb).

The back-end solver uses

- Cellerator (www.igb.uci.edu/servers/sb.html, www.cellerator.info),
- Mathematica (www.wolfram.com),
- JLink (www.wolfram.com), and
- Systems Biology Markup Language (SBML, http://sbml.org).

The back-end database uses

- UML (www.uml.org),
- XML,
- PostgreSQL,
- Anything from XMI Generator (AXgen, http://axgen.sourceforge.net/),
- Velocity Template Language (VTL, http://jakarta.apache.org/velocity),
- OJB, and
- Linux.

Sigmoid software components are available through www.igb.uci.edu/servers/sb.html or www.sigmoid.org.

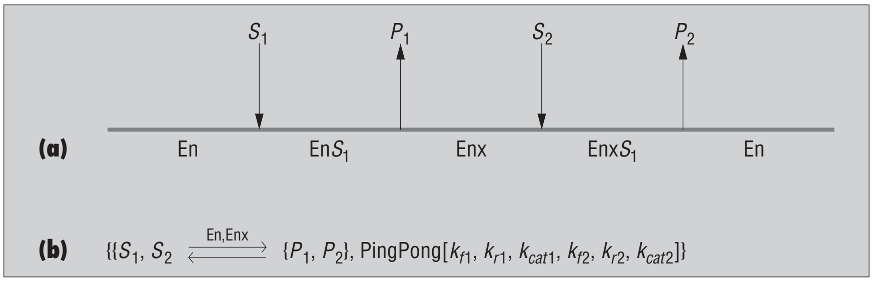

S_1
P_1
S_2
P_2
(a)
En
EnS_1
Enx
EnxS_1
En
(b)
$\{\{S_1, S_2 \xrightleftharpoons{\text{En,Enx}} \{P_1, P_2\}, \text{PingPong}[k_{f1}, k_{r1}, k_{cat1}, k_{f2}, k_{r2}, k_{cat2}]\}$

Figure 4. (a) A representation of the ping-pong bi-bi mechanism of an enzyme's action upon substrate (input) molecules S_1 and S_2 to create product (output) molecules P_1 and P_2. (b) The input syntactic representation for this ping-pong bi-bi model in kMech. S_1 and S_2 are substrates; P_1 and P_2 are products; En is the free enzyme; Enx is the modified enzyme intermediate; {…} delimits ordered pairs or n-tuples; k_{f1} and k_{f2} are rate constants of the enzyme-substrate associations for S_1 and S_2, respectively; k_{r1} and k_{r2} are rate constants of the enzyme substrate dissociations for S_1 and S_2, respectively; and k_{cat1} and k_{cat2} are the catalytic rate constants for the formation of products P_1 and P_2, respectively.

acts with molecular species B to produce molecular species C according to the mass action kinetic law expressed by the differential equation $dC/dt = kAB$, where the rate of C's production is proportional to the product of the concentration of reactants A and B.

This isn't a problem of numerical analysis—several packages permit solving fairly large systems of such equations—but rather one of model management and scalability. So, we selected Mathematica, a symbolic-mathematical-language and numerical solver based on computer-algebra objects and a rich set of well-implemented mathematical operations. Indeed, we implemented Cellerator as a Mathematica notebook, designed to facilitate biological modeling via automated equation generation.[4]

We've implemented many molecular-interaction models in Cellerator using various formalisms. These include differential equations or stochastic molecular simulation formalism and range from the law of mass action and simple Michaelis-Menten models to more complex models of enzyme reactions (such as the Monod-Wyman-Changeaux model for allosteric enzymes) and gene regulation.[6] Cellerator reaction models continue to proliferate, along with a library of actual pathway models comprising sets of coordinated reactions, with parameters derived from the literature whenever possible. Useful models for studying signaling and cancer, for instance, include G-coupled protein receptor activation, phosphoinositol metabolism reproducing an observed peak in membrane-bound PIP3 (phosphatidyl inositol-3,4,5-triphosphate), MAPK (Mitogen Activated Protein Kinase) cascade[7,8] with feedback and possible oscillations, NFκB pathway[9] with feedback and observed oscillations, and published models of yeast cell cycle checkpoints.

kMech (kinetic Mechanisms) is an extended set of enzyme mechanism models for single- and multisubstrate, positively and negatively regulated, and allosteric enzymes.[10] To illustrate how Cellerator works, let's consider the kMech model of the synthesis of the amino acids leucine, isoleucine, and valine within the bacterium *E. coli*.[10] An important reaction in that pathway is the *ping-pong bi-bi* mechanism of the enzyme α-acetohydroxyacid synthase, which catalyzes the condensation of one pyruvate molecule and one α-ketobutyrate molecule to form one α-aceto-α-hydroxybutyrate molecule.

This enzyme's substrates bind in an ordered fashion: a pyruvate molecule must bind to the enzyme, react with a thiamine pyrophosphate cofactor to form an active acetaldehyde group (CH_3CO), and release the first product, CO_2,

Table 1. Elementary Cellerator Arrows.

Cellerator arrow	Name of Reaction, differential equations	Typical biochemical notation
$S \to P$	Conversion: $\frac{dP}{dt} = kS, \frac{dS}{dt} = -kS$	$S \xrightarrow{k} P$
$A + B \to C$	Unidirectional reaction: $\frac{dA}{dt} = -kAB,$ $\frac{dB}{dt} = -kAB, \frac{dC}{dt} = kAB$	$A + B \xrightarrow{k} C$
$A + B^n \to C$	Unidirectional reaction with cooperativity *n* (*n* must be an integer): $\frac{dA}{dt} = \frac{dB}{dt} = -kAB^n = -\frac{dC}{dt}$	$A + nB \xrightarrow{k} C$
$A + B \leftrightarrow C$	Bidirectional reaction: $\frac{dA}{dt} = \frac{dB}{dt} = -\frac{dC}{dt} = -k_f AB + k_r C$	$A + B \underset{k_r}{\overset{k_f}{\rightleftarrows}} C$
$\varnothing \to A$	Creation: $\frac{dA}{dt} = k$	$\xrightarrow{k} A$
$A \to \varnothing$	Annihilation: $\frac{dA}{dt} = -kA$	$A \xrightarrow{k}$
$S \overset{E}{\rightleftarrows} P$	Enzymatic (catalytic) reaction: $\frac{dS}{dt} = -k_f SE + k_r X, \frac{dP}{dt} = kX,$ $\frac{dX}{dt} = -\frac{dE}{dt} = k_f SE - (k + k_r)X$	$S + E \underset{k_r}{\overset{k_f}{\rightleftarrows}} X \xrightarrow{k} E + P$

before the second substrate, α-ketobutyrate, can bind to the enzyme. Next, the enzyme-bound active acetaldehyde group must be transferred to the second substrate, α-ketobutyrate, to form the second product, α-aceto-α-hydroxybutyrate, in order to release the free enzyme. This is a bi-bi mechanism because it has two substrates and two products. It is a ping-pong mechanism because the enzyme shuttles between free and substrate-modified intermediate states (see figure 4).

kMech parses this input into basic association-dissociation reactions (in this case, two single-substrate single-product reactions) that are readable by Cellerator, which then translates them to differential equations and variable definitions using mass-action kinetics. Cellerator then solves these equations by invoking Mathematica's numeric solver function (NDSolve), and generates time plots, phase plots, numerical datasets, or other requested output forms. The Sigmoid Pathway Modeling Database stores this enzyme mechanism's parameters.

So, as table 1 shows, Cellerator converts symbolic reactions to mathematical equations and solves these using Mathematica. As we'll see shortly, it then sends the simulation results to the middleware controller. We can also write Cellerator models in SBML (Systems Biology Markup Language, an XML-based protocol for systems biology information interchange) for exchange with other cell simulation systems.

The Sigmoid Pathway Database

We drew upon several existing open-source projects to implement the Sigmoid Pathway Database. This lightened the core software development load considerably, so we could focus on tying the components together to produce the specific software products we needed. PostgreSQL, a leading open source database, provides the database implementation. A UML schema (see figure 5) defines the pathway model database with four major class hierarchies: Reactions, Reactants, Models, and Knowledge Sources. Reactions use Reactants for their products, substrates, and enzymes, and parameterized Reactions compose Models. These three class hierarchies use Knowledge Sources to reference external information about themselves.

We built the initial Sigmoid database by hand to get started but also sought a method to automatically transform the class descriptions in the high-level UML diagram into a set of instantiable objects upon which we could build applications. Our solution uses the open-source software AXgen to read UML diagrams and provide an API to the

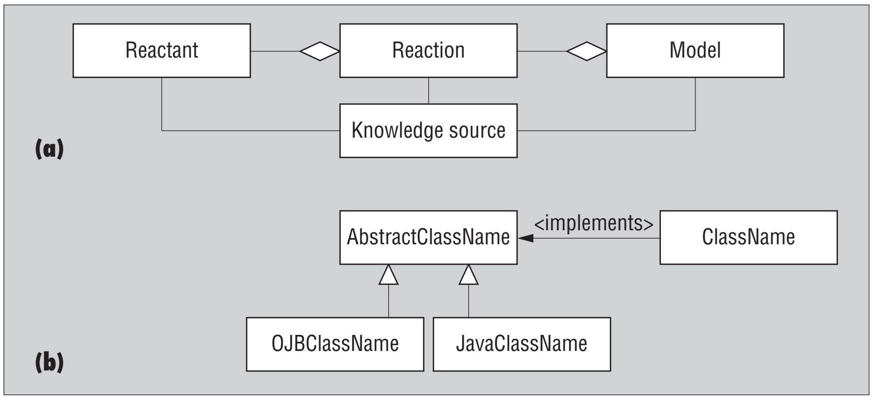


Figure 5. (a) High-level interaction of the Sigmoid schema's four main class hierarchies and (b) implementation classes derived from a single UML class, "ClassName."

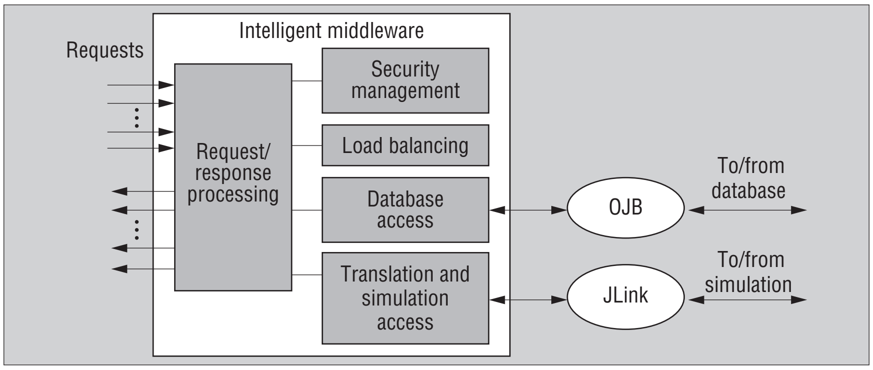


Figure 6. The Sigmoid intelligent-middleware structure. The request-response processing module receives requests from and returns results to the client. The database access and translation and simulation access modules interface with their corresponding back-end components via OJB and JLink, respectively.

diagram's structure, thus permitting autogeneration of software components from a master UML diagram. AXgen has interfaces to both the Novosoft UML library (nsuml) and the NetBeans MetaData Repository (MDR) library, so we can use one tool to read a much wider variety of UML than would otherwise be possible. Once a UML diagram is loaded, a set of Java classes is generated for each corresponding UML class.

Apache Velocity Template Language simplified the process of generating classes by letting us create templates that interact with live Java code. In addition to the Java object class hierarchy, the auto-generation framework creates the necessary auxiliary files. In Sigmoid's current implementation, this includes SQL files, creating a database for the schema defined in the master UML diagram and a mapping from the generated Java classes to the database. This mapping uses OJB, an open-source XML-based object relational bridge. Eventually, we may also be able to autogenerate user interface widgets for each class.

So, we replaced the entire Sigmoid schema and hand-coded database of over 100 relations with autogenerated, functionally equivalent code. Throughout this process, the 100-table Sigmoid database schema (largely a hierarchy of biological reaction and reactant types) functioned as a remotely accessible database. We've also successfully populated Sigmoid's generative version with several existing models—for example, of the MAPK and NFκB pathways—and with selected metabolic pathways from *E. coli* for biosynthesis of branched-chain amino acids. These models contain parameter sets and initial condition sets that engender qualitatively different behaviors, such as oscillations' presence or absence. Populator programs currently under development will import data from other sources (including SBML files and other databases) into Sigmoid. This will increase Sigmoid's power considerably by capturing diverse community input and making it available to end-user biologists in an integrated manner.

Intelligent middleware for scalable services

We built an intelligent, distributed Web middleware layer to access the Sigmoid database and translate reaction sets into Cellerator's input language. The middleware then calls Cellerator for model generation and simulation and receives output data and plots in response, which it passes to the GUI. Although it requires some development and maintenance effort, this layer provides a gateway between Sigmoid's front and back ends that allows each to evolve independently as long as it maintains its middleware interface. Most important, we can scale the middleware to serve more simultaneous users simply by increasing the computational and database server resources.

Large-scale distributed systems use several middleware technologies including SOAP, Common Object Request Broker Architecture (CORBA), Java Remote Method Invocation (RMI), and the Windows Distributed Component Object Model (DCOM). We used Apache SOAP 2.3.1, hosted by Tomcat Application Server 4.1.27, to implement Sigmoid's middleware. SOAP's advantages include openness, simplicity, seamless and natural integration with widely accessible Internet infrastructures, and full language and platform independence. These features meet our requirements for biological pathway modeling. Because SOAP is built on cross-platform technology standards such as HTTP and XML and uses lightweight XML to encode and transmit data between applications through HTTP port 80, it enables cross-platform, cross-language communication over the Internet and through firewalls.

We created and integrated the following middleware services, shown in figure 6:

- The *request-response processing* module interfaces with the client, dispatching its requests to the database or simulation engines and returning the responses to the client. Client applications can call this module via SOAP-based RPC (Remote Procedure Call) as if calling a local function.
- The *security management* module checks the client's identity and password.
- The *load balancing* module assigns client requests evenly to different database or computation servers.
- The *database access* module fetches data from and stores data in the Sigmoid database via OJB. OJB provides an object-oriented interface to the relational database, mapping

between the Sigmoid object and relational database schemas.

- The *translation and simulation access* module translates biological pathway objects into Cellerator text-encoded commands and sends them via JLink to Mathematica's model generation-simulation engine. (JLink is a Mathematica add-on that lets Java applications call Mathematica functions.)

These modules are exposed as Web services that Java application programs and other clients can access.

Available Java Web services are expanding steadily and currently include **getModel**, **updateModel**, **saveModel**, **getModelList**, **simulateModel**, **simulateModelTime**, **simulateModelPhase**, and **simulateModelTransfer**. These can also serve as open APIs for third-party client programs to communicate with Sigmoid's back-end knowledge database and simulation engines.

The Sigmoid Model Explorer interface

The last system component to be initiated, and the most recent to achieve demonstration-level functionality, is the SME Web-compatible GUI. This Java application incorporates several useful elements that support browsing and selecting objects from the model database, editing objects and their attributes including numerical parameters, and displaying and editing network layouts:

1. The Tree Viewer displays biological objects, such as pathways, reactions, and their constituent reactants and parameters, in a compositional hierarchy (see figure 7a, left panel).
2. A biological-network layout view uses JGraph libraries for semiautomated layout of small circuits (see figure 7a, center panel), with a window that lets users edit and save layouts as bipartite labeled graphs with user-definable mapping of object types to icons (see figure 7a, far right panel).
3. Alternative displays for the left panel include a folder hierarchy of displayable simulation results.
4. An optional output previewer permits side-by-side comparison of multiple plots from multiple simulations (see figure 7b).
5. A full output viewer (using JAI, Java Advanced Imaging, for good image quality) allows annotation of simulation outputs for a primitive scientific

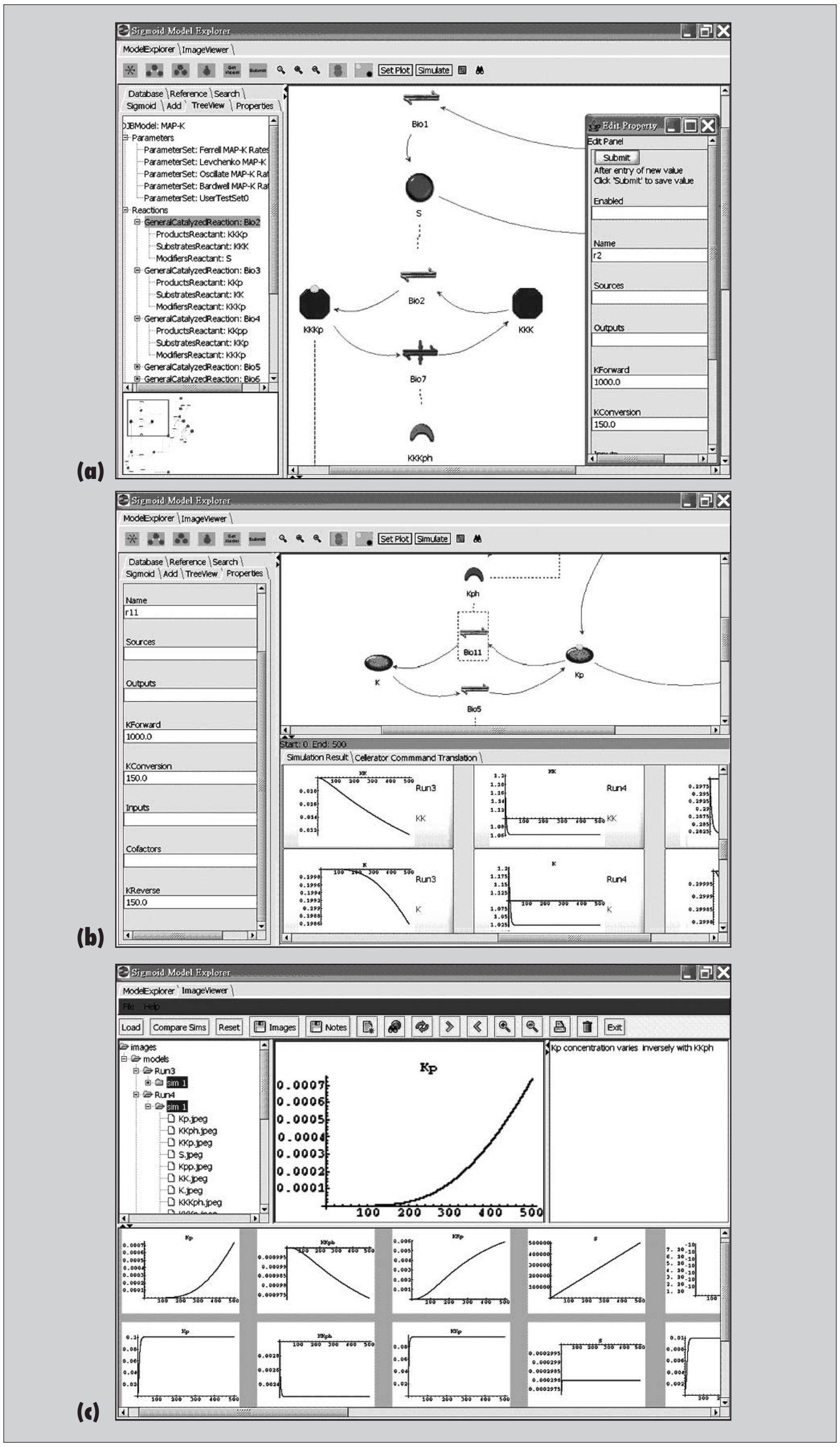
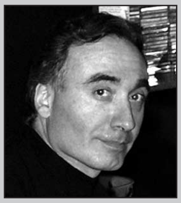


Figure 7. These views of the Sigmoid Model Explorer display a portion of the MAPK pathway. (a) Three panels show, from left to right, the tree view of compositional hierarchy, a network layout visualization, and the parameter-editing panel. Along the top are various action buttons for saving and running the model, and for switching the main panel to view output plots. The user can select reaction icons. (b) The optional output plot preview panel permits comparison of plots from multiple simulations. (c) The full output display mode allows annotation.

The Authors

Jianlin Cheng is a PhD candidate in computer science at the Institute for Genomics and Bioinformatics at the University of California, Irvine. His research interests include algorithms and applications for bioinformatics, systems biology, and machine learning. He has an MS in computer science from Utah State University. Contact him at the School of Information and Computer Sciences, Univ. of California Irvine, Irvine, CA 92697-3425; jianlinc@uci.edu; www.ics.uci.edu/~jianlinc.

Lucas Scharenbroich is a graduate student in the School of Information and Computer Science at the University of California, Irvine. He is also affiliated with the Machine Learning Systems group at NASA's Jet Propulsion Laboratory. His research interests include probabilistic modeling and its application to bioinformatics, remote sensing, and planetary science. He has a BSEE from the University of Minnesota, Duluth. Contact him at the School of Information and Computer Sciences, UCI, Irvine, CA 92697-3425; ischaren@uci.edu; www.ics.uci.edu/~lscharen.



Pierre Baldi is a professor in the School of Information and Computer Science and the Department of Biological Chemistry at the University of California, Irvine. He is also director of UCI's Institute for Genomics and Bioinformatics and is a layer leader at CalIT2, the California Institute for Telecommunications and Information Technology. His research focuses on AI, machine learning, and bioinformatics. He received a PhD in mathematics from the California Institute of Technology. Awards include the 1993 Lew Allen Award at the Jet Propulsion Laboratory and a Laurel Wilkening Faculty Innovation Award at UCI. Contact him at the Inst. for Genomics and Bioinformatics, School of Information and Computer Sciences, UCI, Irvine, CA 92697-3425; pfbaldi@ics.uci.edu; www.ics.uci.edu/~pfbaldi.

Eric Mjolsness is Systems Biology Program Leader for the Institute for Genomics and Bioinformatics and a professor in the Computer Science Department at the University of California, Irvine. His research interests include constructing scientific inference systems and applying techniques from machine learning, pattern recognition, image understanding, nonlinear optimization, and statistical physics to computational biology and other sciences. He received his PhD in physics from the California Institute of Technology. Contact him at Inst. for Genomics and Bioinformatics, School of Information and Computer Sciences, UCI, Irvine, CA 92697-3425; emj@uci.edu; www.ics.uci.edu/~emj.

notebook, with EPS publication-quality output (see figure 7c).
6. User-selectable JPEG/GIF icon sets display biological object types.

SME also provides back-end connections to the middleware, including database and simulator access, and uses Java reflection to discover the current Sigmoid object hierarchy and database schema for displayable objects and model-editing methods. These features let users

- visualize, design, edit, and store pathway models, parameters, and initial conditions and their properties;
- simulate the models by calling the simulator through the middleware; and
- view and compare simulated models' properties—for instance, by viewing the temporal evolution of chemical species' concentration under different conditions.

Users can download and automatically update SME via the Web. It runs from any Web browser as a Java Web Start or local client program and enables simulations to execute remotely, and other software platforms besides SME can use Sigmoid through its Web services.

Refining the infrastructure

We have made available initial versions of the main components providing an infrastructure that already yields biologically relevant results. For instance, the *E. coli* metabolic pathway model correctly predicts the effect of certain mutations, and the MAPK cascade model shows that, on the basis of the parameter sets and initial conditions chosen, it can generate a switch-like or graded input-output relationship, or even produce oscillatory behavior.

Ongoing development

Development and expansion of Sigmoid continues at all levels. Since the GUI mediates the user experience, it will attract the largest number of feature requests. Because the overall architecture is now functional, many of these requests can be met at reasonable levels of effort and cost. New back-end modules under development—for instance, to optimize models or learn from data—will enhance end-user capabilities.

We also need new reaction types in Cellerator to deal with various kinds of (nontranscriptional) feedback. New reaction types—for example, those used in modeling metabolic pathways (such as Generated Monod Wyman Changeux [GMWC])—will need to be exposed to model new pathways beyond MAPK and NFκB. Scaling Sigmoid up will require expert evaluation of allowed and suggested mappings from biological reaction mechanisms to mathematical reaction models.

With such powerful tools, the models created in Sigmoid will be of unique value to biologists. Within Sigmoid such models can guide the design of scientific experiments that discriminate between alternative hypotheses. Models will be exportable, using SBML, to other cell simulation and pathway database environments so that they aren't confined to a particular software environment.

Likewise, we continue to expand and populate the Sigmoid database. Database populator codes import relevant data from other reaction and pathway data sources (depending on their accessibility to software agents), such as the Kyoto Encyclopedia of Genes and Genomes (KEGG), SiBML/GeneNet, Cytoscape, Reactome, the Saccharomyces Genome Database, BioCyc, and BioModels. These connections can use a systems biology communications infrastructure such as SBML, BioPAX, Systems Biology Workbench, BioSpice, Monod, and many others. These each have their own strengths and specializations. For example, the current versions of KEGG and BioPAX are better developed for metabolic pathways than for signal transduction pathways, Sigmoid's initial focus. We can increase standardization and interoperability by using SBML, Gene Ontology, and other de facto standards as they emerge.

Scalability issues

While it's sound practice to have several parallel efforts across multiple research groups, we believe Sigmoid's most distinctive advantage is its scalability. A simple pathway contains a dozen or more reactions that a set of about 50 elementary equation models can represent, taken from a growing library of reaction model types as in figure 4. To reverse-engineer cells, this number must scale up progressively to 500, 5,000, 50,000, and perhaps even 500,000 and beyond.

Clearly, this scale-up requires software that can help generate models of increasing size, complexity, and sophistication. Likewise, as our understanding of systems biology progresses, the schema we use to store models and biological information must evolve. The ability to automatically regenerate entire databases from a modified schema becomes essential. Generative tools, such as autogeneration of the database and compatible object APIs from a UML schema and of mathematical models from reaction notation using computer algebra, have already proven effective in curbing the cost of adding new features to the Sigmoid architecture.

These principles of self-generation and scalability lie at the heart of the Sigmoid architecture, and database software engineering might benefit generally from autogeneration of object-relational database code from UML. We are also exploring whether fundamental software classes for nodes, networks, and communications can be reused across different networks, from the biological reaction networks we wish to simulate to the machine learning networks we use to reverse-engineer them.[11] In short, can generative network software classes enable significant advances in intelligent systems design?

We've made encouraging progress, but several challenges remain. On the modeling side, we haven't yet found a complete mathematical formalism for scale-up. How should we handle and integrate multiple temporal and spatial scales over several orders of magnitude, and complex combinations of continuous, stochastic, and discrete events with different levels of compartmentalization? On the GUI side, network layout and editing scalability remain unresolved. Can we rapidly zoom up and down, from molecular reactions to cellular function? As research proceeds, we hope to answer these questions. ■

Acknowledgments

This work has been supported by NIH grants GM069013 and T15LM007443, NSF grant EIA-0321390, a Laurel Wilkening faculty innovation award, a UC Systemwide Biotechnology Research and Education Program #2002-06 award, NCI Director's Challenge support to Children's Hospital of Los Angeles, NASA Intelligent Systems Program funding, and the Institute for Genomics and Bioinformatics at UCI. We thank students, programmers, and colleagues who provided us with valuable feedback or helped implement particular components of the infrastructure. These include Lee Bardwell, Ben Bornstein, Ben Compani, G. Wesley Hatfield, Peter Hebden, Andre Levchenko, Elliot Myerowitz, Doug Molina, Kirill Petrov, Sonia Ralaivola, Bruce Shapiro, Trent Su, Diane Trout, Barbara Wold, and Chin-Ran Yang.

References

1. T. Ideker et al., "Integrated Genomic and Proteomic Analysis of a Systematically Perturbed Metabolic Network," *Science*, vol. 292, 2001, pp. 929–934.
2. P. Baldi and G. Wesley Hatfield, *DNA Microarrays and Gene Regulation: From Experiments to Data Analysis and Modeling*, Cambridge Univ. Press, 2002.
3. E. Mjolsness and D. DeCoste, "Machine Learning for Science: State of the Art and Future Prospects," *Science*, vol. 293, 14 Sept. 2001, pp. 2051–2055.
4. B.E. Shapiro et al., "Cellerator: Extending a Computer Algebra System to Include Biochemical Arrows for Signal Transduction Simulations," *Bioinformatics*, vol. 19, no. 5, 2003, pp. 677–678.
5. G. Booch, I. Jacobson, and J. Rumbaugh, *The Unified Modeling Language User Guide*, Addison-Wesley, 1998; www.uml.org.
6. I.H. Segel, *Enzyme Kinetics*, John Wiley and Sons, 1993.
7. B.E. Shapiro, A. Levchenko, and E.D. Mjolsness, "Automatic Model Generation for Signal Transduction with Applications to Map Kinase Pathways," *Foundations of Systems Biology*, H. Kitano, ed., MIT Press, 2002, pp. 145–162.
8. L. Bardwell, "A Walk-through of the Yeast Mating Pheromone Response Pathway, *Peptides*, vol. 25, no. 9, Sept. 2004, pp. 1465–1476.
9. A. Hoffmann et al., "The IκB-NFκB Signaling Module: Temporal Control and Selective Gene Activation," *Science*, vol. 298, 2002, pp. 1241–1245.
10. C.R. Yang et al., "An Enzyme Mechanism Language for the Mathematical Modeling of Metabolic Pathways," *Bioinformatics*, vol. 21, no. 6, March 2005, pp. 774–780.
11. P. Baldi, Y. Chauvin, and V. Mittal Henkle, "Software Foundation Libraries for the Design of Intelligent Systems," *Proc. 10th Italian Workshop Neural Nets*, Springer-Verlag, 1998, pp. 17–39.