

Sigmoid: Towards an Intelligent, Scalable, Software Infrastructure for Pathway Bioinformatics and Systems Biology

Jianlin Cheng^{1,2}, Lucas Scharenbroich^{1,2}, Pierre Baldi^{1,2,3}, and Eric Mjolsness^{1,2},

1. Institute for Genomics and Bioinformatics, University of California, Irvine.
2. Department of Computer Science, Donald Bren School of Information and Computer Sciences, University of California, Irvine.
3. Department of Biological Chemistry, University of California, Irvine.

Abstract

We describe a generative, scalable software infrastructure for pathway bioinformatics and systems biology. The Sigmoid modeling system is a three-tier architecture comprising distributed modules that implement pathway/cell model generation and simulation (Cellerator), a pathway modeling database (Sigmoid proper), a Web service-oriented middleware, a graphical user interface, and in the future, parameter optimization and other datamining technologies. Key to the design of the infrastructure is its scalability ensured by leveraging symbolic computer algebra and self-generation of database and other code from high-level representations such as UML schema. All Sigmoid modeling software components are available through: <http://www.igb.uci.edu/servers/sb.html>.

1. Introduction

Although there are many kinds and levels of biological systems, such as immune systems, nervous systems, and ecosystems, the expression “systems biology” is used today mostly to describe attempts at unraveling molecular systems, above the traditional level of single genes and single proteins, focusing on the level of pathways and groups of pathways in a cell. A basic methodological roadmap for systems biology has been outlined by [Ideker et al. 2001] and consists of the following three idealized steps (see also Chapter 8 in Baldi and Hatfield 2002, and references therein):

1. Identify all the players of the system, that is all the components that are involved (e.g. genes, proteins, compartments);
2. Perturb each component through a series of genetic or environmental manipulations and record the global response using high-throughput technologies (e.g. microarrays);
3. Build a global model and generate new testable hypothesis. Return to 2, and in some cases to 1 when missing components are discovered.

Here we describe Sigmoid--a generative, scalable software infrastructure for systems biology to facilitate the third step above. Sigmoid supports the process of cycling between model building, hypothesis generation, and biological experimentation and data gathering, by integrating the hypothesis and discovery phases in the research process (Figure 1).

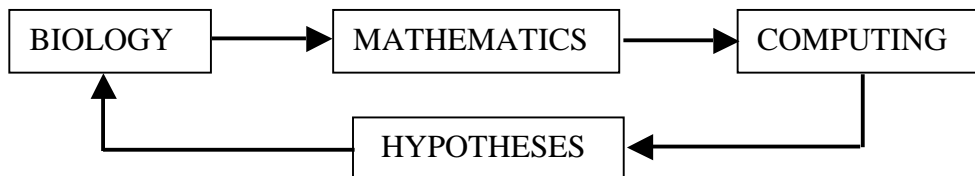


Figure 1: Basic systems biology inference cycle.

Scalability of the software architecture is an essential and pervasive intelligence requirement given the underlying complexity of biological systems brought about by evolutionary tinkering and a large number of components and modules operating at multiple spatial and temporal scales. The scalability must be reflected in each component of the infrastructure.

In particular, we address the problem of creating an expert assistance system for modeling biological pathways, using current software technology to decrease the difficulty and cost of creating the system. The reason for building such a system 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. While the primary focus of the infrastructure is reverse-engineering biological circuits, in the long-run we expect it to become applicable also to bio-engineering projects, that is for the more or less *de novo* design of complex sets of molecular interactions with a particular computational, biomedical, or bio-synthetic focus. Thus the situation is also reminiscent of the early stages of development of CAD tools and tools for modeling and designing VLSI and other circuits with some important differences that will be highlighted below.

2. Overview of the Software Infrastructure

The Sigmoid modeling system consists of distributed modules implementing pathway/cell model generation and simulation (Cellerator [Shapiro et al. 2003]), a pathway modeling database, a Web service-oriented middleware, a graphical user interface friendly to a biologist user, and in the future, parameter optimization and other datamining technologies (Figure 2).

We have taken an innovative approach to coordinating the development of various software modules in Sigmoid by using the Universal Modeling Language (UML) [Booch et al. 1998] to diagram the most important biological objects--notably reactions and molecular reactants--and their relationships, in consultation between bioinformaticians and biologists. This UML diagram then actually becomes source code from which several parts of our system are automatically generated, in particular a realization of the Sigmoid pathway modeling database (in Structured Query Language, SQL) and the corresponding Java object hierarchy along with support files for facilitating the object-relational mapping and end-user documentation. Also the Graphical User Interface (GUI) uses "reflection" to automatically discover much of what it needs to know about the Sigmoid schema. Thus there is a guarantee that the software actually implements something very close to the agreed-upon biological objects.

At the software implementation level, this results in classical three-tier architecture with a back-end, middle layer, and front-end depicted in Figure 3. The back-end includes the database, the simulator, and other model manipulators. The GUI front-end does not access the back-end modules directly but rather via a Web service middleware module. The small development overhead introduced by the middle layer is more than compensated by the advantages in terms of distributed computing, performance, flexibility, and scalability. The middleware layer brokers all communications between the GUI and the back-end components and also between the backend components themselves. This infrastructure was created in close collaboration with biologists by having the design of many of the essential software objects and their relationships be visible to them as implementation proceeded.

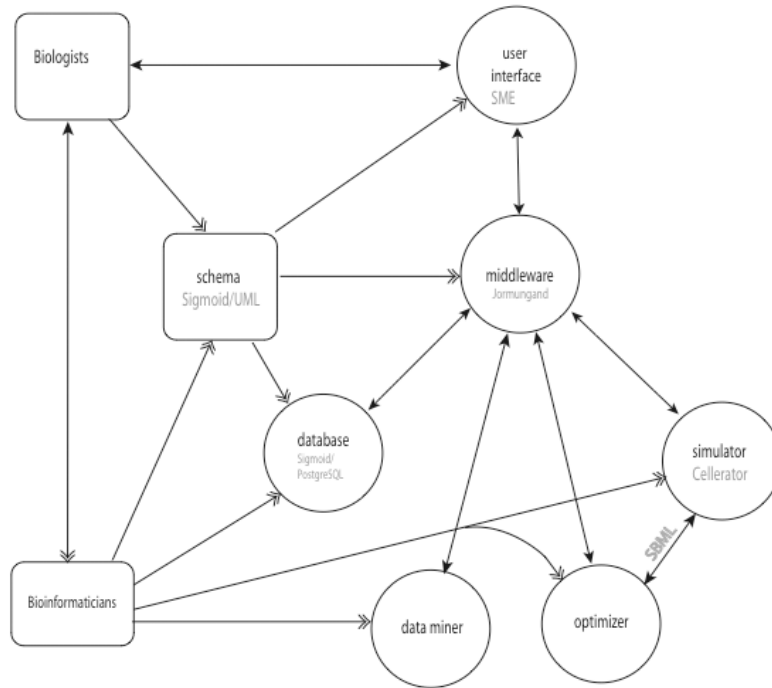


Figure 2. Sigmoid architecture, top-level view. Separation of modules into a communicating distributed system increases scalability of the architecture. Our “simulator” is the Cellerator model generator/simulator; “database” is Sigmoid (autogenerated from UML schema not shown); “user interface” is Sigmoid Model Explorer (SME). Solid arrows: runtime communication. Open double arrows: design/compile time communication.

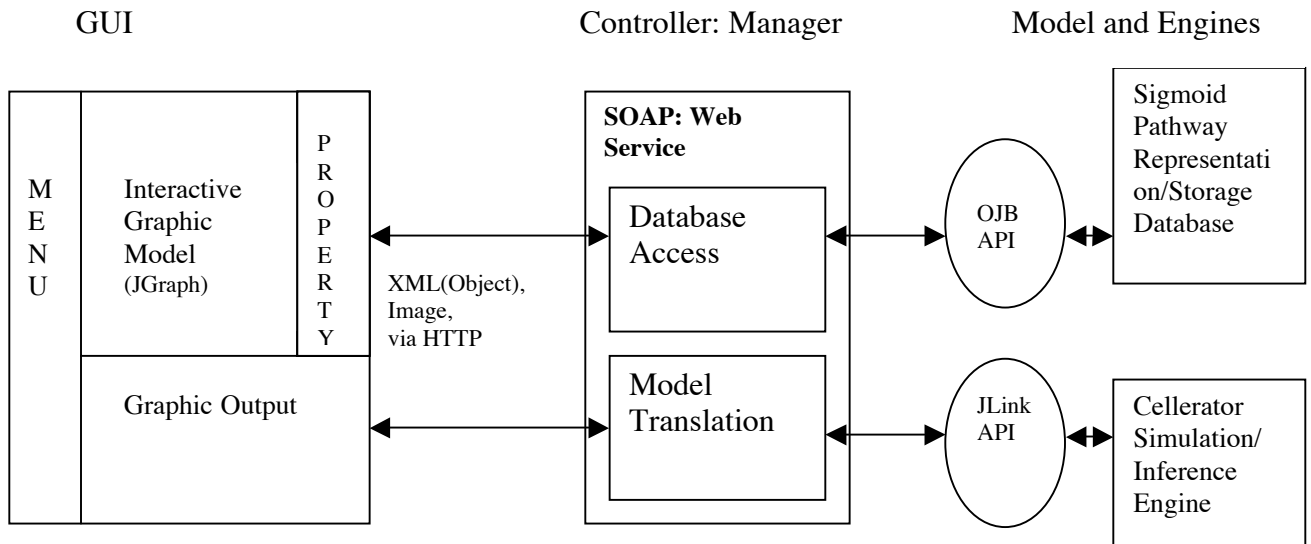


Figure 3: The three-tier architecture.

To keep the infrastructure flexible and manageable as it grows, we resorted to an approach that may be called “generative” in that it seeks to partially automate the generation of both executable code and mathematical models, within as many of the modules in Figure 3 as possible, starting from high-level inputs such as UML diagrams and reaction notations understandable to non-computer scientists. We will now briefly describe the various components of this generative infrastructure, its main features, requirements, and current state of development. While we are developing the various components of this architecture together, it is important to notice that some components are more mature than others and that individual components which are more mature, such as the database or the simulator, are self-sufficient and can be used independently of the GUI or the middleware.

In overview, the main software, languages, and tools that are used in the architecture include (see [URLs] in the References):

- Front-end GUI: Java, Java reflection, JGraph, HyperText Markup Language (HTML), JavaScript, Extensible Markup Language (XML), Java Web Start, Web browser;
- Middleware: Java, Apache Simple Object Access Protocol (SOAP), Java Servlet, Java Server Pages (JSP), XML, Apache Webserver, Tomcat, Object Relational Bridge (OBJ), JLink;
- Back-end solver: JLink, Mathematica, Cellerator, Systems Biology Markup Language (SBML);
- Back-end database: UML, Anything from XMI Generator (AXgen), PostgreSQL, OBJ, XML, Velocity Template Language (VTL) and Linux.

We use publicly available open source, tools as much as possible. Sigmoid software components are available through: <http://www.igb.uci.edu/servers/sb.html> .

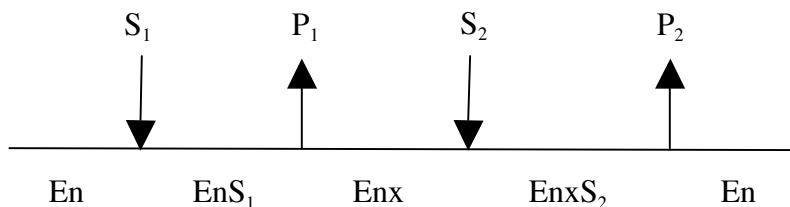
3. Model Generation and Simulation: Cellerator

Simulating a biological pathway often involves simulating dozens if not hundreds or thousands of elementary chemical reactions. Regardless of the details of the equations (typically differential equations) used to model an individual reaction, building a model containing a large numbers of reactions is a tedious and error-prone process if to be performed more or less manually. Note that unlike electronic circuits, such as those found in a television or a computer, and comprising only a small number of elementary building blocks, chemical reactions in biology come in a large variety of elementary forms. What is needed therefore is to build a library of re-usable reaction models that can be expressed in a simple, higher-level language, specifying the molecular species and the type of reaction. For example one can use syntax similar to “ $A+B \rightarrow C$; mass action with rate k ” to specify that molecular species A interacts with molecular species B to produce molecular species C according to the mass action kinetic law expressed by the differential equation $dC/dt=kAB$, whereby the rate of production of C is proportional to the product of the concentration of the reactants A and B . The primary problem is not a problem of numerical analysis—there are several packages that can be used to solve fairly large systems of such equations. The primary problem is a problem of model management and scalability. This problem is best addressed by using a symbolic mathematical language

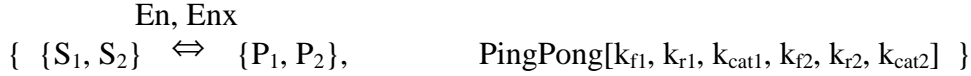
and numerical solver, such as Mathematica, based on computer- algebra objects and a very rich set of well-implemented mathematical operations. Indeed, Cellerator [Shapiro et al. 2003] is implemented as a Mathematica notebook and is designed to facilitate biological modeling via automated equation generation.

Many models of molecular interactions have been implemented in Cellerator using different formalisms, such as differential equations or stochastic molecular simulation formalism and ranging from the law of mass action and simple Michaelis-Menten models to more complex models of enzyme reactions (e.g. the Monod-Wyman-Changeaux or MWC model for allosteric enzymes) and gene regulation (Segel 1993). The list of Cellerator reaction models continues to expand, along with a library of actual pathway models comprising sets of coordinated reactions with parameters derived from the literature whenever possible. Existing models that are particularly relevant, for instance, for studying signaling and cancer include: G-coupled protein receptor activation, phosphoinositol metabolism reproducing an observed peak in membrane-bound PIP3, MAPK cascade (Shapiro et al. 2002, Bardwell 2004) with feedback and possible oscillations, NFkB pathway (Hoffmann 2002) with feedback and observed oscillations, and published models of yeast cell cycle checkpoints. An extended set of enzyme mechanism models for single and multi-substrate, positively and negatively regulated and allosteric enzymes, called kMech, has been written for Cellerator (Yang et al., 2004). Many other models have been expressed in Cellerator as well.

To illustrate Cellerator utility concretely, consider the kMech model for the synthesis of the amino acids leucine, isoleucine and valine within the bacterium *E. coli* [Yang et al. 2004]. One important reaction in that pathway is the “ping-pong bi-bi” enzyme mechanism of the enzyme α -aceto-hydroxyacid synthase, an enzyme that catalyzes the condensation of one molecule of pyruvate and one molecule of α -ketobutyrate to form one molecule of α -aceto- α -hydroxybutyrate. The substrates of this enzyme bind in an ordered fashion. That is, 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 , 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 a free and a substrate-modified intermediate state.



A kMech input syntactic representation for this Ping Pong Bi Bi model is



where 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; $\{ \dots \}$ delimits ordered pairs or n-tuples; k_{f1} and k_{f2} are

Elementary Cellerator Arrows

Cellerator Arrow	Name of Reaction, Differential Equations	Typical Biochemical Notation
$S \rightarrow P$	Conversion $\frac{dP}{dt} = kS, \quad \frac{dS}{dt} = -kS$	$S \xrightarrow{k} P$
$A + B \rightarrow C$	Unidirectional Reaction $\frac{dA}{dt} = -kAB, \quad \frac{dB}{dt} = -kAB, \quad \frac{dC}{dt} = kAB$	$A + B \xrightarrow{k} C$
$A + B^n \rightarrow C$	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 $\frac{dA}{dt} = \frac{dB}{dt} = -\frac{dC}{dt} = -k_f AB + k_r C,$	$A + B \xrightleftharpoons[k_r]{k_f} C$
$\emptyset \rightarrow A$	Creation: $\frac{dA}{dt} = k$	$\xrightarrow{k} A$
$A \rightarrow \emptyset$	Annihilation: $\frac{dA}{dt} = -kA$	$A \xrightarrow{k}$
$S \xrightleftharpoons{E} P$	Enzymatic (Catalytic) Reaction $\frac{dS}{dt} = -k_f SE + k_r X, \quad \frac{dP}{dt} = kX, \quad \frac{dX}{dt} = -\frac{dE}{dt} = k_f SE - (k + k_r)X$	$S + E \xrightleftharpoons[k_r]{k_f} X \xrightarrow{k} E + P$

Figure 4: A small subset of Cellerator reaction models and conversion from symbolic representation to differential equations.

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; k_{cat1} and k_{cat2} are the catalytic rate constants for the formation of products P_1 and P_2 , respectively. kMech interprets this input and parses it into basic association-dissociation reactions (in this case, two single-substrate single-product reactions) defined in Cellerator, which Cellerator then translates to differential equations using mass action kinetics. The resulting differential equations and variable definitions are passed to Mathematica where they are solved by the numeric solver function (NDSolve) and time plots are generated. The parameters for this enzyme mechanism are stored in the Sigmoid Pathways Database. In short, Cellerator converts symbolic reactions to mathematical equations, and solves the corresponding equations. As we shall see, the results of simulations are then sent to the middleware controller. In addition, Cellerator models can be written out in SBML for exchange with other cell simulation systems.

4. Sigmoid Pathway Database

The pathway model database is defined by a UML schema, depicted in Figure 5, with four major class hierarchies: Reactions, Reactants, Models, and Knowledge Sources. Reactions utilize Reactants for their products, substrates, and enzymes, Models are composed of parameterized Reactions, and these three class hierarchies utilize Knowledge Sources in order to reference external information about themselves. While initial versions of the Sigmoid database were implemented by hand, we wished to automatically transform the class descriptions contained in the high-level UML diagram of this hierarchy into a set of instantiable objects upon which applications may be built. Our current approach to the process of auto-generating software components from a master UML diagram relies on the capabilities of several existing open-source projects. These pre-existing projects remove much of the core software development responsibilities and allow us to focus on tying them together to produce the specific software products needed for our own use.

We regenerated 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) was continually functioning as a remotely-accessible database. Object-relational database code autogeneration from UML is itself a contribution of potentially general interest in database software engineering. The current version of Sigmoid is implemented using PostgreSQL the main OpenSource database software.

In more detail, we currently use the AXgen, (<http://axgen.sourceforge.net/>) open-source tool for reading UML diagrams and providing an Application Programming Interface (API) to access the diagram's structure. AXgen provides interfaces to both the Novosoft UML library (nsuml) as well as the NetBeans MDR library. This allows us to use one tool to read a much wider variety of UML than we would be able to otherwise. The AXgen API also provides many convenience functions for the process of auto-generating code from the UML. Once a UML diagram is loaded, a set of Java classes are generated

for each corresponding UML class. As a spin-off we submitted new UML-interpretation features to the AXgen project to support field multiplicity as well as general code base improvements.

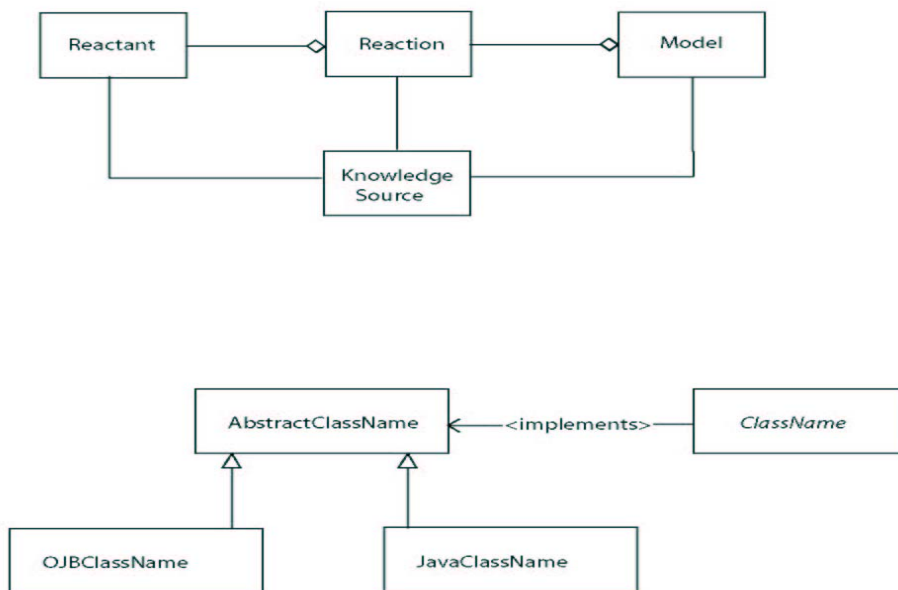


Figure 5: (a) High-level interaction of the four main class hierarchies of the Sigmoid schema (b) Implementation classes derived from a single UML class named “ClassName”.

The actual process of generating the various classes is simplified by leveraging the Apache Velocity project and its associated Velocity Template Language (VTL). VTL allows one to create templates that interact with live Java code. In addition to the Java object class hierarchy, the auto-generation framework is also responsible for generating any auxiliary files. In the current implementation, this encompasses the creation of SQL files which create a database for the schema defined in the master diagram as well as a mapping (using the open-source OJB XML-based “object relational bridge”, <http://db.apache.org/ojb/>) from the generated Java classes to the database. In the future, we may also be able to auto-generate UI widgets for each class.

The generative version of Sigmoid has been successfully populated with several existing models, including models of the MAPK and NFkB pathways, as well as selected reactions from of *E. coli* metabolic pathways for biosynthesis of branched chain amino acids. These models contain a variety of parameter sets and initial condition sets which give rise to qualitatively different behaviors including, for instance, the presence or absence of oscillations. In addition, we are developing “populator” programs which take data available from other sources and bring it into Sigmoid. This will considerably increase the power of Sigmoid by capturing community input from diverse sources and making it available to a biologist end-user in an integrated manner.

5. Intelligent Sigmoid Web Middleware for Distributed Computing and Web Services

A new distributed Web middleware layer was built which accesses the Sigmoid database and translates reaction sets into the input language of our Cellerator cell model generator, then calls Cellerator with requests for model generation and simulation and receives output plots in response. All these functions are exposed as Web services available to Java application programs and/or other clients. In addition to load balance and security management, the middleware provides a gateway between the front-end and the back-end of the architecture, allowing each one to evolve independently as long as the interface to the middleware is properly maintained. Furthermore, the middleware allows scalability in terms of the number of users that can be served simultaneously simply by increasing the computational and database server resources.

Several middleware technologies including SOAP, Common Object Request Broker Architecture (CORBA), Java Remote Method Invocation (RMI), and the Windows Distributed Component Object Model (DCOM) have been widely used in the development of large-scale distributed system. We use Apache SOAP (Simple Object Access Protocol)--the international standard for Web services ratified by W3C) version 2.3.1 hosted by Tomcat Application Server 4.1.27 as a framework to implement the intelligent middleware. The advantages of SOAP 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. SOAP is built on cross-platform technology standards such as HyperText Transfer Protocol (HTTP) and XML. It uses lightweight XML to encode data and transmits data between applications through HTTP port 80. This enables cross-platform, cross-language communication over the internet and through firewalls. The structure of the intelligent middleware, as we have implemented it, is shown in Figure 6.

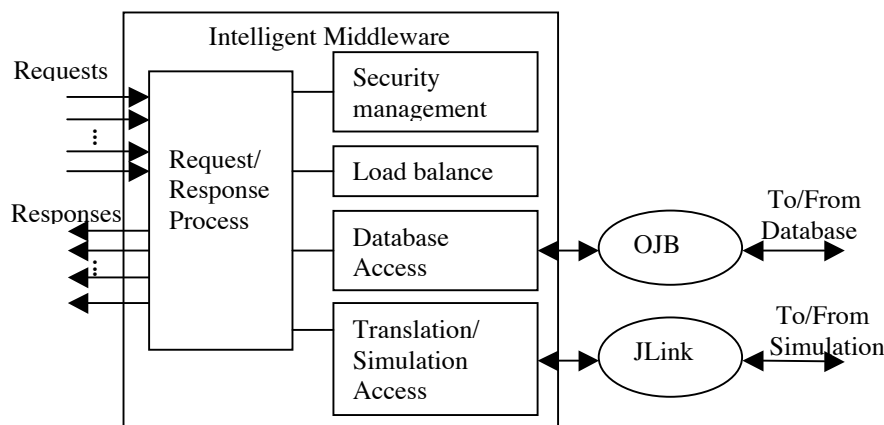


Figure 6: Structure of Sigmoid intelligent middleware.

In this Web middleware architecture, the request/response processing module is the Web service interface for client application. It dispatches the client's requests to database or simulation engines, and sends the responses to client. Client applications can call the Web services provided by this module via SOAP-based RPC (Remote Procedure Call) as if calling a local function. The security management module checks client's identity and password. The load balancing module assigns the client's request evenly to different servers of database or computation services. The database access module fetches data from and stores data in the Sigmoid database via OJB (Object Relational Bridge). OJB provides the mapping between the Sigmoid object schema and relational database schema, and provides an object-oriented interface to the relational database. The translation and simulation access module translates biological pathway objects into Cellerator text-encoded commands, which requires domain knowledge, and then sends them via JLink to the model generation/simulation engine running within Mathematica. JLink is a Java add-on to Mathematica that enables Java applications to call Mathematica functions. These interfaces and implementations were all created and integrated into the architecture (Figure 2).

The Web services currently implemented in Java are steadily growing and currently include: `getModel`, `getCelleratorModel`, `updateModel`, `saveModel`, `getModelList`, `simulateCelleratorModel`, and `simulateModel`. These Web services provided by the intelligent middleware can also be used as an open API for third party client programs to communicate with the back-end knowledge database and simulation engines.

6. The Graphical User Interface: Sigmoid Model Explorer (SME) User Interface

The last component of the system to be initiated, and the most recent to achieve demonstration-level functionality, is the SME Web-compatible Graphical User Interface. The GUI allows the user to visualize, design, edit, and store pathway models, parameters, and initial conditions and their properties, to simulate the models by calling the simulator through the middleware, and to view and compare the properties of simulated models, for instance by viewing the temporal evolution of the concentration of chemical species under different conditions. The GUI runs from any Web browser as a Java Webstart or as a local client program.

The Sigmoid Model Explorer (SME) GUI is a Java application that is "aware" of the current Sigmoid object schema by using Java reflection. The SME GUI can be downloaded and also (as a Webstart) automatically updated through the Web. In addition Sigmoid uses Web-compatible Internet communication protocols (XML and SOAP) to perform three-tier distributed computing through the intelligent Web services" middleware which in turn communicates with the Sigmoid database and with Cellerator. Thus a variety of software platforms in addition to the SME Java application could use Sigmoid through its Web services. The SME GUI can display biological modeling objects in a compositional hierarchy, supports browsing and selection from the model database, and supports editing of numerical parameters. It also supports display and editing of network layouts as bipartite labeled graphs with a user-definable mapping of

object types to icons. Finally SME enables a simulation to execute remotely and return sets of plots for side-by-side comparison with previous plot sets.

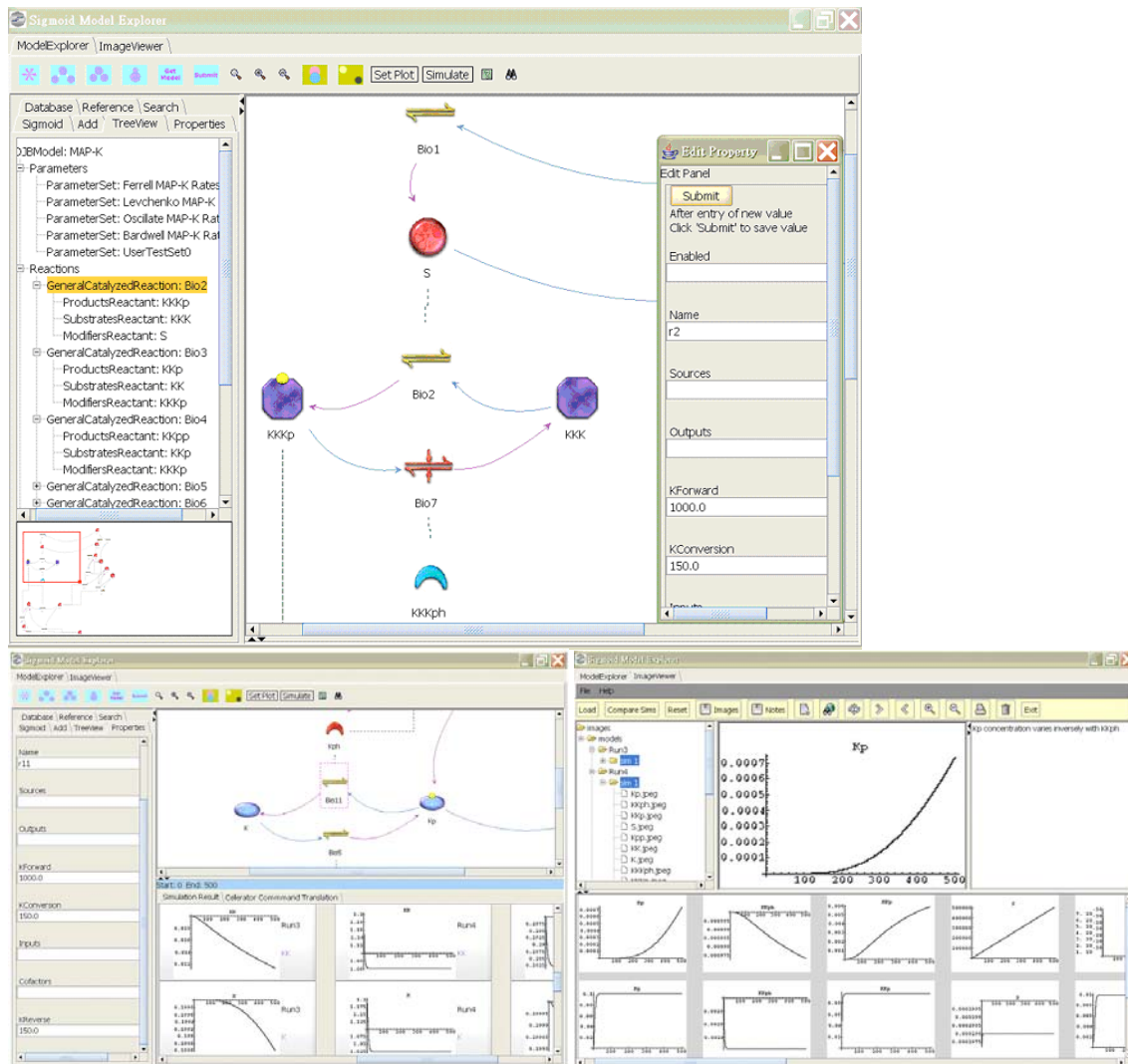


Figure 7: Sigmoid Model Explorer showing portion of MAPK pathway. (a) From left to right: TreeView of compositional hierarchy; network layout visualization; parameter-editing panel. Along the top are various action buttons for saving and running the model, and for switching the main panel to view output plots. User can select reaction icons. (b) Bottom left, optional output plot preview panel. (c) The full output displays mode allows annotation.

SME is a cross-platform Java application containing multiple elements: (1) a Tree Viewer for biological objects (e.g. pathways, reactions, and their constituent reactants and parameters); (2) biological network layout view (using JGraph libraries) in the main right panel including semi-automated layout of small circuits and user editing (and saving) of layout and network reactions and connections; (3) alternative displays for the left panel including a folder hierarchy of displayable simulation results; (4) an optional output previewer that allows side-by-side comparison of multiple plots each from multiple simulation runs; (5) a full output viewer (using JAI for good image quality) that allows

annotation of simulation outputs for a primitive scientific notebook, with Encapsulated Postscript (EPS) publication-quality output; (6) user-selectable JPEG/GIF icon sets for displaying biological object types; (7) back-end connections to the Middleware including database and simulator access; (8) use of Java reflection to discover the current Sigmoid object hierarchy and database schema for displayable objects, as well as model-editing methods.

7. Conclusions

We have described the Sigmoid intelligent software infrastructure for systems biology. An initial version of each of the main components is available today and there are clear signs that the infrastructure can already be used to yield biologically relevant results. For instance, the *E. coli* metabolic pathway model correctly predicts the effect of certain mutations and the MAP Kinase cascade model shows that, depending on the parameter sets and initial conditions chosen, it can generate a switch-like or graded input-output relationship, or even produce oscillatory behavior.

Development and expansion of Sigmoid continues at all levels. As the mediator of the user experience with Sigmoid, the GUI is bound to attract the largest number of feature requests from users. Because the overall architecture is now functional, many of these requests can be met at reasonable levels of effort and cost. There is also a need for new reaction types in Cellerator to deal with various kinds of (non-transcriptional) feedback. Other reaction types already in Cellerator, for example those used in modeling metabolic pathways (MWC, ping-pong, etc.) will need to be exposed for new pathways beyond MAPK and NFkB. An essential aspect of the scale-up of Sigmoid will be expert curation of the allowed and suggested mappings from biological reaction mechanisms to mathematical reaction models. New modules, for instance to optimize models or learn from data, are under development in the back-end and are expected to enhance end-user capabilities. With such powerful tools, the models created in Sigmoid will be of unique value to biologists. Such models will be used within Sigmoid to 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 are not confined to a particular software environment.

Likewise, we continue to expand and populate the Sigmoid database. It is possible to develop database “populator” codes for importing relevant data from other reaction and pathway data sources, depending on their accessibility to software agents, such as Kyoto Encyclopedia of Genes and Genomes (KEGG), SiBML/GeneNet, Cytoscape, Reactome, Saccharomyces Genome Database (SGD), and Biocyc. These connections will be able to take advantage of systems biology communications infrastructure such as Systems Biology Markup Language (SBML), BioPAX, Systems Biology Workbench (SBW), BioSpice, Monod., and many others. All these systems biology projects each have their own strengths and specializations; for example, the current versions of KEGG and BioPAX (level 1, version 1.2) are better developed for metabolic pathways than for the signal transduction pathways which have been the initial focus of Sigmoid. Increased standardization and inter-operability will be achieved through, for instance, use of SBML

(an XML-based protocol for systems biology information interchange [<http://www.sbml.org> for further information]), Gene Ontology (GO) and other *de facto* standards as they emerge.

While it is sound to have a number of parallel efforts across multiple research groups, we believe Sigmoid's most distinctive feature and advantage is its scalability. A simple pathway contains on the order of a dozen reactions representable by a set of 50 elementary equation models or so, taken from a growing library of reaction model types as in Figure 4. If we are to reverse engineer cells, this number must scale up progressively to 500, 5000, 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. Being able to automatically regenerate entire databases from a modified schema becomes essential. Generative tools such as autogeneration of the database and compatible object API's from a UML schema, and autogeneration of mathematical models from reaction notation using computer algebra, have already proven to be an effective way to limit the cost of updating the Sigmoid architecture with new features. These principles of self-generation and scalability lie at the heart of the Sigmoid architecture. We are also exploring whether fundamental software classes for nodes, networks, and communications can be reused across different kinds of networks, from the biological reaction networks we wish to simulate to the machine learning networks we use to reverse engineer them (Baldi et al. 1998). In short, can generative network software classes play a central role in the design of intelligent systems?

The current progress is very encouraging, but considerable challenges remain ahead. Two of the main challenges are at the modeling and GUI levels. On the modeling side, the right mathematical formalism for scale-up is not yet clear. How should one handle and integrate multiple temporal and spatial scales over several orders of magnitude, as well as complex combinations of continuous, stochastic, and discrete events with different levels of compartmentalization? On the GUI side, the scalability of network layout and editing is far from solved. Can we rapidly zoom up and down, from molecular reactions to cellular function?

Acknowledgements

This work has been supported by NIH grant GM069013 to EM; NSF grant EIA-0321390 and NIH grant T15 LM007443 to PB; a Laurel Wilkening faculty innovation award to PB; a UC Systemwide Biotechnology Research and Education Program #2002-06 award to PB; NCI Director's Challenge support to Children's Hospital Los Angeles for EM; NASA Intelligent Systems Program support of EM, and by the Institute for Genomics and Bioinformatics at UCI. We would like to thank students, programmers, and colleagues that have provided us with valuable feedback or have helped implement particular components of the infrastructure. These include Lee Bardwell, 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

P. Baldi, Y. Chauvin, and V. Mittal Henkle. Software Foundation Libraries for the Design of Intelligent Systems. Neural Nets WIRN Vietri 98. Proceedings of the 10-th Italian Workshop on Neural Nets. M. Marinaro and R. Tagliaferri Editors. Springer Verlag Series on Perspectives on Neural Computing Series, J. Taylor Series Editor, (1998).

P. Baldi and G. Wesley Hatfield. DNA Micorarrays and Gene Regulation—From Experiments to Data Analysis and Modeling. *Cambridge University Press*, (2002).

L. Bardwell. A Walk-through of the Yeast Mating Pheromone Response Pathway. (Review) Peptides, in press, (2004).

G. Booch, I. Jacobson, and J. Rumbaugh, The Unified Modeling Language User Guide. *Addison-Wesley*, (1998). See also <http://www.uml.org/>.

A. Hoffmann, A. Levchenko, M. L. Scott, and D. Baltimore. The I_B-NF_B Signaling Module: Temporal Control and Selective Gene Activation. *Science*, 298(5596):1241-1245, (2002).

T. Ideker, V. Thorsson, J. A. Ranish, R.Christmas, J. Buhler, J. K. Eng, R. Bumgarner, D. R. Goodlett, R. Aebersold, and L. Hood. Integrated Genomic and Proteomic Analysis of a Systematically Perturbed Metabolic Network. *Science*, 292:929-934, (2001).

I. H. Segel. *Enzyme Kinetics*. John Wiley & Sons, 1993.

B. E. Shapiro, A. Levchenko, E. M. Meyerowitz, B. J. Wold, and E. D. Mjolsness. Cellerator: Extending a Computer Algebra System to Include Biochemical Arrows for Signal Transduction Simulations. *Bioinformatics*, 19(5):677-678, (2003).

B. E. Shapiro, A. Levchenko, and E. D. Mjolsness. Automatic Model Generation for Signal Transduction with Applications to Map Kinase Pathways. In: *Foundations of Systems Biology*, H. Kitano, ed, MIT Press, pages 145-162, (2002).

URLs for the software technologies listed include: JGraph: <http://www.jgraph.com/>; Java Web Start: <http://java.sun.com/products/javawebstart/>; Apache: <http://apache.org/>; SOAP (Simple Object Access Protocol): <http://www.w3.org/TR/soap12-part1/>; Tomcat: <http://jakarta.apache.org/tomcat/> ; OJB (Object Relational Bridge): <http://db.apache.org/ojb/>; JLink and Mathematica: <http://www.wolfram.com/>; Cellerator: <http://www.igb.uci.edu/servers/sb.html> , SBML (Systems Biology Markup Language): <http://sbml.org>; UML (Universal Modeling Language): <http://www.uml.org/> and the Object Management Group's Model-Driven Architecture <http://www.omg.org/mda/>;

Axgen (Anything from XMI Generator): <http://axgen.sourceforge.net/>; VTL (Velocity Template Language): <http://jakarta.apache.org/velocity/>.

C. R. Yang, B. E. Shapiro, E. D. Mjolsness, and G. W. Hatfield. An Enzyme Mechanism Language for the Mathematical Modeling of Metabolic Pathways. *Bioinformatics*, in press. Supporting material available at available at <http://www.igb.uci.edu/servers/coli/kmech.html> .