Improving the Development Process for Eukaryotic Cell Cycle Models with a Modeling Support Environment

Nicholas A. Allen
Clifford A. Shaffer
Naren Ramakrishnan
Marc T. Vass
Layne T. Watson
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106
shaffer@vt.edu

Theoretical molecular biologists attempt to describe cellular processes and regulatory networks with continuous and discrete mathematical models. Previous practice has been to develop models largely by hand and then to validate them primarily by comparing time-series plots versus the observed experimental results. The authors report their experiences in designing and building a modeling support environment for cell cycle models. They describe improvements to the development process for molecular network models by (a) identifying the key elements of the existing modeling process, (b) incorporating simulation methodology into a revised modeling process, and (c) building and testing software that supports the revised modeling process.

Keywords: Support environment, systems biology, hybrid systems, regulatory networks

1. Introduction

Mathematical models of biochemical control systems attempt to derive observed physiological properties of cells from the underlying molecular regulatory networks. Examples include growth and division, chemotaxis, secretion, and circadian rhythms. The hope is that creating such models will lead to a higher level understanding of the biological processes involved. The common form of such models is (at some point in the process) systems of differential equations with discrete switching. A number of independent groups have been developing tools to support aspects of the modeling process; a sampling includes Loew and Schaff [1], Mendes [2], and Sauro [3]. BioSPICE [4] is a major effort by DARPA to provide a new generation of interoperable modeling and simulation tools. It seeks to improve the quality of pathway modeling by providing the community with common languages for expressing models [5, 6] and interoperability between various model description editors, simulators, and analysis tools.

SIMULATION, Vol. 79, Issue 12, December 2003 674-688

DOI: 10.1177/0037549703040944

However, the recent state of pathway modeling has been largely ad hoc and labor intensive, as most modelers have not tried existing tools, or those tools have proved inadequate for their needs. Many modelers still work by hand-sketching their ideas (see the discussion of wiring diagrams below) and then manually converting those sketches to differential equations. Analysis often involves visual comparisons between time-series plots and experimentally collected results.

This article describes how introducing appropriate modeling tools can improve the speed and accuracy of the model development process (which directly permits the creation of larger, more complex models) and also can lead to a more disciplined approach to the model life cycle.

We present a brief description of typical molecular regulatory network models in section 2. The original modeling process observed in a representative portion of the community is described in section 3. Section 4 describes how the conical methodology [7] relates to the observed modeling process and can be used to improve the modeling life cycle. Section 5 briefly describes the JigCell modeling support environment (MSE) developed for cell cycle modeling and related problems and compares JigCell to existing pathway modeling software. Section 6 describes

the impact that the Systems Biology Markup Language standardization effort and the BioSPICE project, of which JigCell is a part, have had on the modeling community. Section 7 presents our conclusions.

2. Molecular Regulatory Networks

A simple example of a regulatory network is the set of reactions controlling the activity of a mitosis-promoting factor (MPF) in frog eggs (see Fig. 2). Such networks are often represented as graphs, where vertices represent substrates and products (collectively referred to as species), and labeled directed edges connecting vertices represent the reactions. Chemical reactions cause the concentrations of the chemical species (C_i) to change in time according to the equation

$$\frac{dC_i}{dt} = \sum_{j=1}^{R} P_{ij} v_j, \quad i = 1, \ldots, N,$$

where R is the total number of reactions, v_j is the velocity of the jth reaction in the network, and P_{ij} is the stoichiometric coefficient of species i in reaction j ($p_{ij} < 0$ for substrates, $p_{ij} > 0$ for products, $p_{ij} = 0$ if species i does not take part in reaction j). The full set of rate equations is a mathematical representation of the temporal behavior of the regulatory network. These equations are then solved numerically [8], and the model behavior is interpreted from the time-course output of the species concentrations. Modelers are faced with many computational problems: accurately and efficiently solving equations when velocities are characterized by widely varied time constants, finding steady-state solutions, estimating rate constants by fitting numerical solutions to experimental data, and identifying bifurcation points in the multidimensional parameter space.

For example, a current model of the budding yeast cell cycle consists of about 30 differential equations containing 100 rate constants. The parameters are estimated from the cell cycle behavior of more than 100 mutants defective in the regulatory network. Simulating the entire set takes from a few minutes to an hour on a desktop PC for one choice of kinetic constants. Fitting the model to the data by nonlinear regression will likely require thousands of repetitions of the full calculations. A model of such complexity (10-100 equations) represents the upper limit of what a dedicated modeler can produce "by hand" with a good numerical integrator such as LSODE [9]. To adequately describe fundamental physiological processes (such as the control of cell division) in mammalian cells will require models of at least 100 to 1000 equations. Handling this next generation of dynamical models will require sophisticated software to automate the modeling cycle: network specification, equation generation, simulation and data management, and parameter estimation.

Ongoing efforts such as the DARPA BioSPICE initiative [4] aspire to support the necessary increase in model size. This project supports approximately 15 research groups, including our own. The goals of BioSPICE are to advance the broad efforts of biochemical pathway modeling by supporting modeling efforts, experimentation efforts, and software tool-building efforts together. Software efforts are not just to provide sets of tools to modelers but to make the tools developed by the various groups interoperable. In addition, tools developed as part of BioSPICE are made freely available under an open-source license. While it is not known yet whether the interoperability goals will be met, the BioSPICE project is generating the potential for interaction through the definition of application program interfaces (APIs) for communication between tools, as well as language definitions for data such as model definitions and simulation outputs.

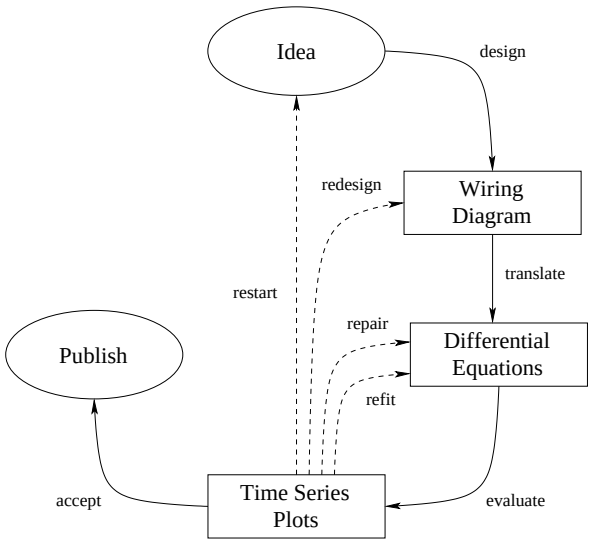

Figure 1. Original modeling process

3. Original Modeling Process

Figure 1 shows a modeling process that has successfully developed several biochemical pathway models [10]. The process evolved over more than 10 years of developing models. It is not based on formalisms or documented; new modelers learn the process through demonstration and mentoring. Until recently, the modelers' tools were primarily off the shelf for solving and analyzing differential equations and not specialized for pathway modeling.

Before a model can be developed, there must be a problem that the model intends to solve. Problem formulation includes analysis of requirements, identification of a solution method, and specification of modeling objectives [11]. Without a formulated problem, the modeler risks inadequately solving the problem or solving the wrong problem.

A notable feature of the original modeling process is that it deals solely with model development and does not contain problem formulation. Over the past 2 years of our observation, this particular group of modelers has not formulated a completely new problem (in the sense of wishing to develop a model for a new organism or apply new solution techniques to a previously developed model). Instead, they expand existing models by attempting to match additional experimental observations. Is this infrequent reformulation an inherent property of the problems these modelers are attempting to solve or a side effect of the current modeling process? Observations with different modeling processes might answer this question in the future.

The original modeling process has four primary stages: design, translate, evaluate, and accept. Models are created and refined in the design and translate stages. Testing occurs during the evaluate stage. The accept stage produces a presentable model from the information the modeler has recorded. This may be the simplest process that could successfully build a model.

In the ensuing text, a stage labeled x in a diagram will be denoted by $\xrightarrow{x}$, and such symbols are used to mark the paragraph where those stages are discussed. Stages drawn with solid lines in diagrams indicate successful completion of a process. Stages drawn with dashed lines indicate error recovery activities.

$\xrightarrow{design}$ The design stage begins with creating a wiring diagram from an idea of how a biological process is carried out. The wiring diagram is a graph that captures the chemical species (also known as products and reactants) at the nodes and represents interactions that create, destroy, and convert these species at the arcs. In addition, the wiring diagram may note the kinetic information for a reaction: describing the mechanics of the reaction and the rate at which the reaction occurs. Figure 2 is a wiring diagram depicting posttranslational modification of cyclin, an important regulation process, in frog egg extracts based on the model in Marlovits et al. [25].

The notation for wiring diagrams is not currently standardized. Modelers often invent ad hoc notation to express abstractions, replication, and unusual processes. Kinetic information is frequently presented separately from the wiring diagram or must be inferred from figures. Without full information about the rate laws and constants, the model can be structurally analyzed but not simulated.

Since the wiring diagram often lacks full details of the kinetics, it is typical for our modelers to first rewrite the model as a series of chemical reaction equations, along with appropriate kinetic functions to describe the reaction.

$\xrightarrow{translate}$ The translate stage is the process of converting the reaction equations to systems of ordinary differential equations. For each species in the system, the modeler creates a differential equation. Reactions that involve the species determine the right-hand side of the differential equations. Parameters for the differential equations are set according to the kinetic information for the reaction. Frequently, exact values for these parameters are not known. In this case, estimates are made for the parameter values and updated as the model is developed.

In some cases, a protein is never created nor destroyed but is converted between different forms. When this occurs, the quantity is said to be conserved, and one of the differential equations is replaced with an algebraic expression called a conservation relation. In addition to the continuous differential equation model, there is also a discrete event model. Certain cellular processes, such as cell division, are modeled by discrete events that set species values, alter rate laws or constants, and switch between sets of differential equations driving the continuous model. Table 1 shows a system of ordinary differential equations and conservation relations produced from Figure 2.

$\xrightarrow{evaluate}$ The evaluate stage begins by generating time-series plots of important species concentrations from the model. These plots correspond to experimental observations of the process in the lab. The modeler compares the time-series plots with the experimental observations and judges whether the model adequately represents the biological process. In addition to determining if a time series matches observed concentrations, the modeler might also seek to determine if gross physical behavior has been reproduced, such as an appropriate mass at cell division or death of the cell at the appropriate stage in the cell cycle.

$\xrightarrow{accept}$ The accept stage is an assertion that the model adequately represents the biological process and consists of final preparations for archiving and disseminating the model.

The remaining stages in the original modeling process are error recovery stages. Errors are detected by informal examination of time-series plots. The modeler must infer the nature and location of the error from experience. Because the developed models are typically underspecified, the modeler cannot always accurately identify the cause of an error. Laboratory experiments can test hypotheses but are extremely expensive. In general, the goal of the modelers is to match the existing collection of laboratory results.

$\xrightarrow{redesign}$ The redesign stage corrects errors in the wiring diagram. Reactions are added and deleted based on the modeler's developing intuition about what mechanisms must be included in the model to adequately reproduce the desired experimental behavior.

$\xrightarrow{repair}$ The repair stage corrects errors made in the translation between the wiring diagram and differential equations. Manually creating differential equations is time-consuming and prone to error. Tedious checking between the wiring diagram and differential equations is required to detect and correct errors in translation.

$\xrightarrow{refit}$ The refit stage corrects errors made in the assignment of differential equation parameters. New estimates are made for the kinetic rate constants based on com-

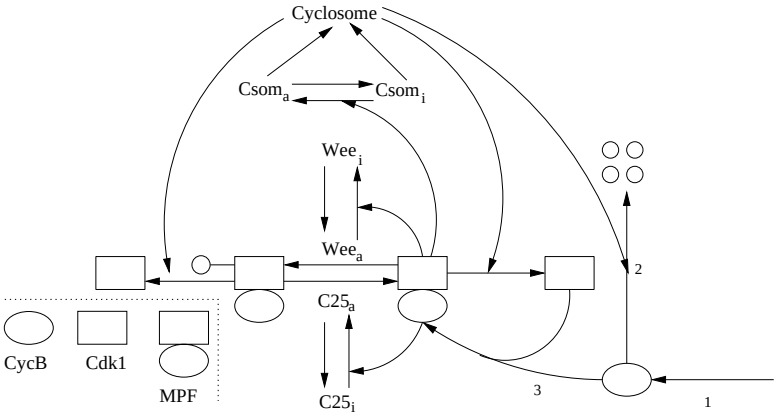

Figure 2. Wiring diagram

Table 1. Equations from wiring diagram

$\frac{d\,\mathrm{C25}_a}{dt}$	=	$\frac{k25_f*(\mathrm{MPF}_a+\epsilon_1)*\mathrm{C25}_i}{J25_f+\mathrm{C25}_i} - \frac{k25_r*\mathrm{C25}_a}{J25_r+\mathrm{C25}_a}$
$\mathrm{C25}_i$	=	$1.0 - \mathrm{C25}_a$
$\frac{d\,\mathrm{CycB}}{dt}$	=	$k_1 - \mathrm{CycB} * (\mathrm{Cyclosome} + k_3 * \mathrm{Cdk1})$
Cyclosome	=	$(k'_2 * \mathrm{Csom}_i + k''_2 * \mathrm{Csom}_a)$
$\frac{d\,\mathrm{MPF}_a}{dt}$	=	$k_3 * \mathrm{CycB} * \mathrm{Cdk1} + (k'_c\mathrm{C25}_i + k''_c * \mathrm{C25}_a) * \mathrm{MPF}_i$
		$- (k'_w * \mathrm{Wee}_i + k''_w * \mathrm{Wee}_a + \mathrm{Cyclosome}) * \mathrm{MPF}_a$
$\frac{d\,\mathrm{MPF}_i}{dt}$	=	$(k'_w * \mathrm{Wee}_i + k''_w * \mathrm{Wee}_a)$
		$- (k'_c * \mathrm{C25}_i + k''_c * \mathrm{C25}_a + \mathrm{Cyclosome}) * \mathrm{MPF}_i$
Cdk1	=	$1.0 - \mathrm{MPF}_a - \mathrm{MPF}_i$
$\frac{d\,\mathrm{Wee}_a}{dt}$	=	$\frac{k\mathrm{Wee}_r*\mathrm{Wee}_i}{J\mathrm{Wee}_r+\mathrm{Wee}_i} - \frac{k\mathrm{Wee}_f*(\mathrm{MPF}_a+\epsilon_2)*\mathrm{Wee}_a}{J\mathrm{Wee}_f+\mathrm{Wee}_a}$
Wee_i	=	$1.0 - \mathrm{Wee}_a$
$\frac{d\,\mathrm{Csom}_a}{dt}$	=	$\frac{k\mathrm{cy}_f*(\mathrm{MPF}_a+\epsilon_3)*\mathrm{Csom}_i}{J\mathrm{cy}_f+\mathrm{Csom}_i} - \frac{k\mathrm{cy}_r*\mathrm{Csom}_a}{J\mathrm{cy}_r+\mathrm{Csom}_a}$
Csom_i	=	$1.0 - \mathrm{Csom}_a$

parison with known experimental results. The modeler typically changes only a small number of rate constants at each iteration due to their potential interactions.

$\xrightarrow{restart}$ The restart stage is the termination of a particular model and marks the start of the next idea of how a biological process is carried out.

4. Applying a Methodology

The original modeling process has successfully developed models that define the current state of the art. However, the modeling community recognizes that they are at the limit of the complexity that their current methodology can support, which is driving many new efforts in tool development such as BioSPICE. Methodologies assist in understanding the model development process and indicate requirements for supporting that process [13]. Formal methodological approaches provide well-defined and tested techniques for the model development process.

On the basis of our experience with the original modeling process, we enumerate some capabilities that a methodology needs to support this modeling community. Modelers have an ultimate goal of producing models that are validated and accepted. Demonstrating that their models are valid and should be accepted requires performing verification, validation, and testing (VV&T). Modelers should

employ VV&T frequently to minimize wasted effort on bad models. Models developed by the original modeling process have proved extremely long-lived and are repeatedly adapted to meet changes in specification. We expect this to continue and so require a modeling process that is capable of introducing change at any stage without undue cost. Computational technology is also expected to change significantly during the lifetime of a model; models and the modeling process need to be insensitive to the runtime host and adaptable to high-performance computing techniques of the next 10 years. Using the terminology of Nance and Arthur [14], our modeling process must support correctness and testability; support adaptability, maintainability, and portability; and test throughout the model life cycle.

We select the conical methodology [7] as a methodology supportive of our requirements and with sufficient adaptability to capture our original and desired modeling processes. Balci [11] describes a model life cycle compatible with the conical methodology; we will use similar terminology to describe our modeling process. The primary objectives of the conical methodology are correctness, testability, adaptability, reusability, and maintainability [14]. This is a good match with our primary and secondary requirements. The conical methodology prescribes a top-down model definition phase followed by a bottom-up model specification phase. As we are creating a domain-specific MSE, we significantly reduce the amount of work required in the definition phase by predefining constructs in our tools.

4.1 Goals for Improving the Process

After we documented the original process, examined methodological frameworks, and listened to the concerns of modelers, we identified four areas for which the modeling process needs to be improved: documentation, testing, standardization, and automation. These four areas are important for developing models quickly and accurately. Independent verification and validation are testing activities performed by someone other than the model developer with the goal of improving the quality of the model [15]. Independent testing reduces potential modeler bias in evaluation, promotes earlier error detection, reduces error cost, and enhances operational correctness. We want to introduce independence into the modeling process at each iterative cycle, with the goal of supporting independence for all testing activities. This level of support requires significant advances in the four outlined areas. We believe that making these improvements will ultimately lead to an increased rate of model accreditation and acceptance.

The goal of documentation is to record critical information about the modeling process. Model documentation is needed at every stage of the modeling process and is critical for future planning of modeling tasks. We want to record the model itself each time the description of the model is transformed. We want to record the procedure used for testing to support automated testing and review of VV&T methods. We want to record the results from testing for presentation and for comparison against future tests.

Comparison with experimental data has been the main testing technique used for validating these models. However, the quantity and quality of experimental data available for a particular system may be limited. It is a major expense to conduct new laboratory experiments for further testing or expansion of the model. In contrast, modeler time is relatively cheap. We emphasize verification during model construction to prevent the introduction of errors that strain our limited testing resources. The goal of testing is to introduce VV&T activities as soon as possible into the modeling process and to continuously monitor for introduced errors. We codify several indicators of model credibility [16] as automated tests that can be performed continuously during model development. When working with the wiring diagram, we want to verify that the graph structure of the diagram corresponds to the modeler's understanding of the structure. We want to verify that the names of species, rate laws, and constants are used consistently across the diagram. When building the executable model, we want to verify that the simulator can properly execute our model and that all required information is available. We want to perform the primary testing activities from a recorded plan that can be defined by an agent independent of the design or specification teams.

The goal of standardization is to adopt uniform notations and processes that reduce burdens on communication and development. For our modelers, a wiring diagram is the initial abstract representation for the model. Unfortunately, no standards exist for the graphical language of wiring diagrams, though the representation of Kohn [17] is becoming increasingly popular. The pathway modeling community is currently involved in standardizing the Systems Biology Markup Language (SBML), an XML-based representation of models at the chemical reaction level [6, 18]. While it is hoped that SBML will apply to wiring diagrams in the sense that model editing tools should be able to convert between SBML and wiring diagram representations, SBML files are not meant to be directly edited by modelers. The main purpose of SBML is to facilitate model exchange between modeling groups, who will then load the models into ad hoc editing tools. Biological models can have operations repeated across multiple reactions and incorporate subcomponents developed as part of other models. Sufficient support for interchanging model fragments would allow replacing both with well-tested black box subcomponents.

We can also employ standardization at each stage of the modeling process by using domain-specific information to construct uniform sequences of tasks. A uniform process reduces the developmental tasks required of modelers and can prevent some errors in the planning stage of the modeling process.

Most of the existing modeling process consists of work that is performed repeatedly. The goal of automation is

to have the computer perform some of these repetitive tasks and speed up other tasks substantially. VV&T activities exist as automatable tasks throughout the modeling process; supporting these activities with automated tools can significantly reduce the time and effort for model testing [19]. Modelers repeatedly modify parameter and initial condition sets at each modification, comparing the revision against experimental data. We want to perform regression testing as frequently as possible and repeat testing activities from previous iterative cycles to ensure that model quality is maintained after each model transformation. Our testing activities should be numerous and specific so that when an error is introduced, we can identify what stage the error was introduced at and in what component of the model the error is located. When the user is engaged in modifying the model, the testing process should be conducted automatically, and feedback on the relative performance of the modification should be supplied automatically.

4.2 Revised Modeling Process

The revised modeling process (see Fig. 3) begins with an already defined problem. This problem definition includes an analysis of requirements and an identification of modeling objectives. We must make an assumption here that our modeling tools are adaptable to the solution technique chosen as part of the problem definition. Domain specificity allows us to make this assumption: the tools were developed specifically to meet the needs of biological modelers who have a large class of problems of interest.

From the problem description, modelers begin to develop model ideas that they believe will satisfy the problem requirements. The process of realizing and testing these ideas is extended from the original modeling process.

$\xrightarrow{design}$ Starting with a conceptual model for a biological process, the modeler must first produce a model that can be understood by others. In addition to the wiring diagram or reaction equations, a complete model requires rate laws, constants, and the discrete event model that will control switches in the differential equation model. Models at this stage can be structurally tested and checked for completeness and consistency of kinetic information.

$\xrightarrow{translate}$ The model must then be translated from a human-understandable form to an executable form. For the domain of biological modeling, we possess a significant amount of information about this transformation process. A sufficiently described model translates by a mechanical process. The modeler only needs to tweak the control parameters for the evaluation process. However, the model must still be verified to ensure that the model was sufficiently described before translation, self-consistent, and tolerant of the numerical errors to which the chosen simulation process is susceptible.

$\xrightarrow{evaluate}$ After a model is simulatable, the modeler tests it against the requirements and objectives in the problem definition. VV&T activities in the evaluate stage likely account for a significant portion of the model development time and should have computer assistance to automate the test process [19]. Entering the problem definition only once and automating the testing process from previous iterative cycles of the modeling process are important for reducing model development time. In addition, there should be provisions for independent execution of the evaluate stage to prevent modeler bias in the testing process.

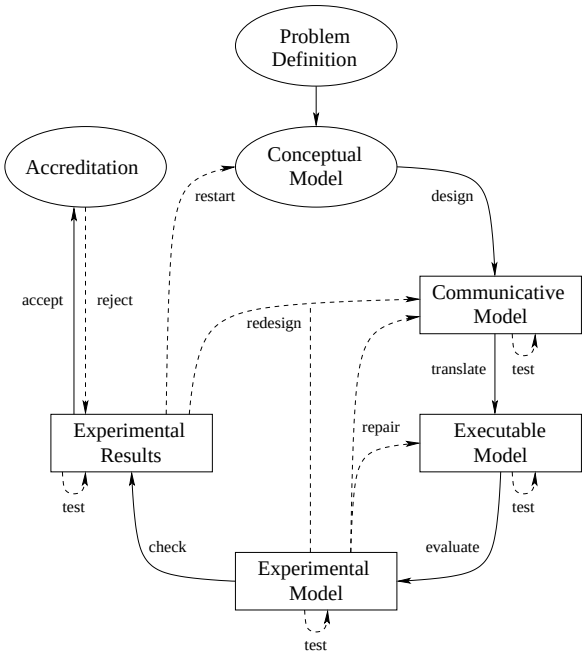

Figure 3. Revised modeling process

$\xrightarrow{check}$ A model that meets the requirements and objectives stated in the problem might still be rejected. In the biological domain, two reasons why this occurs are that the model is insufficiently based on established biological processes or that the model is not significantly better than an existing, simpler model. The check stage addresses these issues. Comparing the proposed model against accepted models [20] representing similar processes can test the first reason. Performing a statistical analysis between the proposed model and a collection of models for the same system can test the second reason. Both techniques test the model against other models of biological systems.

$\xrightarrow{accept}$ and $\xrightarrow{reject}$ The accept stage is relatively unchanged from the original modeling process. We formalize the preparations in the original accept stage as the process of creating documentation and presentations to show that the model is sufficiently accurate for its intended purpose [11]. In addition, we introduce the reject stage for models that pass all of our tests but are rejected by decision makers.

The testing phases of the modeling process have been considerably augmented from the original process.

Models are tested at every stage that creates or transforms a recorded model description. In addition, the results of model testing more specifically point to causes of errors and direct the modeler to an appropriate stage for correcting the error.

$\xrightarrow{test}$ Test stages represent VV&T activities that take place in real time during model development. Continuous verification is especially important when the amount or quality of experimental data is limited. A test stage operates concurrently with tool use and indicates errors found in model information entered in the tool. Errors found and corrected in a test stage do not propagate to other stages of the modeling process. This reduces the amount of time to correct the error and reduces unnecessary switching between tools.

$\xrightarrow{redesign}$ and $\xrightarrow{repair}$ We have condensed the error recovery stages to redesign and repair, which correspond to activities that correct errors detected by validation and verification, respectively. When describing the software that implements this revised modeling process, we will once again enumerate specific types of error recovery activities.

The location, scope, and frequency of testing activities are the most significant difference between the original and revised modeling processes. Modelers get immediate feedback about errors detected during the process of transforming the model from one form to another. For errors that cannot be automatically detected, we attempt to identify more specifically the source and type of error so that the modeler spends less time diagnosing the problem. We hope that increasing the specificity of error reporting leads to a smaller average error recovery time. Moreover, even though we cannot always automatically detect where and how an error was introduced, we can automatically perform modeler-defined diagnostics to determine that there definitely was an error at some point in the last iteration of the process.

5. JigCell

JigCell is a domain-specific MSE for biological pathway modeling, intended ultimately to become a problem-solving environment (PSE) in the sense of Ramakrishnan et al. [21] and Watson et al. [22]. JigCell's user workflow (Fig. 4) corresponds closely with the modeling process we have identified. Table 2 lists support for our defined modeling goals in this MSE.

We have constructed a tailored environment rather than basing it on an existing, general-purpose MSE [23, 24]. We intend to support users in biology and related fields who do not have significant experience in formal modeling (but who are domain experts). During the development of JigCell, we supplied the software to biologists and modelers for testing. As well as finding errors, they gave us feedback about new features and priorities for development.

We incorporate off-the-shelf components such as numerical libraries, visualization tools, and communications

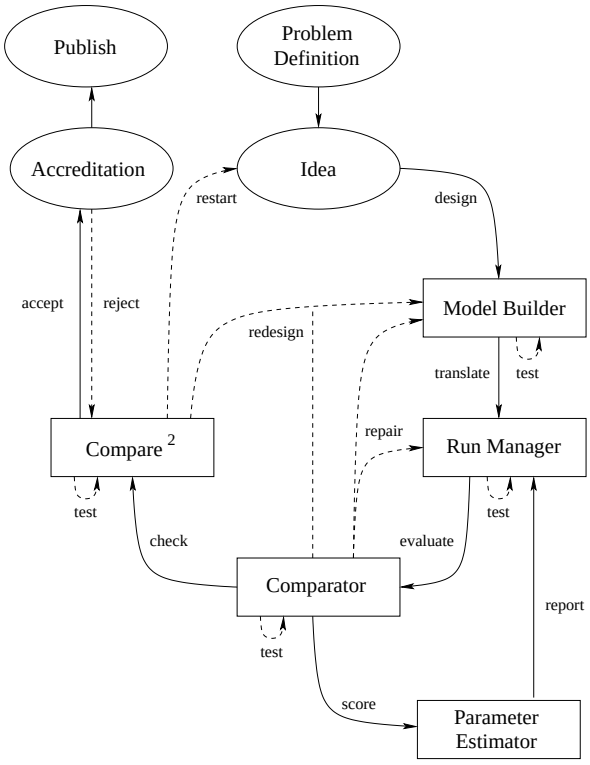

Figure 4. JigCell user workflow

Table 2. Support for modeling goals

General	Record model artifacts	Partial
	Record model testing procedure	Full
	Record model testing results	Full
Design	Verify consistent naming	Partial
	VV&T model structure	
	Abstract repeated operations	Partial
	Abstract compound operations	
	Identify rate laws and constants	Full
	Standardize notation	Full
Translate	Generate differential equations	Full
	Find conservation relations	Full
	Verify execution requirements	Full
	Verify well-formedness	Partial
Evaluate	Make VV&T repeatable	Full
	Support independent VV&T	Full
	Identify causes of errors	Partial
	Standardize process	Full
Check	Make covalidation repeatable	Partial
	Support independent covalidation	
	Identify causes of errors	Partial
	Standardize process	Partial

Note. VV&T = verification, validation, and testing.

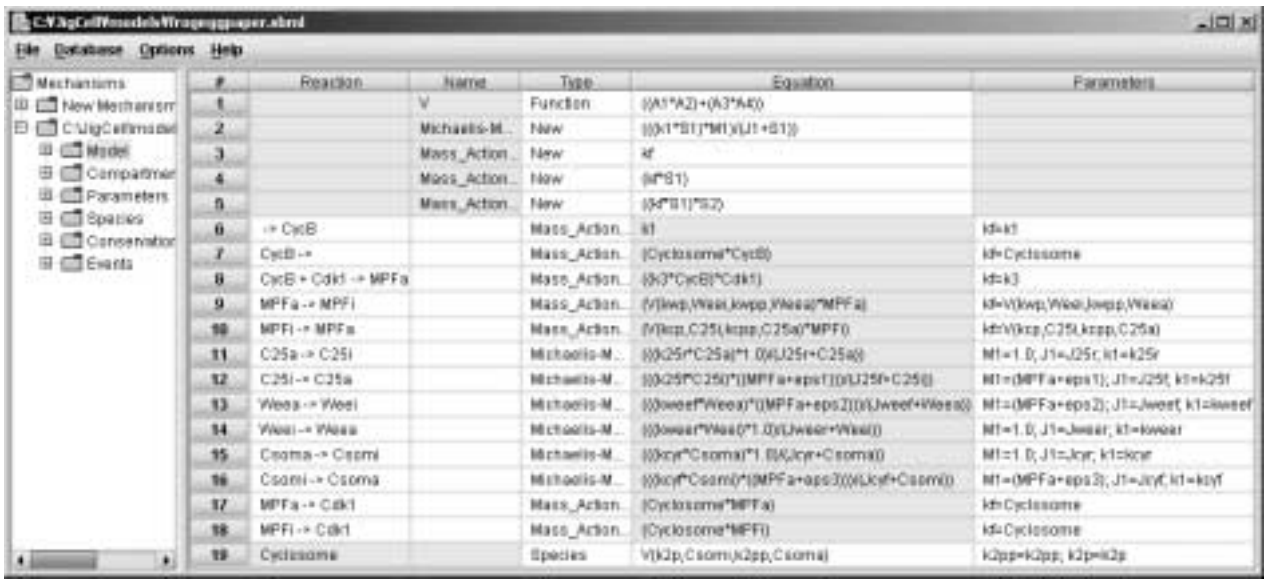

Figure 5. Model Builder interface with equations from wiring diagram

protocols in which quality implementations exist and technical specifics about the component can be hidden from the user. This approach has not been a significant drawback: the majority of development work relates to domain-specific support rather than the modeling infrastructure.

Development versions of JigCell are freely available in Allen et al. [25]. Official releases of JigCell are available as part of the BioSPICE releases [4].

5.1 JigCell Tools

The *Model Builder* creates a model specification that incorporates the wiring diagram, kinetic information, and discrete event model. A spreadsheet interface organizes the information of the wiring diagram and kinetics as a collection of chemical reaction equations. Each row of the Model Builder spreadsheet (Fig. 5) specifies a chemical reaction equation, including substrates, products, kinetic rate law, and kinetic rate constants. Chemical equations are a natural representation for many biological processes of interest and are applicable to a wide variety of fields outside biological modeling.

Restrictions are placed on the class of discrete event models: events can only be triggered based on algebraic conditions of species values and can only modify parameters, constants, and species values in the continuous model. However, these discrete models are sufficient for the biological systems we have studied and are easily and directly created by domain experts without modeling experience. For example, the discrete model can easily handle such processes as cell division: mass at time $t + \Delta t$ is set to one-half the mass at time t when a particular species crosses a threshold value. The Model Builder both reads and writes its models in the form of SBML, which is becoming the standard interchange language for this modeling community.

Species names and kinetic information are checked continuously during model entry, with color highlights indicating portions of the model that are not correctly specified. No mechanisms are included for testing overall model structure, and limited support is provided for representing stochastic and spatial models, which represent specific subdomains within the broader modeling community that we intend to support in the future. Division of the modeled cell into multiple topological compartments (volumes) is possible, but equations cannot contain spatial variables. Abstractions are possible in the sense that rate laws can be defined and reused. However, components in the form of black-box submodels are not currently supported since the community (in terms of the SBML standardization effort) has not yet defined mechanisms for this.

The *Run Manager* translates a model specification into an executable form. Each row in the Run Manager spreadsheet (Fig. 6) specifies how to simulate a certain experiment, including the model to use, parameter and initial condition sets, and the appropriate simulator settings. Parameter sets can contain a value for every parameter in the model or contain only the values changed in relation to another parameter set. For example, suppose we wish to describe a series of simulations for yeast cell mutants. On the first line, we describe how to simulate the wild-type yeast cell using a basal parameter set. On the second line,

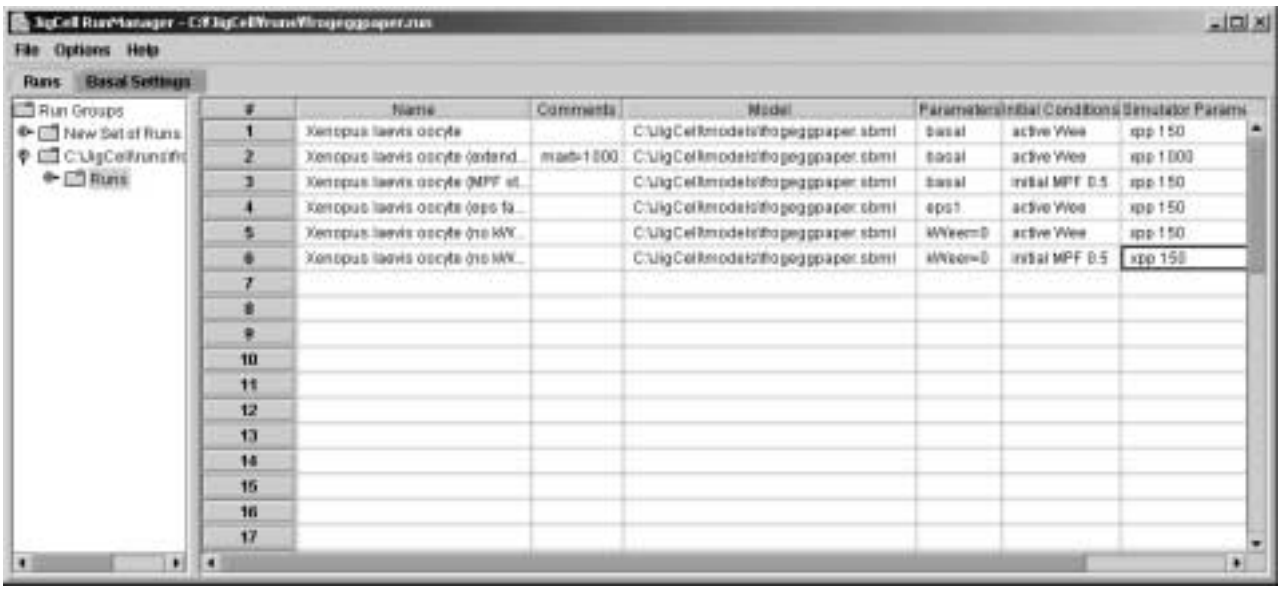

Figure 6. Run Manager interface for model in Model Builder

we describe a knockout mutant by using a parameter set that copies from the basal parameter set but changes the rate of synthesis of the knocked-out protein to zero. On the third line, we describe a mutant with a double copy of a gene by using a parameter set that copies from the basal parameter set and multiplies the synthesis rate associated with the gene by 2. If the modeler then develops a better estimate for the rate constants, she or he only needs to change the basal parameter set. Modelers can more easily explore the parameter space because the Run Manager does the bookkeeping to adjust the simulation specification.

Differential equations and code to handle the discrete event model are automatically generated from the specification. Modelers select the dependent variables of conservation relations, but the relations are automatically detected and generated. Automatic model translation insulates modelers from changes in simulation techniques and the runtime environment. New simulators are added by providing additional translators. This automation step represents a major improvement for our immediate modeling community, who previously converted reactions by hand in an error-prone and laborious process. This should also reduce the amount of model conversion work required of modelers to perform stochastic or spatial simulations in the future.

The *Comparator* and *Compare*[2] are tools for model testing and evaluation. Tests in the Comparator are assertions about a model or comparisons between model performance and experimental data. A test evaluates either operational accuracy or the accuracy in transforming the model. Performance on each test is scored according to a user-defined objective function that represents the goodness of fit between the expected result and the model result. We prefer objective functions that associate model performance with a degree of accuracy rather than a binary result. If the modeler is only told whether the result was acceptable, it is difficult to determine how robust the model is to parameter changes. Moreover, we prefer continuous objective functions, such as distance functions, as these are more amenable to automated optimization. The modeler chooses criteria for the objective functions based on the requirements and purpose of the model. Integrated editors support defining assertions, experimental data, procedures for transforming model results, and objective functions. Since we cannot predict all of the objective functions the modeler might wish to use, we allow the modeler (possibly with the assistance of a computer programmer) to add new functions. The modeler can automatically rerun a defined testing procedure in the Comparator.

The Comparator interface is a tabbed series of screens (Fig. 7) that describe the testing procedure. First, the modeler enters the experimental, or expected, results that the model should reproduce. Then, the modeler associates each experimental result with a procedure for generating an equivalent result from the model. The Run Manager typically executes the model. Finally, the modeler describes how to compute an objective function and specifies what constitutes an acceptable function value. Unacceptable fits are highlighted in the display so that the modeler can quickly see where the model is having problems.

Tests in *Compare*[2] compare performance between the currently proposed model and a collection of other mod-

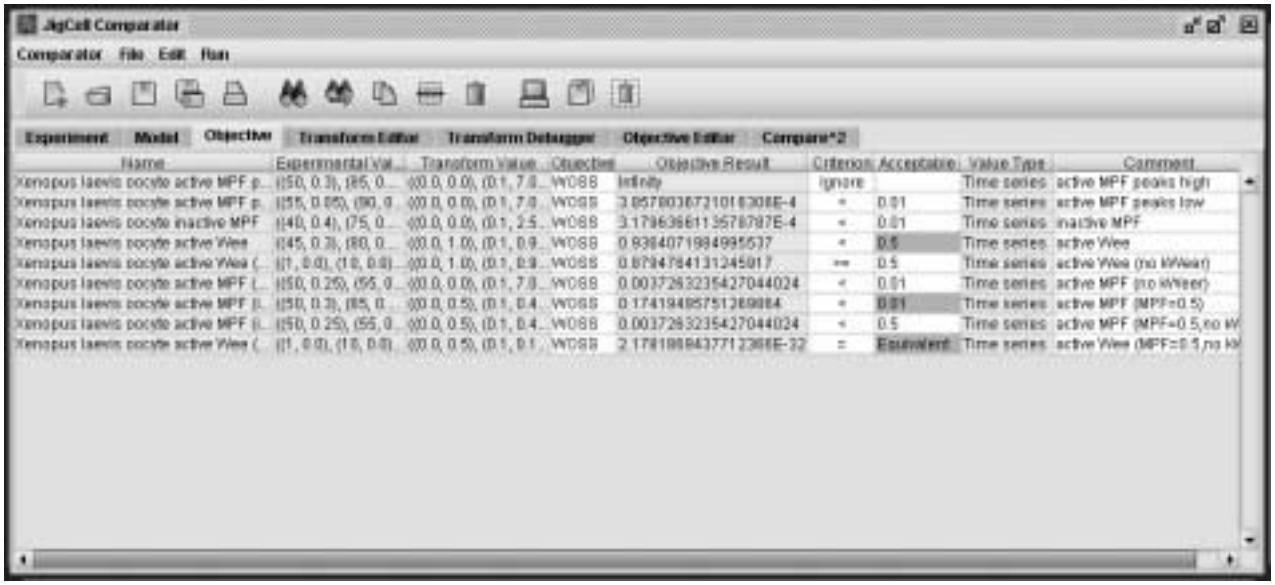

Figure 7. Comparator interface for model in Model Builder

els. Models come from past revisions of the current model, independent models of the same system, and models with subsystems in common with the current model. *Compare*[2] performs ranking and selection among these models based on the same criteria defined in the Comparator. For each objective function defined in the Comparator, models are scored and ranked. We use rankings rather than absolute results as we have found that many objective functions were not designed to finely distinguish results on an absolute scale. Selection of the best model is made by a function combining the objective functions, such as a sum of rankings. Development work on *Compare*[2] is still ongoing. A drawback of this tool is that obtaining meaningful results requires building a significant collection of models. This is also a limiting factor on development since, without test models, it is unclear what tasks are most critical to automate. We hope to incorporate other automated model analyses that could reduce the startup costs of this tool.

By automating the comparison process, we make an additional task possible: parameter estimation. Some rate constants used in the continuous model are not experimentally determined or have a significant range of possible values. Without automated fitting, the modelers must calibrate the model parameters by manually searching for valid and optimal regions of the parameter space. An expert in the biology and mathematics of the model must repeatedly try parameter guesses, using experience to determine acceptability and to select the next guess. This activity has consumed a major part of the model development time in the past. Humans should not perform it at all. A significant obstacle is that inputting the domain expert's intuitive understanding of the model into a computer system is difficult. In addition, when the model calibration procedure involves more than simple curve fitting, there are few general-purpose techniques. We seek to overcome these obstacles by customizing the domain-specific portions of our software to modelers and by selecting mathematical software that has been tested on common problems in the domain. Experienced modelers can extend or replace the interface and computation engines when they encounter limitations with what we have preconstructed.

Although the stages involving parameter estimation included in Figure 4 are a subset of the evaluate and repair stages, we separate them because of their impact on the model development process.

$\xrightarrow{score}$ and $\xrightarrow{report}$ The score stage defines an algorithm that determines whether one set of parameters produces a more acceptable model than another set. The algorithm requires experimental data, an executable model, the range of parameters for the executable model, and a user-defined objective function. The report stage injects the fitted parameters back into the modeling process for study and testing.

The *Parameter Estimator* finds unknown rate constants by fitting the model to experimental data. The data are typically not a solution to a differential equation but rather a complicated, nonlinear functional of the differential equation solution. Furthermore, both the dependent and independent variables involved in these functionals are subject to experimental error. The Parameter Estimator performs both global and local searches during optimization.

The global optimizer, named DIRECT [26], is a variant of Lipschitzian methods for constrained global optimiza-

tion. We want to find the minimum value of our objective function, $\min_{x \in D} f(x)$, where $D = \{x \in E^n | \ell \leq x \leq \mu\}$ is defined by bound constraints only, and the objective function is Lipschitz continuous on D, satisfying $|f(x_1) - f(x_2)| \leq L\|x_1 - x_2\|$ for all $x_1, x_2 \in D$. The Lipschitz optimization method has had many practical applications in science and engineering. Unlike some other methods, the Lipschitz method requires only a few parameters and does not rely on derivatives or other more analytical information about a system. However, the Lipschitz constant of a particular function is often unknown and difficult to estimate. The DIRECT method is guaranteed to converge to the global optimum without knowledge of the Lipschitz constant [27].

The algorithm takes its name from one of its key steps: dividing rectangles (these rectangles are more commonly referred to as boxes). DIRECT is a pattern search method, which takes moves based on objective function values at a pattern of points. The points are the centers of the boxes. Center sampling is generally advantageous to corner sampling when the number of dimensions is expected to be large, as in the problems we are attempting to solve. A box is potentially optimal if there exists a value of the Lipschitz constant for which that box is the most likely to contain the global minimum. Each iteration subdivides all of the potentially optimal boxes. DIRECT can operate in an exploratory mode, which emphasizes searching untested boxes, or in an exploitation mode, which emphasizes searching boxes with better objective function values.

DIRECT is robust to noise in objective function values [26]; convergence to the minimum is limited by the amount of noise in the objective function. This makes DIRECT well suited for stochastic models, which intentionally introduce noise into the model evaluation. Current implementations of DIRECT cannot handle integer variables or constraints other than simple bound constraints. Also, DIRECT is relatively inefficient for finding an accurate value of the minimum. Rather, we would expect to run it in the exploration mode and use the local optimizer to find the minimum from a collection of candidate starting points.

The local optimizer uses ODRPACK [28–30] as the underlying mathematical software. ODRPACK does not assume that the measurement errors are all in the dependent variables [31]. Rather, it seeks to minimize the weighted sum of orthogonal distances between the model and the data. The weighting factors scale the residuals and express the modeler's confidence in the reliability of particular observations. The output of ODRPACK gives a locally optimal parameter vector and a measure of the goodness of fit of the parameter vector. We can then compare the locally optimal solutions for the starting points picked by the global optimizer.

ODRPACK uses a trust region Levenberg-Marquardt method. The Levenberg-Marquardt method starts with the steepest descent method and smoothly changes to Newton's method when approaching the solution. The trust region implementation determines the step size based on the confidence in a local model of the objective function. At each step, the optimizer compares the expected improvement for taking the step with the actual improvement. This can cause slow convergence if the objective function is not differentiable near the optimum solution since ODRPACK's expected improvement is based on estimated derivatives. Step functions in objectives can appear when matching categorical observations, such as whether a mutant is viable. However, we believe it is unlikely that optimum solutions will be located near such discontinuities in real biological systems. If that were the case, the organism would be sensitive to minute environmental changes, which is unfavorable for survival.

5.2 Evaluating JigCell

Several levels of evaluation are possible for MSEs. Microlevel studies employ formal usability testing [32, 33], which benchmarks performance for completing a task. Requirements of the modelers, frequency of use within the domain, and criticality of need determine the chosen tasks. An example of a critical task in the Model Builder is entering the kinetic information for a chemical reaction. The Model Builder could not function without supporting this task. Success or failure is determined by comparing results against a benchmark performance for the task. In formal usability testing, Nielsen [34] shows that studying three to five people finds 80% of the usability problems.

A microlevel study [35] determined JigCell's effect on error rates when converting the wiring diagram to a set of differential equations. For a collection of models, participants either constructed differential equations manually or by using the software. The number of errors in the generated sets of differential equations was then measured. The results indicate a sixfold reduction in errors over the manual method of creating differential equations. Individual tools in JigCell, as well as interactions between tools, can be studied similarly.

Macrolevel studies incorporate benchmarking and assess how well the MSE meets the specified needs of users. MSEs such as JigCell attempt to make knowledgeable users more productive and help them produce creative products. Vass, Carroll, and Shaffer [36] suggest a methodology for evaluating MSEs using flow. Flow is an automatic, effortless, and focused state of consciousness. Creativity is more likely to result from flow states [37]. Detection of flow would indicate support for creativity.

We can evaluate flow in JigCell in the following manner. By specifying assistance for each workflow in Vass, Carroll, and Shaffer [36], we classify support for problem solving and flow in JigCell. Users receive an instrumented version of JigCell; this version also sporadically prompts users to fill out a questionnaire targeting the characteristics of flow [38]. The collected data determine where flow is occurring in the user's work. If JigCell demonstrates

Table 3. Support for modeling goals in other modeling support environments

		Virtual Cell	BSP	Gepasi	Jarnac
General	Record model artifacts	Partial	Partial	Partial	Partial
Design	Verify consistent naming	Partial	Partial	Partial	Partial
	Abstract repeated operations				Full
	Abstract compound operations				Full
	Identify rate laws and constants	Full	Full	Full	Full
	Standardize notation	Full	Partial	Full	Full
Translate	Generate differential equations	Full	Partial	Partial	Partial
	Find conservation relations	Full	Partial	Full	Full
	Verify execution requirements	Full	Full	Full	Full
	Verify well-formedness	Partial	Partial	Partial	Partial

flow in all workflows and meets the formative evaluation requirements, then JigCell indicates support for creativity.

5.3 Other Cell Cycle MSEs

We have reviewed several biochemical pathway MSEs and classify them as wizards, graphical editors, or textual editors. JigCell has primarily text-based editors. Table 3 lists support for our defined modeling goals in these MSEs. Modeling goals met by none of the MSEs are omitted; the evaluate and check stages are omitted entirely as none of the reviewed MSEs directly supports these processes.

Virtual Cell [1] is a graphical editor that directly mimics the wiring diagram. Virtual Cell also supports entering models directly as differential equations. The Virtual Cell simulator supports volume and spatial models. A server-side database records models and associated simulation results with the option to download data to the user's system. Bio Sketch Pad (BSP) [39] is also a graphical editor. The simulator for BSP supports volume models and cellular automata. Gepasi [2] uses a wizard interface. A series of dialog boxes leads the user through creating a reaction network. The simulator supports volume models and can perform parameter estimation. Jarnac [8] uses a textual interface. The textual editor is a programming environment for scripting simulations and other modeling tasks. Jarnac integrates with a graphical editor, JDesigner, to view wiring diagrams. The simulator supports volume models.

6. Tool Interoperability

Tools for doing biochemical pathway modeling have been around for many years, but until recently, there has been little interoperability between them. We describe two projects that have taken as their goal support for interoperability between various biochemical pathway modeling groups. The first is SBML, and the second is DARPA's BioSPICE project (which funds the JigCell project team).

SBML [18] is an XML-based language for describing biochemical pathway models. SBML does not use a formal standards process (though the SBML community is beginning to investigate affiliating with a standards organization). The Software Platforms for Systems Biology Forum, which has been meeting semiannually since April 2000, guides development. The SBML project began with the forum's first meeting, and the definition for SBML Level 1 was published in 2001 [40]. That version supported a relatively small number of biochemical pathway modeling tools. An updated version, SBML Level 2, was published in 2003 [41]. Another language effort similar to SBML is CellML [5]. CellML is also an XML-based markup language for describing biological systems. CellML's scope is broader than SBML. While it is somewhat more mature than SBML, it has not achieved much acceptance within the biochemical pathway modeling community in the way that SBML has, probably due to its broader focus.

Before SBML, there was virtually no interchange of models between working groups. Tools were primarily built for a particular research group with little intention that the models developed within a tool would be directly transferable to other groups with other tools. Models were exchanged only through publication. With the adoption of SBML and the beginning stages of support for that language by many tools, the true exchange of models became possible. However, early tool builders primarily supported SBML through export capabilities and were not able to import SBML models produced by other research groups. SBML initially suffered from an inability to express a number of existing models, leading tool builders to adopt naming and formatting conventions to handle their special cases (see, e.g., [42]). Since these conventions were unique to each tool, few tools supported actual exchange of models. The SBML community (and, by extension, the BioSPICE community) now actively promotes a model testbed and the exchange of models between working groups, with the goal of more rigorous testing of both modeling tools and their SBML language support.

An important standards-setting decision for BioSPICE was determining the model definition language to support interchange of models within the community. After examination of the potential for defining a language for BioSPICE, the decision was made to adopt SBML.

The critical mass achieved by the BioSPICE community joining with the already existing SBML community has had a profound impact on the SBML development process. Initially, SBML could be characterized as defining a language that was the intersection of a small group of similar tools. Most users of SBML converted their models to systems of ordinary differential equations, perhaps with discrete events. Current efforts are moving in the direction of defining a language that is the union of a larger group of more disparate tools. This means support for a broader class of models, including stochastic simulations and spatial models. Structural model analyses such as flux balance analysis and bifurcation analysis are also anticipated. SBML is already at the stage where no single simulation tool will support the full range of features that SBML models can have. Some balance must be struck between the different types of modeling needs, the difficulties encountered by tool builders, and the risk of the language becoming so large that it is no longer truly a mechanism for common exchange.

The BioSPICE community charged itself with designing and building a toolset that could both support current modeling efforts and serve as a base for future modeling efforts. Since it is unlikely that any single tool, or small collection of tools, will satisfy the wide variety of modelers and modeling efforts, any specific task has numerous tools available. Thus, the BioSPICE community is engaged in two types of standardization efforts for tools: data formats and programmatic interfaces.

The exchange of formatted data within the BioSPICE community is difficult because a large number of data types are of interest to at least some part of the community. Popular data types, such as models and time series, were quickly identified and standardized by the community as high-priority items. However, there is also interest in exchanging data for gene expression, molecular interactions, imaging, protein mass spectroscopy, kinetics, flow cytometrics, and flux balance. The standards creation process is driven by the community and mostly distributed. There is a clearinghouse for archiving agreed-on standards, but there is no corresponding authority for creating standards. Tools must have some internal format for representing data. Initially, this is their standard for that data format. If no one else is interested in using this type of data, there is no need to promulgate the standard. However, another group interested in using the same type of data will have its own internal format. At some point, there is a desire to interchange data between the tools. The groups then meet and decide on a commonly acceptable interchange format. When a format achieves a plurality within its domain, it is recommended to the community as a standard for adoption.

Although data formats solve the problem of static interactions between tools, they cannot coordinate computations between tools. The number of possible tool-tool interactions is too great to define individual standards. Instead, standards are defined for the most commonly performed interactions, and a formal tool description language is proposed for describing tool capabilities not captured by one of the standards. Common interactions include tasks such as performing a simulation and parameter estimation. The tool description language is referred to as a meta-interface: it is intended that any particular tool be describable in terms of operations provided by the language. The operations are functions applied to the data types standardized by the community. However, this tool description language only specifies the syntax for tools, not the semantics. It is possible to connect tools that have compatible data types but a nonsensical meaning to their combined computation.

7. Conclusions

We have described our experiences documenting and improving the modeling process of a group of theoretical biologists. The revised modeling process is based on the observed process and incorporates a disciplined methodological approach along with proven techniques for reducing the cost of errors, reducing development time, and making iterations of the modeling process more consistent. The primary improvements come from introducing multiple forms of testing throughout the modeling life cycle and by making modelers explicitly address previously undocumented modeling tasks. The modeling support environment we have built for this revised process meets many of our defined modeling goals and is testable for its effects on the modeling process. Further case studies of modelers using this process will give clear guidance for future improvements, along with the unfulfilled modeling goals.

Further progress in pathway modeling desperately needs new tools and a better modeling process as described here. The BioSPICE community estimates that the size and complexity of existing models (which are about as large as the modeling process prior to what BioSPICE could handle) must grow by two orders of magnitude to capture the control mechanisms of important processes in mammalian cells. Currently, manual parameter estimation and manual error diagnosis consume a significant portion of the model development time. We hope that by automating these tasks, tools such as JigCell and others developed as part of BioSPICE can supply one of the two needed orders of magnitude. Supporting the additional complexity required is an open question.

8. Acknowledgments

Thanks are due to John Tyson and members of his modeling laboratory for making themselves available for study and reviewing our observations of their modeling process.

The work reported herein was partly sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory (AFRL), Air Force Materiel Command, USAF, under agreement number F30602-02-0572. The U.S. government is authorized to reproduce and distribute reprints for governmental purposes

notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, AFRL, or the U.S. government.

9. References

[1] Loew, Leslie M., and Jim C. Schaff. 2001. The Virtual Cell: A software environment for computational cell biology. *Trends in Biotechnology* 19:401-6.

[2] Mendes, Pedro. 1997. Biochemistry by numbers: Simulation of biochemical pathways with Gepasi 3. *Trends in Biochemical Sciences* 22:361-3.

[3] Sauro, Herbert M. 2000. *Animating the cellular map: Proceedings of the 9th International Meeting on BioThermoKinetics*. Stellenbosch, South Africa: Stellenbosch University Press.

[4] BioSPICE. 2003. The BioSPICE Development Project [Online]. Available: http://www.biospice.org

[5] Hedley, Warren J., Melanie R. Nelson, et al. 2001. CellML specification [Online]. Available: http://www.cellml.org/public/specification/20010810/

[6] Hucka, Michael, Andrew Finney, Herbert M. Sauro, Hamid Bolouri, et al. 2003. The Systems Biology Markup Language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics* 19 (4): 524-31.

[7] Nance, Richard E. 1994. The conical methodology and the evolution of simulation model development. *Annals of Operations Research* 53:1-45.

[8] Shampine, Lawrence F., and Marilyn K. Gordon. 1975. *Computer solution of ordinary differential equations: The initial value problem*. San Francisco: W. H. Freeman.

[9] Hindmarsh, Alan C. 1983. ODEPACK: A systematized collection of ODE solvers. In *Scientific computing*, edited by R. S. Stepleman, 55-64. Amsterdam: North Holland.

[10] Tyson, John J., and Bela Novak. 2001. Regulation of the eukaryotic cell cycle: Molecular antagonism, hysteresis, and irreversible transitions. *Journal of Theoretical Biology* 210:249-63.

[11] Balci, Osman. 1998. Verification, validation, and accreditation. In *Proceedings of the 30th Conference on Winter Simulation*, pp. 41-8.

[12] Marlovits, Gabor, Christopher J. Tyson, Bela Novak, and John J. Tyson. 1998. Modeling M-phase control in *Xenopus* oocyte extracts: The surveillance mechanism for unreplicated DNA. *Biophysical Chemistry* 72:169-84.

[13] Balci, Osman, Richard E. Nance, E. Joseph Derrick, Ernest H. Page, and John L. Bishop. 1990. Model generation issues in a simulation support environment. In *Proceedings of the 22nd Conference on Winter Simulation*, pp. 257-63.

[14] Nance, Richard E., and James D. Arthur. 1988. The methodology roles in the realization of a model development environment. In *Proceedings of the 20th Conference on Winter Simulation*, pp. 220-5.

[15] Arthur, James D., and Richard E. Nance. 2000. Verification and validation without independence: A recipe for failure. In *Proceedings of the 32nd Conference on Winter Simulation*, pp. 859-65.

[16] Balci, Osman. 1986. Credibility assessment of simulation results. In *Proceedings of the 18th Conference on Winter Simulation*, pp. 38-44.

[17] Kohn, Kurt W. 1999. Molecular interaction map of the mammalian cell cycle control and DNA repair system. *Molecular Biology of the Cell* 10 (8): 2703-34.

[18] Hucka, Michael, Andrew Finney, et al. 2003. SBML Systems Biology Markup Language [Online]. Available: http://www.sbml.org/

[19] Balci, Osman, Richard E. Nance, James D. Arthur, and William F. Ormsby. 2002. Expanding our horizons in VV&Aresearch and practice. In *Proceedings of the 34th Conference on Winter Simulation*, pp. 653-63.

[20] Wright, Sammuel A., and Kenneth W. Bauer. 1997. Covalidation of dissimilarly structured models. In *Proceedings of the 29th Conference on Winter Simulation*, pp. 311-8.

[21] Ramakrishnan, Naren, Layne T. Watson, Dennis G. Kafura, Cal J. Ribbens, and Clifford A. Shaffer. 2002. Programming environments for multidisciplinary Grid communities. *Concurrency and Computation: Practice and Experience* 14:1241-73.

[22] Watson, Layne T., Vinod K. Lohani, David F. Kibler, Randy L. Dymond, Naren Ramakrishnan, and Clifford A. Shaffer. 2002. Integrated computing environments for watershed management. *Journal of Computational Civil Engineering* 16:259-68.

[23] Balmer, David W., and Ray J. Paul. 1990. Integrated support environments for simulation modelling. In *Proceedings of the 22nd Conference on Winter Simulation*, pp. 243-9.

[24] Mathewson, S. C. 1989. *Computer modelling for discrete event simulation*. New York: John Wiley.

[25] Allen, Nicholas A., Marc T. Vass, Jason W. Zwolak, et al. 2003. The eukaryotic cell cycle collaborative problem-solving environment group [Online]. Available: http://gnida.cs.vt.edu/∼cellcyclepse/

[26] He, Jian, Layne T. Watson, Naren Ramakrishnan, Clifford A. Shaffer, Alex Verstak, Jing Jiang, Kyung Bae, and William H. Tranter. 2002. Dynamic data structures for a direct search algorithm. *Computational Optimization and Applications* 23:5-25.

[27] Jones, D. R., Cary D. Perttunen, and Bruce E. Stuckman. 1993. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications* 79 (1): 157-81.

[28] Boggs, Paul T., Richard H. Byrd, Janet R. Donaldson, and Robert B. Schnabel. 1989. Algorithm 676—ODRPACK: Software for weighted orthogonal distance regression. *ACM Transactions on Mathematical Software* 15 (4): 348-64.

[29] Boggs, Paul T., Richard H. Byrd, Janet E. Rogers, and Robert B. Schnabel. 1992. *User's reference guide for ODRPACK version 2.01: Software for weighted orthogonal distance regression*. Gaithersburg, MD: Center for Computing and Applied Mathematics, U.S. Department of Commerce.

[30] Boggs, Paul T., Richard H. Byrd, and Robert B. Schnabel. 1987. A stable and efficient algorithm for nonlinear orthogonal distance regression. *SIAM Journal on Scientific and Statistical Computing* 8 (6): 1052-78.

[31] Carroll, Ray J., David Ruppert, and Lenny A. Stefanski. 1995. *Measurement error in nonlinear models*. New York: Chapman & Hall/CRC.

[32] Hix, Debbie, and H. Rex Hartson. 1993. *Developing user interfaces: Ensuring usability through product & process*. New York: John Wiley.

[33] Schneiderman, Ben. 1998. *Designing the user interface: Strategies for effective human-computer-interaction*. Reading, MA: Addison Wesley Longman.

[34] Nielsen, Jakob. 1994. Heuristic evaluation. In *Usability inspection methods*, edited by Jakob Nielsen and Robert L. Mack, 25-62. New York: John Wiley.

[35] Vass, Marc T., and Pete Schoenhoff. 2002. Error detection support in a cellular modeling end-user programming environment. In *IEEE 2002 Symposia on Human Centric Languages and Environments*, pp. 104-6.

[36] Vass, Marc T., John M. Carroll, and Clifford A. Shaffer. 2002. Supporting creativity in problem solving environments. In *Proceedings of the Fourth Creativity & Cognition Conference*, pp. 31-7.

[37] Csikszentmihalyi, Mihaly. 1990. *The psychology of optimal experience*. New York: Harper & Row.

[38] Webster, Jane, Linda K. Trevino, and Lisa Ryan. 1993. The dimensionality and correlates of flow in human computer interactions. *Computers in Human Behavior* 9 (4): 411-26.

[39] Webb, Johnathan. 2002. Bio Sketch Pad [Online]. Available: http://bio.bbn.com/biospice/biosketchpad

[40] Hucka, Michael, Andrew Finney, Herbert M. Sauro, Hamid Bolouri, John C. Doyle, Hiroaki Kitano. 2001. Systems

Biology Markup Language (SBML) Level 1: Structures and facilities for basic model definitions [Online]. Available: http://www.sbml.org/specifications/sbml-level-1/version-2/sbml-level-1-v2.pdf
[41] Finney, Andrew, Michael Hucka, John C. Doyle, Hiroaki Kitano. 2003. Systems Biology Markup Language (SBML) Level 2: Structures and facilities for basic model definitions [Online]. Available: http://www.sbml.org/specifications/sbml-level-2/version-1/sbml-level-2.pdf
[42] Mendes, Pedro. 2002. Gepasi and SBML [Online]. Available: http://www.gepasi.org/gep3sbml.html

Nicholas A. Allen *is a PhD candidate in the Department of Computer Science at Virginia Tech (VPI&SU). He received BS degrees (magna cum laude) in mathematics and computer science from Virginia Tech in 1999 and MS degrees in mathematics and computer science from Virginia Tech in 2001. His research interests include software architecture and design, distributed computing, graph theory, and mathematical software.*

Clifford A. Shaffer *has been an associate professor in the Department of Computer Science at Virginia Tech since 1987. He received his PhD from the University of Maryland in 1986. His current research interests include problem-solving environments, bioinformatics, component architectures, visualization, algorithm design and analysis, and data structures.*

Naren Ramakrishnan *is an assistant professor of computer science at Virginia Tech. His research interests include problem-solving environments, mining scientific data, and personalization. He received his PhD in computer sciences from Purdue University.*

Marc T. Vass *is a PhD candidate in the Department of Computer Science at Virginia Tech. He received the BS degree in computer science from Virginia Tech in 2000 and the MS degree in computer science from Virginia Tech in 2001. His research interests include creativity and cognition, problem-solving environments, flow, and theoretical human computer interaction.*

Layne T. Watson *is a professor of computer science and mathematics at Virginia Tech. His research interests include fluid dynamics, structural mechanics, numerical algorithms, parallel computation, mathematical software, and image processing. He has worked for USNAD Crane, Sandia National Laboratories, and General Motors Research Laboratories and has served on the faculties of the University of Michigan and Michigan State University, East Lansing, before coming to Virginia Tech. He received his BA (magna cum laude) in psychology and mathematics from the University of Evansville, Indiana, and his PhD in mathematics from the University of Michigan, Ann Arbor.*