Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

IMPROVING THE DEVELOPMENT PROCESS FOR EUKARYOTIC CELL CYCLE MODELS WITH A MODELING SUPPORT ENVIRONMENT

Nicholas A. Allen
Clifford A. Shaffer
Marc T. Vass
Naren Ramakrishnan
Layne T. Watson

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106, U.S.A.

ABSTRACT

Biological pathway modelers attempt to describe cellular processes and regulatory networks with continuous and discrete models of the cell cycle. Previous practice has been to develop these models largely by hand, and then to validate models primarily by comparing time series plots versus the observed experimental results. This paper reports our experiences in designing and building a modeling support environment (MSE) for cell cycle models. We describe improvements to the development process for cell cycle models by (a) identifying the key elements of the existing modeling process, (b) applying simulation methodology to construct a revised modeling process, and (c) building and testing software that supports the revised modeling process.

1 INTRODUCTION

Biological pathway models attempt to reproduce observed phenomenon in cellular processes such as the concentrations of proteins, and gross behavior of cells such as mass at division or phase of death in the cell cycle. 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. Many groups have been developing tools to support aspects of the modeling process (for example, Sauro 2000, Mendes 1997, Loew and Schaff 2001). BioSPICE (BioSPICE 2003) 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 (Hucka et al. 2003) and interoperability between various model description editors, simulators, and analysis tools.

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 convert those sketches to differential equations. Analysis often involves visually comparing time series plots and experimentally collected results.

This paper describes how introducing appropriate modeling tools can improve the speed and accuracy of the development process (which directly permits the creation of larger, more complex models), and also can lead to a more disciplined approach to the model lifecycle. The original modeling process observed in a portion of the community is described in Section 2. Section 3 describes how the conical methodology (Nance 1994) relates to the observed modeling process, and can be used to improve the modeling lifecycle. Section 4 briefly describes the JigCell modeling support environment (MSE) developed for cell cycle modeling and related problems. Section 5 presents our conclusions.

2 ORIGINAL MODELING PROCESS

Figure 1 shows the modeling process observed in the Virginia Tech laboratory of John Tyson, a well-known member of the pathway modeling community. This process, while not using the most advanced modeling tools available, has permitted them to develop some of the most sophisticated pathway models in the current literature (Tyson and Novak 2001). The process evolved over more than 10 years of developing models. It is not based on formalisms nor documented. New modelers learn the process through demonstration and mentoring. The tools are primarily off-the-shelf components and not specialized for pathway modeling.

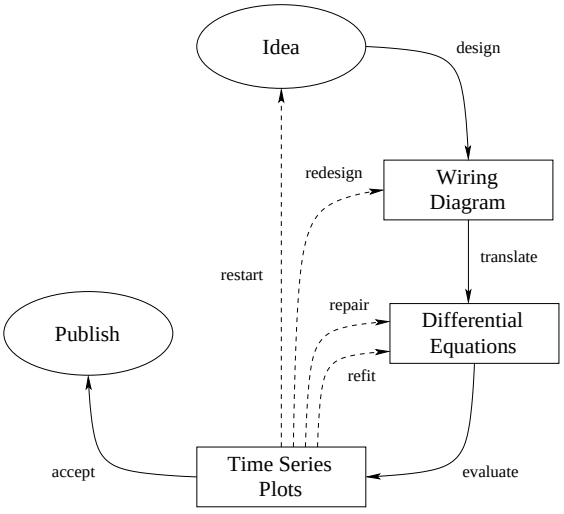

Figure 1: Original Modeling Process

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 (Balci 1998). 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 two 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?

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. In this paper, 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.

$\xrightarrow{design}$ 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. Additionally, the wiring diagram may show the kinetics for a reaction, describing how and at what rate the reaction takes place. Figure 2 is a wiring diagram based on the model of Marlovits et al. (1998).

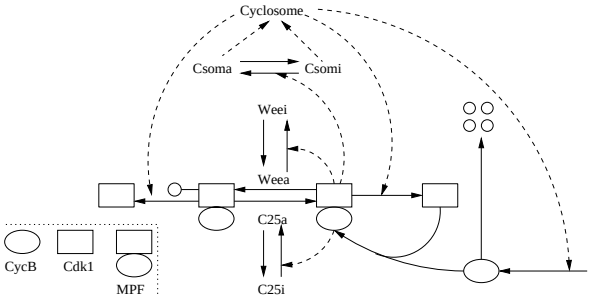

Figure 2: Wiring Diagram

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, our modelers 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 equation. 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, or switch between sets of differential equations driving the continuous model.

$\xrightarrow{evaluate}$ 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 mass at division or phase of death in the cell cycle.

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

The remaining stages in the original modeling process are error recovery stages. Errors are detected because of informal testing from the time series plots. The modeler must infer the nature and location of the error from experience looking at plots. Because the systems are typically underspecified, the modeler cannot always accurately identify the cause of an error without laboratory testing (which is expensive).

$\xrightarrow{redesign}$ The redesign stage corrects errors in the wiring diagram. Reactions are added and deleted based on the modeler's developing intuition about the mechanisms that 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 wiring diagram and differential equations. Manually creating differential equations is a time-consuming and error-prone task. 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 comparison 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.

3 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 their current methodology can support, which is driving many new efforts in tool development such as BioSPICE. One purpose of methodologies is to assist understanding the model development process and indicate requirements for supporting that process (Balci et al. 1990). Formal methodological approaches provide well-defined and tested techniques for the model development process.

Based on our experience with the original modeling process, we enumerate capabilities that this modeling community needs from a methodology. Modelers have the 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 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 the high performance computing techniques of the next 10 years. Using the terminology of Nance and Arthur (1988), our modeling process must primarily support correctness and testability, secondarily support adaptability, maintainability, and portability, and permit testing throughout the model lifecycle.

We select the conical methodology (Nance 1994) as a methodology supportive of our requirements and with a sufficient adaptability to capture our original and desired modeling processes. Balci (1998) describes a model lifecycle 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 (Nance and Arthur 1988). 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 domain-specific constructs in our tools.

3.1 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 improvement: 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 (Arthur and Nance 2000). 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 is the main testing technique for validating these models. However, the quantity and quality of experimental data available for a particular system may be limited. Conducting new laboratory experiments for further testing or expansion of the model is a major expense; modeler time is cheap in comparison. We emphasize verification during model construction to prevent the introduction of errors that strain our 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 wish to codify indicators of model credibility (Balci 1986) as automated tests to 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 (1999) 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 (Hucka et al. 2003). 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. SBML's main purpose is facilitating model exchange between modeling groups, who will then load the models into ad hoc editing tools. Biological models can have subcomponents duplicated within a model 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 planning.

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 (Balci et al. 2002). 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 at what stage the error was introduced and in what component of the model the error is located. When the user is modifying the model, testing should be automatically conducted to give feedback on the relative performance of the modification.

3.2 Revised Modeling Process

The revised modeling process, Figure 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, is self-consistent, and is 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

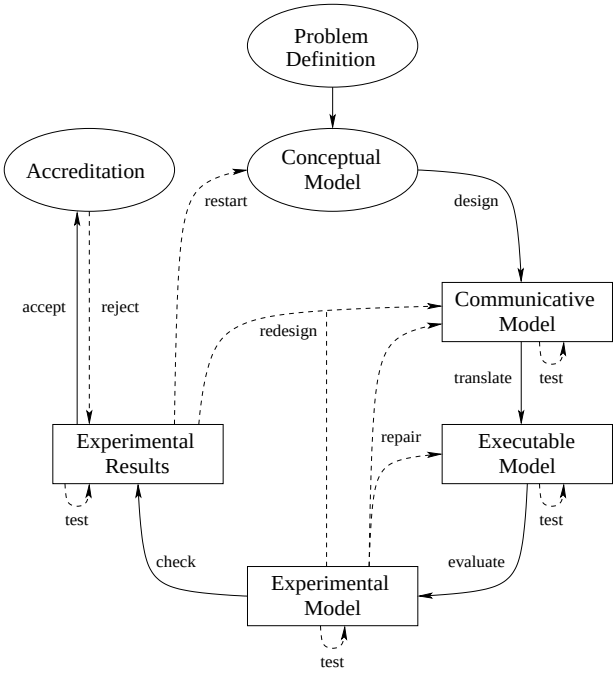

Figure 3: Revised Modeling Process

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 testing process (Balci et al. 2002). Entering the problem definition only once and automating the testing process from previous iterative cycles of the modeling process are important for reducing the model development time. Additionally, there should be provisions for independent execution of the evaluate stage to prevent modeler bias in testing.

$\xrightarrow{check}$ A model that meets the requirements and objectives stated in the problem might still be rejected. In the biological domain, two reasons 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 representing similar processes can test the first (Wright and Bauer 1997). Performing a statistical analysis between the proposed model and a collection of models for the same system can test the second.

$\xrightarrow{accept}$ 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 (Balci 1998). Additionally, 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. Additionally, 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.

4 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. (2002) and Watson et al. (2002). JigCell's user workflow, Figure 4, corresponds closely with the modeling process we have identified. Table 1 lists support for the modeling goals of Section 3.1 in this MSE.

We have constructed JigCell as a tailored environment rather than basing it on an existing, general-purpose MSE. JigCell made extensive use of participatory design interaction with John Tyson, the leader of the modeling group. We intend to support experts in biology and related fields who do not have significant experience in formal modeling. We incorporate off-the-shelf components such as numerical libraries, visualization tools, and communications protocols where 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 modeling infrastructure.

The **Model Builder** creates a model specification that incorporates structural information, kinetic information, and the discrete event model. A spreadsheet interface organizes the structural and kinetic information as a collection of chemical reaction equations. 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

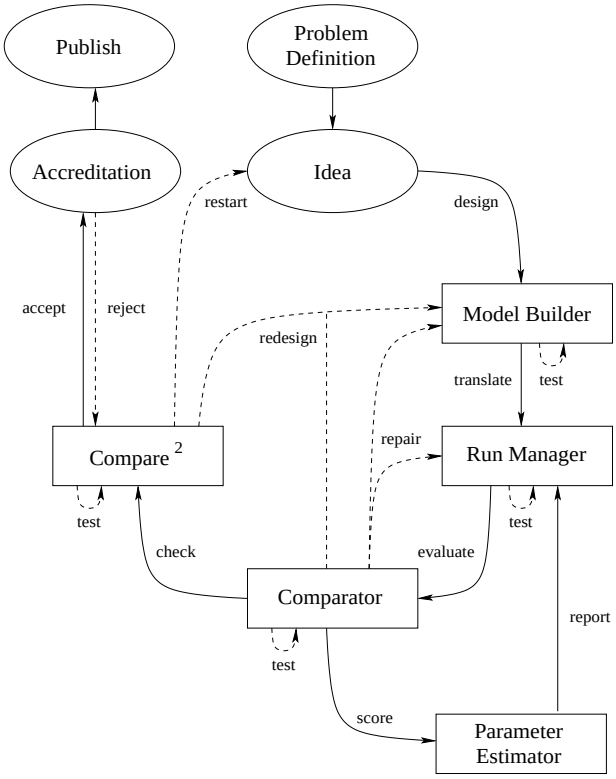

Figure 4: JigCell User Workflow

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. The model builder both reads and writes its models in the form of SBML, which is the standard interchange language for the 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 yet for testing overall model structure, and limited support is so far provided for representing stochastic and spatial models, which represent specific sub-domains within the modeling community that we intend to support. 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. Incorporation of black-box subcomponents is 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. 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

Table 1: 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

translation insulates modelers from changes in simulation techniques and the runtime environment. 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. Objective functions associate model performance with a degree of accuracy rather than a binary result. 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. The modeler can automatically rerun a defined testing procedure in the Comparator.

Tests in Compare^2 compare performance between the currently proposed model and a collection of other models. 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. A drawback of Compare^2 is that obtaining meaningful results requires building a significant collection of models. We hope to in-

corporate 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 manually search for valid and optimal regions of the parameter space. This activity has consumed a major part of the model development time in the past. It should not be performed by humans at all.

Although the stages involving parameter estimation shown 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 scoring algorithm requires experimental data, an executable model, a definition of the parameter space, and a metric for evaluating model quality. 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, DIRECT (He et al. 2002), is a variant of Lipschitzian methods for constrained global optimization. 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 (Jones, Perttunen, and Stuckman 1993). DIRECT can operate in an exploratory mode, which emphasizes searching untested regions of parameter space, or in an exploitation mode, which emphasizes searching regions with better objective function values. 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 points.

The local optimizer uses ODRPACK (Boggs et al. 1989) as the underlying mathematical software. ODRPACK does not assume that the measurement errors are all in the dependent variables. It instead 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. 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 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.

4.1 Evaluating JigCell

Several levels of evaluation are possible for MSEs. Micro-level studies employ formal usability testing (Hix and Hartson 1993), 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 performance benchmark.

A micro-level study determined JigCell's effect on error rates converting the wiring diagram to a set of differential equations (Vass and Schoenhoff 2002). For a collection of models, participants either constructed differential equations manually or using the software. The number of errors in the generated sets of differential equations was then measured. The results indicate a six-fold reduction in errors over the manual method of creating differential equations. Other tasks in JigCell can be studied similarly.

Macro-level 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 (2002) 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 (Csikszentmihalyi 1990). Detection of flow would indicate support for creativity.

The following procedure evaluates flow in JigCell. We classify support for problem solving and flow in JigCell by specifying assistance for each workflow of Vass, Carroll, and Shaffer (2002). Users receive an instrumented version of JigCell that periodically prompts users to fill out a questionnaire targeting the characteristics of flow (Webster, Trevino, and Ryan 1993). The collected data determines where flow is occurring in the user's work. If JigCell demonstrates flow in all workflows and meets the formative evaluation requirements, then JigCell indicates support for creativity.

5 CONCLUSIONS

We have described our experiences documenting and improving the process of a group of biological modelers.

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 JigCell MSE we have built for this revised process meets many of our defined modeling goals and is testable for its effects on the modeling process. Unfulfilled modeling goals and the results of testing will give clear guidance for future modeling process improvements.

Further progress in the field desperately needs such improvements 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 BioSPICE could handle) must grow by two orders of magnitude to capture the control mechanisms of important processes in mammalian cells. We hope that the tools currently envisioned by BioSPICE can supply one of the two needed orders of magnitude. Supporting the additional complexity required is an open question. However, the results of the currently planned improvements should permit modelers to greatly advance the state of their art.

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.

REFERENCES

Arthur, J. D., and R. E. Nance. 2000. Verification and validation without independence: a recipe for failure. In *Proceedings of the 32nd Conference on Winter Simulation*, ed. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 859–865. Piscataway, NJ: IEEE.

Balci, O. 1986. Credibility assessment of simulation results. In *Proceedings of the 18th Conference on Winter Simulation*, ed. J. Wilson, J. Henriksen, and S. Roberts, 38–44. Piscataway, NJ: IEEE.

Balci, O. 1998. Verification, validation, and accreditation. In *Proceedings of the 30th Conference on Winter Simulation*, ed. D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, 41–48. Piscataway, NJ: IEEE.

Balci, O., R. E. Nance, J. D. Arthur, and W. F. Ormsby. 2002. Expanding our horizons in VV&A research and practice. In *Proceedings of the 34th Conference on Winter Simulation*, ed. E. Yucesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, 653–663. Piscataway, NJ: IEEE.

Balci, O., R. E. Nance, E. J. Derrick, E. H. Page, and J. L. Bishop. 1990. Model generation issues in a simulation support environment. In *Proceedings of the 22nd Conference on Winter Simulation*, ed. O. Balci, R. P. Sadowski, and R. E. Nance, 257–263. Piscataway, NJ: IEEE.

BioSPICE 2003. The BioSPICE development project. Available online via `http://www.biospice.org` [accessed February 4, 2003].

Boggs, P. T., R. H. Byrd, J. R. Donaldson, and R. B. Schnabel. 1989. Algorithm 676 - ODRPACK: Software for weighted orthogonal distance regression. *ACM Transactions on Mathematical Software* 15 (4): 348–364.

Csikszentmihalyi, M. 1990. *The psychology of optimal experience*. New York, NY: Harper & Row.

He, J., L. T. Watson, N. Ramakrishnan, C. A. Shaffer, A. Verstak, J. Jiang, K. Bae, and W. H. Tranter. 2002. Dynamic data structures for a direct search algorithm. *Computational Optimization and Applications* 23:5–25.

Hix, D., and H. R. Hartson. 1993. *Developing user interfaces: Ensuring usability through product & process*. New York, NY: John Wiley & Sons, Inc.

Hucka, M., A. Finney, H. M. Sauro, H. Bolouri et al. 2003. The Systems Biology Markup Language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics* 19 (4): 524–531.

Jones, D. R., C. D. Perttunen, and B. E. Stuckman. 1993. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications* 79 (1): 157–181.

Kohn, K. W. 1999. Molecular interaction map of the mammalian cell cycle control and DNA repair system. *Molecular Biology of the Cell* 10 (8): 2703–2734.

Loew, L. M., and J. C. Schaff. 2001. The Virtual Cell: A software environment for computational cell biology. *Trends in Biotechnology* 19:401–406.

Marlovits, G., C. J. Tyson, B. Novak, and J. J. Tyson. 1998. Modeling M-phase control in Xenopus oocyte extracts: the surveillance mechanism for unreplicated DNA. *Biophysical Chemistry* 72:169–184.

Mendes, P. 1997. Biochemistry by numbers: Simulation of biochemical pathways with Gepasi 3. *Trends in Biochemical Sciences* 22:361–363.

Nance, R. E. 1994. The conical methodology and the evolution of simulation model development. *Annals of Operations Research* 53:1–45.

Nance, R. E., and J. D. Arthur. 1988. The methodology roles in the realization of a model development environment. In *Proceedings of the 20th Conference on Winter Simulation*, ed. M. Abrams, P. Haigh, and J. Comfort, 220–225. Piscataway, NJ: IEEE.

Ramakrishnan, N., L. T. Watson, D. G. Kafura, C. J. Ribbens, and C. A. Shaffer. 2002. Programming environments for multidisciplinary grid communities. *Concurrency: Practice and Experience* 14:1241–1273.

Sauro, H. M. 2000. Jarnac: A system for interactive metabolic analysis. In *Animating the Cellular Map: Proceedings of the 9th International Meeting on Bio-ThermoKinetics*, ed. J.-H. S. Hofmeyr, J. M. Rohwer, and J. L. Snoep, 221–228: Stellenbosch University Press.

Tyson, J. J., and B. Novak. 2001. Regulation of the eukaryotic cell cycle: Molecular antagonism, hysteresis, and irreversible transitions. *Journal of Theoretical Biology* 210 (2): 249–263.

Vass, M. T., J. M. Carroll, and C. A. Shaffer. 2002. Supporting creativity in problem solving environments. In *Proceedings of the Fourth Creativity & Cognition Conference*, ed. J. Mottram, L. Candy, and T. Kavanagh, 31–37. New York, NY: ACM Press.

Vass, M. T., and P. Schoenhoff. 2002. Error detection support in a cellular modeling end-user programming environment. In *IEEE 2002 Symposia on Human Centric Languages and Environments*, 104–106. Piscataway, NJ: IEEE.

Watson, L. T., V. K. Lohani, D. F. Kibler, R. L. Dymond, N. Ramakrishnan, and C. A. Shaffer. 2002. Integrated computing environments for watershed management. *Journal of Computational Civil Engineering* 16:259–268.

Webster, J., L. Trevino, and L. Ryan. 1993. The dimensionality and correlates of flow in human computer interactions. *Computers in Human Behavior* 9 (4): 411–426.

Wright, S. A., and K. W. Bauer. 1997. Covalidation of dissimilarly structured models. In *Proceedings of the 29th Conference on Winter Simulation*, ed. S. Andradottir, K. J. Healy, D. H. Withers, and B. L. Nelson, 311–318. Piscataway, NJ: IEEE.

AUTHOR BIOGRAPHIES

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. His email address is <nallen@acm.org>.

CLIFFORD A. SHAFFER is an associate professor in the Department of Computer Science at Virginia Tech since 1987. He received his PhD from University of Maryland in 1986. His current research interests include problem solving environments, bioinformatics, component architectures, visualization, algorithm design and analysis, and data structures. His email address is <shaffer@vt.edu>.

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. His email address is <mvass@vt.edu>.

NAREN RAMAKRISHNAN is an assistant professor of computer science at Virginia Tech. His research interests include problem solving environments, mining scientific data, and personalization. Ramakrishnan received his Ph.D. in computer sciences from Purdue University. Contact him at <naren@cs.vt.edu>.

LAYNE T. WATSON is a professor of computer science and mathematics at Virginia Tech. His research interests include fluid dynamics, structural mechanics, homotopy 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, Ind., and his PhD in mathematics from the University of Michigan, Ann Arbor. His email address is <ltw@cs.vt.edu>.