Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

BUILDING MODELING TOOLS THAT SUPPORT VERIFICATION, VALIDATION, AND TESTING FOR THE DOMAIN EXPERT

Nicholas A. Allen
Clifford A. Shaffer
Layne T. Watson

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

ABSTRACT

Successfully building a model requires a combination of expertise in the problem domain and in the practice of modeling and simulation (M&S). Model verification, validation, and testing (VV&T) are essential to the consistent production of models that are useful and correct. There are significant communities of domain experts that build and use models without employing dedicated modeling specialists. Current modeling tools relatively underserve these communities, particularly in the area of model testing and evaluation. This paper describes several techniques that modeling tools can use to support the domain expert in performing VV&T, and discusses the advantages and disadvantages of this approach to modeling.

1 INTRODUCTION

The successful construction of a model requires combining expertise in the domain being modeled with expertise in the practice of building models. Many domain experts that practice modeling have at least some background knowledge in the area of modeling and simulation (M&S). This knowledge is gained somewhat through formal learning, but mostly from the experience of building models in the problem domain.

Although domain experts are often not modeling specialists, they can successfully complete modeling projects by using the right tools. Modeling tools are software programs built from the knowledge and experiences of modeling specialists that assist with M&S tasks. The domain expert draws upon these resources by using modeling tools to solve a problem in their domain. Unfortunately, current modeling tools relatively underserve these communities, particularly in the area of model testing and evaluation.

Model verification, validation, and testing activities are often referred to jointly as model VV&T. These activities assess the accuracy of a model. The practice of VV&T is essential to the consistent production of models that are useful and correct. Balci (1998) and Sargent (2004) provide a comprehensive overview of model VV&T.

Verification is the process of certifying that the fidelity of a model is maintained when the model is transformed from one form to another. This ensures that the model is transformed as intended, and that the model's accuracy is preserved over time. Model verification checks the process of building the model.

Validation is the process of determining whether a model sufficiently approximates the real system. The definition of "sufficiently" depends upon the purpose of the model. Increasing the validity of a model has cost, so it is most efficient to evaluate the model with respect to its intended application (Balci and Sargent 1981). Therefore, the purpose of the model dictates which aspects are important to validate, and what standards to apply.

Testing is the process of checking for errors in the model. Model testing determines if the model is functioning properly by subjecting the model to controlled inputs. A model test is designed to perform verification, validation, or both activities.

Domain experts often do not perform sufficient model VV&T due to a lack of accessible support for these activities in modeling tools. An increase in model VV&T would improve the likelihood of success for modeling efforts. This paper describes several techniques by which modeling tools can be adapted to better support the domain expert. Section 2 presents several usage scenarios in which domain experts find themselves needing to perform model VV&T. Section 3 describes techniques that modeling tools can use to support the domain expert. Section 4 gives potential consequences of having a domain expert perform VV&T. Concluding remarks are given in Section 5.

2 USAGE SCENARIOS

The target audience for general purpose modeling tools is often the modeling specialist. These tools can be inadequate for the domain expert because of assumptions made for the target community. Assumptions include the languages used for expressing models and modeling concepts, support for features not relevant to the expert's problem domain, and lack of modeling guidance for the tool user.

A software modeling tool is unlikely to accommodate domain expert users unless the software designer includes domain experts as a user class. If actual user representatives are not available, it is helpful to construct personas that typify how a user class might use a software product (Kujala and Kauppinen 2004). Software builders can employ these personas when considering how development decisions will impact users of the modeling tool. We include here a small sample of scenarios that demonstrate domain experts performing VV&T.

Scenario 1: The domain expert wants to use a model created by someone else, and needs to check that the model is suitable.

Abel is a biochemist trying to understand signaling in a tissue culture. He has researched the literature for suitable models, and has obtained electronic versions of several models that appear interesting. The documentation does not include enough details about how the models were validated and tested, so Abel wants to make sure that their performance is acceptable for his application. Also, Abel knows that integrating the published models together is likely to be an error-prone process, so he will need to verify that the integration is performed correctly.

Scenario 2: The domain expert is acting as an independent agent to perform verification and validation activities on a model under development.

Joan is a process engineer with experience in semiconductor fabrication. A manufacturer is using a simulation study to reduce the cycle time of one of their products, and has hired Joan to help validate the models that they are creating. Joan will perform face validation (review of simulation output for reasonableness), and participate in model reviews and walkthroughs. She will need to understand how the models work, and test them by performing experiments.

Scenario 3: The domain expert has decided to develop a model without the assistance of a modeling specialist.

Steven is a physicist working on a new theory of particle interaction. It is difficult to predict some of the consequences of his theory, so Steven has decided to use a discrete event model to test his intuition. This is a side project for Steven with little budget so he decides to build the model himself rather than hiring a modeling specialist. Table 1 gives a variety of other reasons that Steven may have for building the model himself. The outcome of the simulation is a little different than Steven had expected, but now he is unsure if the error is in his model or his intuition.

Table 1: Why Steven Didn't Employ a Modeling Specialist

- Cost of employing specialist
- Difficulty of finding specialist
- Value of employing specialist not understood
- Model turnaround time
- Confidentiality or secrecy of information
- Unable to transfer domain knowledge or requirements
- Modeling objectives in flux
- Prototype or experimental product
- Limited scale or scope of modeling project
- Expert learning from model behavior or construction process

Each scenario describes a domain expert performing VV&T. However, the users all have different M&S goals. Abel is attempting to reuse an existing model. Joan is part of a larger simulation study with a particular business objective. Steven is constructing a new model. The scenarios for using a modeling tool determine which techniques will help support the domain expert. Therefore, it is important to consider how the modeling tool will be used when deciding whether to incorporate a particular technique.

3 SUPPORTING THE DOMAIN EXPERT

Many techniques exist for better adapting modeling tools to a domain expert's needs. The focus here is on techniques that aid the performance of VV&T. Table 2 summarizes the techniques included in this paper. This list is not intended to be comprehensive. The authors have experimented with each of these techniques as part of the simulation study described by Allen et al. (2003A) and Allen et al. (2003B). Many other modeling tools have also employed these techniques, with a positive effect towards supporting the domain expert.

We estimate the impact, applicability, and difficulty of using a technique based on our past experiences. There are problem domains for which these estimates will be less accurate. For example, the use of domain terms and concepts for models has a difficulty rating of 'high' because we have observed many problem domains that have rich vocabularies for describing models with variation among domain experts. Modeling tools aimed at a narrowly defined problem domain with a controlled vocabulary for describing models would find implementing this technique less difficult.

3.1 Using Domain Terms and Concepts For Models

The need for custom languages for M&S has been recognized for decades (Dahl and Nygaard 1966). Using a language designed for M&S reduces the amount of information that needs to be entered as compared to a general purpose programming language. Moreover, raising the level

Table 2: Summary of Impact, Applicability, and Difficulty of Implementing Techniques that Support a Domain Expert in Verification, Validation, and Testing Activities (L = Low, M = Medium, H = High)

Technique	Impact	Applicability	Difficulty
Using domain terms and concepts for models	H	H	H
Using domain terms and concepts for modeling	H	M	H
Structuring data and validating data entry	H	M	M
Integrating modeling tools into an environment	M	M	H
Maintaining a model test plan	H	H	M
Actively monitoring model quality	M	L	M
Diagnosing model errors with a knowledge base	M	L	H
Keeping historical records of model development and testing	M	H	L
Presenting multiple visualizations of models and model outputs	M	M	M

of abstraction of a language removes some of the error-prone and resource-intensive mappings that a user needs to translate their ideas into the language. Many modeling languages are for general purpose modeling, or are specialized according to a particular technique, such as discrete event simulation (Wonnacott and Bruce 1996).

Introducing domain concepts into a modeling language to produce a domain-specific modeling language (DSL) is a powerful technique for making models accessible to the domain expert. A DSL allows domain experts to better understand, modify, develop, and test programs written in the language (van Deursen, Klint, and Visser 2000). Testing and debugging a model requires an in-depth understanding of how the model is constructed and behaves. Using a DSL makes the structure and function of a model more readily apparent.

CellML (Cuellar et al. 2003) and Systems Biology Markup Language (SBML) (Hucka et al. 2003) are DSLs for biophysical models. Both languages are structured, textual languages that support the description of models using domain concepts, such as molecular counts, chemical reactions, and cellular compartments. Previously, models intended for simulation were primarily developed in computer code, and then converted to text, figures, and equations for publication. This conversion frequently introduces errors that impede the replication of results and further development of a published model (Lloyd, Halstead, and Nielsen 2004). Domain experts benefit from the use of a DSL by having access to the original model description in a more readily apparent form.

Building a DSL requires selecting a problem domain, gathering knowledge about the domain, and reducing that knowledge to semantic objects and operations. DSLs can be textual languages, graphical languages (Balci et al. 1998), spreadsheet languages, or other forms convenient for the domain expert. The language is constructed by hand, or by using meta-modeling tools (Agrawal, Karsai, and Ledeczi 2003).

Selecting a problem domain involves making a tradeoff between the focus and size of the language. A language that represents a large domain or scope can only weakly specialize to any particular aspect of the domain. In contrast, a language that tightly focuses on a small domain may have a limited number of interested users. Developing a DSL often requires several iterations between prototyping a language and having experts in the targeted domain give feedback about the applicability and ease-of-use of the language (Kienle 2001).

The development of SBML is a good example of the tradeoff between focus and size. There is continuous pressure to directly support additional types of models in the language, such as flux-balance models. However, the committee responsible for designing SBML must struggle with the problem of adding this support without placing an undue burden on the tool builders using the language. There is also concern that making the language too broad will lead to communities using disjoint subsets of the language with no real communication between them. As SBML becomes larger and more complicated, it becomes harder to understand models written in the language.

3.2 Using Domain Terms and Concepts For Modeling

Along with customizing the modeling language, it is also desirable to use the language of the domain expert in modeling tools. When using a DSL, model editing tools naturally adopt domain terms as the easiest way to express the model to the user. The DSL is unlikely to describe the inputs and actions of model testing tools however. DSLs typically describe models rather than activities done with a model, although problem solving environments are specifically intended to do both (Allen et al. 2003A, Watson et al. 2002). It may be necessary to elicit the preferred terminology from the domain expert.

The model and modeling tools must be able to accommodate the types of experiments that the domain expert wishes to perform on the model. Valentin and Verbraeck (2002) present guidelines for creating a DSL that include consideration of this issue. Using domain terms and concepts in modeling tools aids the domain expert in matching the functionality of the tool to their needs.

JigCell (Allen et al. 2003B) is a modeling environment designed for cell cycle modeling and research. Users of early versions of JigCell found the terminology used for modeling activities confusing. In particular, JigCell used M&S terms, such as "experimental model", that were too similar to terms used in the domain, causing conflicts. Renaming the terms displayed in the user interface improved user understanding of the modeling tools' capabilities. Additionally, some of the sequences of steps involved in modeling activities were restructured to be more analogous to the process that the domain experts used when evaluating a model by hand.

3.3 Structuring Data and Validating Data Entry

Desk checking, inspections, reviews, and walkthroughs are verification and validation methods that involve the careful scrutiny of a model. Syntactic and typographic errors are a distraction during these model testing activities, and can potentially hide serious errors. After fixes are applied for the syntactic and typographic errors, the model must be retested to make sure that the errors were corrected and that no new errors were introduced. Prevention or early detection of these types of errors reduces the testing burden of the domain expert. Once all of the superficial errors have been corrected, the domain expert can search for more fundamental errors in the model.

Two important techniques for preventing the introduction of syntactic and typographic errors are the use of validating and structured data editors. Validating editors check that user input is reasonable before applying it to the model. Structured editors break the task of entering data into a predefined collection of attributes, values, and relations (Frank and Szekely 1998, Hsieh and Shipman 2002). A structured editor changes the organization of data from text to a more communicative form. Schank and Hamel (2004) indicate that this has the additional benefit of making model modification more accessible to the domain expert. Since it is difficult to validate input unless the class of valid inputs is restricted and well-defined, validating editors are frequently also structured editors.

3.4 Integrating Modeling Tools Into an Environment

Integrated modeling environments supply tools that support multiple parts of the M&S life cycle. An integrated environment also supplies a modeling methodology by selecting a particular set of tools and controlling how those tools are used in concert (Balmer and Paul 1990).

Comprehensive environments reduce the need to locate a modeling tool for a particular activity. Carefully selecting the available tools in a comprehensive environment also reduces the risk of the domain expert using an inappropriate M&S technique. Errors from using an inappropriate technique are normally difficult to detect and correct. Cohesive environments have well-tested exchange of model information between tools, reducing the loss of fidelity when the model is transformed.

The integration of tools in a modeling environment is a continuum from loose to tight coupling. Loosely coupled environments are easier to make comprehensive. Tightly coupled environments are easier to make cohesive.

The Simulation Model Development Environment (SMDE) (Balci and Nance 1992) is an example of an environment that is loosely coupled and intended to be comprehensive. SMDE is designed to support the conical methodology by selecting tools appropriate for each step of that methodology's model life cycle. Tools are included for model generation, translation, verification, analysis, and management. The environment can be specialized by adding tools for a particular modeler or domain. As a research environment, SMDE uses loose coupling to ease the creation and testing of prototype modeling tools.

Integrating modeling tools is a requirement for integrating the verification and validation functions of those tools (Caughlin 2000). Having integrated verification and validation is important because it improves the coverage of VV&T activities along the model life cycle. As modeling work moves from tool to tool, the domain expert never loses the ability to evaluate model quality. Transferring model testing information between tools also reduces the startup costs of using the VV&T capabilities of a modeling tool.

3.5 Maintaining a Model Test Plan

The core of a model test plan is a repeatable collection of scenarios for model testing. A scenario, called a test case, describes a sequence of actions to perform on the model, and an expected outcome for each action. Failure to observe the expected outcome implies that the model contains an error.

A description of the expected outcome may be complex if the model contains stochastic components. Nondeterministic models can produce many equally correct outputs from a single set of inputs. In this case, the observations of model behavior must be treated as coming from a sample space and statistically analyzed.

Model credibility is improved by documenting that the model passes the test plan and justifying why these test cases demonstrate that the model is suitable for a particular purpose. By maintaining a model test plan, the next user of the model benefits from the body of evidence developed during model accreditation (Conwell, Enright, and Stutzman 2000). Balci et al. (2000) give an organization for a formal and comprehensive plan of testing and accreditation. This level of detail is not necessary for all uses of a model, although it is instructive to review the types of information that can be collected about model testing.

Maintaining a model test plan also assists the domain expert in developing a model. During the model development process, much iteration will be made between refinement and evaluation of the model. Without a repeatable test plan, model VV&T is a scatter shot approach that is unlikely to add significant value. Including support for a test plan in a modeling tool provides a way to measure progress made during model development.

3.6 Actively Monitoring Model Quality

VV&T of a model must be performed throughout the model life cycle. Detecting and fixing errors as early as possible reduces the total cost of producing a correct model. Errors left uncorrected can cascade until it is too late to do anything but restart the modeling process. Modeling tools that continuously and actively search for model errors aid the domain expert by reducing the need to diagnose and debug the cause of an error.

Ideally, an error is detected and reported immediately after it is introduced, allowing the user to fix the error before it spreads to other parts of the model. Balci (1998) gives a comprehensive list of model verification and validation techniques and their applicability to the model life cycle. Since many of these techniques can only be used during a specific part of the model life cycle, it is often necessary to combine several techniques to provide full coverage.

Having a computer-understandable model test plan allows for automation of testing. Automatic test plan evaluation encourages the modeler to perform testing more frequently and can be incorporated into the continuously run tests for model quality (Allen et al. 2003B). However, automatic evaluation has cost in terms of the time required to specify the test plan and the run time required for execution. Overstreet (2002) notes that these costs can make automatic evaluation unattractive even when model correctness is crucial.

3.7 Diagnosing Model Errors With a Knowledge Base

Determining the reason that a model test fails and localizing the model fault is particularly difficult. Improving the skill of diagnosing or debugging the source of error in a model is hard. In general software programming, fault localization is the most difficult part of debugging and requires extensive knowledge to perform (Ducasse and Emde 1988). For a domain expert working on a model, this can result in an excessive expenditure of time correcting model errors.

The tenet of knowledge bases is that a model built to solve a particular problem is not constructed from scratch and then discarded. Instead, modelers can reuse the knowledge generated by building a model in a domain by treating the creation of successive models as an ongoing process (Delen, Benjamin, and Erraguntla 1998). A knowledge base records the experience gained by a domain expert or modeling specialist when performing model diagnosis. This information can then be passed on and researched by another domain expert attempting to build or modify a model.

Birta and Ozmizrak (1996) describe using a knowledge base to generate new experiments and model tests. This provides similar support for model diagnosis while reducing the need for the domain expert to construct a comprehensive suite of model test cases.

3.8 Keeping Historical Records of Model Development and Testing

Historical records of model development and testing provide crucial information about the modeling process. Keeping a historical record assists with the credibility of a model, and is useful for planning future modeling tasks. Information is added to the historical record when the model is transformed, modified, or tested. The historical record can be used in support of automated testing, and helps the domain expert review VV&T methods. Having a historical record makes the model more accessible to experimentation. A historical record lists past experiments done with a model, and makes it easier to undo model changes when an error is detected.

WBCSim (Goel et al. 1999) is a simulation package for wood-based composite models. The intended users of WBCSim are manufacturers and wood scientists. Users construct a wood composite model, perform experiments on the model, and receive visualizations of the experimental results. The addition of a historical record to WBCSim improved usability by allowing searches and comparisons of past model experiments (Shu et al. 2004). A database stores the user's notes, model modifications, simulation setup, and simulation results.

3.9 Presenting Multiple Visualizations of Models and Model Outputs

Visualizations are graphical representations of models and model results. Graphical model representations, such as block diagrams, are useful for communicating the high-level relationships in a model (Sargent 1986). Animations and plots are common examples of visualizations of model outputs. Graphical representations are often combined with textual representations; the structure of the model is depicted by the graphical representation, but the details are filled out in the textual representation. Sanchez and Langley (2003) present an example of this hybrid modeling approach.

The use of visualizations improves understandability by distinctively representing patterns that the modeler considers important. VV&T is aided when visualizations are used by making models and model results more understandable, and by clearly showing the incorrect behavior when the model is not working correctly (Bishop and Balci 1990).

Visualizations reduce the need for the domain expert to master the technical and abstruse specification languages often used for models. However, it is important to remember that the simplicity of visualization can come from hiding important model details. A poorly chosen visualization may give a misleading impression of the model.

One concern about providing multiple visualizations for a model is the expense of maintaining multiple model representations. This expense can be avoided by maintaining a single model from which multiple presentations are generated. Padmanaban, Benjamin, and Mayer (1995) illustrate the use of a knowledge base to store pertinent data about a model, which is then used to create different model views. This approach prevents the introduction of inconsistencies between model representations. Otherwise, it is necessary to evaluate whether the improvement to model understandability is worth the increased cost of development and execution. Nance, Overstreet, and Page (1999) report that the redundancy of multiple model representations can be automatically eliminated in some cases, significantly improving the execution time.

4 DISADVANTAGES

Although we strongly believe that keeping domain experts in mind when building modeling tools is beneficial to the modeling community, it would be unfair to omit discussion of some of the costs of including this support. The most obvious cost of modeling tools that support domain experts is the cost of designing, building, and maintaining these tools. Including domain experts as a targeted user class may require sacrifices in quality, timeliness, efficiency, reliability, robustness, testability, and usability for other user classes. These tradeoffs make it important to consider how, and why, domain experts would want to use a modeling tool.

A domain expert acting alone also incurs cost in the form of risk of failure of the modeling project. Modeling projects that fail have opportunity costs in addition to the expense of the project. Starting the process over again drains resources from other worthwhile activities and delays the return on investment of using M&S. Finally, producing an incorrect model is costly. Detecting model errors through operational use is difficult and time-consuming. In the meantime, money and credibility may be spent on the results of a model that is later found to be invalid.

4.1 Cost Of Not Having Independence in Model Testing

Independent verification and validation (IV&V) is the performance of these activities by someone other than the model developer. Arthur and Nance (2000) emphatically conclude that IV&V is an important technique for mitigating risk in model development. Reducing this risk lowers the expected cost of model development, use, and maintenance. Additionally, it can be expected that model quality and operational correctness are improved by the incorporation of independence into the modeling process.

A domain expert that is developing a model might find that employing outside help for VV&T is cost-effective. This is particularly true if the model is of "critical" nature, has a high cost of failure, or has a cost for error detection and maintenance that exceeds the cost of IV&V. However, the use of independence does not preclude the need for modeling tools that support domain experts. The independent agent may also be a domain expert instead of a modeling specialist.

4.2 Cost Of Selecting Inappropriate Modeling Techniques

When not employing a modeling specialist, the domain expert increases the risk of selecting an inappropriate M&S technique. Although modeling tools provide guidance about which techniques to employ, it is ultimately the tool user that must make the decision. Using inappropriate techniques can introduce model errors that are difficult to detect. Correcting these errors may require discarding some of the work done on the model. This is an expense of time and money that can lead to the failure of the modeling project.

5 CONCLUSIONS

Domain experts have long struggled to make use of modeling tools designed for the modeling specialist. In particular, domain experts lack accessible modeling tools that support model verification, validation, and testing. This paper described several techniques that modeling tools can employ to aid the domain expert in model testing and evaluation. The list of techniques is not intended to be comprehensive. Modeling tool builders should consider the needs of domain experts and provide additional support whenever possible. Although building modeling tools that support the domain expert can be costly, these tools have high utility and are vital to the successful completion of modeling projects.

ACKNOWLEDGMENTS

The work reported herein was partly sponsored by the DARPA and AFRL, Air Force Materiel Command, USAF, under agreement number F30602-01-2-0572, by AFOSR grant F49620-02-1-0090, and by NSF grants DMI-0422719 and DMI-0355391.

REFERENCES

Agrawal, A., G. Karsai, and A. Ledeczi. 2003. An end-to-end domain-driven software development framework. In *Companion of the 18th Annual ACM SIGPLAN Confer-*

ence on Object Oriented Programming, Systems, Languages, and Applications, 8–15: ACM Press.

Allen, N. A., L. Calzone, K. C. Chen, A. Ciliberto, N. Ramakrishnan, C. A. Shaffer, J. C. Sible, J. J. Tyson, M. T. Vass, L. T. Watson, and J. W. Zwolak. 2003A. Modeling regulatory networks at Virginia Tech. *OMICS: A Journal of Integrative Biology* 7 (3): 285–299.

Allen, N. A., C. A. Shaffer, N. Ramakrishnan, M. T. Vass, and L. T. Watson. 2003B. Improving the development process for eukaryotic cell cycle models with a modeling support environment. *SIMULATION: Transactions of The Society for Modeling and Simulation International* 79 (12): 674–688.

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

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

Balci, O., and R. E. Nance. 1992. The Simulation Model Development Environment: an overview. In *Proceedings of the 1992 Winter Simulation Conference*, ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, 726–736: ACM Press.

Balci, O., W. F. Ormsby, J. T. Carr III, and S. D. Saadi. 2000. Planning for verification, validation, and accreditation of modeling and simulation applications. In *Proceedings of the 2000 Winter Simulation Conference*, ed. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 829–839: Society for Computer Simulation International.

Balci, O., and R. G. Sargent. 1981. A methodology for cost-risk analysis in the statistical validation of simulation models. *Communications of the ACM* 24 (4): 190–197.

Balci, O., C. Ulusarac, P. Shah, and E. A. Fox. 1998. A library of reusable model components for visual simulation of the NCSTRL system. In *Proceedings of the 1998 Winter Simulation Conference*, ed. D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, 1451–1460: IEEE Computer Society Press.

Balmer, D. W., and R. J. Paul. 1990. Integrated support environments for simulation modelling. In *Proceedings of the 1990 Winter Simulation Conference*, ed. O. Balci, R. P. Sadowski, and R. E. Nance, 243–249: IEEE Computer Society Press.

Birta, L. G., and F. N. Ozmizrak. 1996. A knowledge-based approach for the validation of simulation models: the foundation. *ACM Transactions on Modeling and Computer Simulation* 6 (1): 76–98.

Bishop, J. L., and O. Balci. 1990. General purpose visual simulation system: a functional description. In *Proceedings of the 1990 Winter Simulation Conference*, ed. O. Balci, R. P. Sadowski, and R. E. Nance, 504–512: IEEE Computer Society Press.

Caughlin, D. 2000. An integrated approach to verification, validation, and accreditation of models and simulations. In *Proceedings of the 2000 Winter Simulation Conference*, ed. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 872–881: Society for Computer Simulation International.

Conwell, C. L., R. Enright, and M. A. Stutzman. 2000. Capability maturity models support of modeling and simulation verification, validation, and accreditation. In *Proceedings of the 2000 Winter Simulation Conference*, ed. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 819–828: Society for Computer Simulation International.

Cuellar, A. A., C. M. Lloyd, P. F. Nielsen, D. P. Bullivant, D. P. Nickerson, and P. J. Hunter. 2003. An overview of CellML 1.1, a biological model description language. *SIMULATION: Transactions of The Society for Modeling and Simulation International* 79 (12): 740–747.

Dahl, O.-J., and K. Nygaard. 1966. SIMULA: an ALGOL-based simulation language. *Communications of the ACM* 9 (9): 671–678.

Delen, D., P. C. Benjamin, and M. Erraguntla. 1998. Integrated modeling and analysis generator environment (IMAGE): a decision support tool. In *Proceedings of the 1998 Winter Simulation Conference*, ed. D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, 1401–1408: IEEE Computer Society Press.

Ducasse, M., and A.-M. Emde. 1988. A review of automated debugging systems: Knowledge, strategies and techniques. In *Proceedings of the 10th International Conference on Software Engineering*, 162–171: IEEE Computer Society Press.

Frank, M. R., and P. Szekely. 1998. Adaptive forms: an interaction paradigm for entering structured data. In *Proceedings of the 3rd International Conference on Intelligent User Interfaces*, 153–160: ACM Press.

Goel, A., C. Phanouriou, F. A. Kamke, C. J. Ribbens, C. A. Shaffer, and L. T. Watson. 1999. WBCSim: A prototype problem solving environment for wood-based composites simulations. *Engineering with Computers* 15 (2): 198–210.

Hsieh, H., and F. M. Shipman. 2002. Manipulating structured information in a visual workspace. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology*, 217–226: ACM Press.

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.

Kienle, H. M. 2001. Using smgn for rapid protoptyping of small domain-specific languages. *SIGPLAN Notices* 36 (9): 64–73.

Kujala, S., and M. Kauppinen. 2004. Identifying and selecting users for user-centered design. In *Proceedings of the Third Nordic Conference on Human-Computer Interaction*, 297–303: ACM Press.

Lloyd, C. M., M. D. B. Halstead, and P. F. Nielsen. 2004. CellML: its future, present and past. *Progress in Biophysics and Molecular Biology* 85 (2–3): 433–450.

Nance, R. E., C. M. Overstreet, and E. H. Page. 1999. Redundancy in model specifications for discrete event simulation. *ACM Transactions on Modeling and Computer Simulation* 9 (3): 254–281.

Overstreet, C. M. 2002. Model testing: is it only a special case of software testing? In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yucesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, 641–647: IEEE Computer Society Press.

Padmanaban, N., P. C. Benjamin, and R. J. Mayer. 1995. Integrating multiple descriptions in simulation model design: a knowledge based approach. In *Proceedings of the 1995 Winter Simulation Conference*, ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman, 714–719: ACM Press.

Sanchez, J. N., and P. Langley. 2003. An interactive environment for scientific model construction. In *Proceedings of the International Conference on Knowledge Capture*, 138–145: ACM Press.

Sargent, R. G. 1986. The use of graphical models in model validation. In *Proceedings of the 1986 Winter Simulation Conference*, ed. J. Wilson, J. Henriksen, and S. Roberts, 237–241: ACM Press.

Sargent, R. G. 2004. Validation and verification of simulation models. In *Proceedings of the 2004 Winter Simulation Conference*, ed. R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, 17–28: IEEE Computer Society Press.

Schank, P., and L. Hamel. 2004. Hiding UML and promoting data examples in NEMo. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, 574–577: ACM Press.

Shu, J., L. T. Watson, N. Ramakrishnan, F. A. Kamke, and B. G. Zombori. 2004. An experiment management component for the WBCSim problem solving environment. *Advances in Engineering Software* 35 (2): 115–123.

Valentin, E. C., and A. Verbraeck. 2002. Guidelines for designing simulation building blocks. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yucesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, 563–571: IEEE Computer Society Press.

van Deursen, A., P. Klint, and J. Visser. 2000. Domain-specific languages: an annotated bibliography. *SIGPLAN Notices* 35 (6): 26–36.

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 Computing in Civil Engineering* 16 (4): 259–268.

Wonnacott, P., and D. Bruce. 1996. The APOSTLE simulation language: Granularity control and performance data. In *Proceedings of the Tenth Workshop on Parallel and Distributed Simulation*, 114–123: IEEE Computer Society Press.

AUTHOR BIOGRAPHIES

NICHOLAS A. ALLEN is a PhD candidate in the Department of Computer Science at Virginia Tech. He received BS degrees (magna cum laude) in Mathematics and Computer Science in 1999, and MS degrees in Mathematics and Computer Science in 2001. His research interests include software design, distributed computing, systems biology, modeling and simulation, 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>.

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>.