Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

ERROR DETECTION AND DISPLAY FOR GRAPHICAL MODELING ENVIRONMENTS

Robert F. Gordon
Paul G. Loewner
Edward A. MacNair
Haojin Wang

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

Kurtiss J. Gordon
James F. Kurose

Department of Computer and Information Science
University of Massachusetts
Amherst, Mass. 01003

ABSTRACT

Many performance modeling systems are now graphically based, enabling the user to build a model by directly drawing and manipulating a pictorial model diagram on the computer display. These modeling systems generally provide a menu driven interface and an icon palette with which the user specifies the model by selecting and linking icons and then providing associated textual attribute information. The resulting combination of graphics and text forms the model specification and requires that certain rules be obeyed in terms of content, completeness, syntax and semantics. Some classes of user errors can be completely prevented with this type of interface; others need to be detected and identified to the modeler, ideally as the model is constructed. Within the visual modeling paradigm, the detection and display of errors is significantly more complex than in text based modeling languages; the display problem is particularly acute in the case of multi-level, hierarchically structured models. In this paper, we describe the error detection and display capabilities we believe are needed in visual modeling tools and describe the implementation of these capabilities in the Research Queueing Package Modeling Environment (RESQME).

1 INTRODUCTION

In the last five years, there has been a rapid development of graphically based modeling systems (Browne et al. 1985; Conway and Maxwell 1986; Conway et al. 1987; Cox 1988; Davis and Pegden 1988; Gilman and Watremez 1986; Hurrion 1986; Kurose et al. 1986; Melamed and Morris 1985; Miles et al. 1988; Roberts and Flanagan 1988; Standridge and Pritsker 1987). These modeling systems generally provide a menu driven interface and icons with which the user specifies the model by selecting and linking the icons and by providing the associated attribute information. The resulting combination of graphics and text forms the model specification and requires that certain rules be obeyed in terms of content, completeness, syntax and semantics.

Some classes of errors can be completely eliminated with this type of interface, while others need to be detected and identified to the modeler as the model is constructed. In this paper, we describe the error detection and display capabilities which we believe are needed to support these graphical interfaces, and discuss their implementation in the Research Queueing Package Modeling Environment (RESQME) (Gordon et al. 1986, 1987, 1990, 1991), using this implementation as an example.

There is complexity in the specification of the model even with the ease-of-use that graphical interfaces provide. To develop realistic models, often a large number of icons need to be displayed and/or the graphical network may contain submodels that represent a subnetwork of multiple icons in a hierarchical model. The specification may be further complicated with a large number of routing choices and complex routing conditions. In addition, some textual input is necessary to fully specify the model; specifying attribute information (such as initial conditions, run length, number of servers, variable names, and routing conditions,) is better done textually than graphically. This textual information may be subject to syntactic and semantic requirements. The above complexity can lead to user errors both in omitting necessary information and in entering erroneous information.

In this paper, we describe how the interface can *incrementally* (i.e., while the model is being constructed) eliminate some classes of errors and detect

and assist in correcting the other syntactical errors *interactively* as the user builds the model. We distinguish this incremental avoidance, detection, and notification of errors from the case in which a model is first completely constructed and then "compiled" or checked for errors in much the same way that a program written in traditional programming languages is compiled. SIMAN (Pegden et al. 1990) is an example of the traditional programming language approach of error checking the whole model during compilation. We also distinguish this incremental error checking from a purely local checking for syntax errors, as is done by packages such as BLOCKS (Pegden et al. 1990). We will describe the need for a more comprehensive error checking.

A number of other graphics oriented performance modeling tools provide varying degrees of error avoidance, detection and notification. Both PAW (Melamed and Morris 1985) and BONES (Shanmugan et al. 1988) adopt an avoidance approach towards errors. An icon or submodel (in the case of BONES) can not be defined unless it is completely and correctly specified (to the extent that their incremental parsers can detect errors). Additional errors are then detected at *compile* time, when the model is evaluated. An advantage of an avoidance oriented approach is that any detectable errors are localized; they can only be in the icon or submodel currently being defined. A disadvantage is that the modeling process itself is constrained; partially completed icons or submodels cannot be defined, and textual information that is temporarily in error (e.g., an arithmetic expression which relies on a parameter, global variable or icon name that has yet to be defined) is not permitted. Some existing graphical modeling tools have thus opted for a more flexible approach. For example, WITNESS (Gilman and Watremez 1986) allows the modeler to partially complete an icon definition (although those attributes specified must be correct) before moving on to specify or modify another part of the model. RESQME provides additional flexibility in that incorrectly specified modeling elements can be left in an error state while additional model elements are created or modified. It is of interest to note that the creation of these additional elements may then correct these errors, a fact that should be detected by an incremental parser. In either case, a more flexible approach than strict error avoidance requires more sophisticated parsing and error notification procedures; these procedures are described in the latter sections of this paper.

The remainder of this paper is structured as follows. Section 2 describes methods to eliminate classes of errors. Section 3 describes the need for interactive error checking and some of the problems related to incremental parsing. Section 4 describes the interface for detecting and informing the user of errors. We summarize the results and benefits in Section 5.

2 ERROR AVOIDANCE

Graphical interfaces allow the modeler to create the model by directly manipulating and linking icons into a network. The resulting network diagram and the corresponding attribute information comprise the model specifications. The alternative is to fully specify the model textually in a procedural language, where each statement has a syntax requirement and therefore a chance for error. With graphical interfaces, more of the specification can be accomplished through drawing the network, thus reducing the syntax requirements of the user.

The graphical interface replaces some of the text entry with pointing and selecting. For example, graphics can eliminate the syntax requirements related to routing connections. These connections can be specified by pointing to the "from" and "to" nodes on the diagram to specify each link. Furthermore, the environment can check that the connection semantics are permissible based on the type of nodes at the end of each link, so that for example, routing lines cannot be connected to a source node nor emanate from a sink node. (On the other hand, the conditional or probabilistic predicates for these connections need to be specified textually and are therefore subject to syntax errors.)

Another way that the interface can help the user avoid errors is to prompt the user for input specific to the object being selected. For example, selecting a service center would cause prompts related to the service time distribution and the queueing discipline to be displayed. An icon that represents an invocation of a parameterized submodel would have the names of its parameters automatically displayed as prompts. Defaults would be provided for many of the replies to the prompts, so that the user need only enter changes for many replies. Presenting a form containing prompts for the required attribute information reduces or eliminates the chances that the modeler will omit this information.

Not only can the attribute prompts be dependent on the icon selected, but also the attribute prompts can be structured, so that the prompts are context sensitive, i.e., dependent on the previous replies. For example, if a priority discipline is selected, then a prompt specifying priority attributes would be dis-

played, whereas this prompt would not be shown for a first-come-first-served discipline. Additionally, whenever there is only a restricted number of replies for a given prompt, the replies would be selectable from a list rather than requiring input. This technique is appropriate in certain situations, as in specifying a queueing discipline, but would obviously not be appropriate for indicating the number of servers.

In addition, any order dependencies can be eliminated (at least as far as the user is concerned). The user can construct the model in any order, adding details and making modifications at will. The model can be worked on at one level and then submodels introduced, or existing submodels may be connected to form a higher level model. Some run parameters may be introduced after the basic model is completed and others before the model is constructed. Some nodes may be connected and then others added and reconnected. Since the software has access to all the information the user has entered about the model, the software can scan and interpret the information in whatever order it requires.

The above error avoidance capability is accomplished in RESQME by an object oriented user interface. RESQME provides two levels of information for every object: graphics and textual attributes. An icon represents a specific elementary building block or a copy (invocation) of a submodel and can be selected by the user from the icon palette. In terms of object oriented programming, each icon *on the palette* represents a class (for example an active queue). Each class has specific attribute requirements. When the icon is selected from the icon palette and placed on the modeling surface, it becomes an instance (object) of that class. Its specific attribute requirements appear in a pop-up window when that object is either placed on the modeling surface or selected for modification. Filling in its attribute information fully specifies the object as an instance of its class. For example, the object "teller1" with 2 servers, first-come-first-served queueing discipline, and exponential service time with mean one is an instance of the class active queue.

When linking these objects together, RESQME checks the class type for compatibility. When selecting an object, the attributes are shown in a pop-up window, the prompts are specific to the object, the replies are selectable whenever possible, and replies affect the other prompts. The process of selecting, linking objects, and specifying attributes of the objects can be done in any sequence.

3 INTERACTIVE ERROR DETECTION

In this section, we discuss the need for interactive error detection with incremental parsing, and the practical requirement to globally parse the model on user demand, as motivation for the error display and location capability we describe in Section 4.

When creating a model graphically, the modeler can make syntax errors in naming objects (for example duplicate names), in declaring variables (or forgetting to declare them), and in writing expressions whether for routing, specifying distributions or assigning values. Immediate error detection, just as when writing a program, allows (but does not necessarily require) the modeler to correct the error while entering that statement, avoiding having to locate it again and avoiding the problem of propagating the error. The modeler is assured that the entries are syntactically correct before proceeding, and can be prompted and assisted as soon as an error is detected.

In order to detect as many errors as possible as soon as possible, incremental parsing is employed. In RESQME, every identifier is entered into a symbol table which contains the symbol type along with other information. As textual information is entered, its correctness is checked both in a local fashion (within the attribute information of that object) and in a more global fashion. The information is checked to be correct for the specific prompt that is being answered, to be consistent with the other information associated with the object, and to use any referenced identifiers correctly. All identifiers are checked for uniqueness. Each type of object has specific types of parsing to be performed, and each line of an object has a unique parse code for determining whether the information supplied is correct or not.

Incremental parsing may not identify all errors, and there is clearly a trade-off between the level of complexity of the parser and the set of errors that can be immediately detected. For example, some errors result from the order in which information was entered and/or modified. As a design decision, the modeling environment may be programmed to detect some of these errors, while not others. For example, in RESQME if an undefined identifier is used in an arithmetic expression in an object, an error will be detected. If the identifier is then defined, the previous error will still be displayed as an error until a global parsing of the entire model is performed.

The global consistency problem can be addressed by globally parsing the entire model every time a line of information is entered, but this may adversely af-

fect performance especially as the size of the model and resulting symbol table increases. In RESQME, we decided to balance this trade-off by checking globally for duplicate names on each entry and checking only locally for all other information associated with an object.

We provide the modeler with the option at any point to invoke global parsing, which will then resolve any inconsistencies. From that point, the user can continue with incremental parsing until the need for another global parsing. The parsing results are saved so that further incremental parsing will take into account the global checks to that point as well as to allow error checking of previously saved models.

To further illustrate the above discussion, we present a few examples of how immediate error detection is implemented in RESQME. As an example of automatic global checking, if a user enters a duplicate name, all objects with that name are immediately flagged to the user as having an error (displayed in red). As the user corrects the duplicate name in an object or perhaps deletes an object with the duplicate name, all objects which are no longer duplicates are immediately flagged as correct (displayed in green).

As an example of incremental checking, as the user enters any arithmetic expression, it is immediately checked for the correct syntax. This includes checking that all identifiers used have been previously defined. Any incorrect information is displayed to the user with a message about the error.

Additionally, any required information not yet completed will be flagged to the user for entry. For example, a queue without a service time distribution would be flagged as being in error.

4 ERROR DISPLAY AND LOCATION

In text based modeling languages, a performance model definition is specified textually (typically in a file) and submitted to a *compiler*, which then checks the model specification for syntactic and semantic errors. Any errors are then listed in a file (possibly interspersed among the model specification statements themselves). Since text based model definitions are essentially sequential (being a series of lines in a file), a natural way for the modeler to locate and correct errors is to sequentially move (*scroll*) through the model definition and error listing with an editor, correcting errors as the error messages are encountered.

In the case of graphically constructed models, the problem of error display and recovery is significantly more complex for two reasons. First, the model diagram is (at a minimum) two dimensional and thus there is no natural counterpart to the notion of *scrolling* through the model definition. The problem is even more complex in visual modeling environments like RESQME in which the entire model diagram may not always be visible on the screen. The model may be hierarchical (in which case only the current level of the model is displayed) and/or only a portion of the current level of the model may be on the screen (in which case a panning command may be required to move the window over the modeling canvas). A second complication arises from the very fact that the principal interface to the model is graphical. The picture of the model serves as the primary model view, with textual attributes essentially hidden *within* the model's icons and only viewable via pop-up windows. The textual portion of the model specification is then viewable only by selection and only in object size pieces, preventing the user from seeing (and changing) the whole model specification in a single document. Our experience has been that most modeling errors that can not be prevented using the methods discussed above are associated with the specification of the textual attributes. In the remainder of this section we discuss possible solutions to these problems of error *display* (i.e., indicating the existence of specific errors to the modeler) and error *location* (i.e., the process by which the modeler can locate mistakes in the model).

In some cases, displaying the existence of errors in a graphical representation of a model can be quite simple and can exploit the pictorial nature of the model. If the textual attributes of an icon which is currently displayed on the screen are in error, for example, the icon can be displayed in a reserved color like red to visually flag the error. If there is a mistake in a routing chain definition, the routing chain (or individual link) may be drawn in red. If there is no icon associated with certain textual attributes of the model (e.g., the specification of which method is to be used to generate confidence intervals in a simulation solution), the menu item which would be selected to specify that information can be displayed in red.

In each of the cases above, in addition to showing that the graphical object is in error by color, color can be used in the pop-up attribute window to indicate the existence (or non-existence) of errors in the specific line(s) of the underlying textual attributes. Furthermore, to explain the specific errors associated with those attributes, a context-sensitive error message should be displayed.

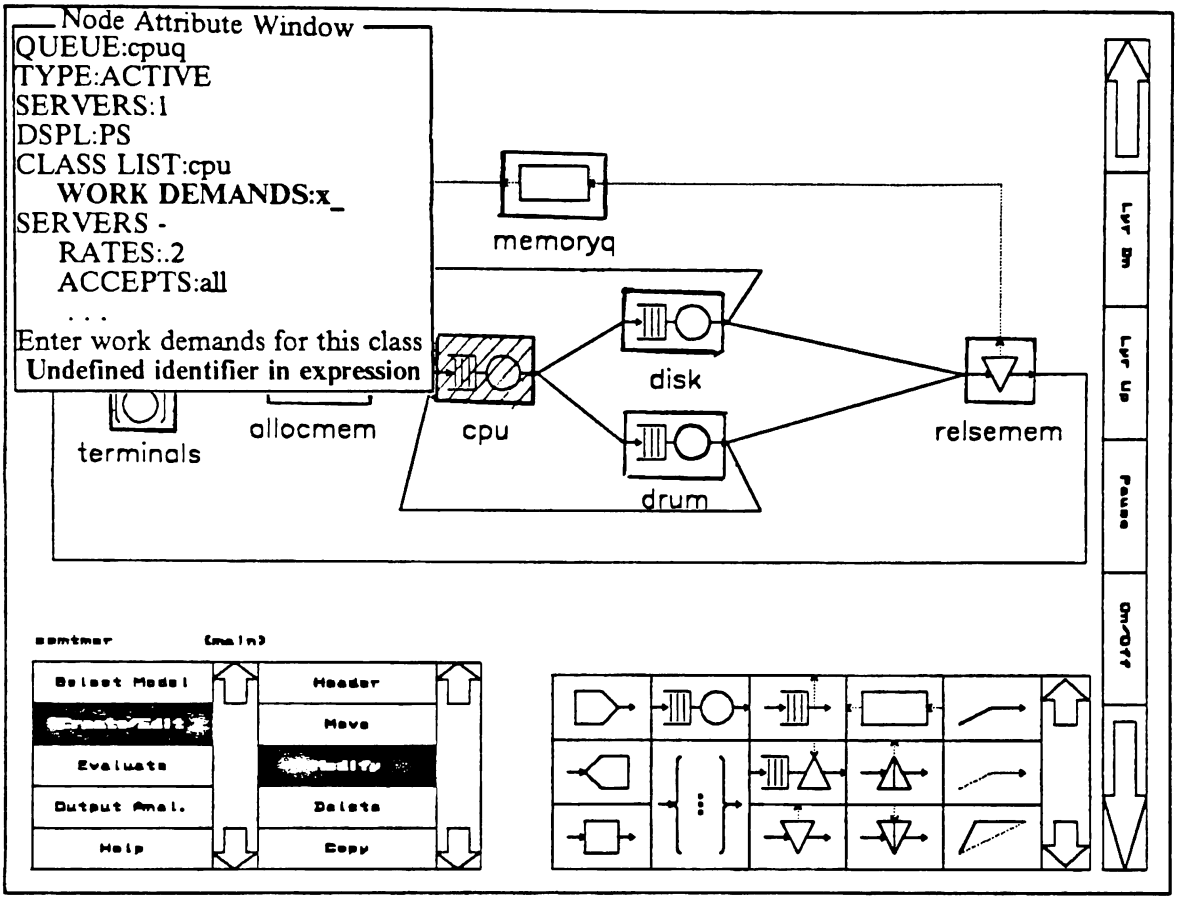

Figure 1. RESQME Screen Illustrating an Error

As discussed in Section 2, a graphic modeling environment may eliminate order dependence, allowing the user to create and modify objects in any sequence. An additional problem this creates for error display is that changing certain text attributes (e.g., an icon's name) can either correct or create errors that extend well beyond the icon itself, so that a single change in a text attribute may significantly influence the model's correctness and hence the colors used in its visual display. This requires some mechanism to reparse objects, other than the one currently being modified by the user, and to redisplay the colors of the affected objects. Ideally, only those objects that can be affected by a change should be reparsed. For example, if parameters in a submodel are modified, these changes have to be reflected in all copies (invocations) of the submodel where the parameters are assigned values.

We illustrate the techniques and solutions for the problem of error display as discussed above, by showing how they are implemented in RESQME. In RESQME, to view the attributes associated with an object, the modeler displays the attributes in a pop-up window. This is done by first selecting the MODIFY command and then selecting (pointing to) an icon, chain, or menu item in error. When the textual attributes are displayed, those attributes in error are shown in red. Also, if the current line (i.e., the line containing the text cursor) is in error, a descriptive error message is displayed at the bottom of the text pop-up window. The modeler can thus visually identify the textual attributes in error (through the use of color), move the cursor to the line in error, view the associated error message, and then correct the error. Figure 1 shows the icon cpu in red (shaded). Selecting MODIFY and then the icon shows the pop-up attribute window, which explains x as undefined.

When a textual attribute has been changed, the object containing the textual attribute and all other objects that have been influenced by the change in this object are incrementally reparsed and if correct, the prompt and reply lines for this textual attribute (which were previously shown in red) are re-lettered (in white). Additionally, if the correction of a text attribute error results in the entire text attribute

pop-up window becoming error-free, either the associated icon is redisplayed with a non-red (green) background or the associated menu item is relettered (in white). On the other hand, if a previously-correct text attribute is changed and becomes in error, the incorrect text line(s) is shown in red and the associated icon color or menu item is set to red.

The discussion in this section has focused thus far on displaying error indicators for either graphical objects or text that are currently displayed on the screen. In some cases, a model may be hierarchically structured, being composed of different levels (layer or submodels) which can only by displayed layer by layer. At times because of size or panning or zooming, only a part of a single level of the model is visible on the screen. Errors may then exist *off the screen*, and some mechanism is needed to visually flag the existence of these errors.

In RESQME, a menu command, LIST ERRORS, is provided to display a list of icons in error within the current (sub)model. Selecting any item from this list will then display that portion of the model diagram containing the icon in error. The text attributes for the object can then be selected for editing.

The existence of errors in other layers of the model is indicated to the user by showing the LAYER DOWN and/or LAYER UP menu commands in red based on which layer(s) is in error. When traversing to different layers (by selecting the LAYER DOWN or LAYER UP menu item), the user is presented with a pop-up of the layer names and can view the desired layer by selecting the name from the pop-up. If a layer has an error, its name is shown in red. After moving to a layer containing errors, the modeler can follow the visual flags discussed above to determine the precise location and to correct each error within the layer.

5 SUMMARY

In this paper we have discussed the importance of interactive error detection and display in graphical modeling environments. One of the greatest strengths of graphical modeling environments is that, to the extent that a graphical representation of a model is syntax free, many kinds of potential errors are avoided. However, the complete specification of a model contains both graphical and textual components, and these textual components still present opportunities for syntax errors. The goal of the modeling environment should be to detect these errors as they are introduced by the user and quickly alert the user to their existence and location. In this way the user can correct individual errors as they occur, rather than having them accumulate and perhaps interact in complex ways.

We have also discussed the kinds of errors which may occur. Some are structural errors which can be detected immediately and locally. Others are contextual or entry order dependent, and can only be found with a global analysis of the model.

When the model is defined hierarchically and when it is too large to display on the screen all at once, we have shown how to direct the user to the error by the development of on-screen surrogates or pointers. We have illustrated techniques for interactive error detection and display to help users more quickly develop useful, error free models.

REFERENCES

Browne, J.C., D. Neuse, J. Dutton, and K.-C. Yu. 1985. Graphical programming for simulation of computer systems. In: *Proceedings of the 18th Annual Simulation Symposium*, ed. A. Miller, 109-126. IEEE Computer Society, Los Alamitos, California.

Conway, R. and W.L. Maxwell. 1986. XCELL: a cellular, graphical factory modelling system. In: *Proceedings of the 1986 Winter Simulation Conference*, eds. J.R. Wilson, J.O. Henriksen, and S.D. Roberts, 160-163. IEEE, Piscataway, New Jersey.

Conway, R., W.L. Maxwell, W. McClain, and S. Worona. 1987. *Users guide to XCELL+ factory modeling system.* Redwood City, California: Scientific Press.

Cox, S.W. 1988. GPSS/PC graphics and animation. In: *Proceedings of the 1988 Winter Simulation Conference*, eds. M.A. Abrams, P.L. Haigh, and J.C. Comfort, 129-135. IEEE, Piscataway, New Jersey.

Davis, D.A. and C.D. Pegden. 1988. Introduction to SIMAN. In: *Proceedings of the 1988 Winter Simulation Conference*, eds. M.A. Abrams, P.L. Haigh, and J.C. Comfort, 61-70. IEEE, Piscataway, New Jersey.

Gilman, A.R. and R.M. Watremez. 1986. A tutorial on SEE WHY and WITNESS. In: *Proceedings of the 1986 Winter Simulation Conference*, eds. J.R. Wilson, J.O. Henriksen, and S.D. Roberts, 178-183. IEEE, Piscataway, New Jersey.

Gordon, R.F., E.A. MacNair, P.D. Welch, K.J. Gordon, and J.F. Kurose. 1986. Examples of using the RESearch Queueing Package Modeling

Environment (RESQME). In: *Proceedings of the 1986 Winter Simulation Conference*, eds. J.R. Wilson, J.O. Henriksen, and S.D. Roberts, 504-510. IEEE, Piscataway, New Jersey.

Gordon, R.F., E.A. MacNair, K.J. Gordon, and J.F. Kurose. 1987. A visual programming approach to manufacturing modeling. In: *Proceedings of the 1987 Winter Simulation Conference*, eds. A. Thesen, H. Grant, and W.D. Kelton, 465-471. IEEE, Piscataway, New Jersey.

Gordon, R.F., E.A. MacNair, K.J. Gordon and J.F. Kurose. 1990. Hierarchical modeling in a graphical simulation system. In: *Proceedings of the 1990 Winter Simulation Conference*, eds. O. Balci, R.P. Sadowski, and R.E. Nance, 499-503. IEEE, Piscataway, New Jersey.

Gordon, K.J., J.F. Kurose, R.F. Gordon and E.A. MacNair. 1991. An extensible visual environment for construction and analysis of hierarchically-structured models of resource contention systems. *Management Science* 37: 714-732.

Hurrion, R.D. 1986. Visual interactive modelling. *European Journal of Operational Research* 23: 281-287.

Kurose, J.F., K.J. Gordon, R.F. Gordon, E.A. MacNair, and P.D. Welch. 1986. A graphics-oriented modeler's workstation environment for the RESearch Queueing Package (RESQ). In: *1986 Proceedings Fall Joint Computer Conference*, 719-728. Dallas.

Melamed, B. and R.J.T. Morris. 1985. Visual simulation: the performance analysis workstation. *IEEE Computer* 18: 87-94.

Miles, T., R.P. Sadowski, and B.M. Werner. 1988. Animation with CINEMA. In: *Proceedings of the 1988 Winter Simulation Conference*, eds. M.A. Abrams, P.L. Haigh, and J.C. Comfort, 180-187. IEEE, Piscataway, New Jersey.

Pegden, C.D., Shannon, R.E., and R.P. Sadowski. 1990. "Introduction to simulation using SIMAN". New York: McGraw-Hill.

Roberts, S.D. and M.A. Flanigan. 1988. Simulation modeling and analysis with INSIGHT: a tutorial. In: *Proceedings of the 1988 Winter Simulation Conference*, eds. M.A. Abrams, P.L. Haigh, and J.C. Comfort, 180-187. IEEE, Piscataway, New Jersey.

Shanmugan et al. 1988. Block-oriented network simulator. In: *Proceedings of IEEE Globecom '88 Conference*, 1679-1684. IEEE, Piscataway, New Jersey.

Standridge, C.R. and A.A.B. Pritsker. 1987. *The extended simulation support system.* New York: John Wiley & Sons, Halsted Press.

AUTHOR BIOGRAPHIES

KURTISS J. GORDON is on the staff of the University Computing Services of the University of Massachusetts. His research interests include user interface design for information systems with graphical content. He is a member of ACM, IEEE, and Sigma Xi.

ROBERT F. GORDON is manager of modeling and analysis software systems at the IBM T.J. Watson Research Center. His research interests are in the areas of decision support systems and graphical environments. He is an adjunct professor in the Business Computer Information Systems and Quantitative Methods Department of Hofstra University.

JAMES F. KUROSE is an associate professor in the Department of Computer and Information Science at the University of Massachusetts at Amherst. His research interests include computer networks, modeling and performance evaluation, and real-time systems. Professor Kurose is the Editor-in-Chief of the IEEE Transactions on Communication and Program Co-Chair of the IEEE Infocom'92 conference. He is a member of ACM, IEEE and Phi Beta Kappa.

PAUL G. LOEWNER is an advisory programmer in the Computer Science Department at the IBM Thomas J. Watson Research Center. His interests are in graphical systems, compilers, and mathematical analysis. He is a member of ACM, SIAM, AMS, MAA, IEEE CS.

EDWARD A. MACNAIR is a Research Staff Member in the Computer Science Department at the IBM Thomas J. Watson Research Center. His research interests are performance modeling tools and simulation output analysis. He is a member of ORSA, TIMS, and ACM, and was *Proceedings* Editor for the 1989 Winter Simulation Conference.

HAOJIN WANG is a Ph.D. candidate in the Department of Computer Science at Texas A&M University. His research interests are design and development of expert system and software engineering in computer simulation.