Proceedings of the 1987 Winter Simulation Conference
A. Thesen, H. Grant, W. David Kelton (eds.)

EXPLORING THE FORMS OF MODEL DIAGNOSIS IN A SIMULATION SUPPORT ENVIRONMENT

Richard E. Nance
Systems Research Center
Virginia Tech
Blacksburg, VA 24061

C. Michael Overstreet
Computer Science Department
Old Dominion University
Norfolk, VA 23529-0162

ABSTRACT

Our purpose is to explain the categorization of diagnostic assistance, using an example, in order to address the central question for a simulation support environment:

> In what forms can computer assistance be provided to improve the simulation model development task?

This question must be answered in the context of the economic and technical realities of today, for the near future, and the long term. We present several examples of computer assistance which indicate that a significant degree of assistance is possible.

1. INTRODUCTION

Several years ago the Computer Science and Engineering Research Study, an NSF sponsored project to produce an accessible record of past research and future projections, published their 900+ page report under the intriguing title, "What Can be Automated?" (Arden, 1980). The authors of this comparatively modest piece (in size and scope) suggest that perhaps the time is appropriate to raise a similar question with regard to simulation model development. Prompted more by discrete event than continuous simulation, the authors have welcomed the interest in utilizing artificial intelligence concepts within simulation modeling and analysis, but admit to some discomfort with the expressions of interest at times appearing to take an unbridled and uninformed "rush to embrace."

1.1. Objective

With that somewhat dampening first paragraph, we intend that this brief treatment address a few preliminary and incomplete answers to a question that has motivated our investigations over the past eight years:

> In what form can computer assistance be provided to improve the simulation model development task?

Note that this question is far more restrictive (and far less ambitious) than, "What can be automated?" An answer to this question is not nearly so demanding as a response to "What can be automated in the application of discrete event simulation?"

Restricted to *model development*, and ignoring the equally important aspects of *model execution*, *model recognition*, and *model classification*, the answers nevertheless are instructive but certainly not conclusive. We hope others can find a motivational spark in our rather pedestrian comments and through their reactions provide more insightful understanding.

1.2. Approach to an Answer

The research leading to this paper began with an attempt to bring some order to the acquisition and use of "computerized" models by the Federal government. Problems cited in an earlier report (USGAO 1976) appear to reflect root causes in the acquisition and development functions and in the documentation requirements (Nance 1977, 1979), (Nance et al. 1981). Zeigler's landmark book points toward the potential for analysis of formal model specifications (Zeigler 1976). Computer assistance in model specification and the derivation of documentation in the development process are recognized as essential requirements (Nance 1979, pp. 95-96), and a methodological foundation for simulation support is given in the conical methodology (Nance 1981a, 1987) and the doctoral dissertation of Overstreet (1982). One set of examples of tangible forms of computer assistance emerge with the definition of a model development environment (MDE) based on the conical methodology (Balci 1986).

Diagnostic assistance following model specification but prior to model execution is embodied in the Model Analyzer, a software tool that accepts a model representation produced using the Model Generator as input and furnishes information about that representation as output. This diagnostic assistance can be allocated to one of three categories: (1) analytical, (2) comparative, or (3) informative. These terms are discussed below.

The following section seeks to describe the basic requirements for a diagnosable representation and provides a review of the Condition Specification using a simple example presented in Section 3; see Overstreet and Nance (1985). Section 4 describes the three categories of assistance, explaining the benefits realized through classification of diagnostic approaches. A brief summary and conclusions in Section 5 completes the treatment.

2. EXAMPLE OF A DIAGNOSABLE REPRESENTATION

Beginning with a model specification as a quintuple (Overstreet and Nance 1985, p. 193):

> < input specifications,
> output specifications,
> object definition set,
> indexing attribute,
> transition specification >.

in which the object oriented paradigm is clearly reflected in the first three elements; see Cox (1983). The fourth and fifth elements impart the distinguishing characteristic of discrete event simulation: the preeminence of time in model representation.

2.1. Producing a Diagnosable Representation

The indexing attribute, commonly designated as *system time*, requires no explanation. The *transition specification* includes three components: (1) an initial state wherein the attributes of all model objects are assigned values, (2) terminal conditions for the conclusion of a model instantiation, and (3) the

Table 1: Syntax and Function of Condition Specification Primitives (from Oversteet and Nance (1985, p. 197)).

Name	Syntax	Function
Value change description	Not specified	Assign attribute values
Set Alarm	SET ALARM(< alarm name > [(< argument list >)], < time delay >)	Schedule an alarm
When Alarm	WHEN ALARM(< alarm name exp > [(< parameter list >)])	Time sequencing condition
After Alarm	AFTER ALARM(< alarm name > & < Boolean exp > [(< parameter list >)])	Time sequencing condition
Cancel Alarm	CANCEL ALARM(< alarm name > [, < alarm id >])	Cancel scheduled alarm
Create	CREATE(< object type > [, < object id >])	Generate new model object
Destroy	DESTROY(< object type > [, < object id >])	Eliminate a model object
Output	Not specified	Produce output
Stop	Not specified	Terminate simulation experiment
Comment	{ < any text not including a "}" > }	Comment

dynamic structure by which each object attribute is assigned a value at some value of the indexing attribute (system time) and, in turn, affects the value of other object attributes at a subsequent value of system time; see Nance (1981b).

The *condition specification* (CS) of a model is a primitive specification language that includes precise requirements for four component specifications: object, transition, interface, and report (Overstreet and Nance 1985, pp. 196-197). The syntax and function of the CS primitives are summarized in Table 1.

The *object specification* is an enumeration and typing of all attributes for all model objects. The *transition specification* is based on a condition action semantics similar to a rule-based construct, i.e.

{boolean condition = **true**}→{actions performed},

which is described as a *condition action pair* (CAP).

Prototypes of the Model Generator use the CAP as the ultimate target for capturing the dynamic structure of a model. The grouping of CAPs with identical conditions produces an *Action Cluster*, (AC), i.e., a set of actions all of which are performed when the same condition is encountered. The following example helps to convey the simplicity and expressive power of the CS.

3. THE SINGLE SERVER QUEUE IN THE CONDITION SPECIFICATION

A single server queue illustrates model specifications using a CS. An object specification for the single server queue is given in Table 2.

The attribute types include those of standard programming languages and the additional type, "time-based signal." Attributes of this type are used for time sequencing. This is illustrated below.

Action Clusters for the single server queue are given in Table 3. In this example, mean interarrival time for parts arrivals and the average service time required to service each part are read at each instantiation of the model. In addition, termination occurs when a specified number of parts have been serviced, this number also read with each instantiation. The amount of time required to service the specified number of parts is reported at model termination.

The syntax used for Action Clusters is similar to that of Pascal. Each action cluster in Table 3 is given a one or two word name and a two letter identifier in a comment at the beginning of each Action Cluster. The two letter identifiers are used as node names in subsequent graphs.

Table 2: Single Server Queue Object Specification

Object	Attribute	Type
System	SystemTime	Nonnegative real
	StopNum	Nonnegative integer
Server	ServerStatus	{ busy, idle }
	MeanServiceTime	Nonnegative real
	EndService	Time-based signal
	NumServed	Nonnegative integer
Parts	NumWaiting	Nonnegative integer
	MeanInterarrivalTime	Nonnegative real
	Arrival	Time-based signal

Table 3: Single Server Queue Action Clusters

Condition	Action
{ Initialization (IN) } initialization	SystemTime = 0 ServerStatus = idle NumWaiting = 0 NumServed = 0 read(MeanInterarrivalTime, MeanServiceTime StopNum) SetAlarm(Arrival, 0)
{ Termination (TE) } NumServed >= StopNum	Output(SystemTime) Stop
{ Arrival (AR) } WhenAlarm(Arrival)	NumWaiting = NumWaiting + 1 SetAlarm(Arrival, NegExp(MeanInterarrivalTime))
{ Begin Service (BS) } NumWaiting > 0 & ServerStatus = idle	NumWaiting = NumWaiting - 1 ServerStatus = busy SetAlarm(EndService, NegExp(MeanServiceTime))
{ End Service (ES) } WhenAlarm(EndService)	NumServed = NumServed + 1 ServerStatus = idle

To interpret part of the above table, a "Begin Service" occurs whenever the value of *NumWaiting* is positive and the value of *ServerStatus* is "busy." The actions which occur when this is true are: *NumWaiting* is decremented by 1, the status of the server is changed to "busy," and an alarm is set to signal an end-of-service. After the specified amount of time has elapsed, the condition "WhenAlarm(EndService)" will test true, so that an "End Service" occurs. At that time *NumServed* is incremented by 1 and the server status is changed to "idle."

4. FORMS OF DIAGNOSTIC ASSISTANCE

The rapid prototyping strategy, employed in the model development environment (MDE) project, forces an assessment of functional priorities and design costs. That is, the initial prototype must reflect *basic* needs and the scope of the effort defined sufficiently for estimating development costs; see Jenkins (1983, pp. 8-9). Consequently, the capability for applying graphical operations, and for deriving graphical characterizations, is an obvious requirement for the initial Model Analyzer prototype.

Model diagnosis should not require a significant investment of time and effort beyond that demanded of the modeler for specification and documentation of the model. Consequently, to the extent possible, elements of model diagnosis should automatically derivable. The condition specification offers this possibility. The following terms are defined to support derivation:

> An attribute x is a *control attribute* of an action cluster if x appears in a condition expression of the action cluster.
>
> An attribute x is an *output attribute* of an action cluster if the actions of the action cluster can change the value of attribute x.
>
> An attribute x is an *input attribute* of an action cluster if the value of x affects the value of the output attributes of the action cluster.

The control, input, and output attributes for the single server queueing model are shown in Table 4. This table is the basis for the two graphs which follow. The first column uses the action cluster identifiers of Table 3.

Table 4: Single Server Queue
Action Cluster - Attribute Relationship Table

AC Id	Control Attributes	Input Attributes	Output Attributes
IN	initialization	MeanInterarrivalTime MeanServiceTime StopNum	SystemTime ServerStatus NumWaiting NumServed MeanInterarrivalTime MeanServiceTime StopNum Arrival
TE	NumServed StopNum	SystemTime	
AR	Arrival	NumWaiting MeanInterarrivalTime	NumWaiting Arrival
BS	NumWaiting ServerStatus	NumWaiting MeanServiceTime	NumWaiting ServerStatus EndService
ES	EndService	NumServed	NumServed ServerStatus

For a condition specification, the following definitions are required to develop the graphs of subsequent sections:

$T = \{t_1, t_2, \ldots, t_k\}$ be the set of all time-based signals in the CS;

$A = \{a_1, a_2, \ldots, a_m\}$ be the set of all other attributes in the CS; and

$AC = \{ac_1, ac_2, \ldots, ac_n\}$ be the set of all action clusters in the CS.

4.1. Graphical Characterization — A Fundamental Need

The conical methodology defines a model of a system as "*objects* and the *relationships* among objects"; see Nance (1981b, p. 175). Such a definition imparts as much importance to the description of relationships as to the related objects. Graph theory is the most obvious modeling mechanism for the portrayal of relationships; therefore, the prominence of directed graphs as a derived model representation is not surprising. The two graphs playing major roles in the initial prototype of the Model Analyzer are the action cluster attribute graph (ACAG) and the action cluster incidence graph (ACIG).

4.1.1. The Action Cluster Attribute Graph

The *Action Cluster Attribute Graph* (ACAG) for a CS is defined as follows. Assume a condition specification with k time-based signal attributes, m other attributes, and n action clusters. Then G, a directed graph with $k + m + n$ nodes, is constructed as follows:

G has a directed, labeled edge from node i to node j if

(1) node i is a control or input attribute for node j, an AC, or

(2) node j is an output attribute for node i, an AC.

Thus G is bipartite with one set of nodes representing ACs and the other, attributes. Each edge of G in the first set above is labeled as "control" or "input" to reflect the attribute relationship; thus, G is a M-level digraph following the definition of Burns and Winstead (1985, p. 344).

The ACAG depicts the interaction between the ACs and attributes in a model specification, showing both the potential for actions of an AC to change the value of an attribute, and the reverse (an attribute's influence on the actions of an AC). In

order to distinguish interactions that occur instantly from those involving a time delay, edges from an AC to a time-based signal (attribute) are depicted with a dashed line; other edges with a solid line. The ACAG for the single server queueing model is shown in Figure 1.

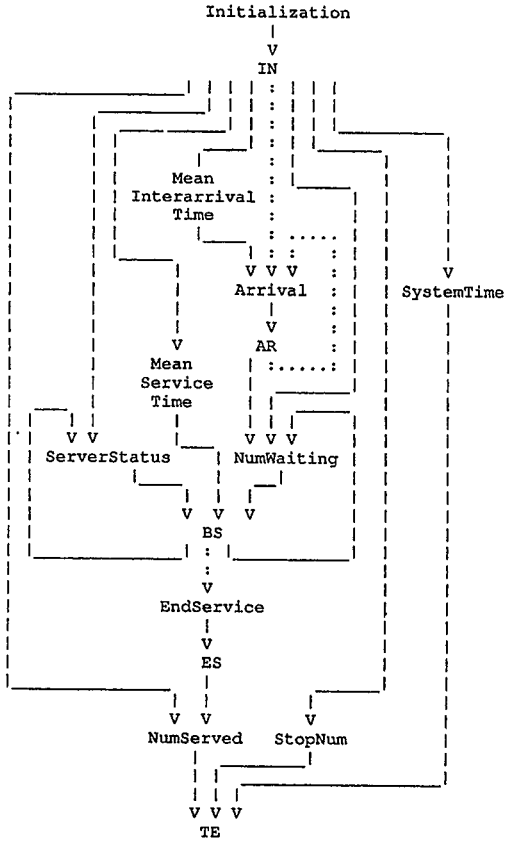

Figure 1: Single Server Queue Action Cluster Attribute Graph

4.1.2. Action Cluster Incidence Graphs

The Action Cluster Incidence Graph (ACIG), depicts a particular type of interaction among ACs. The idea is simple: if an action of AC_i has the potential to cause AC_j to occur (by changing the value of a control attribute), then a directed edge leads from AC_i to AC_j. As before, dashed and solid edges differentiate between time delay and immediate occurrences.

The automatic construction of an ACIG from a CS is easily accomplished following the procedure given below. However, this procedure produces a graph which may include many potential edges. That is, an edge from AC_i to AC_j might be constructed although the application of additional knowledge (from the modeler or through diagnosis) would reveal that AC_i could never cause AC_j to occur. Potential edges removed by the application of additional knowledge are termed infeasible. Overstreet (1982, p. 271) shows that no algorithm can exist to produce from a model specification an ACIG which never includes infeasible edges. Even so, in many cases the number of infeasible edges is not excessive.

An ACIG for a CS consisting of a set of ACs $\{ac_1, ac_2, \cdots, ac_n\}$, can be constructed as follows:

(1) For each $1 \le i \le n$, let node i represent ac_i

(2) For each ac_i, partition the attributes into three sets:
T_i = {control attributes which are time-based signals}
C_i = {all other control attributes}
O_i = {output attributes}

(3) For each $1 \le i \le n$,
For each $1 \le j \le n$,
Construct a solid edge from node i to node j if $O_i \cap C_j \neq \phi$
Construct a dashed edge from node i to node j if $O_i \cap T_j \neq \phi$.
END for each $1 \le j \le n$
END for each $1 \le i \le n$.

For the single server queueing model, the ACIG produced by the above algorithm is shown in Figure 2. Note the prevalence of immediate change edges in the graph. Nance and Overstreet (1986) describe a procedure of deletion of infeasible edges by employing knowledge-based analysis.

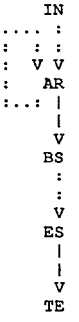

Figure 2: Singer Server Queue Action Cluster Incidence Graph

4.2. Analytical Diagnosis

The ACAG supplies a simple and convenient mechanism for diagnosis of *attribute utilization*, the identification of an attribute node with out-degree of zero. Such an attribute could not influence model behavior. The existence of such an attribute might emanate from an early perception of its need by the modeler during the model definition process followed by a later unrecognized decision that the attribute is unnecessary. Note, however, that an attribute serving strictly a reporting ("statistical") function would also have out-degree of zero; so that one cannot immediately class this characteristic as an error.

Another diagnostic readily performed on the ACAG is *action cluster completeness*, the determination that at least one control attribute of the AC must also be an output attribute of that AC if the condition expression contains attributes that are not time-based signals. This diagnostic assists in the identification of a potential "infinite loop" situation.

Both attribute utilization and action cluster completeness are examples of *analytical diagnosis*: determination that a model representation possesses a defined property. Analytical diagnosis constitutes the simplest form of examination, that form which is most easily understood and most readily automated. The information conveyed to the modeler is direct: the model representation has the property or does not.

4.3. Comparative Diagnosis

In contrast with analytical diagnosis, *comparative diagnosis* furnishes a relative measure of a representation characteristic. The intended connotation of this term is of measures that in isolation may be difficult to interpret, but have potential utility when different values (perhaps from different systems) can be compared. Measures of model complexity are an obvious example of relative measures. The definition of the control and transformation metric, developed specifically for discrete event simulation models, can be found elsewhere in this volume (Wallace 1987).

The validity and utility of *software complexity metrics* has been rightfully questioned (Kearney et al. 1986), and the scarcity of published studies of simulation model complexity might suggest that this research community has been reluctant to pursue this approach. Among the few examples are:

(1) an information theoretical approach (Davies 1976), (Mathewson 1977)

(2) digraph representation and digraph morphisms (Zeigler 1976, pp. 378-389)

(3) finite state machine and tree formalisms (Zeigler 1984, Chapter 2),

(4) resource-entity interaction categorized by levels (Evans 1984, 1986), and

(5) control and transformation analysis (Wallace and Nance 1985), (Wallace 1987).

The utility of the model complexity metric depends on the availability of historical data that convey aspects of the difficulty of a modeling task (number of staff-days to create, total development cost, cost per error corrected) and the success of the metric in reflecting on a linear scale the synthesis of several measures. For prediction of future effort and costs in model development, a model complexity metric sustained by an ongoing database offers invaluable management assistance.

Cohesion and coupling are software engineering attributes of programs; the latter relates the degree of modular independence, the former, the singularity of function. Transforming the ACIG into an interaction matrix enables coupling and cohesion measures to be determined through powers of the interaction matrix (see Nance and Overstreet (1986)).

Comparative diagnosis lacks the clear, undeniable foundation demonstrated by the analytical techniques. Fundamental work in this area is needed for many questions remain to be answered, but the interactive appeals is persuasive. Implementation in an automated form, while clearly accomplishable, is more involved with less obvious immediate benefits than for the analytical forms.

4.4. Informative Diagnosis

The third category of diagnostic assistance is the least defined and the most difficult to automate. This category, informative diagnosis, includes model derivations or extracts that conceivable could assist in the verification of correctness. For example, a list of attributes by type could help the modeler to recognize redundancy, i.e. the definition of two attributes when only one is necessary.

Utilizing the fact that the dynamic behavior of a simulation model is bounded by the initiation and termination status, the ACIG presented in section 4 identifies the sequence of actions which are possible during model execution. The ACIG for the single server queueing model of Figure 2 readily conveys the precedence structure among ACs and reveals the subsets with cyclic relationships. This is an excellent example of informative diagnosis, for the graph may reveal the existence of specification errors to a knowledgeable modeler: the graph may omit sequences which the model knows should be possible or may include sequences which the modeler knows to be improper. No analytic tool could, in general, identify these omissions or inclusions as errors.

Simple measures such as the number of time-delayed and immediate edges or a display showing the submodel decomposition are further examples of informative diagnosis. This category includes those techniques which rely most heavily on the modeler's ability to recognize something informative. Relegated to this category are the forms of *potential and partial* assistance that offer the least prospect for automation. However, these forms might prove to be the most significant source of guidance in model verification.

5. SUMMARY AND CONCLUSION

A simulation support environment must provide assistance to those charged with the development of large, complicated models. Diagnostic assistance in several forms can be employed:

(1) to determine conclusively that a model representation demonstrates a desirable property,

(2) to admit comparisons among different model representations or between different models sharing a common representation, and

(3) to extract or derive model characteristics perceived to offer the modeler information useful in the verification of correctness.

A summary of the three categories described above is given in Table 5. Some examples are not discussed in the prior sections, but the brief explanation presented in the table should be sufficient.

The two principal conclusions of the research underlying this paper are conveyed forcefully in Table 5. The first is that the recognition of the three categories, and the subsequent allocation of each proposed diagnostic to a category, is invaluable in the assignment of priorities so important in a prototyping effort. All too easily is the intelligence of the modeler ignored in the "rush to automate," and less dramatic techniques that augment human capabilities are sacrificed.

The second principal conclusion is that graph-based representations are exceedingly important. Certainly, this finding should not be surprising, for the Conical Methodology in its architectural guidance gives fundamental importance to both *objects* and the *relationships among objects* in the composition of a model.

REFERENCES

Arden, B. W. (1980). *What Can Be Automated?* The MIT Press.

Balci, O. (1986). Requirements for Model Development Environments. *Computers & Operations Research* **13**, 1, Jan-Feb, 53-67.

Burns, J. R., and Winstead, W. H., (1985). M-Labeled Digraphs: An Aid to the Use of Structural and Simulation Models. *Management Science* **31**, 3, Mar., 343-357.

Table 5: Categorized Summary of Diagnostic Assistance

Category of Diagnostic Assistance	Properties, Measures, or Techniques Applied to the Condition Specification (CS)	Basis for Diagnosis
1) Analytical: Determination of the existence of a property of a model representation.	a) Attribute Utilization: No attribute is defined that does not affect the value of another unless it serves a statistical (reporting) function.	Action Cluster Attribute Graph (ACAG)
	b) Attribute Initialization: All requirements for initial value assignment to attributes are met.	ACAG
	c) Action Cluster Completeness: Required state changes within an action cluster are possible	ACAG
	d) Attribute consistency: Attribute Typing during model definition is consistent with attribute usage in model specification.	ACAG
	e) Connectedness: No action cluster is isolated.	Action Cluster Incidence Graph (ACIG)
	f) Accessibility: Only the initialization action cluster is unaffected by other action clusters.	ACIG
	g) Out-complete: Only the termination action cluster exerts no influence on other action clusters.	ACIG
	h) Revision Consistency: Refinements of a model specification are consistent with the previous version.	ACAG
2) Comparative: Measures of differences among multiple model representations.	i) Attribute cohesion: The degree to which attribute values are mutually influenced.	Attribute Interaction Matrix (originates with the ACAG)
	j) Action Cluster cohesion: The degree to which action clusters are mutually influenced.	Action Cluster Interaction Matrix (originates with the ACAG)
	k) Complexity: a relative measure for the comparison of a CS to reveal differences in specification (clarity, maintainability, etc.) or implementation (run-time overhead) criteria.	ACIG
3) Informative: Characteristics extracted or derived from model representations.	l) Attribute Classification: Identification of the function of each attribute (e.g input, output, control, etc.)	ACAG
	m) Precedence Structure: Recognition of sequential relationships among action clusters.	ACIG
	n) Decomposition: Depiction of subordinate relationships among components of a CS.	ACIG

Cox, B. J. (1983). The Message/Object Programming Model. In: *Proceedings of Softfair*. IEEE, Piscataway, N.J., 51-60.

Davies, N. R. (1976). On the Information Content of a Discrete Event Simulation Model. *Simulation* **27**, 4, 123-128.

Evans, J. B. (1984). A Characterisation of Discrete Simulation Complexity. Publication No. TR-A6-84, Centre of Computer Studies and Applications, University of Hong Kong.

Evans, J. B. (1986). Simulation Complexity Revisited. Publication No. TR-B4-86, Centre of Computer Studies and Applications, University of Hong Kong.

Jenkins, A. M. (1983). Prototyping: A Methodology for the Design and Development of Application Systems. Discussion Paper #227, Graduate School of Business, Indiana University, April.

Kearney, J. K., Scellmeyer, R. L., Thompson, W. B., Gray, M. A., and Adler, M.A. (1986). Software Complexity Measurement. *Communications of the ACM* **29**, 11, Nov., 1044-1050.

Mathewson, S. C. (1977). Technical comment On the Information Content of a Discrete Event Simulation Model. *Simulation* **28**, 3, 96.

Nance, R. E. (1977). The Feasibility of and the Methodology for Developing Federal Documentation Standards for Simulation Models. Final Report to the National Bureau of Standards.

Nance, R. E. (1979). Model Representation in Discrete Event Simulation: Prospects for Developing Documentation Standards. In: *Current Issues in Computer Simulation*, N. Adam and A. Dogramaci (eds.), Academic Press, 83-97.

Nance, R. E., Mezaache, A. L. and Overstreet, C. M. (1981). Simulation Model Management: Resolving the Technological Gaps. In: *Proceedings of the 1981 Winter Simulation Conference* (Atlanta, GA, Dec. 9-11), IEEE, Piscataway, N.J., 173-179.

Nance, R. E. (1981a). Model Representation in Discrete Event Simulation: The Conical Methodology. Technical Report CS81003-R, Department of Computer Science, Virginia Tech, Blacksburg, VA.

Nance, R. E. (1981b). The Time and State Relationships in Simulation Modeling. *Communications of the ACM* **24**, 4, Apr., 173-179.

Nance, R. E. and Overstreet, C. M. (1986). Diagnostic Assistance Using Digraph Representation of Discrete Event Simulation Model Specification. Technical Report SRC-86-001, Systems Research Center, Virginia Tech, Blacksburg, VA, Mar.

Nance, R. E. (1987). The Conical Methodology: A Framework for Simulation Model Development. In: *Proceedings of the Conference on Methodology and Validation* (1987 ESC, Orlando, FL, Apr. 6-9). Published as *Simulation Series 19* **1** (Jan. 1988), 38-43. SCS, San Diego, CA.

Overstreet, C. M. (1982). Model Specification and Analysis for Discrete Event Simulation. Ph.D. Dissertation, Virginia Tech, Blacksburg, VA, Dec.

Overstreet, C. M. and Nance, R. E. (1985). A Specification Language to Assistant in Analysis of Discrete Event Simulation Models. *Communications of the ACM* **28**, 2, Feb., 190-201.

United States Government Accounting Office (1976). Ways to Improve Management of Federally Funded Computerized Models. LCD-75-111, Washington, D.C.

Wallace J. C. (1987). The Control and Transformation Metric: Toward the Measurement of Simulation Model Complexity. In *Proceedings of the 1987 Winter Simulation Conference* (Atlanta, GA, Dec. 14-16).

Wallace J. C. and Nance, R. E. (1985). The Control and Transformation Metric: A Basis for Measuring Model Complexity. Technical Report TR-85-15, Department of Computer Science, Virginia Tech, Blacksburg, VA, Mar.

Zeigler, B. P. (1976). *Theory of Modelling and Simulation.* John Wiley.

Zeigler, B. P. (1984). *Multifacetted Modelling and Discrete Event Simulation,* Academic Press.

AUTHORS' BIOGRAPHIES

RICHARD E. NANCE is the Director of the Systems Research Center and Professor of Computer Science at Virginia Polytechnic Institute and State University. He serves as Principal Investigator for the Navy-funded project in Model Development Environments. He has chaired the TIMS College on Simulation and Gaming and the ACM Special Interest Group on Simulation (SIGSIM). He has held editorial positions involving simulation publications in *Operations Research* and *IIE Transactions.* He is a former member of the WSC Board. He currently serves on the editorial panels of the *Communications of the ACM* for research contributions in simulation and statistical computing and the *Journal of Operations Research and Computer Science* for contributions in simulation.

Richard E. Nance
Systems Research Center
Virginia Polytechnic Institute and
State University
Blacksburg, VA 24061, U.S.A.
(703) 961-6144

C. MICHAEL OVERSTREET is an Assistant Professor of Computer Science at Old Dominion University. He received his Ph.D. in 1982 at Virginia Polytechnic Institute and State University. He is currently Principal Investigator for a Navy-funded project in the development of software analysis tools and vice-chair of the ACM Special Interest Group on Simulation (SIGSIM). Dr. Overstreet is a member of ACM, IEEE CS, and SCS.

C. Michael Overstreet
Computer Science Department
Old Dominion University
Norfolk, VA 23529-0162, U.S.A.
(804) 440-4545