Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

MODEL DIAGNOSIS USING THE CONDITION SPECIFICATION: FROM CONCEPTUALIZATION TO IMPLEMENTATION

C. Michael Overstreet

Computer Science Department
Old Dominion University
Norfolk, VA 23529-0162

Ernest H. Page
Richard E. Nance

Systems Research Center
and
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106

ABSTRACT

Creating efficient implementations for discrete event models with complex behaviors is often difficult. The task of correctly specifying intended behaviors even without dealing with implementation and run-time details is challenging. We describe ongoing work with Condition Specifications intended to support a modeler in the creation of a model specification. A Condition Specification includes no low-level implementation details and is amenable to static analysis which can 1) identify some types of specification errors, 2) provide feedback to the model specifier which might allow the modeler to detect specification errors, or 3) assist creation of an efficient implementation. We review previous work on Condition Specifications, present some important results related to what is achievable in analysis of specifications, describe how analysis can be performed and discuss some current activities in this area.

1 INTRODUCTION

Advancements in computing technology have influenced the *problem-solving disciplines* in a unique fashion. Engineering, operations research, management science and areas of computer science, all of which rely on modeling activities and the existence of a model to effect a solution, see changes in computing technology as creating mutually accelerative and prohibitive pressures. Accelerative pressure stems from the fact that models unyielding to manual techniques are now amenable to solution via automation. But the ability to use models of increasing size and complexity feeds the desire to tackle problems even more challenging. Problems and models of ever-increasing complexity have ushered in the realization that only by harnessing computing technology to assist in model development can the limitations of human memory and understanding be overcome.

Simulation models provide some of the largest and most complex examples. Under various labels, e.g. Simulation Model Development Environment (Balci and Nance 1987) and Computer Aided Simulation Modeling (Balmer and Paul 1986), an integrated set of software utilities, generally following an underlying methodology, assist model builders in coping with the complexity of their task. Within this software toolset, argues Balci, must be a Model Analyzer (1986, p. 63) – a utility that performs diagnostic analysis on the model specification produced by the Model Generator and "effectively assists in model verification." Model analysis is also cited as crucial in the Knowledge Based Simulation, where introspective analysis exposes relationships among model components (Baskaran and Reddy 1984).

In this paper, a specification language created to support model analysis and diagnosis is described. The language – the Condition Specification (CS) – defined over a decade ago, is reviewed in terms of its evolution and current status. In Section 2, the original motivation and development history of the CS is briefly recounted. Section 3 provides a differentiation of the concepts *specification* and *implementation*. The CS structure and syntax and a small example of model development using the CS appears in Section 4, and the provisions for model analysis and execution are discussed in Section 5. Concluding remarks appear in Section 6.

2 A BRIEF HISTORY OF THE CS

In an early GAO report (USGAO 1975) on the management of government funded computer models, an inability of users to understand how to make minor model changes and adaptations is identified. In a response to this recognition, Nance (1977) identifies the criteria for a Simulation Model Specification and Documentation Language (SMSDL) intended to: 1) provide independence of model specification from model

implementation, 2) permit and support hierarchical model specification, and 3) extract model documentation as a byproduct of the specification process. From this context emerged the Conical Methodology as the first simulation modeling methodology (see (Nance 1981, 1994)).

The Conical Methodology (CM) defines a role for model specification but does not prescribe its form. Overstreet (1982) defines a formalism for simulation model specification congruent with the principles outlined by the CM. The primary goal of this formalism, the Condition Specification (CS), is to provide a world-view-independent model representation that is expressive enough to represent any model but sufficient to facilitate automated diagnosis of the model representation. Analysis of the specification can be used to: 1) identify errors in a specification early in the development process, 2) assist in the creation of efficient model implementations, and 3) provide information to the modeler which may assist in developing a deeper understanding of the system being simulated.

An important property of the CS is the precise and explicit delineation of time and state within a model representation. Given a CS, all model dynamics are easily identifiable as time-based, state-based or a mixture of the two. The CS is not generally intended to function as a language with which the modeler *directly* works when constructing a model. Several efforts have addressed techniques for *extracting* a CS from a modeler via dialog-driven Model Generators within the context of a Simulation Model Development Environment (Barger 1986, Hansen 1984, Page 1990). In these approaches, the Model Generator provides a buffer between the modeler and the low-level syntax of the CS. Still, after years of investigation, several details regarding the nature of the conceptual framework for the Model Generator in the environment remain unresolved (Balci et al. 1990).

3 MODELING CONCEPTS

Fundamental to the development of the CS are the precise characterizations of, and differentiation between, a simulation model specification and a simulation model implementation.

3.1 Simulation Model Specification

A *simulation model specification*, or simply model specification (MS), is a quintuple: $\langle \Phi, \Omega, \Gamma, \tau, \Theta \rangle$ where:

Φ is the *input specification.* The input specification provides a description of the information the model receives

Ω is the *output specification.* The output specification provides a description of the information the environment receives from the model.[1] Attributes used in the output specification serve two functions: 1) If the model is part of a larger model, they provide information needed to coordinate model components. 2) Reporting of model behavior a) to support the study model objective(s), and b) to support model validation.

Γ is the *object definition set.* An object definition is an ordered pair, $\langle O, A(O) \rangle$, where O is the object and $A(O)$ is the object's *attribute set.* During a simulation run, several *instances* of the same object "type" may exist.

A *model attribute set,* $A(M,t)$ is the union of all object attribute sets for a model M that exists at system time t. This set is not time invariant and is based not only on a particular simulation run for the model, but for a point in time for that run.

The *state of an object,* $S(O,t)$ at system time t for and object O is defined by the values of all its attributes. Likewise, the *state of the model,* $S(M,t)$ is defined by the values of the attributes in $A(M,t)$. A change in the value of an attribute constitutes a *state change* both in the model and the object with which the attribute is associated.

A model attribute set cannot be assumed to provide a basis for a set of state variables, defined as (Overstreet 1982, p. 52): "A set of variables for a system form a *state set* if the set, together with future system inputs, contain enough information to completely determine system behavior." In order to establish a set of state variables, the model attribute set must be augmented with "system variables" such as those required to implement scheduling statements, list management, and so on.

τ is the *indexing attribute.* Commonly this attribute is referred to as *system time.* While not mandatory, system time is usually one of the model inputs and if so, the model does not describe how it changes value. τ provides a partial ordering of model action during any simulation run.

Θ is the *transition function.* A transition function contains each of the following: 1) An *initial state* for the model. The initial state defines values for

[1] The input specification and the output specification can be combined to form a *boundary* specification.

all attributes of objects that exist at initiation (model "start up") including an initial value for system time. It must also include the scheduling of at least one determined event[2] 2) A *termination condition,* and 3) a *definition of the dynamic behavior* of the model, describing the effect each model component has on other components, the model response to inputs, and how outputs are generated.

3.2 Simulation Model Implementation

Let $A(M,t)$ be the model attribute set for a model specification M at time t. A model specification is a *model implementation* if: 1) for any value of system time t, $A(M,t)$ contains a set of state variables, and 2) the transition function describes all value changes of those attributes. Thus, if "system variables" have been added to the object specification set so that $A(M,t)$ must always contain a state set, then the transition description also contains a complete description of how these additional attributes change value.

Since $A(M,t)$ typically does not contain a set of state variables, a primary function of a simulation programming language (SPL) is to augment the attributes of the model specification as necessary to create a state set and to augment the transition function as necessary to accommodate the additional attributes.

4 LANGUAGE DESCRIPTION

A Condition Specification provides a particular syntax and semantics for each component in the tuple defining a model specification.

- The *Interface Specification* identifies input and output attributes by name, data type and communication type (input or output). After the transition specification (defined below) is complete, the communication interface description can be generated from the internal dynamics of the model and the object specification. Any CS must have at least one output attribute (Overstreet and Nance 1985).

- The *Object Specification* is a list of all model objects and their attributes. The CS enforces typing for each attribute similar to other strongly-typed languages.

- The *Transition Specification* is a set of ordered pairs called *condition-action pairs.* Each pair includes a condition and an associated action. A *condition* is a boolean expression composed of model attributes and the CS sequencing primitives, WHEN ALARM and AFTER ALARM. Model *actions* come in five classes: 1) a value change description, 2) a time sequencing action, 3) object generation (or destruction), 4) environment communication (input or output), or 5) a simulation run termination statement. The transition specification can be augmented by the definition of side-effect-free functions to simplify the representation of model behavior.

  Condition-action pairs (CAPs) with equivalent conditions are brought together to form *action clusters.* Action clusters (ACs) represent all actions which are to be taken in a model whenever the associated condition is true.

  Besides WHEN ALARM and AFTER ALARM, the CS provides other primitives: SET ALARM, and CANCEL manipulate the values of attributes typed as time-based signals (the example below illustrates their use), CREATE and DESTROY provide instance manipulation for "temporary" objects, and INPUT and OUTPUT provide communication with the model environment. Recent extensions to the CS provide the operations INSERT, REMOVE, EMPTY, MEMBER and FIND to facilitate the CM provisions for model sets (Page 1994).

  Two conditions appear in every CS: *initialization* and *termination.* Initialization is true only at the start of a model instantiation (before the first change in value of system time). The expression for termination is model dependent and may be time-based, state-based, or mixed (both time- and state-based).

- The *Report Specification.* Overstreet separates the report specification from the transition specification since typically many "computations" are required to gather and report statistics that in-and-of-themselves do not define model behavior (Overstreet 1982). Page (1994) describes a syntax for the report specification similar to that provided by extant simulation programming languages.

4.1 Example

Examples of CS model specifications may be found in (Barger 1986; Nance and Overstreet 1987a, 1987b; Overstreet 1982, 1985; Overstreet and Nance 1985;

[2] An *event* is an instant in time in which at least one model attribute changes value. Events are *determined* if their occurrence, once scheduled, depends only on the value of τ. An event is *contingent* if its occurrence depends on attributes other than τ.

Page 1990, 1994; Puthoff 1991). In most of these sources, model specification is effected in the context of model development under the Conical Methodology. For medium- to large-scale models, the processes of model definition and model specification are intimately connected as the model evolves through successive elaboration and refinement. While the nature of this development cannot be adequately demonstrated within the limited scope of this paper, an example of the language application is nonetheless warranted. Figure 1 contains a CS transition specification for an M/M/1 queueing model.

The semantics of the constructs used the example are largely intuitive. Time advance is provided by the *alarm* components: SET ALARM is an action and is used to schedule a particular alarm (all alarms are named) to go off at a future time; WHEN ALARM is a condition (hence evaluates to true or false) and may only be true when the named alarm has been set by a SET ALARM action, and only at the instant of time for which the alarm is scheduled. WHEN ALARM is used to describe actions which can be scheduled to occur and whose occurrence then only depends on the value of the simulation clock. AFTER ALARM also depends on a SET ALARM action, but is used as part of a compound condition to describe the situation when some time must pass and after that, other conditions must be satisfied for the associated actions to occur. For more detailed examples, refer to the previously cited sources.

```
{Initialization}
initialization:
     INPUT(arrival_mean, service_mean, max_served)
     CREATE(server)
     queue_size := 0
     server_status := idle
     num_served := 0
     system_time := 0.0
     SET ALARM(arrival, 0)

{Arrival }
WHEN ALARM(arrival):
     queue_size := queue_size + 1
     SET ALARM(arrival, negexp(arrival_mean))

{Begin Service}
queue_size > 0 and server_status = idle:
     queue_size := queue_size - 1
     server_status := busy
     SET ALARM(end_of_service, negexp(service_mean))

{End Service}
WHEN ALARM(end_of_service):
     server_status := idle
     num_served := num_served + 1

{Termination}
num_served ≥ max_served:
     STOP
     PRINT REPORT
```

Figure 1: M/M/1 Transition Specification.

5 MODEL ANALYSIS AND EXECUTION

A key issue for model analysis is the notion of *model specification equivalence.* Since we are interested in the automated creation of alternative implementations for the same specification, a basis for identifying equivalence-preserving transformations is required. Two model specifications are *structurally equivalent* with respect to a set of model attributes if: 1) the condition sets are equivalent with respect to those attributes, and 2) identical model actions (if stochastic, variates must be from the same distribution) affecting the set of model attributes are specified for corresponding conditions. Two model specifications are *externally equivalent* with respect to a set of model attributes if they specify identical output for those attributes when provided identical input (Overstreet 1982).

5.1 Condition Specification Model Decomposition

An obvious way of organizing CAPs is by grouping them into action clusters as described in the previous section. Still, an AC-oriented CS may have on the order of hundreds or thousands of ACs. In addition to the action cluster aggregation, several transformations, or *decompositions,* have been defined. A CS may be decomposed into an equivalent specification: 1) based on the model objects, 2) reflecting one of the traditional world views, or 3) representing a collection of strongly connected components (see (Overstreet 1982; Overstreet and Nance 1985)). Each of these decompositions is based on the graphical and matrix representations of the CS described below.

5.2 Graph-Based Model Diagnosis

The following taxonomic description of attributes within CAPs provides the basis for much of the subsequent discussion (Overstreet 1982, p. 120): *control attributes* provide the information needed to determine when the action should occur. These are the at-

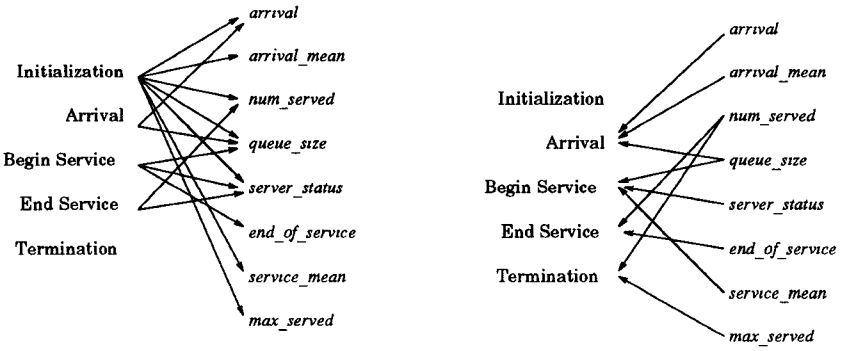

Figure 2: The Action Cluster Attribute Graph for the M/M/1 Model.

tributes that occur in the condition expression. *Output attributes* change value due to the action, and *input attributes* provide the data to be used to set new values for output attributes or schedule future actions.

Most of the analyses provided by the CS utilize graph representations (and the matrix equivalents of the graphs). The most useful graph forms are described below. For further details see (Nance and Overstreet 1987a, 1987b; Puthoff 1991; Wallace 1985). A summary of the analyses defined for these representations is given in Table 1 (adapted from (Nance and Overstreet 1987b)).

5.2.1 Action cluster attribute graph

The *action cluster-attribute graph* (ACAG) is defined as follows. Given a Condition Specification with k time-based signals, m other attributes, and n action clusters, then G, a directed graph with $k + m + n$ vertices is constructed as follows:

G has a directed, labeled arc from i to j if
1) i is a control or input attribute for j, an AC,
2) j is an output attribute for i, an AC.

The ACAG represents the interactions between action clusters and attributes in the CS; specifically, the potential for actions of one AC to change the value of an attribute and the influence of an attribute on the execution of an AC are shown in the ACAG. Figure 2 is the ACAG for the M/M/1 specification of the previous section.

Since the ACAG is a bipartite graph, it may be represented using two Boolean matrices: the *attribute-action cluster matrix* (AACM) and the *action cluster-attribute matrix* (ACAM). For a CS with m action clusters $(ac_1, ac_2, ..., ac_m)$, and n attributes $(a_1, a_2, ..., a_n)$ The AACM is an n by m Boolean matrix in which:

$$b(i,j) = \begin{array}{ll} 1 & \text{if edge}(a_i, ac_j) \text{ exists in the ACAG} \\ 0 & \text{otherwise} \end{array}$$

And the ACAM is an m by n Boolean matrix where:

$$b(i,j) = \begin{array}{ll} 1 & \text{if edge}(ac_i, a_j) \text{ exists in the ACAG} \\ 0 & \text{otherwise} \end{array}$$

Two other matrices may be formed from these matrices, the *attribute interaction matrix* (AIM), defined as AACM $\times$ ACAM, and the *action cluster interaction matrix* (ACIM), defined as ACAM $\times$ AACM

5.2.2 Action cluster incidence graph

An *action cluster incidence graph* (ACIG) is a directed graph in which each node corresponds to an AC in the CS. If, for an implementation based on the CS, the actions in one action cluster, AC_i, can cause the condition for another action cluster, AC_j, to become true (at either the same simulation time at which AC_i is executed or at some future time by setting an alarm) then a directed arc leads from AC_i to AC_j. By convention this arc is depicted as a dotted line if AC_i sets an alarm that is used in the condition for AC_j, otherwise the arc is depicted as a solid line. If the condition on AC_j is a WHEN ALARM then AC_j is referred to as a *time-based successor* of AC_i. If the condition on AC_j is an AFTER ALARM then AC_j is referred to as a *mixed successor* of AC_i. Otherwise AC_j is referred to as a *state-based successor* of AC_i. The ACIG for the M/M/1 model is given in Figure 3.

One may construct an ACIG for a CS consisting of ACs $ac_1, ac_2, \ldots, ac_n$ using to the algorithm of Figure 4. Note that an ACIG completely depicts the

Table 1: Summary of Diagnostic Assistance in the Condition Specification.

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

potential sphere of influence of each AC in the specification, that is when an output attribute of an action cluster is a (state-based or time-based) control attribute of another (not necessarily distinct) action cluster. However, many of these "interactions" may never occur. For instance if AC_i has an output attribute that is involved in the Boolean expression on the condition, denoted p, for AC_j, then there is a solid arc in the ACIG from AC_i to AC_j. However, if the *postcondition* for AC_i (the values of model attributes following the "execution" of AC_i) implies $\neg p$, then the execution of AC_i can never cause the execution of AC_j and the arc can safely be removed from the graph. Overstreet (1982) shows that no algorithm can exist which removes all such edges. However, Puthoff (1991) describes an expert system approach to this type of precondition/postcondition analysis for ACIG simplification, noting near-optimal results for the model specifications considered.

5.2.3 Limits of model analysis

The theme of this research is to understand how we can utilize analyses of specifications to support the modeling process. For example, analysis of a model might assist in choosing a more efficient implementation among several alternative approaches. We also desire analysis tools which identify problems with a specification. However, several questions we might like answered about a particular model specification are undecidable; that is, no algorithm can be written which can determine if an arbitrary CS has a partic-

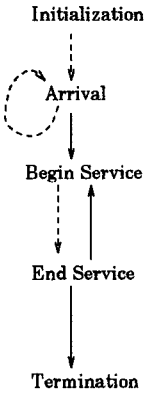

Figure 3: The Action Cluster Incidence Graph for the M/M/1 Model.

For each $1 \leq i \leq n$, let node i represent ac_i
For each ac_i, partition the attributes into 3 sets:
  T_i = {time-based signals (control attributes)}
  C_i = {all other control attributes}
  O_i = {output attributes}
For each $1 \leq i \leq n$,
  For each $1 \leq j \leq n$,
    Construct a solid edge from node i to node j if $O_i \cap C_j \neq \emptyset$
    Construct a dashed edge from node i to node j if $O_i \cap T_j \neq \emptyset$

Figure 4: Algorithm for Constructing an Action Cluster Incidence Graph.

ular property.

For example, during the execution of a CS, the conditions of several ACs might be simultaneously true (a common occurrence), so that they all could execute. If the execution of AC_1 followed by AC_2 leaves the system in a different state than execution of AC_2 followed by AC_1 (perhaps because they both the modify the same attribute), then AC_1 and AC_2 are *order dependent*. This is not necessarily a problem (perhaps their conditions can never be simultaneously true so their consecutive execution can never occur), but *may* indicate a problem with the specification. Determination of order dependency is undecidable (Overstreet 1982).

5.2.4 Support for model execution

Page (1994) describes algorithms for the direct execution of action cluster (DEAC) simulation. The algorithms utilize the ACIG as a model of computation, thereby minimizing the number of conditions which must be tested at any instant during model execution. Algorithms suitable for parallel execution are also presented, and a method for estimating the *inherent parallelism* in a CS based on the critical path through the ACIG is defined. Data flow analysis techniques similar to Weiser's program slicing (Weiser 1986) can automatically identify causality and sequential relationships among model components or can identify model components which can execute in parallel.

6 CONCLUSIONS

Our primary goal for Condition Specifications has been to support analysis of model specifications so that: 1) decisions about many implementation details are not included in the model specification and implementation choices can be based on analysis, and 2) detection of several types of problems in specifications can be effected earlier than is usually possible. This approach requires use of a specification language with carefully defined semantics. We have shown that some properties of specifications which are of interest are in general not decidable. We have also identified several graphs, directly derivable from a Condition Specification, which can be used to determine several important properties of a specification.

Our experience with Conditions Specifications shows that the separation of specifications from implementations is feasible and supports significant error detection and implementation guidance. Future efforts will focus on analyses which can assist in automating or guiding the creation of efficient implementations for both serial and parallel executions.

REFERENCES

Balci (1986). "Requirements for Model Development Environments," *Computers and Operations Research,* **13**(1), pp. 55-67.

Balci, O. and Nance, R.E. (1987). "Simulation Model Development Environments: A Research Prototype," *Journal of the Operational Research Society,* **38**, pp. 753-763.

Balci, O., Nance, R.E., Derrick, E.J., Page, E.H. and Bishop, J.L. (1990). "Model Generation Issues in a Simulation Support Environment," In: *Proceedings of the 1990 Winter Simulation Conference,* pp. 257-263, New Orleans, LA, December 9-12.

Balmer, D.W. and Paul, R.J. (1986). "CASM - The

Right Environment for Simulation," *Journal of the Operational Research Society,* **37**(5) pp. 443-452.

Barger, L.F. (1986). "The Model Generator: A Tool for Simulation Model Definition, Specification, and Documentation," M.S. Thesis, Department of Computer Science, Virginia Tech, Blacksburg, VA, August.

Baskaran, V. and Reddy, Y.V. (1984). "An Interactive Environment Environment for Knowledge-Based Simulation," *Proceedings of the 1984 Winter Simulation Conference,* pp. 645-651, Dallas, TX, November 28-30.

Hansen, R.H. (1984). "The Model Generator: A Crucial Element of the Model Development Environment," Technical Report CS84008-R, Department of Computer Science, Virginia Tech, Blacksburg, VA, August.

Nance, R.E. (1977). "The Feasibility of and Methodology for Developing Federal Documentation Standards for Simulation Models," Final Report to the National Bureau of Standards, Department of Computer Science, Virginia Tech, Blacksburg, VA, June.

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

Nance, R.E. (1994). "The Conical Methodology and the Evolution of Simulation Model Development," *Annals of Operations Research,* Special Volume on Simulation and Modeling, O. Balci, (Ed.), To appear.

Nance, R.E. and Overstreet C.M. (1987a). "Diagnostic Assistance Using Digraph Representations of Discrete Event Simulation Model Specifications," *Transactions of the Society for Computer Simulation,* **4**(1), pp. 33-57, January.

Nance, R.E. and Overstreet, C.M. (1987b). "Exploring the Forms of Model Diagnosis in a Simulation Support Environment," *Proceedings of the 1987 Winter Simulation Conference,* Atlanta, GA, December 14-16, 590-596.

Overstreet, C.M. (1982). "Model Specification and Analysis for Discrete Event Simulation," PhD Dissertation, Department of Computer Science, Virginia Tech, Blacksburg, VA, December.

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

Page, E.H. (1990). "Model Generators: Prototyping Simulation Model Definition, Specification, and Documentation Under the Conical Methodology," M.S. Thesis, Department of Computer Science, Virginia Tech, Blacksburg, VA, August.

Page, E.H. (1994). "Simulation Modeling Methodology: Principles and Etiology of Decision Support," PhD Dissertation, Department of Computer Science, Virginia Tech, Blacksburg, VA, expected.

Puthoff, F.A. (1991). "The Model Analyzer: Prototyping the Diagnosis of Discrete-Event Simulation Model Specification," M.S. Thesis, Department of Computer Science, Virginia Tech, Blacksburg, VA, September.

U. S. General Accounting Office (1975). "Ways to Improve Management of Federally Funded Computerized Models," LCD-75-111, Washington, DC.

Wallace, J.C. (1985). "The Control and Transformation Metric: A Basis for Measuring Model Complexity," M.S. Thesis, Department of Computer Science, Virginia Tech, Blacksburg, VA, March.

Weiser, Mark (1984) "Program Slicing," *IEEE Transactions on Software Engineering,* **SE-10**(4), pp. 352-357, July.

AUTHOR BIOGRAPHIES

C. MICHAEL OVERSTREET is an Associate Professor of Computer Science and Graduate Program Director for Computer Science at Old Dominion University. He is currently chair of the Special Interest Group in Simulation (SIGSIM) of ACM. He received his B.S. from the University of Tennessee in 1966, an M.S. from Idaho State University in 1968, and an M.S. and Ph.D. from Virginia Polytechnic Institute and State University in 1975 and 1982. He has been a visiting research faculty member at the Kyushu Institute of Technology in Japan. His current research interests are in model specification and analysis, high performance networking, and static code analysis in support of software maintenance tasks. He is currently a principal investigator in tasks funded by ICASE at NASA Langley, the National Science Foundation, and the U.S. Navy. Dr. Overstreet is a member of ACM, and IEEE CS.

ERNEST H. PAGE is a Research Associate with the Systems Research Center and a Ph.D. candidate in the Department of Computer Science at Virginia Polytechnic Institute and State University (VPI&SU). He received B.S. and M.S. degrees in Computer Science from VPI&SU in 1988 and 1990. His research interests include discrete event simulation, parallel and distributed systems, and software engineering. He is a member of ACM, ACM SIGSIM, IEEE CS, SCS, and Upsilon Pi Epsilon.

RICHARD E. NANCE 's biography appears elsewhere in these proceedings.