Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

REDUNDANCY IN MODEL REPRESENTATION: A BLESSING OR A CURSE?

Richard E. Nance

Systems Research Center
Virginia Polytechnic Institute
and State University
Blacksburg, VA 24061-0251 USA

C. Michael Overstreet

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

Ernest H. Page

C3 Modeling & Simulation Center
Mitre Corp.
7525 Colshire Drive
McLean, VA 22102-3481 USA

ABSTRACT

With the intent of dispelling the prevailing negative connotations associated with redundancy, we argue that redundancy can effect benefits in model specification as opposed to model execution. Sources of redundancy are classified as accidental or intentional, and several examples are given for each. The comparative benefits and detriments are discussed briefly, and for the most interesting source of redundancy – that induced by a modeling methodology, we demonstrate that automated elimination of redundancy can actually improve the execution time. Although the set of models investigated is small, these results are encouraging for researchers in modeling methodologies using automated model diagnosis.

1 BACKGROUND

As computing technology has advanced, the problems perceived as solvable have grown more challenging, and the consequences have been models of ever-increasing size and complexity. Often, model development efforts focus on the edge of what is currently technically feasible. The need for control and discipline in the specification of such models, accompanied by a recognition of the needs of humans for assistance in performing such complex tasks, have led to the emergence of modeling methodologies and supporting environments. Recognition of the importance of modeling methodologies and computer-assisted support is evident in the prescriptive modeling community through efforts such as ANALYZE (Greenberg 1983; 1987; 1993), GAMS (Bisschop and Meeraus 1982; Brooke, Kendrick and Meeraus 1992) and Structured Modeling (Geoffrion 1987, 1992a, 1992b). Within the discrete event simulation community, the maturation of research in methodology-based support environments is seen in KBSim (Rothenberg 1989), Knowledge Based Simulation (KBS) (Baskaran and Reddy 1984), MODSYN (Rozenblit and Huang 1991), and the Visual Simulation Environment (VSE^{TM}) (Balci et al. 1995). VSE^{TM} is distinct in that it has evolved from research projects originating in 1983 to become a commercial product. Concurrently, simulation language vendors have expanded their software assistance for simulation activities, e.g., SLAMSYSTEM supporting SLAM II and Arena supporting SIMAN.

The consequences of this more expansive view of what modeling support is needed are summarized in (Nance 1994) with particular attention given to the Conical Methodology, the first attempt to provide life-cycle support to simulation activities (Nance 1981, 1987). A major consequence is a shift from program-centric modeling paradigms, where a simulation programming language (SPL) is considered as the only representational form, to model-centric paradigms in which the modeling process encompasses multiple abstraction levels and the focus is directed toward the transformation of model representations through iterative specification.

Model specification is that set of activities performed by a modeler (or modeling team) to create model representations that can can be communicated to humans. A model specification that is executable is considered a model implementation. Note that model specification can employ multiple representations, each at a different level of abstraction. One or more model specification languages can be used in these activities. While a model implementation (in a SPL or any other executable form) is a model specification, the reverse is not true. Since a model specification describes what behavior is to be exhibited whereas an implementation describes how that behavior is to be produced, the former is inherently more abstract than the latter.

The term "redundancy" typically induces negative connotations – something that is unnecessary and exacts a penalty by its inclusion, e.g., superflous constraints in a mathematical program, unexecuted and obsolete code in a simulation program. However, both examples pertain to the model execution, not the model specification. Maintaining unused storage and compiling unexecuted code is wasteful, but redundancy can prove beneficial in model specification. The objectives of this paper are to identify how redundancy is created in simulation model specifications, describe the beneficial influences of redundancy, and demonstrate that the objectionable influences might not be as perceived.

2 REPRESENTATIONAL REDUNDANCY

Redundancy can take several forms; the precise form of interest here is representational redundancy, which we define as the inclusion of any symbols in a model specification that are not required to fulfill the objectives for which the model is developed. "Symbols" can refer to icons in a graphical specification, textual strings used as attribute identifiers, or objects following the object-oriented paradigm. Methods that are never invoked or procedures that remain unexecuted provide additional examples. From this point, all uses of the term "redundancy" are intended to be limited to the "representational" form.

Careful distinctions must be drawn at this point: symbols external to the model specification but included to attain model development objectives are not redundant. An example is the use of embedded assertions to assist in testing and validation with the intent of assuring model correctness as an objective. In contrast, symbols internal to the model specification to support the attainment of objectives such as model reuse do exemplify redundancy.

Given the usual negative associations with redundancy, a typical reaction is that simulation models developed in practice provide few examples. We maintain that such a view can be overly optimistic. In a related paper (Nance, Overstreet and Page. 1996), we identify four major sources of redundancy in model specifications, and offer a number of examples. Because of space constraints, only a few examples are given herein, with the potential sources classified as accidental or intentional.

2.1 Accidental Redundancy

The most prevalent examples of accidental redundancy are those that are tool-induced. In particular, SPLs commonly use defaults on certain functions that force attributes to be carried along even if not used. Consider the TALLY and ACCUMULATE statements in Simscript II.5. Counters are required for computing the ten statistics available in each and are maintained even if the statistic is not accessed. A similar default decision of 12 parameters for a GPSS transaction imposes accidental redundancy if less than 12 are actually used.

Redundancy can be induced accidentally by the modeling methodology. Use of an object-oriented methodology can cause attributes imparted to the inheriting (created) objects to be redundant. For example, the class *vehicle* with an attribute *number_of_axles*, originally intended to distinguish trucks from cars, might be expanded through concatenation to define the class motorized vehicles, which is assigned the attribute type with enumerated values *motorcycle, car, light truck, bus, heavy truck,* and *tandem.* The inherited attribute *number_of_axles* might be superfluous but remain in the model description.

2.2 Intentional Redundancy

A common occurrence of redundancy can be attributed to the objective of reusability. For example, the reuse of an aircraft object to represent a satellite would cause several attributes related to maneuverability and speed changes to become redundant. However, the value of modifying the reused object simply does not argue for doing so.

Potentially, the most prevalent intentional redundancy can be observed in simulations using animation of output behavior. A host of attributes are attached to objects to "jazz up" the output, in the attempt to facilitate validation or promote model acceptance and credibility. Judged strictly from the objectives of the study they are unnecessary. One could assign the cause of redundancy here as stemming from the desire to utilize application domain knowledge to make the model reflect reality beyond that which is required.

Methodology-induced redundancy is clearly the most interesting. Intentionally including redundant description in the form of attributes, methods or even objects as part of the proper way to construct models seems counter-intuitive. However, the use of multiple descriptions to combat complexity is argued by Padmanaban et al. (1995). Incorporating descriptions from multiple domain experts is required "to provide more information than a single description" (Padmanaban, Benjamin, Mayer 1995, p.719). Clearly, the inability to partition these multiple descriptions must lead to redundancy.

The Condition Specification (CS) (Overstreet and Nance 1985), especially in its use within the Conical Methodology, forces even more obvious redundancy through its insistence on the declaration of relational attributes that relate one object to another. An indicative attribute, which conveys some useful information about the object (given the study objectives), can also serve a relational role, and often each of two interacting objects is assigned an attribute with the two attributes being so strongly dependent that one is unnecessary. Fortunately, automated model simplification techniques can be applied to recognize the dependency and eliminate the redundancy.

3 BALANCING THE NEGATIVES AND THE POSITIVES

The argument is put forth in this paper that, contrary to the usual definition of the term, redundancy, at least with respect to model representation, is not entirely bad. Of course, neither is redundancy entirely meritorious. So, can the negatives and positives be properly balanced? An obvious answer to this question seems to be "yes" and that this balancing is the purview of the modeler – such balancing, after all, *is* the essence of modeling. But upon closer examination this answer seems insufficient. As noted in Section 2, the modeler is not the sole mechanism for inducing representational redundancy. The tools that the modeler employs, as well as the modeling methodology itself, may be contributors to redundancy. Does a modeler have dominion over these factors apart from being free (generally) to select from a variety of methodologies and tools? The answer is not clear.

An example of tool-induced redundancy, in the form of default statistical collection routines, in Simscript II.5 and GPSS is given in Section 2.1. Tool-induced redundancy through the use of defaults is commendable as long as: (1) the defaults are reasonable, and (2) overriding the defaults is not prohibitive in time and effort. The tool developer's motive is laudable: provide convenience to the modeler. That convenience should not be attained by making departures from the defaults costly however. In both of the languages noted above the defaults are clearly reasonable. On the other hand, overriding default behaviors in these languages can be difficult.

Animation introduces some controversy as well. Some see it as a costly selling (marketing) gimmick; while others believe that it is essential to validation and model credibility and acceptance. The answer probably lies in the degree to which the animation developer attempts to mimic reality. Beyond a given point, no benefit is likely to accrue.

Redundancy created in the attempt to accomplish study objectives such as reusability is rarely to be criticized. The savings in reuse for instance clearly offset any costs incurred from storage loss or added computation. The inclusion of redundant documentation within the simulation model – internal documentation to promote maintainability as a study objective – seems eminently justifiable. On the other hand, redundant documentation could potentially *increase* the cost of maintenance since a single modification in design might require multiple changes in the documentation.

Redundancy in the representation emanating from the model development methodology probably requires the most careful defense. For what purpose are the instances of redundancy created? Is the redundancy limited to specifications that are not executable? How great is the penalty on execution performance if redundancy is produced in the implementation? In the section that follows we provide a demonstration of how methodology-induced representational redundancy can be circumvented to permit efficient model execution.

4 EXECUTION IMPROVEMENT: AN UNEXPECTED BENEFIT

The role of the CS, as with any specification language, is to operate at a level of abstraction above the implementation details; While the CS does not specify a particular time-flow mechanism (e.g., activity scan or event scheduling), such mechanisms are easily provided. Since the actions to be taken whenever a condition is satisfied can be prescribed in a level of detail similar to a programming language such as C, we have experimented with direct execution of action clusters to measure the performance benefits of diagnostic analysis through elimination of methodology-induced redundancy. The transformation required for direct execution of action clusters (DEAC) is minor, and a variety of algorithms for DEAC simulations can be defined; see (Page 1994, pp. 189-194).

Examples of CS model specifications may be found in many of the references cited herein. The most recent extension of the CS, to provide a more complete specification for parallel execution, is given in (Page 1994). 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. Selection of the four models used here is based on three criteria: (1) published in the literature, well known or easily understood; (2) not so lengthy as to require excessive space, and (3) provide among them a range in size and complexity for examining that factor. The Single Server System is too widely used to give a single reference. The Harbor Model appears in (Buxton and Laski 1963 and Schriber 1974). The MVS Computer Systems is from (Balci 1988). The Machine Repair has appeared in several published works, going back to (Nance 1971).

The DEAC algorithm uses the Action Cluster Incidence Graph (ACIG) described in (Overstreet 1982, Overstreet and Nance 1985, Overstreet and Nance 1986, Nance and Overstreet 1987a, Nance and Overstreet 1987b, Puthoff 1991). Space limitations restrict complete discussion of the graph, but the ACIG distinguishes between models actions whose occurrence is based strictly on the value of the simulation clock (called time-based actions), actions whose occurrence is based only on the values of model attributes other than the simulation clock (called state-based), and those actions which depend on both time and state (called mixed). In essence, model behavior is driven by testing to see if the condition controlling an action is true, and executing the action if so. Inefficient implementations may make many unnecessary tests of model conditions.

Figure 1 presents the DEAC algorithm used in these studies. This algorithm assumes a CS with no mixed ACs. A list, $\mathcal{A}$, of scheduled alarms is maintained as well as a list of state-based action clusters, $\mathcal{C}$, whose conditions should be tested within the current context of model execution. Note that the execution of the termination AC causes an exit from the simulation proper.

While the algorithmic construction of a minimal ACIG (which would produce the minimal number of tests) for any model specification is unsolvable (Overstreet 1982), we have identified techniques for simplifying ACIGS which, though not guaranteeing minimality, are effective and their benefits are illustrated in this section.

The eliminated edges in the ACIG represent modeling relationships that are specified following the CM so that concepts can be captured in a communicative form without regard to issues of efficiency, either in representation or execution. Redundancy abounds, but by recognizing redundant or unnecessary relationships prior to the implementation, no execution penalty is incurred. Note that eliminations ranging from 50% to 78% portend considerable savings in both programming effort and execution time.

4.1 Execution Results

For each of four models, we execute versions based on two ACIGs. The naive version (called "before" below) uses all state-based ACs as successors for each AC and provides a base case for comparison. The second version (called "after" below) incorporates the benefits of analysis by reducing the successor sets for individual Action Clusters based on analysis.

Table 1 presents data about each of the models, the effectiveness of the analysis techniques, and the effect of this analysis on performance.

The first set of data show the simplification of

Let $\mathcal{A}$ be the list of scheduled alarms.
Let $\mathcal{C}$ be the set of state-based clusters whose conditions should be tested immediately.
Let σ_{i_S} be the set of state-based successors for action cluster σ_i (where $1 \leq i \leq |$ ACs $|$).

Initially

```
    $\forall \sigma_i$, set $\sigma_{i_S}$ (from ACIG)
    $\mathcal{C} = \emptyset$
    $\mathcal{A}$ = (initialization AC, initialization clock time)
```

Simulate

```
    while (true) do

        clock $\leftarrow$ time given by FIRST($\mathcal{A}$)
        while (clock = time given by FIRST($\mathcal{A}$)) do
            let $\sigma_a$ be the AC corresponding to
            FIRST($\mathcal{A}$);
            remove FIRST($\mathcal{A}$)
            perform actions of $\sigma_a$
            add $\sigma_{a_S}$ to $\mathcal{C}$

            while ($\mathcal{C} \neq \emptyset$)
                remove $\sigma_c \leftarrow$ FIRST($\mathcal{C}$)
                if condition of $\sigma_c$ is true
                    perform actions of $\sigma_c$
                    add $\sigma_{c_S}$ to $\mathcal{C}$
                endif
            endwhile
        endwhile

    endwhile
```

Figure 1: DEAC Algorithm for a CS without Mixed ACs

the ACIGs for each of the four sample models. The number of edges which can be removed (that is, are redundant) obviously depends on model structure; in some models, it is possible that no edges could be deleted. For these models, from 50% (6 of 12 for the single server queue) to 78% (91 of 116 for the MVS computing system) are eliminated.

Table 1 also includes counts of the total number of condition tests made during a typical run of each version of each of the four models. These data also indicate one aspect of the differences among the models: the frequency counts on number of tests vary by two orders of magnitude.

A significant indication of the benefit of the analysis is the "Percent True Tests" data. The analysis (data flow and expert system) is an attempt to determine a priori that certain tests need not be performed since they must come out false. So the percent of the tests which evaluate to true is a measure of the efficiency (or lack of wasted effort) of the implementation. Thus improving the percent of true tests for 8.5% to 19.3% (as is the case for the Harbor Model) indicates a more efficient implementation.

The last set of data in Table 1 present execution time data for each version of the four models. All runs are on a SPARCstation 5, running Solaris OS with each run consisting of 20 replications of each model (with different random number seeds so behaviors are not identical. The count and percentage data are from a single run (not the 20 replications); behaviors of each replication are similar and show little variation in counts or percentages.

Not included in this table is an actual count of tests which evaluated to *true* for the different versions of each model since for one model all versions must generate exactly the same count: correct implementation would require that any version perform actions whenever the model is in a state where the condition for the action is *true*. Each version for a single model must have identical counts for *true* tests; only the count of false tests can vary. This is the case with these runs.

Much of the data of Table 1 are represented in Table 2 as a percentage improvement. It compares the two versions of each model by computing $|before - after|/before \times 100$ for each metric. This shows, for example, that the speed-up varies from 27.3% to 53.2% among the four models, and that more than twice as many tests (220.5%) evaluate to *true* for the optimal compared to the base case for the Machine Repair model.

From this small number of models, it is inappropriate to draw general conclusions on the percent improvements likely with these techniques; thus we include no averages since it is unlikely these data represent "average" behavior. But since these models are not preselected in anticipation of these techniques working well with them, we do surmise that this approach is generally effective with a wide class of models.

Table 1: Performance Measures

Metric	Vers.	Sing. S. Q.	MVS Sys.	Har.	Mach. Rep.
ACIG	Before	12	116	67	29
Edges	After	6	25	26	10
Total	Before	12K	1,121K	30K	30K
Tests	After	6K	263K	14K	9K
% True	Before	17.7	8.7	8.5	11.2
Tests	After	33.3	33.9	19.3	35.9
Time	Before	2.2	215.7	4.6	6.2
(secs)	After	1.6	68.8	2.9	2.9

Table 2: Improvement of Optimal over Base: $|before - after|/before \times 100$

Metric	Single Serv. Q	MVS Sys	Harbor	Mach. Repair
Exec. Time	27.3	68.1	37.0	53.2
Num. Tests	50.0	76.5	53.3	69.1
% True	88.1	289.7	127.1	220.5

A comment on the effectiveness of our analysis for these four models: the techniques did find every edge which was valid to delete in the ACIG of each model. That is, for each of the remaining edges, at some point in an execution, the test it suggested might need to be performed did evaluate to *true* at least once. Put another way, if the test had not been performed (to check to see of another action should occur), the implementation would have been invalid. We do not expect the analysis to this effective for all models, but the completeness in these four cases is encouraging.

5 CONCLUDING SUMMARY

We continue to explore techniques and benefits of static analysis of model specifications. In this paper, we have demonstrated how one assumed negative aspect of redundancy, namely poor execution, can sometimes be ameliorated by analysis, so that the redundancy sometimes useful in model specifications to support analysis need not result in poor run-time performance.

REFERENCES

Balci, O. 1988. The implementation of four conceptual frameworks for simulation modeling in high-level languages, In: *Proceedings of the 1988 Winter Simulation Conference*, 287-295.

Balci, O., A. I. Bertelrud, C. M. Esterbrook, and R. E. Nance. 1995. A picture-based object-oriented visual simulation environment, In: *Proceedings of the 1995 Winter Simulation Conference*, 1333-1340, Arlington, VA.

Baskaran, V. and Y. V. Reddy. 1984. An interactive environment for knowledge-based simulation, In: *Proceedings of the 1984 Winter Simulation Conference*, 645-651.

Bisschop, J. and A. Meeraus. 1982. On the development of a general algebraic modeling system in a strategic planning environment, *Mathematical Programming Study*, 20:1-29.

Brooke, A., D. Kendrick and A. Meeraus. 1992. *GAMS: A user's guide, release 2.25*, San Francisco, CA: The Scientific Press.

Buxton, J. N. and J. G. Laski. 1963. Control and simulation language, *Computer Journal*, 5, 194-199.

Geoffrion, A.M. 1987. An introduction to structured modeling, *Management Science*, 33:547-588.

Geoffrion, A. M. 1992a. The SML language for structured modeling: levels 1 and 2, *Operations Research*, 40:38-57.

Geoffrion, A. M. 1992b. The SML language for structured modeling: levels 3 and 4. *Operations Research*, 40:58-75.

Greenberg, H. J. 1983. A functional description of ANALYZE, *ACM Transactions on Mathematical Software*, 9(1):18-56.

Greenberg, H. J. 1987. ANALYZE rulebase, In *Proceedings of NATO ASI: Mathematical Models for Decision Support*, 229-238, Berlin: Springer-Verlag.

Greenberg, H. J. 1993. *A computer-assisted analysis system for mathematical programming models and solutions: a user's guide for ANALYZE*, Boston, MA: Kluwer.

Nance, R. E. 1971. On time flow mechanisms for discrete event simulation, *Management Science*, 18(1):59-73, September.

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.

Nance, R. E. 1987. The conical methodology: a framework for simulation model development, methodology and validation, *Simulation*, 19(1):38-43. Orlando, FL: The Society for Computer Simulation.

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

Nance, R.E. and C. M. Overstreet. 1987b. Exploring the forms of model diagnosis in a simulation support environment, In *Proceedings of the 1987 Winter Simulation Conference,* Atlanta, GA, December 14-16, 590-596.

Nance, R. E. 1994. The conical methodology and the evolution of simulation model development, *Annals of Operations Research* 53:1-46.

Nance, R. E., C. M. Overstreet, and E. H. Page. 1996. Redundancy in model specifications for discrete event simulation, work in progress.

Overstreet, C. M. 1982. Model specification and analysis for discrete event simulation, Ph.D. Dissertation, Department of Computer Science, Virginia Tech, Blacksburg, VA, December.

Overstreet, C. M. and R. E. Nance. 1985. A specification language to assist in analysis of discrete event simulation Models, *Communications of the ACM*, 28(2):190-201, February.

Overstreet, C. M. and R. E. Nance. 1986. World view based discrete event model simplification, In *Modelling and Simulation Methodology in the Artificial Intelligence Era*, Elsevier Science Publishers (North-Holland), 165-179.

Padmanaban, N., P. C. Benjamin and R. J. Mayer. 1995. Integrating multiple descriptions in simulation model design: A knowledge based approach, In *Proceedings of the 1995 Winter Simulation Conference*, 714-719, Arlington, VA.

Page, E. H. 1994. Simulation modeling methodology: principles and etiology of decision support, Ph.D. Dissertation, Department of Computer Science, Virginia Tech, Blacksburg, VA, September.

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.

Rothenberg, J. 1989. The nature of modeling, RAND Note N-3027-DARPA.

Rozenblit, J. W. and Y. M. Huang. 1991. Rule-based generation of model structures, in Multifaceted Modeling and System Design, *ORSA Journal on Computing* 3(4):330-344.

Schriber, T. J. 1974. *Simulation using GPSS*, John Wiley and Sons.

AUTHOR BIOGRAPHIES

RICHARD E. NANCE is the RADM John Adolphus Dahlgren Professor of Computer Science and the Director of the Systems Research Center at Virginia Tech (VPI&SU). Dr. Nance is also Chairman of the Board of Orca Computer, Inc. He received B.S. and M.S. degrees from N.C. State University in 1962 and 1966, and the Ph.D. degree from Purdue University in 1968. He has served on the faculties of Southern Methodist University and Virginia Tech, where he was Department Head of Computer Science, 1973-1979. Dr. Nance has held research appointments at the Naval Surface Weapons Center and at the Imperial College of Science and Technology (UK). Within ACM, he has chaired two special interest groups: Information Retrieval (SIGIR), 1970-71 and Simulation (SIGSIM), 1983-85. He has served as Chair of the External Activities Board and several ACM committees. He is the US representative to IFIP TC7. The author of over 100 papers on discrete event simulation, performance modeling and evaluation, computer networks, and software engineering, Dr. Nance has served on the Editorial Panel of Communications of the ACM for research contributions in simulation and statistical computing, 1985-89, as Area Editor for Computational Structures and Techniques of Operations Research, 1978-82, and as Department Editor for Simulation, Automation, and Information Systems of IIE Transactions, 1976-81. He served as Area Editor for Simulation, 1987-89 and as a member of the Advisory Board, 1989-92, ORSA Journal on Computing. Dr. Nance was the founding Editor-in-Chief of the ACM Transactions on Modeling and Computer Simulation, 1990-1995. He served as Program Chair for the 1990 Winter Simulation Conference. Dr. Nance received a Distinguished Service Award from the TIMS College on Simulation in 1987. In 1995 he was honored by an award for "Distinguished Service to SIGSIM and the Simulation Community" by the ACM Special Interest Group on Simulation. He is a member of Sigma Xi, Alpha Pi Mu, Upsilon Pi Epsilon, ACM, IIE, and INFORMS.

C. MICHAEL OVERSTREET is an Associate Professor of Computer Science and Graduate Program Director for Computer Science at Old Dominion University. He is immediate past 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, distributed simulation, high performance networking, remote instruction technologies, and static code analysis in support of software maintenance tasks. He is currently a principal investigator in tasks funded by the National Science Foundation, DARPA, and ICASE at NASA Langley. Dr. Overstreet is a member of ACM, and IEEE CS.

ERNEST H. PAGE is a Member of Technical Staff at the C3 Modeling and Simulation Center and Mitre Corp. in McLean, VA. He received his Ph.D. in Computer Science in 1994 from 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.