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

SIMULATION SUPPORT: PROTOTYPING THE AUTOMATION-BASED PARADIGM

Osman Balci
Richard E. Nance

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

ABSTRACT

This paper describes our research efforts in prototyping the automation-based software paradigm to provide automated support for discrete-event simulation model development. The automation-based paradigm has been suggested as the software technology in the 1990's. The technology needed to support this paradigm does not yet exist. However, the benefits to be gained are so significant that, if achieved, it could profoundly change the way that simulation models are developed. We have been working to achieve this paradigm in the form of an environment composed of an integrated and comprehensive collection of computer-based tools. Our prototyping efforts have focused on the Model Generator, Model Analyzer, and Assistance Manager tools. The Model Generator tool is crucial for the realization of the paradigm and three prototypes have been developed. Our experimentations with the prototypes indicate that the paradigm can be achieved if a small problem domain is chosen. The problem becomes quite complex in the domain-independent case; nevertheless, we believe that the challenge can be met by way of an evolutionary development of prototypes.

1. INTRODUCTION

This past decade has witnessed the clear reversal of roles in computing economics. Human time clearly is more expensive than computer time, and the differential is apparently widening. While the cost of simulation program execution cannot be ignored, the need to utilize modelers and analysts more effectively is pervasive.

Not only are the simulation modelers the expensive commodity, there is a shortage of those who are adequately trained. A simulation study requires multifaceted and multidisciplinary knowledge and experience. In order to gain the basic knowledge for using simulation correctly, Shannon predicts that a practitioner is required to have about 720 hours of formal classroom instruction plus another 1440 hours of outside study (more than 1 man-year of effort) [Shannon 1986].

Progress in software support for simulation applications has unfortunately not kept up with the incredible pace in computing technology. Despite phenomenal advances in computer hardware and some impressive gains in software technology, simulation still remains a labor intensive, error prone, and costly technique especially for large complex simulation studies. Automated support of a simulation study *throughout its entire life cycle* is undeniably needed to confront the problems identified by Balci [1986].

Balzer et al. [1983] have proposed the automation-based paradigm as the software technology for the 1990's. The benefits to be gained in simulation model development are so significant that, if achieved, this new paradigm could profoundly change the way that simulation models are developed. We have been prototyping the automation-based paradigm in the form of a simulation model development environment (SMDE).

The objective of this paper is to describe these prototyping efforts. Section 2 introduces the automation-based paradigm. The architecture of the current SMDE research prototype is explained in Section 3. Section 4 provides a description of the minimal SMDE tools and our prototyping efforts. Conclusions are given in Section 5.

2. THE AUTOMATION-BASED PARADIGM

Balzer et al. [1983] propose a software life-cycle model, shown in Figure 1, for incorporating the capabilities of automatic programming, program transformation, and a "knowledge-based software assistant." The technology needed to support this paradigm does not yet exist, but recognition of the benefits and the consequent emergence of component principles, concepts and tools is clearly apparent.

This radically different approach proposes a shift from the current informal, person-based software paradigm to a formalized, computer-assisted software paradigm. Today, maintenance constitutes 80 to 90 percent of software life-cycle cost and it is a major problem. Under the automation-based paradigm, maintaining the specification as opposed to the implementation will significantly reduce this problem and the software will finally assume its intended form: "soft" and modifiable rather than becoming ossified and brittle with age.

Table 1 [Balzer et al. 1983] compares the automation-based paradigm with the current paradigm.

Despite the absence of many components of the necessary technology, Balzer [1981] claims to have demonstrated the feasibility of the automation-based approach to building software.

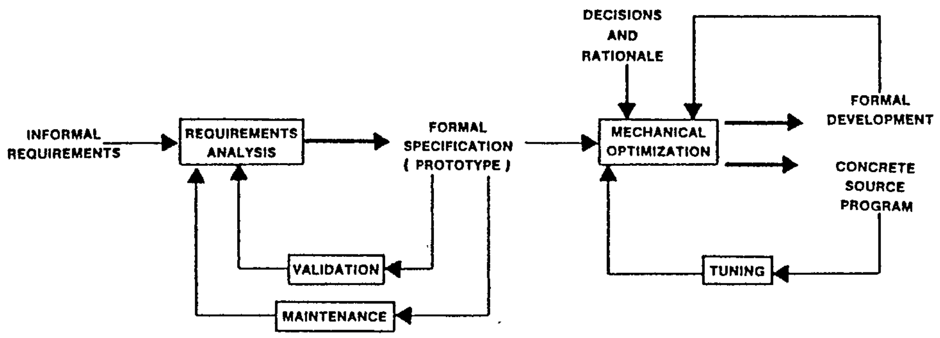


Figure 1. The Automation-Based Paradigm.

Table 1. Comparison of the Automation-Based and Current Paradigms.

Automation-Based Paradigm	*Current Paradigm*
Formal Specification	Informal Specification
Prototyping Standard	Prototyping Uncommon
Specification is the Prototype	Prototype Created Manually
Prototype Validated Against Intent	Code Validated Against Intent
Prototype Becomes Implementation	Prototype Discarded
Implementation Machine-Aided	Implementation Manual
Testing Eliminated	Code Tested
Formal Specification Maintained	Concrete Source Code Maintained
Development Automatically Documented	Design Decisions Lost
Maintenance by Replay	Maintenance by Patching

2.1 Why the Automation-Based Paradigm?

We answer this question by way of describing the major benefits of the automation-based paradigm in comparison with the current paradigm under which software is developed [Balzer et al. 1983].

(1) *Reducing the Cost and Duration of Software Development*: People are becoming more expensive than computers as hardware costs continue to plummet amidst a shortage of adequately trained people. Reliance on software is becoming an economic reality — emphasizing the importance of high quality and less development time. The automation-based paradigm offers the prospect of achieving a reduction both in cost and development time.

(2) *Achieving Maintainability*: Maintenance is a major problem attributable to several flaws in the current paradigm. The automation-based paradigm switches the noncreative aspects of maintenance and modification from man to machine. Program specification becomes the prime focus for maintenance rather than the implementation. By transitioning the focus to a representational form closer to the problem domain, the complexities of code maintenance are eliminated and the intrinsic difficulties are simplified.

(3) *Developing Reusable Software*: Reusability has been a major issue under the current paradigm due to the development of software which is dependent on a programming language and/or an implementation. The automation-based paradigm enhances reusability since specifications and their recorded development are central rather than implementations. Thus, when a module is to be reused, the specification can be modified (or maintained) and its development is changed accordingly.

(4) *Increased User Involvement*: Under the current paradigm systems analysts take the major responsibility for predicting the behavior of the unimplemented system to determine if it matches the user's needs. The evergrowing complexity and specialization of systems make this awesome responsibility grow unrealistically. Under the automation-based paradigm, however, the specification is "operational" in the form of a prototype with which users can experiment to determine the extent to which it matches their requirements. Such an experimentation leads users to improved perceptions of their needs resulting in the development of systems which are more progressively responsive to those needs.

(5) *Reduced Portability Problem*: Under the automation-based paradigm, the machine on which software runs is just one of the decisions made during the implementation process. This decision can be changed like any other decision; therefore, portability disappears as a problem.

3. SMDE ARCHITECTURE

Since June 1983 the MDE project has addressed a complex research problem: prototyping of a discrete-event Simulation Model Development Environment (SMDE) following the automation-based software paradigm described in Section 2. The major research goal has been to provide an integrated and comprehensive collection of computer-based tools to (1) offer cost-effective, integrated, and automated support of model development throughout the model life cycle, (2) improve the model quality by effectively assisting in the quality assurance of the model, (3) significantly increase the efficiency and productivity of the project team, and (4) substantially decrease the model development time.

Guided by the fundamental requirements identified by Balci [1986], the Conical Methodology [Nance 1981, 1987] has furnished the architectural underpinnings of the SMDE research prototype. Prototypes of SMDE tools have been developed using the rapid prototyping technique. See [Balci and Nance 1987] for a description of concepts and principles employed, experiences gained, and guidelines derived in the design and creation of the latest SMDE research prototype.

The prototype SMDE is architectured in four layers as depicted in Figure 2: hardware and operating system, kernel SMDE, minimal SMDE, and SMDEs. Each layer is described in some detail below.

3.1 Layer 0: Hardware and Operating System

A SUN-3/160† color computer workstation running under MC68020 CPU with 4 megabytes of main memory, 380 megabytes of disk subsystem, a 1/4-inch cartridge tape drive, and a 19-inch monitor with 1152×900 pixel resolution constitute the hardware of the prototype SMDE. A laser printer and a line printer accessible via an Ethernet local area network serve the SMDE for producing high quality documents and hard copies of SUN screens and files.

The UNIX‡ 4.2BSD operating system and utilities, multiwindow display manager (SunWindows), device independent graphics library (SunCore), computer graphics interface (SunCGI), visual integrated environment (SunView), Sun programming environment (SunPro), and INGRES relational database management system (SunINGRES) constitute the software environment upon which the SMDE is built. Nance et al. [1984] have evaluated the capabilities of UNIX for hosting an earlier prototype SMDE, noting major and minor deficiencies.

The current prototype, based on the SUN workstation, eliminates certain deficiencies in an earlier version (utilizing a VAX 11/785*) and reduces the negative effects of others.

† SUN-3/160 is a trademark of Sun Microsystems, Inc.
‡ UNIX is a trademark of AT&T Bell Laboratories.
* VAX 11/785 is a trademark of Digital Equipment Corporation.

3.2 Layer 1: Kernel Simulation Model Development Environment

Primarily, this layer integrates all SMDE tools into the software environment described above. It provides Sun-INGRES databases, communication and run-time support functions, and a kernel interface. There are three Sun-INGRES databases at this layer labeled project, premodels, and assistance, each administered by a corresponding manager in Layer 2. All SMDE tools are required to communicate through the kernel interface. Direct communication between two tools is prevented to facilitate maintenance and expansion. The kernel interface provides a standard communication protocol and a uniform set of interface definitions. Security protection is imposed by the kernel interface to prevent any unauthorized use of tools or data.

3.3 Layer 2: Minimal Simulation Model Development Environment

This layer provides a "comprehensive" set of tools which are "minimal" for the development and execution of a model. "Comprehensive" implies that the toolset is supportive of all model development phases, processes, and credibility assessment stages. "Minimal" implies that the toolset is basic and general. It is basic in the sense that this set of tools enables modelers to work within the bounds of the minimal SMDE without significant inconvenience. It is general in the sense that the toolset is generically applicable to various simulation modeling tasks.

Minimal SMDE tools are classified into two categories. The first category contains tools specific to simulation modeling: Project Manager, Premodels Manager, Assistance Manager, Command Language Interpreter, Model Generator, Model Analyzer, Model Translator, and Model Verifier. The second category tools (also called assumed tools, library tools, or host provided tools) are expected to be provided by the software environment of Layer 0: Source Code Manager, Electronic Mail System, and Text Editor.

Figure 3 shows the top-level menu of the current SMDE research prototype from which current prototypes of Minimal SMDE tools can be activated.

3.4 Layer 3: Simulation Model Development Environments

This is the highest layer of the environment, expanding on a defined minimal SMDE. In addition to the toolset of the minimal SMDE, this layer incorporates tools that support specific applications or are needed either within a particular project or by an individual modeler. If no other tools were added to a minimal SMDE toolset, a minimal SMDE would be a SMDE.

The SMDE tools at layer 3 are also classified into two categories. The first category tools include those specific to a particular area of application. These tools might require

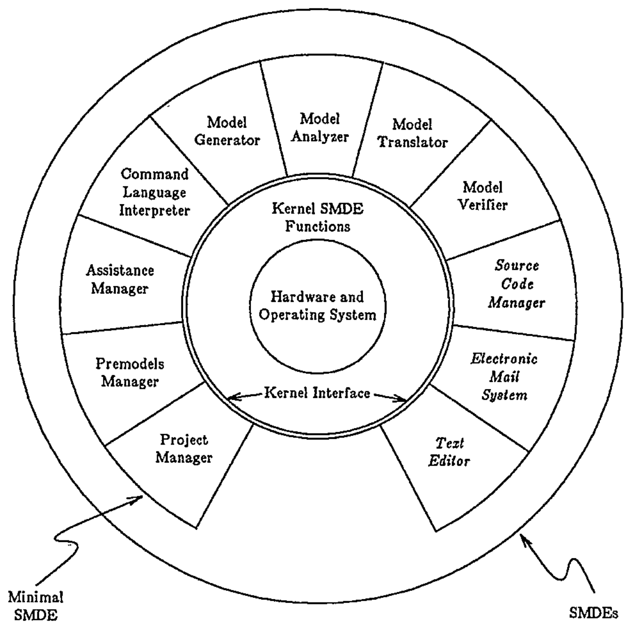
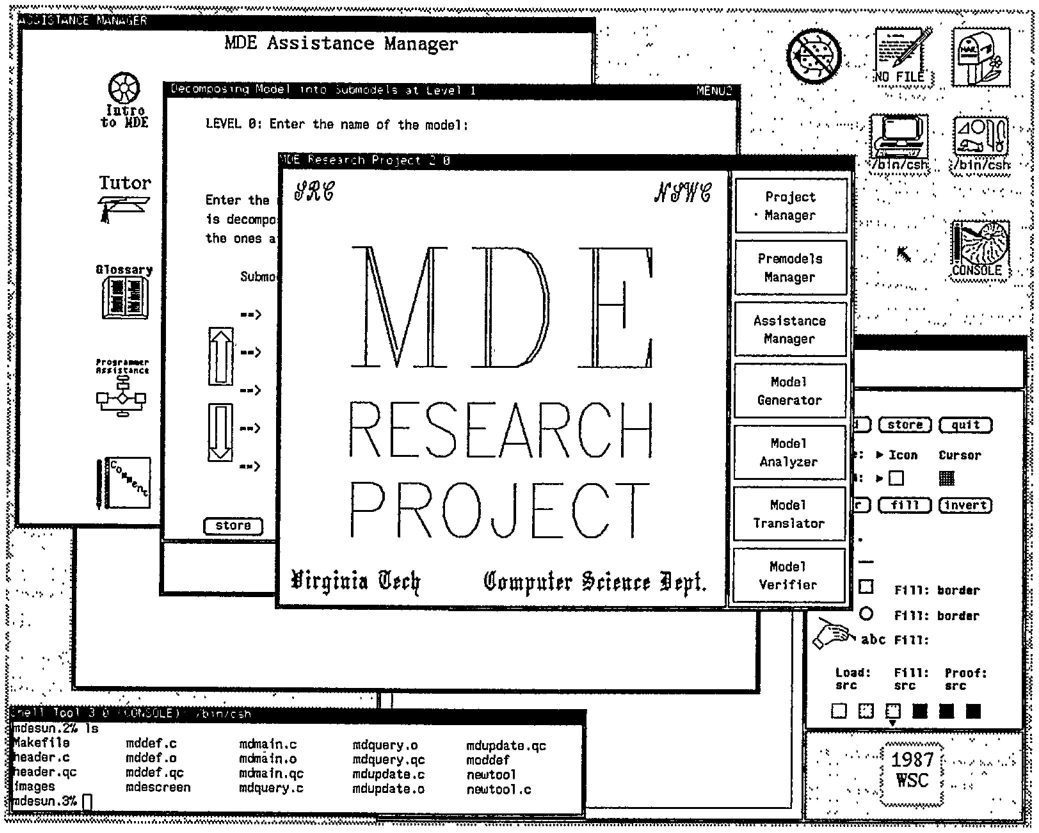


Figure 2. The architecture of the SMDE research prototype.



LEVEL 0: Enter the name of the model:

Enter the
is decompo
the ones a

Submo

Shell Tool 3.0 (CONSOLE) /bin/csh

```
mdesun.2% ls
Makefile        mddef.c         mdmain.c        mdquery.o       mdupdate.qc
header.c        mddef.o         mdmain.o        mdquery.qc      moddef
header.qc       mddef.qc        mdmain.qc       mdupdate.c      newtool
images          mdescreen       mdquery.c       mdupdate.o      newtool.c
mdesun.3%
```

Figure 3. The SMDE Top-Level Menu and Other Tools.

further customizing for a specific project, or additional tools may be needed to meet special requirements. The second category tools (also called assumed tools or library tools) are those expected to be available due to their availability and use in several other areas of application. A tool for statistical analysis of simulation output data, a tool for designing simulation experiments, a graphics tool, a tool for animation, and a tool for input data modeling are some example tools of layer 3.

A SMDE tool at layer 3 is integrated with other SMDE tools and with the software environment of layer 0 through the kernel interface. The provision for this integration is indicated in Figure 2 by the opening between Project Manager and Text Editor.

4. MINIMAL SMDE TOOLS

This section describes our research efforts in prototyping the Model Generator, Model Analyzer, and Assistance Manager. The other minimal SMDE tools are briefly explained.

4.1 Model Generator

The Model Generator (the simulation Model specification and documentation Generator) (MG) is a tool which assists the modeler in: (1) creating a formal model specification, (2) creating multi-level (stratified) model documentation, and (3) model qualification.

Based on the system definition and study objectives, a modeler conceptualizes a model of the system in his or her mind and then converts the conceptual model into a formal specification by way of using the MG tool. The creation of the specification takes place under a conceptual framework. Three essential attributes characterize this specification: (1) it lends itself to formal analysis, (2) it is completely translatable into executable code, and (3) its conceptual framework is **not** domain dependent.

Detecting modeling errors as early as possible within the development life cycle is extremely important for reducing the time and cost of model development and for decreasing the probability of type II error — the error of accepting an invalid model as valid. Therefore, the simulation model should be specified in a form which is amenable to rigorous testing. The Model Analyzer tool is intended to perform such testing described in Section 4.2.

The second attribute is critical for achieving the automation-based paradigm. The Model Translator tool is expected to receive the formal model specification as input and produce an executable model as output. In our experience, the automation-based paradigm can be easily achieved if a restricted problem domain is chosen. For example, if we choose computer networks as the problem domain, we can develop a MG based on the known characteristics of computer networks, extract the required information from the modeler, and create a model specification which can be translated in total into an executable version (experimental model).

The third attribute is the most demanding. Our objective is to build a MG tool which is applicable for any discrete-event simulation problem.

The three attributes of the model specification altogether pose a significant technical challenge. Nevertheless, we are confident that the challenge can be met by way of an evolutionary development of MG prototypes.

To date, three prototypes of the MG tool under the guidance of the Conical Methodology [Nance 1981, 1987] have been developed. The Conical Methodology advocates a top-down model definition and a bottom-up model specification. Prototypes I and II adequately address the definition phase but offer little support for the specification phase. Prototype III supports both the definition and specification phases of the Conical Methodology. A description of Prototype I is given by Hansen [1984]. Prototype III is an improved version of Prototype I and is fully described by Barger [1986].

4.2 Model Analyzer

This tool is intended to diagnose the model specification created by the Model Generator and to effectively assist the modeler in communicative model verification.

Nance and Overstreet [1986] propose several diagnostics which are based on analysis of graphs constructed from a particular form of model specification called *Condition Specification* [Overstreet 1982; Overstreet and Nance 1985]. The diagnostic assistance is partitioned into three categories: (1) analytical—determination of the existence of a property of a model representation, (2) comparative—measures of differences among multiple model representations, and (3) informative—characteristics extracted or derived from model representations. Action cluster attribute graphs, action cluster incidence graphs, and matrix representations extracted from the original graphical forms constitute the basis for the diagnosis.

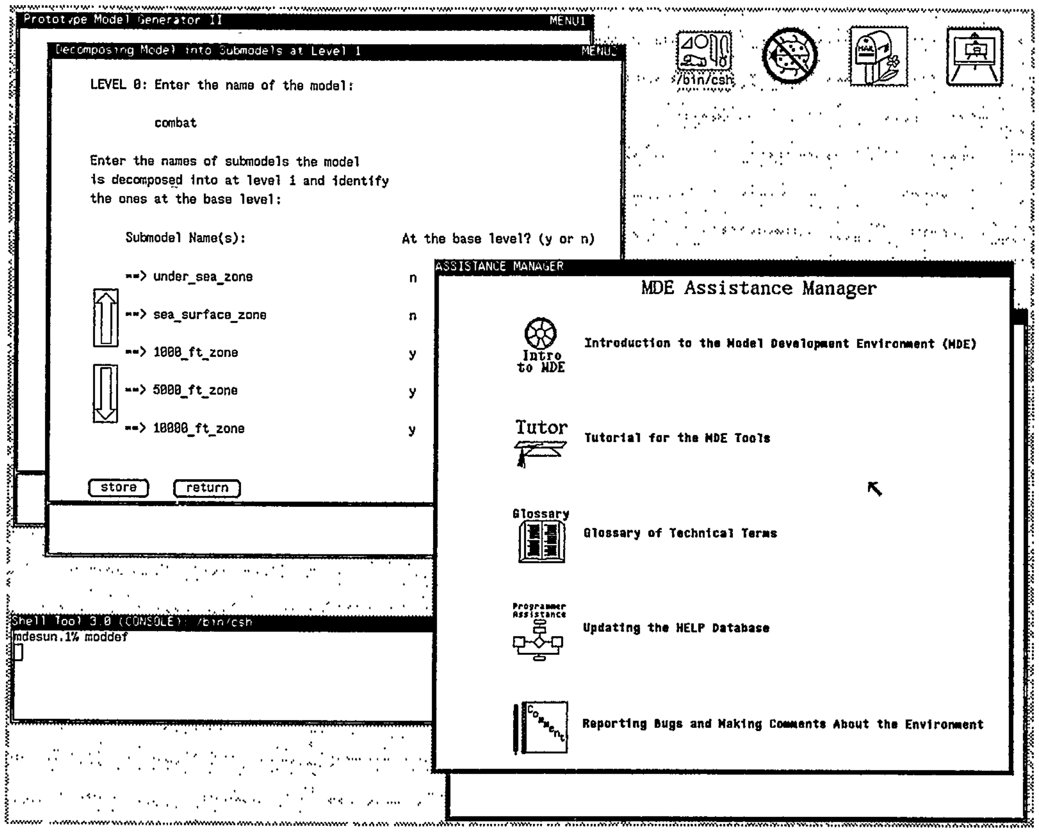
The analytical diagnosis is conducted by measuring the following indicators: attribute utilization, attribute initialization, action cluster completeness, attribute consistency, connectedness, accessibility, out-complete, and revision consistency. The comparative diagnosis is done by measuring attribute cohesion, action cluster cohesion, and complexity. The third category represents the most difficult diagnostics to automate. Examples of informative diagnosis include: attribute classification, precedence structure, and decomposition.

The current research prototype described by Moose and Nance [1985] provides a subset of the graph-based diagnostics and implements a control and transformation metric for measuring model complexity [Wallace and Nance 1985].

4.3 Assistance Manager

The Assistance Manager (AM) is the on-line help facility of the SMDE. A prototype has been developed and is described by Frankel [1987]. The prototype AM has six components as shown in Figure 4: (1) introduction to the SMDE, (2) tutorial, (3) glossary, (4) local (tool specific) help, (5) programmer assistance, and (6) comment facility.

The *Introduction to the SMDE* component provides general information about the SMDE for beginning users. It allows a user to view an on-line document stored in the



LEVEL 0: Enter the name of the model:

combat

Enter the names of submodels the model is decomposed into at level 1 and identify the ones at the base level:

Submodel Name(s):	At the base level? (y or n)
--> under_sea_zone	n
--> sea_surface_zone	n
--> 1000_ft_zone	y
--> 5000_ft_zone	y
--> 10000_ft_zone	y

```
mdesun.1% moddef
```

Introduction to the Model Development Environment (MDE)

Tutorial for the MDE Tools

Glossary of Technical Terms

Updating the HELP Database

Reporting Bugs and Making Comments About the Environment

Figure 4. The Assistance Manager Top-Level Menu.

Assistance Database. A user can obtain information about the tools available in the SMDE and how to invoke a particular tool.

The *Tutorial* component is intended to teach a user: (1) how to use an SMDE tool, (2) how to apply a particular methodology or procedure, and (3) some concepts of modeling and simulation. A user can activate the Tutorial in one window and a specific tool of interest in another. This way, the user can employ the step-by-step instructions and witness the results as the instructions are applied to the tool in the other window.

The *Glossary* component provides the definitions of technical terms. A project member can query the meaning of a term encountered in a project document. The Glossary is intended to alleviate the communication problem among the people involved in the simulation project.

The *Local Help* component provides assistance to a user with regard to the current state of a running tool in the SMDE. The AM provides this capability by giving tool developers the ability to include a help package directly in their code.

The *Programmer Assistance* component allows authorized programmers to embed help packages within their program through the use of the AM and to update and maintain the help database. The AM assists the authorized programmers in terms of updating: (1) documents for the Introduction to the SMDE, (2) tutorials, (3) terms in the Glossary, (4) diagnostic messages, and (5) help text.

The *Comment Facility* component is included for recording user observations and suggestions about the SMDE and its tools. The information gathered by this facility is used for the improvement of tools.

4.4 Other Tools

Project Manager is a tool which (1) administers the storage and retrieval of items in the project database, (2) keeps a recorded history of the progress of the project, (3) triggers messages and reminders (especially about due dates), and (4) responds to queries in a prescribed form concerning project status.

Premodels Manager is a tool which (1) implements a query language, (2) administers the premodels database, (3) provides information on previous modeling projects, and (4) provides a stratified description and several representational forms of models or model components developed in the past.

Command Language Interpreter (CLI) is the language through which a user invokes an SMDE tool. Early in our project, the CLI was prototyped based on the proposal of Moose [1983] and was fully described by Humphrey [1985]. Later, after the acquisition of the SUN workstation, the CLI was replaced by SUN's window management system.

Model Translator translates the model specification into executable version after the quality of the specification is assured by the Model Analyzer. During this translation, code optimization is performed to improve the performance of the executable model.

Model Verifier is intended for the programmed model verification. Applied to the executable representations, it provides: (1) assistance in incorporating diagnostic measures within the source program, (2) a cross-reference map to identify where a particular variable is referenced and where its value is changed, (3) a chart of the programmed model control topology to indicate subprogram invocation relationships, and (4) dynamic analysis procedures for snapshots, traces, breaks, statement execution monitoring, and timing analyses.

Source Code Manager is a tool which configures the run-time system for execution of the programmed model, providing the requisite input and output devices, files, and utilities.

Electronic Mail System facilitates the necessary communication among people involved in the project. Primarily, it performs the task of sending and receiving of mail through (local or large) computer networks. The SUN workstation's MailTool (icon shown in the upper right of Figure 3) is used to communicate with other nodes in the local area network from which large computer networks (e.g., ARPANET, CSNET, BITNET, etc.) can be accessed.

Text Editing is provided by the VI editor and the SUN workstation's TextEditor. Both of these tools are used for preparing technical reports, user manuals, system documentation, correspondence, and personal documents.

5. CONCLUSIONS

The need for automated support in simulation model development is undeniable. The benefits to be gained from the automation-based paradigm are so significant that, if the paradigm is achieved, the development of simulation models could be profoundly changed. Our experience gained by experimenting with the prototype SMDE tools indicate that the automation-based paradigm can be achieved within the context of a very restrictive problem domain. However, the paradigm becomes extremely difficult to achieve in the domain-independent case. Nevertheless, we believe that the challenge can be met by way of an evolutionary development of prototypes.

ACKNOWLEDGMENTS

This research was sponsored in part by the Space and Naval Warfare Systems Command and Naval Research Laboratory under contract N60921-83-G-A165-B030 through the Systems Research Center at VPI&SU. The contributions of the following people to the MDE project are greatfully acknowledged: Lynne F. Barger; Charles W. Box; E. Joseph Derrick; Valerie L. Frankel; Robert H. Hansen; Matthew C. Humphrey; David P. Maynard; Robert L. Moose, Jr.; and Jack C. Wallace.

REFERENCES

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

Balci, O. and Nance, R.E. (1987), "Simulation Model Development Environments: A Research Prototype," To appear in *Journal of the Operational Research Society.*

Balzer, R. (1981), "Transformational Implementation: An Example," *IEEE Transactions on Software Engineering SE-7*, 1 (Jan.), 3-14.

Balzer, R., Cheatham, T.E. and Green, C. (1983), "Software Technology in the 1990's: Using a New Paradigm," *Computer 16*, 11 (Nov.), 39-45.

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., Aug.

Frankel, V.L. (1987), "A Prototype Assistance Manager for the Simulation Model Development Environment," M.S. Thesis, Department of Computer Science, Virginia Tech, Blacksburg, Va., July.

Hansen, R.H. (1984), "The Model Generator: A Crucial Element of the Model Development Environment," M.S. Project, Department of Computer Science, Virginia Tech, Blacksburg, Va., July.

Humphrey, M.C. (1985), "The Command Language Interpreter for the Model Development Environment: Design and Implementation," Technical Report TR-85-17, Department of Computer Science, Virginia Tech, Blacksburg, Va., Mar.

Moose, R.L. (1983), "Proposal for a Model Development Environment Command Language Interpreter," Technical Report CS83032-R, Department of Computer Science, Virginia Tech, Blacksburg, Va., Dec.

Moose, R.L. and Nance, R.E. (1985), "Model Analysis in a Model Development Environment," Technical Report TR-85-27, Department of Computer Science, Virginia Tech, Blacksburg, Va.

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., 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, Fla., Apr. 6-9). Published as *Simulation Series 19*, 1 (Jan. 1988), 38-43. SCS, San Diego, Calif.

Nance, R.E. and Overstreet, C.M. (1986), "Diagnostic Assistance Using Digraph Representations of Discrete Event Simulation Model Specifications," Technical Report TR-86-8, Department of Computer Science, Virginia Tech, Blacksburg, Va., Mar.

Nance, R.E., Balci, O. and Moose, R.L. (1984), "Evaluation of the UNIX Host for a Model Development Environment," In *Proceedings of the 1984 Winter Simulation Conference* (Dallas, Tex., Nov. 28-30). IEEE, Piscataway, N.J., pp. 577-584.

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 Assist in Analysis of Discrete Event Simulation Models," *Communications of the ACM 28*, 2 (Feb.), 190-201.

Shannon, R.E. (1986), "Intelligent Simulation Environments." In *Proceedings of the Conference on Intelligent Simulation Environments* (San Diego, Calif., Jan. 23-25). Published as *Simulation Series 17*, 1 (Jan. 1986), 150-156. SCS, San Diego, Calif.

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.

AUTHORS' BIOGRAPHIES

OSMAN BALCI is an assistant professor of Computer Science at Virginia Polytechnic Institute and State University. He received B.S. and M.S. degrees from Boğaziçi University (Istanbul, Turkey) in 1975 and 1977, and M.S. and Ph.D. degrees from Syracuse University (N.Y.) in 1978 and 1981. He is currently the simulation and modeling category editor of ACM *Computing Reviews*, the program chairman of the SCS conference on Credibility Assessment, and the principal investigator for the U.S. Navy-funded research project in simulation model development environments. He has served as the vice-chairman of ACM SIGSIM (7/85 — 6/87), the program chairman and proceedings editor of the SCS conference on Methodology and Validation, and an associate editor of *Simuletter* (10/83 — 3/86). He has been a consultant for Planning Research Corporation, VM Software Inc., and Central Intelligence Agency. His current research interests center on simulation model development environments, credibility assessment of simulation results, performance evaluation, and software engineering. Professor Balci is a member of Alpha Pi Mu, ACM, IEEE CS, ORSA, and SCS.

Professor Osman Balci
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061
(703) 961-4841

RICHARD E. NANCE is a professor of Computer Science and the director of the Systems Research Center at Virginia Polytechnic Institute and State University. He received B.S. and M.S. degrees from N.C. State University in 1962 and 1966, and 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. Professor 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, the Outstanding Service Awards Subcommittee, the ad hoc Conference Procedures Committee and the ad hoc Film Committee that produced "Computers In Your Life."

He currently serves on the Editorial Panels of *Communications of the ACM* for research contributions in simulation and statistical computing, and *Journal of Operations Research and Computer Science* for contributions in simulation. The author of papers on discrete event simulation, performance modeling and evaluation, and computer networks, Professor Nance has served 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.

Professor Richard E. Nance, Director
Systems Research Center
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061
(703) 961-6144