Proceedings of the 1990 Winter Simulation Conference
Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)

MODEL GENERATION ISSUES IN A SIMULATION SUPPORT ENVIRONMENT

Osman Balci
Richard E. Nance
E. Joseph Derrick
Ernest H. Page
John L. Bishop

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

ABSTRACT

This paper provides an overview of a simulation support environment and presents a discussion of a number of issues related to the generation of a simulation model. The need for automated, computer-based assistance in the generation of simulation models of large, complex systems is undeniable. The required support goes beyond the programming process and extends to all phases of the simulation model development life cycle. In the last several years, we have been developing a domain-independent simulation support environment with four layers on a Sun computer workstation. The software tools of the environment are briefly described. The most challenging tool of the environment has been the Model Generator; the tool with which a modeler creates a simulation model specification under a conceptual framework. Our experience and a discussion of issues in developing the Model Generator software tool are presented.

1. INTRODUCTION

The ever-increasing complexity of systems being simulated mandates cost-effective, integrated and automated support of simulation model development throughout the entire model development life cycle. A simulation programming language supporting just the process of programming is not sufficient for complex, large systems. The required computer-aided assistance goes beyond the process of programming and extends to all phases of the simulation model development life cycle starting with system and objectives definition and culminating with integrated decision support.

Support of all phases of the life cycle is achieved in the form of an environment. An environment is an *integrated* set of hardware and software tools that provide cost-effective, automated support throughout the *entire* development life cycle. A collection of tools that is *not* integrated does not constitute an environment. A set of integrated tools not supporting all phases of the life cycle constitutes an incomplete environment. The supporting hardware tools could be a scanner, optical character recognizer, video digitizer, CD-ROM player, sound digitizer, laser printer, etc. The supporting software tools could be the ones described in Section 2.2.

In the last several years, we have been developing a simulation support environment on a Sun computer workstation. Among the tools that have been prototyped, the Model Generator tool has been the most challenging. The purpose of this paper is to provide an overview of the environment and to discuss the issues related to the development of the Model Generator tool. Section 2 presents the overview of the simulation support environment. Section 3 discusses the methodology, conceptual frameworks, automated assistance, automatic translation, human-computer interface, visualization, interactive multimedia technology, and rapid versus evolutionary prototyping issues in the generation of simulation models. Conclusions are given in Section 4.

2. AN OVERVIEW OF THE SIMULATION SUPPORT ENVIRONMENT

The Simulation Model Development Environments (SMDE) research project at VPI&SU has been developing an environment that can be characterized as a simulation support environment or a computer-aided simulation engineering environment. The SMDE project has addressed a complex research problem: prototyping a domain-independent discrete-event simulation support environment to provide a comprehensive and *integrated* collection of computer-based tools to:

(1) offer cost-effective, integrated and automated support of model development throughout the entire model life cycle;
(2) improve the model quality by 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], prototyping techniques have been used to develop the prototypes of SMDE tools on a Sun 3/160 computer workstation. The object-oriented paradigm enunciated by the Conical Methodology [Nance 1987] has furnished the underpinnings of the SMDE research prototype [Balci and Nance 1987a].

2.1 SMDE Architecture

Figure 1 depicts the architecture of the SMDE in four layers: (0) Hardware and Operating System, (1) Kernel SMDE, (2) Minimal SMDE, and (3) SMDEs.

2.1.1 Layer 0: Hardware and Operating System

A Sun 3/160 computer workstation running under MC68020 CPU with 8 megabytes of main memory, 380 megabytes of disk subsystem, a 1/4-inch cartridge tape drive, and a 19-inch color 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 SunOS 4.0 operating system and utilities, a graphical human-computer interface (SunView), device independent graphics library (SunCore), computer graphics interface (SunCGI), Sun programming environment (SunPro), and INGRES relational database management system (SunINGRES) constitute the software environment upon which the SMDE is built.

2.1.2 Layer 1: Kernel Simulation Model Development Environment

Primarily, this layer integrates all SMDE tools into the software environment described above. It provides SunINGRES databases, communication and run-time support functions, and a kernel interface. There are three SunINGRES 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 make the SMDE easy to maintain and expand. The kernel interface provides a standard communication protocol and a uniform set of interface definitions. Security

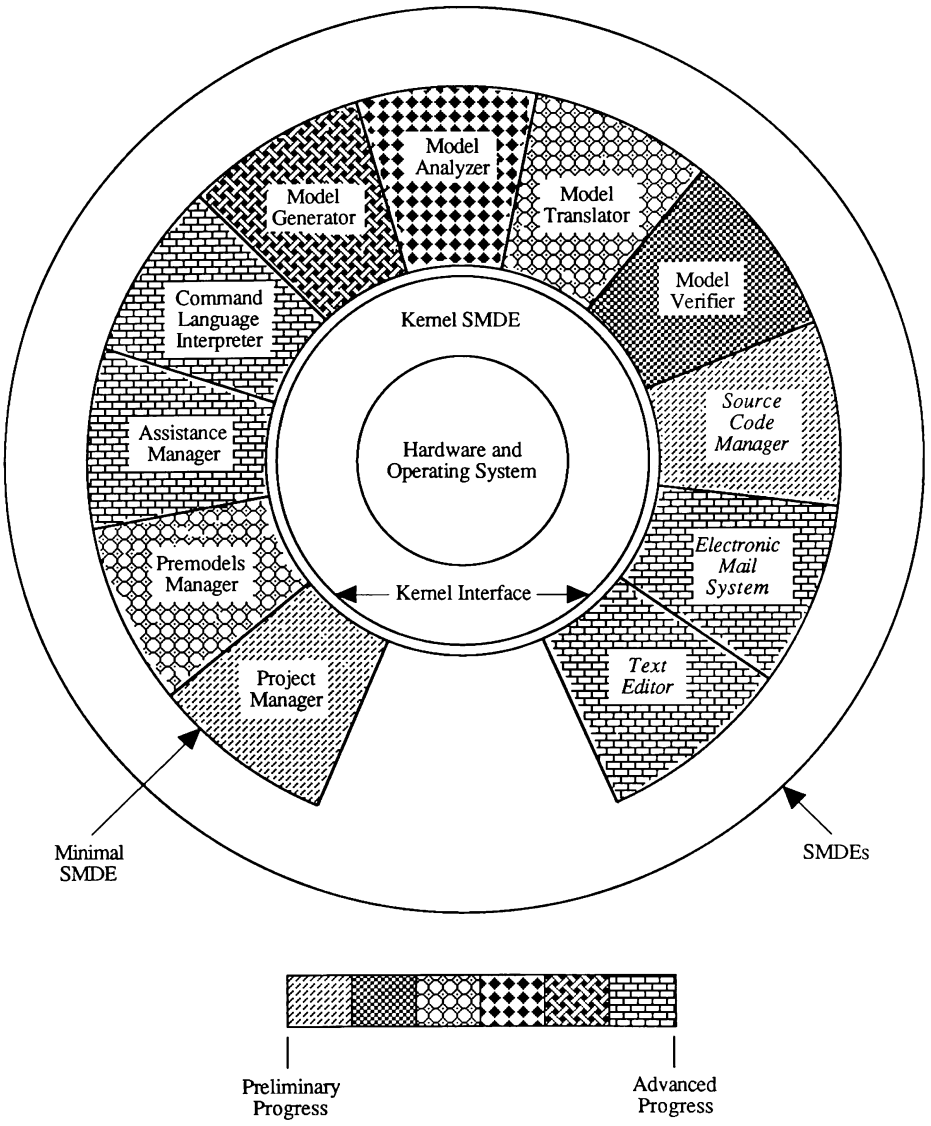

Figure 1. The Architecture of the SMDE and Current Status of Each Minimal SMDE Tool

protection is imposed by the kernel interface to prevent any unauthorized use of tools or data.

2.1.3 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. "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 or library tools) are expected to be provided by the software environment of Layer 0: Source Code Manager, Electronic Mail System, and Text Editor.

The current prototypes of the minimal SMDE tools are described in Section 2.2.

2.1.4 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, it incorporates tools that support specific applications and 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 further customizing

for a specific project, or additional tools may be needed to meet project 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 examples of tools in layer 3.

An 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 1 by the opening between Project Manager and Text Editor.

2.2 Description of Current Prototypes of Minimal SMDE Tools

Figure 1 shows the current status of each minimal SMDE tool on a scale from preliminary progress to advanced progress. Each tool with its current status is described below.

Project Manager is a software tool which: (1) administers the storage and retrieval of items in the project database; (2) keeps a recorded history of the progress of the simulation modeling project; (3) triggers messages and reminders (especially about due dates); and (4) responds to queries in a prescribed form concerning project status. Other than the preliminary design decisions, little work has been done on the Project Manager. Its development is dependent on design decisions taken for the other minimal SMDE tools; therefore, high-level design is being delayed until sufficient progress is achieved on the other tools.

Premodels Manager is a software tool intended to facilitate the reusability of earlier developed models or model components. Using this tool, a modeler searches the premodels database to identify earlier developed model components for reuse in the development of a new simulation model. An early prototype has been developed by Box [1984]. An advanced prototype of the Premodels Manager software tool is being developed.

The **Assistance Manager** software tool has been prototyped [Frankel and Balci 1989] to provide: (1) information on how to use an SMDE tool; (2) a glossary of technical terms; (3) introductory information about the SMDE; and (4) assistance for tool developers in supplying "help" information.

Command Language Interpreter (CLI) is the language through which a user invokes an SMDE tool. An early prototype, based on the design by Moose [1983], is described in [Humphrey 1985]. Following the acquisition of the Sun workstation, the SunView graphical user interface has served the CLI function.

Model Generator (the simulation model specification and documentation generator) is a software tool which assists the modeler in: (1) creating a model specification in a predetermined form which lends itself to formal analysis; (2) creating multi-level (stratified) model documentation, and (3) performing model qualification. Three research prototypes of the Model Generator have been developed. The first [Hansen 1984] implements the definition stage of the Conical Methodology [Nance 1987]. The second builds on the first by adding the specification stage of the Conical Methodology, mostly based on the Condition Specification [Barger 1986; Barger and Nance 1986; Overstreet and Nance 1985; Overstreet et al. 1986]. The third is based on a conceptual framework developed by O. Balci and implemented by Bishop [Bishop and Balci 1990]. Improved prototypes of the Model Generator are needed since it is the most crucial tool of the SMDE. Prior research [Balci 1988; Derrick 1988; Derrick et al. 1989] and Derrick's ongoing Ph.D. investigation is intended to surface new ideas in this critical area. Achieving the automation-based software paradigm [Balci and Nance 1987b] in the simulation modeling domain relies heavily on improved understanding of contributions by conceptual frameworks to knowledge extraction and model representation.

The **Model Analyzer** diagnoses the model specification created by the Model Generator and effectively assists the modeler in communicative model verification. The first prototype is described by Moose and Nance [1987]. The current research prototype implements a control and transformation metric for measuring model complexity [Wallace and Nance 1985], and provides diagnostic assistance using digraph representations of simulation model specifications [Nance and Overstreet 1987; Overstreet and Nance 1983, 1986].

Model Translator translates the model specification into an executable representation after the quality of the specification is assured by the Model Analyzer. Currently, a prototype of the Model Translator is employed within the General Purpose Visual Simulation System developed by Bishop and Balci [1990].

Model Verifier is intended for programmed model verification [Whitner and Balci 1989]. Applied to the executable representations, it provides assistance in substantiating that the simulation model is programmed from its specification with sufficient accuracy. Extensive groundwork [Balci 1987, 1990; Whitner and Balci 1989] has been conducted to prototype the Model Verifier, but development is delayed until a standard executable model representation is adopted for the SMDE.

Source Code Manager is a software tool which configures the run-time system for execution of the programmed model, providing the requisite input and output devices, files and utilities. Its development is being delayed until a standard executable model representation is adopted for the SMDE.

Electronic Mail System facilitates the necessary communication among project personnel. Primarily, it performs the task of sending and receiving mail through (local or large) computer networks. The Sun workstation's MailTool is currently used as the Electronic Mail System of the SMDE. The Sun computer workstation is a node on the Internet computer network with the node name of *mdesun.cs.vt.edu*.

Text Editor is used for preparing technical reports, user manuals, system documentation, correspondence, and personal documents. Currently, the vi editor serves as the text editor for the SMDE.

3. MODEL GENERATION ISSUES

3.1 Methodology

Often used without a clear meaning or as an embellishing substitute for "method," the term "methodology" is defined [Henry et al. 1985, p. 4] as a problem-solving procedure that should:

(1) organize and structure the tasks comprising the (problem-solving) effort to achieve global objectives,
(2) include methods and techniques for accomplishing individual tasks (within the framework of global objectives), and
(3) prescribe an order in which certain classes of decisions are made and the ways of making those decisions that lead to the desired objectives.

A methodology may presume a conceptual framework (described below) or prescribe one. In fact, the relationship between the two terms has become clouded, but the latter, in its original context, connotes a passive explanatory guide to describing the system (world) that is conducive to accomplishing (study) objectives. For example, mathematical programming provides a conceptual framework for solving optimization problems.

On the other hand, a methodology represents a more active, prescriptive set of directions on "how to do it." A methodology includes either a (possibly implicit) conceptual framework. Structured modeling [Geoffrion 1987] is a methodology that utilizes a mathematical programming conceptual framework, but may not be limited to that framework.

3.1.1 Roles of a Methodology

Nance and Arthur [1988] identify the dual roles of a methodology as: (1) contributing to the understanding of the development task and how to accomplish it, and (2) providing the requirements specification for a supporting environment. This latter role can be understood within the context of the Objectives/Principles/Attributes (OPA) framework for methodologies [Nance and Arthur 1988, pp. 221-222], which can be summarized in the following rationale:

> A project effort (modeling or software) has defined *objectives* which are desired. Achievement of the desired objectives can be assured through application of certain *principles* that should govern the effort. An effort governed by

such principles should lead to a product (model or software) that evinces distinguishing *attributes*.

Thus, principles establish the requirements for tools, e.g. a methodology advocating the principle of hierarchical decomposition stipulates the need for modeling utilities that support this top-down refinement of abstraction.

The roles of a methodology might be described as *conceptual*, i.e. the prescription of "how the modeler should see the world" and practical, i.e. setting forth the requirements for helping the modeler express "what is seen." Both roles are requisite in the methodology principles.

3.1.2 Effect of the Conical Methodology

The Conical Methodology has served both the conceptual and practical roles as the foundation for the SMDE. The objectives and principles are described in the work cited above, and the latter are repeated here:

(1) Top-down model definition followed by bottom-up model specification promotes model correctness and testability.
(2) Documentation and specification must be inseparable to assure model maintainability, adaptability, and reusability.
(3) Iterative refinement and progressive elaboration of model description are essential in the abstraction resolution required for assuring the correctness of large, complex models.
(4) Verification must be instituted as early as possible in the model development and continue throughout to enhance model testability and promote model correctness.
(5) Model specification should be independent of model implementation to promote model adaptability, reusability, and maintainability.

Table 1, taken from [Nance and Arthur 1988, p. 224], maps the principles above to the affected SMDE tools shown in Figure 1. Examination of this Table should convincingly demonstrate the practical role. Demonstration of the conceptual role, in a convincing fashion, requires use of an SMDE prototype.

3.2 Conceptual Frameworks

A *Conceptual Framework* (CF) is a structure of concepts under which a modeler is guided to represent a system in the form of a model. A CF is also called a *simulation strategy*, a *world view*, and a *formalism* in the literature. Object-oriented paradigm, Conical Methodology, process interaction, event scheduling, three-phase approach are among the 13 CFs Derrick et al. [1989] compare.

Conceptual Frameworks play an important role in four distinct approaches for the construction of a simulation model generator as depicted in Figures 2 through 5. In approach 1, the model generator employs a CF (e.g., object-oriented paradigm) under the guidance of which a user creates a specification which is then translated into an executable model. The CF of approach 1 is expected to be easily applicable for modeling a large class of problems.

For a particular application domain, a CF can be developed to provide much more detailed guidance and to facilitate model specification much more effectively in comparison with approach 1. This philosophy is adopted by approach 2 shown in Figure 3. For each application domain of interest, a model generator is built using a CF designed based on the domain specific knowledge. SIMFACTORY (factory simulation), COMNET (wide-area voice and data network simulation), NETWORK (computer communications network simulation), and LANNET (local area network simulation) are example commercial products (registered trademarks of CACI, Inc.) that use approach 2.

Several domain specific CFs can be incorporated in a model generator with a selector component in approach 3 illustrated in Figure 4. The CF Selector aids the user in selecting the most appropriate CF for the problem domain of interest. Once a CF is selected, the user generates the executable model following that CF's path that is independent of the paths of the other CFs. Such a model generator, although huge in amount of code, provides easy model specification for a number of problem domains. Potential duplication of the same specification under different CFs contributes to the large size of the generator; however, such duplication facilitates maintainability and modifiability of the generator.

Approach 4, shown in Figure 5, is similar to approach 3 except that the common elements of the CFs are shared within the model generator. The generator size is reduced at the expense of less maintainability and modifiability.

The Model Generator [Bishop and Balci 1990] of the SMDE uses approach 1 employing a new CF which is graphical, object-oriented, and activity-based.

3.3 Automated Assistance

A fundamental human limitation, the *hrair limit*, dictates that a human being cannot *simultaneously* handle more than 7±2 activities. A simulation modeler undoubtedly needs assistance in modeling a complex system with dozens or hundreds of simultaneous activities. The modeler needs methodological, conceptual, and computer-aided assistance to overcome the complexity of simulation modeling. A modeling methodology, a conceptual framework, and a model generator software tool can provide the required assistance in generating simulation models.

3.4 Automatic Translation

The translation of model specification created by the model generator into an executable model is certainly a challenging issue in all four approaches shown in Figures 2 through 5. Balzer et al. [1983] propose an automation-based software paradigm that incorporates the capabilities of automatic programming, program transformation, and a "knowledge-based software assistant." 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 67 to 80 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 can significantly reduce this problem. Balci and Nance [1987b] discuss the automation-based software paradigm for simulation support within the SMDE.

3.5 Human-Computer Interface

A graphical user interface (GUI) is a crucial element in providing the required computer-aided assistance in model generation. Early in the SMDE research project, a dumb-terminal interface to a UNIX computer has proven to be useless. The dumb-terminal interface, although menu driven, has resulted in a tedious, long, and ineffective interaction for extracting the knowledge necessary for model generation. After the acquisition of the SUN workstation in 1985, SunView GUI has provided the needed platform. However, some prototype tools developed under SunView employing a strictly menu-driven interface has also proven to be ineffective. Hence, a graphical, window-driven interface has been designed under a new CF as the GUI of the Model Generator tool in the SMDE.

3.6 Visualization

Two basic types of simulation model visualization exist: *Post-simulation animation* visualizes the input, internal, and output behaviors of a simulation model by using the simulation trace data generated from a completed simulation run. *Simulation-concurrent animation* visualizes the input, internal, and output behaviors of a simulation model as the simulation runs. The second type is required for providing an interactive visualization.

Development of a post-simulation animation requires the instrumentation (insertion of trap codes or hooks for the purpose of gathering data) of the simulation model. The instrumented simulation model is run to completion creating the trace data from which the model visualization is obtained. Simulation-concurrent animation imposes much more challenging demands on the construction of the model, but it provides the interactive capability. The model is constructed with the objective of visualization. Hence, an additional requirement is added to the development process making it more complex and demanding.

The added complexity is justified, however, by the additional

Table 1. Procedural Guidance for Environment Design Tool Functionality

	Conical Methodology Principle	*Procedural Guidance Derived From the CM Principle*	*Environment Tools Affected*
1.	Top-down model definition/bottom-up model specification	1.1 Definition must precede specification	Model Generator Premodels Manager
2.	Documentation and specification are inseparable	2.1 Model documentation is produced during model specification	Assistance Manager Project Manager Model Generator
		2.2 The model specification and consequent documentation should support different views (aspects) of the modeling task	
3.	Iterative refinement and progressive elaboration	3.1 The degree of detail of submodel description should be controlled by the modeler; submodel stubbing should be supported so that later addition of detail is facilitated	Premodels Manager Model Generator
		3.2 The functional expansion (progressive elaboration) of the model should be supported	
4.	Verification must begin with communicative models and continue throughout the development process	4.1 Diagnosis of model representation should begin as early as possible, certainly prior to the program form	Project Manager Model Analyzer Model Verifier
		4.2 Automated or semi-automated diagnosis is a requirement	
5.	Model specification is independent of model implementation	5.1 The execution (implementation) details should be ignored in the model development (specification) process	Model Generator Model Analyzer Model Translator Model Verifier

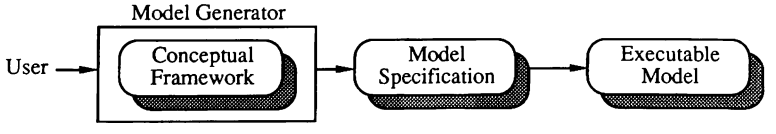

Figure 2. Model Generator Construction Approach 1

benefits to be gained from visualization. Most importantly, visualization facilitates verification, validation, and testing of the simulation model. Other benefits are described in [Bishop and Balci 1990].

3.7 Interactive Multimedia Technology

Interactive multimedia technology refers to the integrated use of video (motion), audio (sound), images, graphics, and text in an interactive fashion on a computer. Such a technology would be useful to educate the modelers about the operation of the system under study. A modeler can, for example, interactively watch a video of how a manufacturing system operates in one window on the computer, and in another window work on the development of its model. Knowledge of the system being modeled is crucial for developing successful simulation models. Therefore, interactive multimedia technology can provide significant assistance in model generation.

3.8 Rapid Versus Evolutionary Prototyping

The prototyping of SMDE tools has employed the design and development strategy generally described as rapid prototyping with

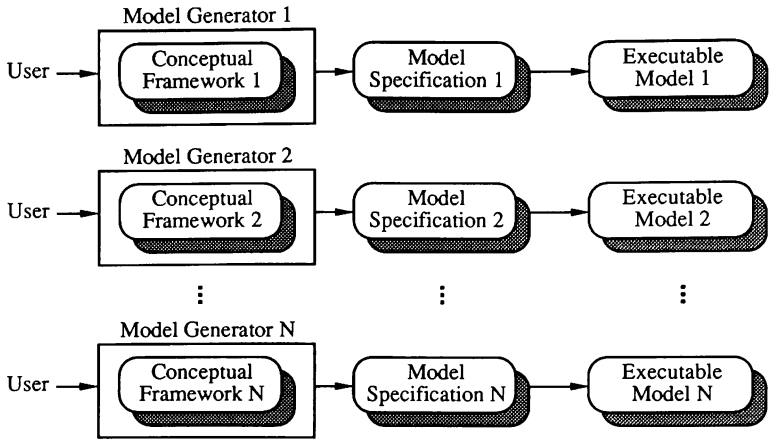

Figure 3. Model Generator Construction Approach 2

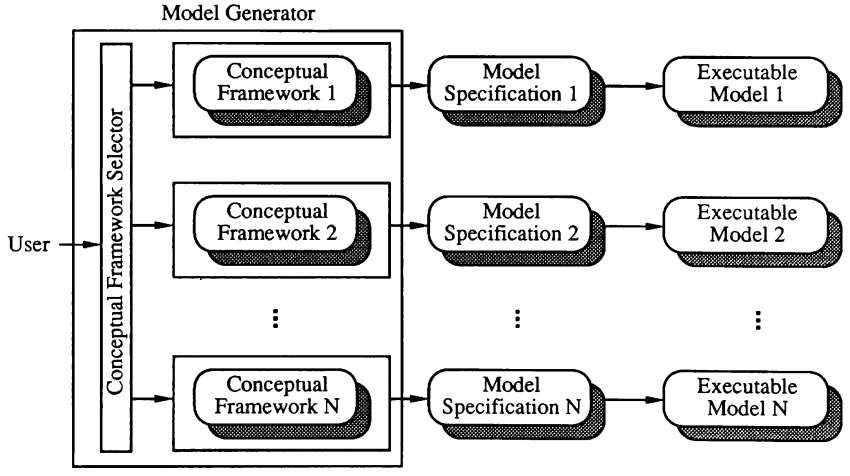

Figure 4. Model Generator Construction Approach 3

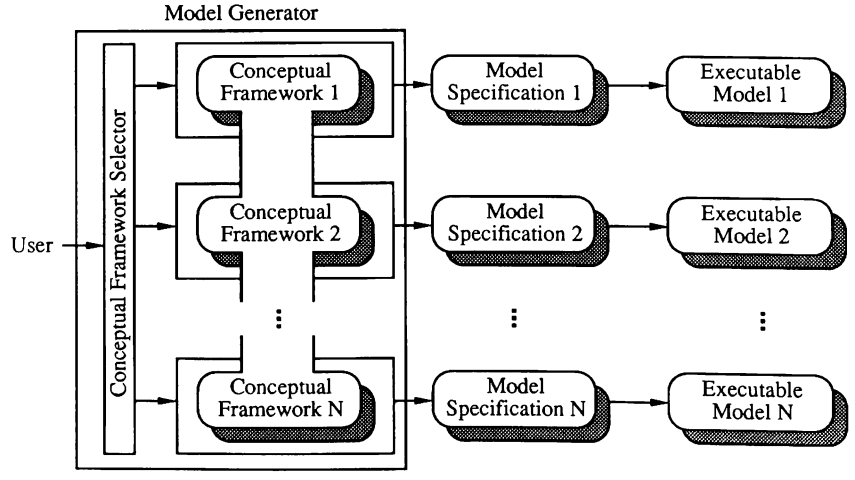

Figure 5. Model Generator Construction Approach 4

one *major* departure. Rapid prototyping maintains that the knowledge gained from development is foremost, and the actual product should be totally discarded so as not to inhibit design creativity in subsequent versions. With "evolutionary prototyping" the intent is to retain those portions of the prototype which represent the kernel for the next version. In essence, a balance is struck between the limitations on creativity and the efficiency of design, and potentially code, reuse. Our experience with evolutionary prototyping has been decidedly positive.

4. CONCLUSIONS

The ever-increasing complexity of systems studied via simulation mandates a software environment providing cost-effective, integrated and automated assistance throughout the entire simulation model development life cycle. The SMDE Research Project described herein is a major step towards such an environment. Development of the Model Generator software tool has been the most challenging in the SMDE Research Project. The Conical Methodology has provided the underpinnings of several Model Generator prototypes. The Conceptual Framework has been recognized as the most crucial element of the Model Generator. Extensive research has been conducted to develop the desired Conceptual Framework. Automated assistance, automatic translation, human-computer interface, visualization, interactive multimedia technology, and prototyping remain essential issues in the generation of simulation models.

ACKNOWLEDGMENTS

This research is based on past work sponsored in part by the U.S. Navy and on current efforts supported by IBM through the Systems Research Center at VPI&SU. The contributions of the following people to the SMDE project are gratefully acknowledged: Lynne F. Barger; Jay D. Beams; Charles W. Box; Valerie L. Frankel; Robert H. Hansen; Matthew C. Humphrey; Kevin E. Martin; David P. Maynard; Robert L. Moose, Jr.; C. Michael Overstreet; and Jack C. Wallace.

REFERENCES

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

Balci, O., Ed. (1987), *Proceedings of the Conference on Methodology and Validation*, SCS, San Diego, CA.

Balci, O. (1988), "The Implementation of Four Conceptual Frameworks for Simulation Modeling in High-Level Languages," In *Proceedings of the 1988 Winter Simulation Conference*, M.A. Abrams, P.L. Haigh, and J.C. Comfort, Eds. IEEE, Piscataway, NJ, 287-295.

Balci, O. (1990), "Guidelines for Successful Simulation Studies" In *Proceedings of the 1990 Winter Simulation Conference*, O. Balci, R.P. Sadowski, and R.E. Nance, Eds. IEEE, Piscataway, NJ,

Balci, O. and R.E. Nance (1987a), "Simulation Model Development Environments: A Research Prototype," *Journal of the Operational Research Society 38*, 8, 753-763.

Balci, O. and R.E. Nance (1987b), "Simulation Support: Prototyping the Automation-Based Paradigm," In *Proceedings of the 1987 Winter Simulation Conference*, A. Thesen, H. Grant, and W.D. Kelton, Eds. IEEE, Piscataway, NJ, 495-502.

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

Barger, L.F. and R.E. Nance (1986), "Simulation Model Development: System Specification Techniques," Technical Report SRC-86-005, Systems Research Center, Virginia Tech, Blacksburg, VA.

Bishop, J.L. and O. Balci (1990), "General Purpose Visual Simulation System: A Functional Description," In *Proceedings of the 1990 Winter Simulation Conference*, O. Balci, R.P. Sadowski, and R.E. Nance, Eds. IEEE, Piscataway, NJ, 504-512.

Box, C.W. (1984), "A Prototype of the Premodels Manager," MDE Project Memorandum, Department of Computer Science, Virginia Tech, Blacksburg, VA.

Derrick, E.J. (1988), "Conceptual Frameworks for Discrete Event Simulation Modeling," M.S. Thesis, Department of Computer Science, Virginia Tech, Blacksburg, VA.

Derrick, E.J., O. Balci, and R.E. Nance (1989), "A Comparison of Selected Conceptual Frameworks for Simulation Modeling," In *Proceedings of the 1989 Winter Simulation Conference*, E.A. MacNair, K.J. Musselman, and P. Heidelberger, Eds. IEEE, Piscataway, NJ, 711-718.

Frankel, V.L. and O. Balci (1989), "An On-Line Assistance System for the Simulation Model Development Environment," *International Journal of Man-Machine Studies 31*, 6, 699-716.

Geoffrion, A.M. (1987), "An Introduction to Structured Modeling," *Management Science 33*, 5, 547-588.

Hansen, R.H. (1984), "The Model Generator: A Crucial Element of the Model Development Environment," Technical Report SRC-85-004, Systems Research Center, Virginia Tech, Blacksburg, VA.

Henry, S.M., J.D. Arthur, and R.E. Nance (1985), "A Procedural Approach to Evaluating Software Development Methodologies," Technical Report SRC-85-008, Systems Research Center, Virginia Tech, Blacksburg, VA.

Humphrey, M.C. (1985), "The Command Language Interpreter for the Model Development Environment: Design and Implementation," Technical Report SRC-85-011, Systems Research Center, Virginia Tech, Blacksburg, VA.

Moose, R.L., Jr. (1983), "Proposal for a Model Development Environment Command Language Interpreter," Technical Report SRC-85-012, Systems Research Center, Virginia Tech, Blacksburg, VA.

Moose, R.L., Jr. and R.E. Nance (1987), "Model Analysis in a Model Development Environment," Technical Report SRC-87-010, Systems Research Center, Virginia Tech, Blacksburg, VA.

Nance, R.E. (1987), "The Conical Methodology: A Framework for Simulation Model Development," In *Proceedings of the Conference on Methodology and Validation,* O. Balci, Ed. SCS, San Diego, CA, 38-43.

Nance, R.E. and J.D. Arthur (1988), "The Methodology Roles in the Realization of a Model Development Environment," In *Proceedings of the 1988 Winter Simulation Conference*, M.A. Abrams, P.L. Haigh, and J.C. Comfort, Eds. IEEE, Piscataway, NJ, 220-225.

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

Overstreet, C.M. and R.E. Nance (1983), "Graph-Based Diagnosis of Discrete Event Model Specifications," Technical Report SRC-85-003, Systems Research Center, Virginia Tech, Blacksburg, VA.

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.

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*, M.S. Elzas, T.I. Ören, and B.P. Zeigler, Eds. North-Holland, Amsterdam, 165-179.

Overstreet, C.M., R.E. Nance, O. Balci, and L.F. Barger (1986), "Specification Languages: Understanding Their Role in Simulation Model Development," Technical Report SRC-87-001, Systems Research Center, Virginia Tech, Blacksburg, VA.

Wallace, J.C. and R.E. Nance (1985), "The Control and Transformation Metric: A Basis for Measuring Model Complexity," Technical Report SRC-85-007, Systems Research Center, Virginia Tech, Blacksburg, VA.

Whitner, R.B. and O. Balci (1989), "Guidelines for Selecting and Using Simulation Model Verification Techniques," In *Proceedings of the 1989 Winter Simulation Conference*, E.A. MacNair, K.J. Musselman, and P. Heidelberger, Eds. IEEE, Piscataway, NJ, 559-568.