Proceedings of the 1984
Winter Simulation Conference
S. Sheppard, U. Pooch, D. Pegden (eds.)

EVALUATION OF THE UNIX[+] HOST FOR A MODEL DEVELOPMENT ENVIRONMENT[*]

Richard E. Nance
Osman Balci
Robert L. Moose, Jr.

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

ABSTRACT

The needs for model development and the functionality of UNIX are reviewed in order to evaluate the capability of UNIX for either hosting an environment or supporting the development of an environment. Based on an ideal system comparison, the deficiencies of UNIX serve to define the second iteration in a rapid prototyping effort.

THE NEED FOR IMPROVED MODEL MANAGEMENT

Almost ten years have past since the General Accounting Office indictment of management and modeling techniques based on a sample of Federally funded computer models [28]. The response from the simulation modeling community, expectably diverse, has been generally supportive of critical examination of modeling practices.

The diversity of responses to the shortcomings in simulation modeling technology is indicative of a broad range of activities and, in truth, a recognition of the need for improvements prior to the appearance of the report. The subject of this paper is a response based on the generalization of critical needs assessments for large simulation projects involving multiple users, a modeling team, and a relatively long development period (one to five years or more). These are considered to be modeling tasks where the challenge mandates the proper use of "model management" [20].

Model Development within Model Management

An earlier paper [21, p. 175] defines a Model Management System (MMS) as "... a set of tools that assist in the efficient creation and use of an effective model whose application is expected to extend in scope and time beyond the original study objectives." The MMS requirements are based on the needs of users who are managers as well as those who are analysts and programmers. The support provided by a MMS extends over the entire model life cycle, which is shown in Figure 1.

Within the model life cycle, the category of activities grouped as the Model Development Phases constitute the most human-intensive efforts. The formulation of a conceptual model of a system, the expression of communicative models, and the resolution of variations among communicative models that are inevitable in a team effort represent intensively creative activities.

By separating the processes of model formulation and representation from the programming process, the distinctive characteristics of a new modeling paradigm are given visibility:

(1) separation of model development from model execution,

(2) assistance to the modeler in the formation and expression of concepts unfettered by syntactic and semantic requirements for program execution, and

(3) employment of automated diagnostic techniques in the attempt to identify and correct errors well before they are submerged in code.

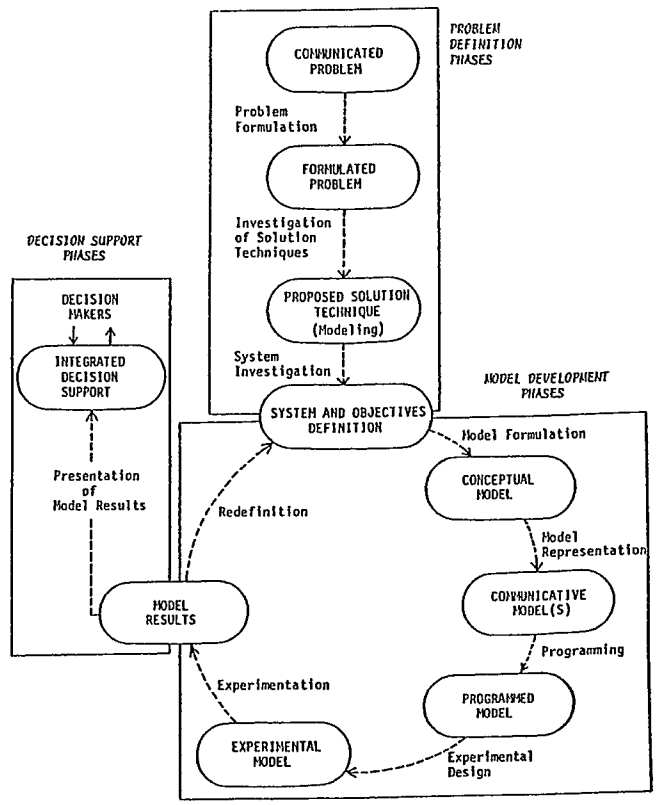

Figure 1. Phases in the Chronological Periods of the Model Life Cycle.

[+] UNIX is a trademark of Bell Laboratories.

[*] Research contributing to this paper was supported in part by the U.S. Navy under Contract No. N60921-83-G-A165 through the Systems Research Center, Virginia Tech.

The Model Development Phases

The arrows of Figure 1 represent processes, wherein computer assistance to the modeler can make the creative task more productive, i.e. completed in less time with fewer errors and greater model credibility. Model development encompasses the activities of producing an initial model, with verification and validation, prior to the generation of results. However, model development *also* includes the activities stemming from the redefinition of a model based on the desire to extend its applicability or to address additional objectives. In meeting the requirements of the MMS, data reflecting the efforts expended in the development processes and the evolving status of model components must be captured and organized for subsequent access.

The model development processes portrayed in Figure 1 emphasize the role of the analyst or simulation software development manager [21] and deemphasize that of the programmer. Assistance is rendered in the expression of concepts and the production of diagnosable communicative model specifications, from which executable model implementations can ultimately be algorithmically generated.

Related Work

Recognition of the need for improved simulation modeling technology began with a study supported by the National Bureau of Standards [17]. A companion study, encompassing techniques other than simulation, led to a series of papers on model validation and evaluation by Gass, see [9] for a culminating description. The concept of a Simulation Model Specification and Documentation Language (SMSDL), advanced in [17], is described in the context of a review of simulation model representation techniques in [18]. More recent indepth treatments of simulation model representation are to be found in [16] and [30].

The relationship of a model development environment to a software development environment (software engineering environment, programming support environment) is readily acknowledged. From this perspective the ideas of Lehman [13, 14, 15] have been influential. The importance of a methodological framework for software development, cited by Henriksen [10, p. 1061], is emphasized by the use of the Conical Methodology [19] in the implementation described in the subsequent paragraphs.

In terms of research in simulation modeling, the paper by Frankowski and Franta [8] has helped to solidify certain concepts. The works of Zeigler [29], Oren and Zeigler [23], and Oren [22] have been influential, and the current research shares some common objectives with the efforts of Unger, Birtwistle, et. al. involved in Project JADE at Calgary [24]. Adelsberger [1] is pursuing a related path confined to the investigation of an Ada Simulation Support Environment.

Finally, the framework provided by the Conical Methodology and the objectives set forth in the MDE requirements [3] promote a paradigm for model development that conforms with the software development paradigm enunciated by Balzer, Cheatham and Green [4]. As a result, tools created for support of the "redefined" programming task are likely to find use in the programming-related processes of the model development activities.

Objective

The objective of this paper is to report the conclusions of an evaluation of UNIX as the host operating system for a Model Development Environment. This evaluation was based on a comparison of the functionality provided by UNIX with the stated needs and requirements for a MDE [3]. In essence, this work used rapid prototyping to answer the following question: to what extent could the MDE requirements be met strictly by the UNIX Programming Environment [12]. The answer to this question serves to provide the groundwork for the design of a MDE hosted by UNIX, which is now in progress.

COMPOSITION OF THE MODEL DEVELOPMENT ENVIRONMENT

Similar to the ADA* Programming Support Environments [2], the MDE is composed of four layers as depicted in Figure 2 (taken from [3]) and briefly described below.

The MDE Components

Computer hardware and the host UNIX operating system at layer 0 constitute the core upon which the MDE is built. The Kernel MDE (KMDE) at layer 1 integrates all MDE tools into the UNIX programming environment. It provides communication and run-time support functions, a Kernel Interface, and Project, Premodels, and Assistance Databases.

The minimal MDE (MMDE) at layer 3 provides a "comprehensive" set of tools which are "minimal" for the development and execution of a simulation model. There are two categories of MMDE tools. The first contains the Project Manager, Premodels Manager, Assistance Manager, Command Language Interpreter, Model Generator, Model Analyzer, Model Translator, and Model Verifier. Source Code Manager, Electronic Mail System, and Text Editor form the second category MMDE tools and are provided by the UNIX programming environment. The MDE at layer 3 incorporates tools that support specific applications and are of special interest only within a particular project or of interest only to an individual modeler. The functionality of each MMDE tool is explained briefly.

The *Project Manager* supervises all activities performed on the project database which acts as the central repository of the MDE. The *Premodels Manager*, employing a query language, assists the user in retrieving a prefabricated "certified" model (component) stored in the premodels database. The *Assistance Manager*, using the information stored in the assistance database, processes all "help" requests issued by any MDE tool. The *Command Language Interpreter* (CLI) is the means by which the user invokes a tool in the environment. The *Model Generator* is an interactive tool

---

* ADA is a registered trademark of the U.S. Department of Defense Ada Joint Program Office.

which, employing the Conical Methodology [19], aids the modeler in creating simulation model specification and documentation in a non-executable but analyzable language. The *Model Analyzer* diagnoses the specification created by the model generator to detect errors as early as possible in the model development life cycle. The *Model Translator* transforms the model specification into the syntax of a (simulation) programming language following the correction of errors detected in the model specification by the model analyzer. The *Model Verifier* aids the modeler in substantiating that the translation of the earlier representations of the model into the programmed model has been done accurately. The *Source Code Manager* supervises the translation of the (simulation) programming language representation of the model into the machine language and performs its execution. The *Electronic Mail System* facilitates the necessary communication among people involved in the project. The *Text Editor* is used for all text processing needs, including the preparation of user manuals, system documentation, correspondence, and personal documents.

Other tools may be required for meeting special requirements of a simulation project. The open-endedness feature of the MDE provides for easy integration of added tools into the environment.

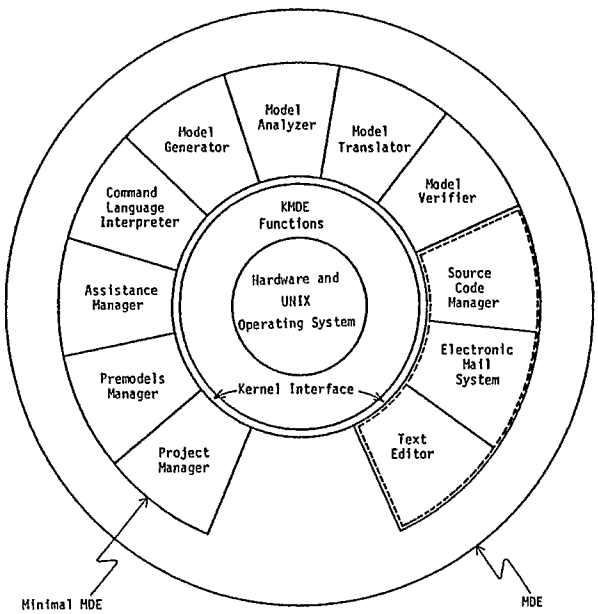

Figure 2. Layered illustration of the components of a Model Development Environment.

Relationships Among the MDE Components

Using a UNIX Shell command, the user activates the CLI to enter the MDE mode. The CLI provides the capability to escape to the UNIX programming environment enabling the user to make a switch and use a tool within a (possibly restricted) UNIX programming environment. The CLI interfaces with all the MMDE tools and any other tool added at the MDE layer which can be invoked directly by the user.

The menu driven Model Generator interacts with the Project Manager during the creation of the model specification and documentation which are stored in the project database. Some of the menus of the Generator contain the "help" option providing local and global assistance to the user. For example, in classifying an attribute as "indicative" or "relational", the user may choose the local "help" option to see the definitions of these classes. In this case, the Model Generator passes the request to the Assistance Manager which in turn displays the definitions and passes the control back to the Generator. By choosing the global "help" option during the use of a tool or by using the CLI, the user can activate the Assistance Manager to get tutorial information about, for example, the Conical Methodology or concerning the use of a MDE tool.

The Model Analyzer interacts with the Project Manager to act upon a representational form of model specification in the project database for the purpose of diagnostic analysis. The Model Translator transforms the specification in the project database by interacting with the Project Manager. The transformed specification which is also stored in the project database can be completed by using the Text Editor if not already complete. The Model Verifier also interacts with the Project Manager to act upon the source code and earlier model representations. The Source Code Manager interacts with the Project Manager in configuring the run-time system.

Relation to the Kernel Functions and the Operating System

Basic run-time support functions are provided by the KMDE for all tools that execute within the MDE. The KMDE provides the necessary functions for accessing the Project, Premodels, and Assistance Databases. The communication capability between the MDE tools and a fixed set of terminal interface control functions are also provided by the KMDE.

The interface between the MDE tools and the KMDE is called the Kernel Interface. All MDE tools communicate with or invoke each other *only* through this interface. An added tool at the MDE layer is integrated into the environment through the Kernel Interface, indicated by the opening between the Project Manager and Text Editor in Figure 2. The Kernel Interface allows the user to interact with the invoked tool and to exercise control over the tool. Protection is imposed within the Kernel Interface to prevent any unauthorized use of a tool or data.

Although the Kernel Interface can be made machine-independent, the MDE is dependent upon the UNIX operating system. Thus, the MDE is as portable as the UNIX system.

THE UNIX OPERATING SYSTEM

Brief Overview of UNIX

Since the UNIX operating system became generally available (around 1974 or 1975 [7, 12]), the number of UNIX installations has experienced a remarkable increase. With vendor supported versions, as well as numerous microcomputer implementations of UNIX and UNIX look-alikes now available, the rate of this increase has reached an explosive level. Kernighan and Pike have estimated that there exist "tens of thousands" of UNIX installations world-wide [12, p. vii]. This section provides a brief introduction to some important aspects of UNIX and its development which contribute to this widespread popularity.

UNIX consists of (1) a kernel which supports "a hierarchical file system ..., compatible file, device, and interprocess I/O, (and) the ability to initiate asynchronous processes" [26, p. 1905], and (2) "over 100 subsystems including a dozen languages" [26, p. 1905] (see also [12, p. 1, p. 201]). The command interpreter, known as the shell[+], is sometimes regarded by users as a distinct and special component of the operating system. As noted by Bourne, the shell "is the most important communication channel between the system and its users." [6, p. 1955] However, it is treated by the kernel as an ordinary system program and thus belongs to the set of UNIX subsystems.

An additional, significant characteristic of UNIX is that most of its kernel and software is written in the high level language C, which has produced the benefits of easier system modification and portability. (Only "a few modules" must be rewritten to transport UNIX to a different machine [25, p. 1964].) The disadvantages of increased size, slower execution, and system compilation and linking overhead were considered acceptable by the UNIX developers [25, pp. 1964-1965].

UNIX has evolved through several versions before reaching its current state (or states). The following milestones, extracted from [7, pp. 2-3], trace this evolution through Bell's Seventh Edition and the VAX 11/780 based versions:

1969-1970: A preliminary kernel was implemented on a PDP-7 by Thompson and then improved, with some utilities added, by Thompson and Kernighan.

1970-1971: UNIX was transported to a PDP-11/20. The First Edition was subsequently documented by Thompson and Ritchie.

1972-1973: The C language was developed and added to UNIX, which was then rewritten in C.

1975: The Sixth Edition became generally available.

1979: Bell's Seventh edition, which resulted from the transportation of UNIX to a Interdata 8/32, became generally available.

1979-1983: The transportation of UNIX to a VAX 11/780 by Reiser and London produced 32V. Further development of 32V was done by the University of California at Berkeley. Bell eventually released its current version, System V.

Philosophy

A number of features in UNIX are motivated by similar features in previously developed operating systems [26, p. 1928], including GENIE, Multics, TENEX, the Cambridge Multiple Access Systems, and CTSS [6, p. 1972]. The development of UNIX is marked by two significant characteristics: (1) designed by programmers for interactive use by programmers, and (2) developed under size constraints, which "encouraged not only economy, but also a certain elegance of design" [26, pp. 1926-1927]. These charactistics influence a philosophy of design and use which stresses consistency (uniformity), modularity, and simplicity.

Files, devices, pipes, and input/output.

Consistency and simplicity in UNIX is evident in the implementation and use of files, devices, and pipes (interprocess communication channels) and the associated input/output mechanisms. The hierarchical (not quite tree structured) file system contains ordinary files, directories, and special files. An ordinary file consists of a sequence of bytes with no structure or size constraints imposed on it by the system. The structural interpretation of the contents of a file depends on the programs which use it. Use of the "read" and "write" I/O calls, which access the bytes of a file sequentially, is straightforward. Additionally, a byte-wise random access capability exists [26, p. 1907, pp. 1911-1913; 25, pp. 1950-1951].

Directories and devices are accessed through the same mechanisms used with ordinary files. Directories are readable but not writable by nonprivileged programs, and the contents of a directory have a highly specific meaning. At least one special file exists for each device. To perform device I/O, the appropriate special file is opened, read, and written as if it were an ordinary file. Accessing a special file causes the activation of the associated device [26, pp. 1908-1910; 25, pp. 1954-1955].

A communication channel between two processes is established through the "pipe" system call. Processes thus connected communicate by treating pipes as open files and using the file system read and write calls [26, pp. 1917-1918].

The Shell.

The UNIX shell is a command language but also provides constructs normally found in high level programming languages. Facilities provided by the shell encourage the use of a certain philosophy of program and system development. Under this philosophy, single purpose tools are created and then combined, using capabilities of the shell, to accomplish more complex tasks. To enhance the utility of this composition technique, basic tools generally do not impose complex format requirements on their input and often produce unstructured output. Then the output of one tool can be used as input for another without intermediate processing.

Standard input and output, input/output redirection, and pipes are central in supporting this philosophy. Each "simple command" to the shell executes in a separate process, which begins with a number of open files. The files with "descriptor" numbers 0 and 1 are known as standard input and standard output respectively, and by default are associated with the user's terminal. These files are available to the command for general I/O. Input/Output redirection is used to associate standard input or output with (disk or device) files other than the terminal. The pipe operation, an extension of the redirection mechanism, causes "the standard output of one command (to be) connected to the standard input of another" [6, p. 1973]. Pipes provide a general composition method in which groups of simple commands are combined to perform high level transformations on the initial input [6, pp. 1973-1974; 25, p. 1957].

Composition of tools into shell scripts (as these composites are known) is facilitated further by the following features of the shell:

[+] The shell commonly used in most of the Bell versions of UNIX has become known as the Bourne Shell, after its creator, S.R. Bourne, to distinguish it from other shells.

- o Input of scripts from disk files.
- o Control constructs including the while loop and the conditional branch.
- o String-valued variables.
- o Parameter passing mechanisms.

Additional features of the shell include asynchronous process initiation and error and fault handling [6, pp. 1973-1986].

Other Subsystems

The remaining subsystems can be divided into a variety of functional categories. Briefly, the following constitute several of the major categories[+] [5, p. 1979]:

- o The C programming language and related utilities, library routines, and system calls.
- o Language implementation subsystems.
- o File manipulation utilities.
- o Text editors.
- o Document formatting subsystems.
- o Status inquiry commands.

HOSTING THE MODEL DEVELOPMENT ENVIRONMENT WITH UNIX

Having established the need for improved model management, characterized the requirements for effective model development, and explained the functionality provided by UNIX, the background exists for assessing the capabilities of UNIX as a MDE host. This evaluation begins with the definition of an ideal MDE host and proceeds with an assessment of UNIX functionality in support of the MDE tool and Kernel requirements. During the research effort leading to this paper, two related evaluation tasks emerged: (1) UNIX as a MDE host and (2) UNIX as a *tool for developing* model development environments. The capabilities of UNIX for supporting both tasks must be addressed, and the underlying reasons are made apparent in the following discussion.

The Ideal MDE Host

Based on the earlier description, the functions of a host operating system can be viewed on three levels:

(1) the operating system support for the operational environment, i.e. task scheduling, resource allocation, file handling, protection for an interactive multi-user application (layer 0);

(2) the support given to the Kernel to enable tool interaction and user flexibility in the employment of MDE tools and operating system utilities (layer 1); and

(3) the support provided to the MDE tools through needed utilities and functions supplied by the host.

Each level can be idealized in terms of the MDE support.

The interactive setting for model development is an absolute requirement, and the ideal host would maintain the quick response time, unaffected by load variations, so essential to support the creative development task. Task scheduling and resource allocation should contribute to maintaining the interactive setting and to the preservation of data and computational integrity. File handling should support a simple, consistent set of options, protecting the MDE user from catastrophic errors of commission or omission.

The ideal host would subsume much of the functionality required of the MDE tools. Not only the text editing, source code management, and electronic mail functions, but also the functionality required for model generation, analysis, translation, and verification would be offered by the host. Furthermore, the host services would extend beyond file handling to include the capabilities for database management required by the three managers (Project, Premodels, and Assistance). User interaction with the operating system would obviate the CLI requirements for menu-driven dialogue and human engineering.

Finally, partitioning the extensive functionality of the host should be unconstrained so that communication services provided by the Kernel would enable broad flexibility in the interactions among MDE tools. Database access functions should represent no constraints on the three managers in their interfacing with users. Terminal control should provide the flexibility needed by any of the MDE tools and furnish any desired level of protection for tool invocation.

UNIX Support of MDE Requirements

No existing operating system can meet the expectations for the ideal host, but selection of UNIX for the initial MDE prototype reflects the conviction that no other alternative offers as many advantages. The strategy employed in the rapid prototyping of the MDE is simple:

(1) learn the extent to which UNIX can meet the expectations of the ideal host,

(2) determine the major and minor deficiencies with respect to each tool and the Kernel,

(3) identify the UNIX tools that can contribute to removing the deficiencies,

(4) evaluate (subjectively) the benefit/effort ratio for obtaining the functionality required by each tool and the Kernel, and

(5) design the next prototype based on the evaluation.

Note that the intent of the initial prototyping effort requires the evaluation of UNIX in supporting both tasks mentioned earlier: (1) as a MDE host and (2) as a system for developing MDEs.

Table 1 provides a brief summary of the quite extensive evaluation. Several points should be noted regarding interpretation of the Table 1 entries:

[+] The omission of any subsystem or category from this list is not meant to imply a lack of importance.

MDE Component	Abstract of Requirements	UNIX Tools Providing Some Functionality	Major(M) and Minor(m) Deficiencies	UNIX Tools Contributing to Development	Benefit/Effort Ratio
Project Manager	Administer the Project Database, record project history, generate "triggers" and "alerters".	dbm, calendar, mkey, hunt, inv	Insufficient capability for "alerters" and "triggers" (M). Relational db support lacking (M). Query language and security lacking (M).	stdio, curses, termcap	high/med
Premodels and Assistance Managers	Administer respective databases: access, security, reorganization.	dbm, mkey, inv, hunt	Second and third from above.	stdio, curses, termcap	med/med
Command Language Interpreter	Menu-driven interface, extensible, human-engineered.	sh, csh, system calls	No menu capability (M). Lack of window management (M).	yacc, lex, ar, curses, termcap	high/high
Model Generator	Create model specification and multilevel documentation, assist in model qualification.	vi, ed	Lack of window management (M). Little capability for interactive dialogue (M).	stdio, curses, termcap	high/high
Model Analyzer	Diagnose model specifications and assist in communicative model verification.	spline, plot graph	Limited capability for graph production (M). No operations on graphs (m).	yacc, lex, grep	high/high
Model Translator	Translate model specification into executable representation for some abstract machine.	None	No model translation capabilities (M).	yacc, lex	high/high
Model Verifier	Verification of program representation based on communicative representations.	adb, sdb, lint, Clib (assert)	Executable code restrictions for sdb (m). Use of adb and sdb for debugging only (m).	stdio, curses, termcap adb, sdb	med/high
Source Code Manager	Construct the run time system for experimentation with the program representation.	None	Absence of simulation programming language (M).	SPL dependent.	high/(low-high)
Electronic Mail System	Facilitate communication among project participants.	mail, write	No deficiencies.	Not applicable.	low/low
Text Editor	Text formatting, creation and production.	vi, ed, sed	No deficiencies.	Not applicable.	high/low
Kernel	Database access and protection; communication among tools, run-time support; terminal controls.	sh, csh, system calls	Limited process communication primitives, other than semaphores in System V (M). Communication is restricted (m).	stdio, sh, csh, ar, system calls	high/med

Table 1. A Brief Summary of UNIX Support of MDE Requirements.
(A brief explanation of each UNIX utility is given in the Appendix.)

(1) Only the UNIX operating system is considered; extensions to broaden the utility of UNIX such as Programmers Workbench (pwb), etc. are not included.

(2) The interest is in "generic" UNIX, i.e. capabilities found in System V or BSD4.2, and not in one version at the exclusion of the other, or in particular manufacturers' hybrid versions.

(3) Utilities that would obviously contribute to the development of every MDE tool, such as the text editors ed and vi or the C language and programming library (Clib) are omitted from the fifth column. Many of the utilities identified in the concluding section describing UNIX are ubiquitous in program development and are omitted.

The absence of entries in the row corresponding to the Source Code Manager reflect the lack of a SPL within UNIX. However, translators for GPSS/H [11] and SIMULA 67 [27] are available from other sources. The range in development effort is based on the approach taken to realize model execution -- purchase (low) or develop (high).

CONCLUDING SUMMARY

Having established the need for improved model management, the investigation described herein focuses on the model development phases of the model life cycle. The Conical Methodology serves to identify a new modeling paradigm, providing a framework for the definition of a Model Development Environment. The implementation of such an environment hosted by the UNIX Operating System is under investigation using rapid prototyping.

A review of MDE requirements and UNIX functionality precedes the description of the ideal MDE host. Judging UNIX against an ideal admittedly forces a harsh comparison, but the severe assessment is necessary in reaching the difficult decisions regarding the MDE tools to be developed in the second prototype. Prototypes of the Command Language Interpreter, Model Generator, Model Analyzer, Model Verifier, and the Kernel are under development.

ACKNOWLEDGMENTS

The authors acknowledge the contributions of C. W. Box, M. Humphrey, T. B. Massie, C. M. Overstreet, and J. C. Wallace during this work. We are indebted to A. V. Box for the excellent formatting, organization, and typing of the manuscript.

APPENDIX: BRIEF DESCRIPTIONS OF SOME UNIX TOOLS

adb:	debugger for use on object code and core image files.
ar:	file library maintenance utility.
calendar:	program to print reminders near specified dates.
Clib(assert):	assertion routine in C subroutine library.
csh:	C shell command language.
curses:	curser movement and screen I/O package.
dbm:	database management routines.
ed:	text editor.
graph:	graph drawing utility.
grep:	program to print lines from a file which match a specified pattern.
hunt:	search and retrieval routine for inverted indeces.
inv:	index creation routine for inverted indeces.
lex:	lexical analyzer generator.
lint:	C program checker which detects possible defects generally undetected by the C compiler.
mail:	interuser mail utility.
mkey:	key maker for inverted indeces.
plot:	graphics output program and subroutines compatible with several types of terminals.
sdb:	symbolic debugger.
sh:	Bourne shell command language.
spline:	curve interpolation program.
stdio:	standard buffered I/O routines for use in C programs.
system calls:	entries to low level system routines (for I/O, process management, and other operations).
termcap:	database of terminal capabilities and routines which use this database.
vi:	screen oriented text editor.
write:	program for interactive, interuser communication.
yacc:	compiler generator.

REFERENCES

[1] Adelsberger, H.H., "Interactive Modeling and Simulation of Transaction Flow or Network Models Using the ADA Simulation Support Environment," *Proc. Winter Simulation Conference,* Arlington, VA, 1983, pp. 561-570.

[2] Advanced Research Projects Agency, "Requirements for ADA Programming Support Environments -- 'STONEMAN'," U.S. DoD, Arlington, VA, 1980.

[3] Balci, O., "Requirements for Model Development Environments," Technical Report CS83022-R, Department of Computer Science, Virginia Tech, Blacksburg, VA, 1983.

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

[5] Bell Telephone Laboratories, "UNIX/32V Time-Sharing System: UNIX Programmer's Manual," Version 1, 1, (with Preface to the Seventh Edition), Bell Telephone Laboratories, Murray Hill, NJ, 1979.

[6] Bourne, S.R., "The UNIX Shell," *Bell System Technical Journal, 57* (6), Part 2, 1978, pp. 1971-1990.

[7] Bourne, S.R., *The UNIX System*, Addison-Wesley, London, 1983.

[8] Frankowski, E.N. and Franta, W.R., "A Process Oriented Simulation Model Specification and Documentation Language," *Software -- Practice and Experience, 10* (9), 1980, pp. 721-742.

[9] Gass, S.I., "Decision-Aiding Models: Validation, Assessment, and Related Issues for Policy Analysis," *Operations Research, 31* (4), 1983, pp. 603-631.

[10] Henriksen, J.O., "The Integrated Simulation Environment (Simulation Software of the 1990's)," *Operations Research, 31* (6), 1983, pp. 1053-1073.

[11] Henriksen, J.O., Personal Communication, May 1984.

[12] Kernighan, B.W. and Pike, R., *The UNIX Programming Environment*, Prentice-Hall, Englewood Cliffs, NJ, 1984.

[13] Lehman, M.M., "The Software Engineering Environment," Research Report, Department of Computing Science, Imperial College, London, 1979.

[14] Lehman, M.M., "Programs, Programming and the System Life Cycle," Research Report 80/6, Department of Computing Science, Imperial College, London, 1980.

[15] Lehman, M.M., "Program Evolution," Research Report 82/1, Department of Computing Science, Imperial College, London, 1982.

[16] Mathewson, S.C., "The Application of Program Generator Software and Its Extensions to Discrete Event Simulation Modeling," *IIE Transactions, 16* (1), 1984, pp. 3-18.

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

[18] Nance, R.E., "Model Representation in Discrete Event Simulation: Prospects for Developing Documentation Standards," In: N. Adam and A. Dogramaci (editors), *Current Issues in Computer Simulation*, Academic Press, New York, NY, 1979, pp. 83-97.

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

[20] Nance, R.E. and Balci, O., "The Objectives and Requirements of Model Management," In: M. Singh (editor-in-chief), *Encyclopedia of Systems and Control*, Pergamon Press, Oxford, to appear in 1985.

[21] Nance, R.E., Mezaache, A.L., and Overstreet, C.M., "Simulation Model Management: Resolving the Technological Gaps," *Proc. Winter Simulation Conference*, Atlanta, GA, 1981, pp. 173-179.

[22] Oren, T.I., "Computer Aided Modeling Systems (CAMS)," Plenary Address, *Simulation '80 Symposium*, Interlaken, Switzerland, 1980.

[23] Oren, T.I. and Zeigler, B.P., "Concepts for Advanced Simulation Methodologies," *Simulation, 32* (3), 1979, pp. 69-82.

[24] Project JADE, "Papers for the Conference on Simulation in Strongly Typed Languages," Department of Computer Science, University of Calgary, Alberta, Canada, 1984.

[25] Ritchie, D.M., "A Retrospective," *Bell System Technical Journal, 57* (6), Part 2, 1978, pp. 1947-1969.

[26] Ritchie, D.M. and Thompson, K., "The UNIX Time-Sharing System," *Bell System Technical Journal, 57* (6), Part 2, 1978, pp. 1905-1929.

[27] "S-Port Simula on VAX/UNIX," *SIMULA Newsletter, 12* (3), 1984, p. 16.

[28] U.S. General Accounting Office, "Ways to Improve Management of Federally Funded Computerized Models," LCD-75-111, Washington, D.C., 1976.

[29] Zeigler, B.P., "Concepts and Software for Advanced Simulation Methodologies," In: T.I. Oren, C.M. Shub, and P.F. Roth (editors), *Simulation with Discrete Models: A State-of-the-Art View*, IEEE, New York, NY, 1980, pp. 25-44.

[30] Zeigler, B.P., "System-Theoretic Representation of Simulation Models," *IIE Transactions, 16* (1), 1984, pp. 19-34.