Technical Report CS83022-R

REQUIREMENTS

FOR

MODEL DEVELOPMENT ENVIRONMENTS*

Osman Balci**

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

25 October 1984

Revised: 6 February 1985

---

*Research partially supported by the Office of Naval Research and the Naval Sea Systems Command under Contract No. N60921-83-G-A165 through the Systems Research Center at VPI&SU.

**Osman Balci is currently an Assistant Professor of Computer Science at VPI&SU. He received a Ph.D. in Industrial Engineering and Operations Research from Syracuse University in 1981. He serves as Co-Principal Investigator for the Navy-funded project in Model Development Environments. He is currently an Associate Editor of *Simuletter*. His main research interests include model development environments, simulation, performance evaluation of computer systems, and software engineering.

Scope and purpose - This paper presents fundamental requirements for Model Development Environments (MDEs) and offers guidance for MDE designers and implementers. A MDE provides an integrated and comprehensive collection of computer-based tools to (1) offer cost-effective, integrated, and automated support of model development throughout its entire 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. Although the requirements are perceived to be generically applicable to all mathematical (or abstract) - specifically, simulation and mathematical programming - modeling tasks, the focus of this paper is on discrete event simulation model development.

Abstract - This paper deals with the initial phase of our ongoing research project on the Definition of a Discrete Event Simulation MDE which started on 1 June 1983. The first phase of the rapid prototyping approach we are using in designing the MDE involves the requirements specification. A literature review revealed eleven current problems in modeling. To address these problems, a MDE was identified as composed of four layers: (1) hardware and operating system, (2) kernel MDE, (3) minimal MDE, and (4) MDEs. Requirements were then perceived for each layer and are reported in this paper. The feasibility of the requirements have been assessed throughout our prototyping efforts. This paper has provided significant guidance to our research group in designing the MDE and its associated tools. We believe that the designers and implementers of other types of MDEs can benefit from the research described herein.

1. INTRODUCTION

The use of computer-based models is becoming an increasingly important activity to solve problems encountered in all areas of business, government, industry, and military. As these problems grow larger, become more complex, and require accurate solutions more rapidly than ever before, the associated modeling activity becomes one which requires more computer assistance in the model development process itself.

There is a great need for automation to support model development *throughout its entire life cycle*. This automated support is needed not so much to fundamentally change the end product, but to reduce the cost of development and increase the quality. This automated support can best be characterized in the form of a Model Development Environment (MDE).

Current modeling problems are reviewed in Section 2. The context and architecture of the MDEs are presented in Section 3. Section 4 contains the fundamental MDE requirements. A comparison of the modeling problems with the MDE tools is given in Section 5. Finally, conclusions are stated in Section 6.

2. CURRENT PROBLEMS IN MODELING

A review of substantial current problems in modeling is presented in this section. The problems recognized are believed to exist in simulation (discrete event, continuous, combined), mathematical programming, econometric, and other types of modeling.

High Cost of Model Development

As reported by Roth, Gass, and Lemoine, "The U.S. Government is the largest sponsor and consumer of models in the world. Estimates have indicated that over one-half billion dollars are being spent annually on developing, using, and maintaining mathematical, simulation, and econometric models in the decision-and-policy-making functions of the Federal Government" [31, p.214].

A report to the U.S. Congress prepared by the General Accounting Office (GAO) [36] said in part:

> "GAO identified 519 federally funded models developed or used in the Pacific Northwest area of the United States. Development of these models cost about $39 million. Fifty-seven of these models were selected for detailed review, each costing over $100,000 to develop. They represent 55 percent of the $39 million of development costs in the models.
>
> Although successfully developed models can be of assistance in the management of Federal programs, GAO found that many model development efforts experienced *large cost overruns*, prolonged delays in completion, and total user dissatisfaction with the information obtained from the model."

An obviously important component of *software development* is computer programming. The cost of programming is becoming more dominant and apparent as hardware costs continue to decline. As indicated by Wasserman and Gutz [38], "There is already a serious shortage of skilled programmers and the cost of such a person is expected to surpass $100,000 a year (salary, benefits, overhead) by the mid-1980's." This shortage is even more serious in the area of *model development* because programming is only one of several costly component activities.

Model Quality Assurance

The importance of the following dictum is not fully recognized:

"Nobody solves *the* problem. Rather, everybody solves the model that he has constructed of the problem." [8]

This dictum places the modeling in its correct perspective. It clearly indicates that it is crucial to assess the quality of the model to claim that it provides a credible solution to the problem.

However, there is no formal, precise definition and means of determining model confidence at the present time as indicated by Gass [9]. The quality becomes much more difficult to assess for models of systems which are nonexistent or future-oriented in which the past is not a good predictor of the future. Even for existing, operational systems which are not completely observable (or from which it is not possible to collect the required data completely), quality assurance still poses a significant technical challenge.

The inability to assess the model quality adequately for nonexistent, future-oriented, and partially or completely unobservable systems, as well as for others, raises the probability of committing the type II error; the error of accepting the results of an invalid model (model user's risk) [4]. The consequences of the type II error are crucial especially when vital decisions are made on the basis of model results. The extreme seriousness of the consequences of the type II error for military applications, such as Navy combat system models, is one of the important factors which has motivated the research described herein.

Lack of Full Life Cycle Support

In spite of the available Simulation Programming Languages (SPLs) and Mathematical Programming Systems (MPSs), model development is still labor intensive and error prone. The current SPLs and MPSs are

supportive of only the programming process. Rarely do they even claim to provide effective tools for programmed model verification. At this time, automated support of model development *throughout its entire life cycle* (see Fig. 1) is nonexistent.

Conceptual and Communicative Models are Built Under the Influence of a SPL or a MPS

Model formulation and representation are usually done under the constraints imposed by the SPL or MPS to be used in the programming process. This may induce substantial errors within the model representation right at the beginning of the model development life cycle. These errors are either caught in much later phases resulting a higher cost of correction or never detected resulting the type II error.

A linear programming MPS, for example, requires all constraints to be linear. If the conceptual and communicative models are built under this assumption, all crucial nonlinear constraints will be approximated to be linear. This may be an unacceptable approximation and may result in an invalid model representation. Similarly, if the modeler is constrained by the world view of a SPL, the conceptual and communicative models may be invalidated due to the incompatibility of the world view for the system under study. This invalidity may easily be carried out until later phases in the development and may even cause the type II error.

Redefinition Does Not Usually Follow the Entire Life Cycle

The experimental model is commonly redefined for an update, obtaining another set of results, maintenance, or other use(s). The changes required are generally made on the programmed or experimental model skipping the formulation and representation processes. This

practice may induce substantial errors especially for large scale complex models.

Inadequate Management

The seriousness of this problem has motivated the U.S. GAO to submit a report to the Congress [36]. Three broad categories of problems were identified in this report as a result of an analysis of 33 federally funded models that had experienced problems during their development: (1) inadequate management planning (70%), (2) inadequate management coordination (15%), and (3) inadequate management commitment (15%).

Inadequate Documentation

In his feature article, Gass [9] indicates that "we do not know of any model assessment or modeling project review that indicated satisfaction with the available documentation." He notes that serious problems exist regarding the production and availability of model documentation.

Most models evolve over an extended period of time. The model is redefined repeatedly to reflect the new and increased understanding of the system, changing objectives, and the availability of new data. This evolutionary change, however, causes the documentation often to become obsolete, incomplete, or inadequate shortly after they are written [2]. The longer the model development the more the documentation deteriorates under the current practice.

Poor Communication

A modeling project involves people with different backgrounds and areas of expertise. Communication problems arise among these people

mostly due to the lack of (1) a conceptual framework, a uniform terminology, and a language for communication, (2) effective communication tools, and (3) stratified documentation.

At this time, the literature on modeling does not display a consistent terminology. Many terms (i.e. assessment, evaluation, verification, validation, quality assurance, credibility, documentation, portability, certification) are interpreted differently depending upon the background of the modeler or the specific area of application.

Modeling Projects Experience Prolonged Delays in Completion

One of the problems identified in the GAO report to the U.S. Congress [36] is the prolonged model development cycles. Large scale and complex model development is still labor intensive and error prone.

Inadequate User Participation

"All too often, model developers simply go off by themselves for a year and then proudly drop the 'completed,' never to be used model on the sponsor's desk." [2] Insufficient user involvement causes the model to be unresponsive to user needs resulting user dissatisfaction with the information obtained from the model [36].

Prefabricated Models and Past Experience are not Effectively Utilized

Earlier modeling projects are not studied in detail and are not fully utilized. An already existing usable model (part) is sometimes rebuilt from scratch duplicating the development effort and unnecessarily increasing the cost and time of development. Even if existing models or model parts are not usable in the new project, a modeler (especially an inexperienced one) can study them to learn from past experience. Such a study may be extremely beneficial especially for

the management, planning, and resource allocation for the new project.

There exists other problems which affect the success of a modeling project. Most important of all are: (1) failure to define a set of achievable objectives [2], (2) insufficient problem formulation [36], (3) inadequate user participation in defining the problem [36], (4) failure to identify the best solution technique [36], and (5) ineffective presentation of model results [30]. These problems, however, should be addressed within a Model Management System (MMS) [22] rather than within a model development system which is a subset of the MMS. Balci and Nance [3] introduced the formulated problem verification as an explicit requirement of model credibility. Nance and Balci [21] emphasized that a MMS should be concerned with the problem definition phases as well as the model development and decision support phases by stating the objectives and requirements to address these problems.

3. MODEL DEVELOPMENT ENVIRONMENTS

In order to provide a proper framework to determine the context of MDEs, it is convenient to divide the life cycle of model development into six phases [20] as depicted in Fig. 1. The six phases are shown by oval symbols. A dashed arrow describes a process by which an input phase is translated into an output phase. A solid arrow refers to a credibility assessment stage. The life cycle presented in Fig. 1 generically applies to all types of abstract model development some of which are stated as follows with example programming languages used for development: (1) discrete event simulation model development (GPSS [12], SIMSCRIPT [17]), (2) continuous simulation model development

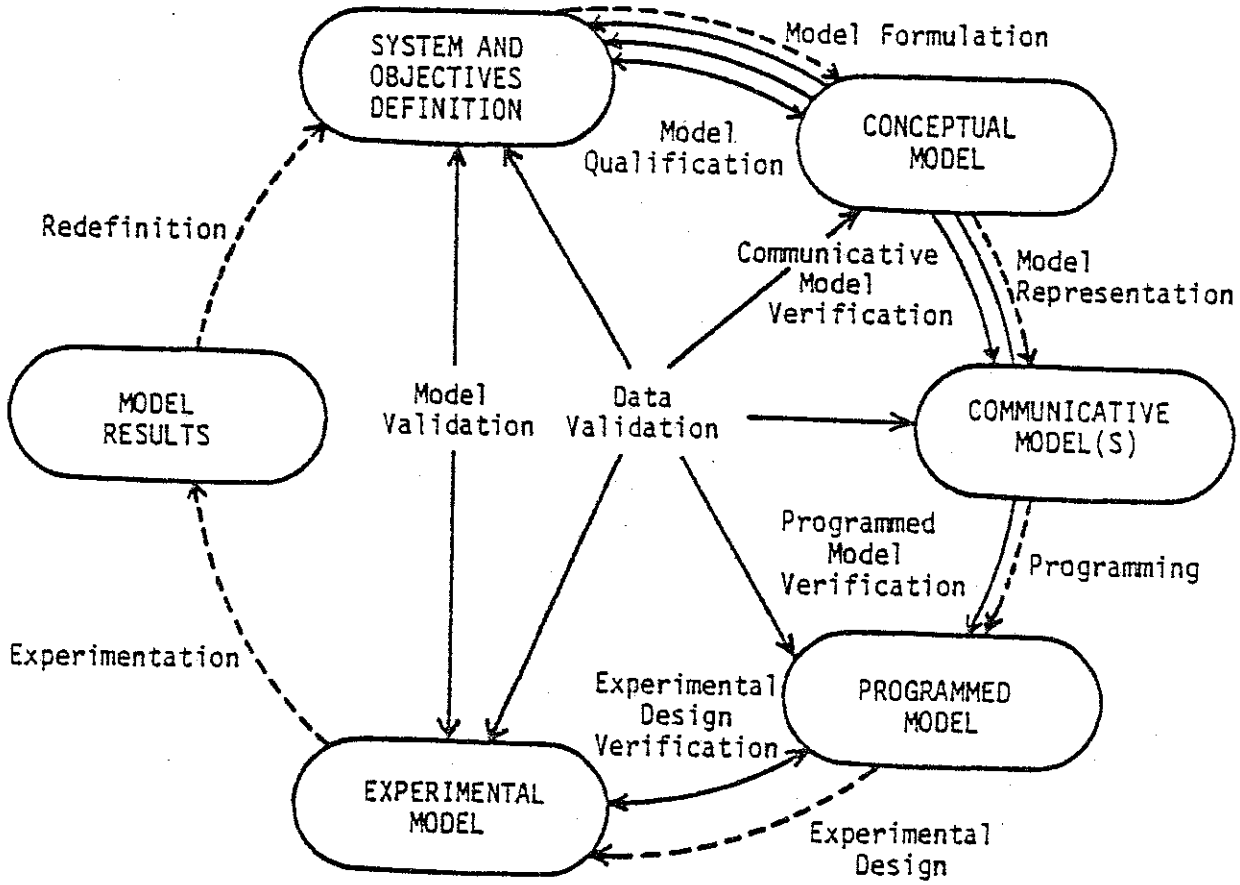

Fig. 1. The life cycle of model development.

(DYNAMO [29], CSMP [13]), (3) combined (discrete and continuous) simulation model development (C-SIMSCRIPT [5], SLAM [28]), and (4) mathematical programming (Linear Programming - LP, Integer LP - ILP, Mixed ILP, etc.) model development (FMPS [33], MPSX [14]).

The processes of model development should in no way be interpreted as sequential. Model development is iterative in nature and we bounce back and forth between the phases during the development. Modeling starts with the given definition of the system under study and explicitly stated study objectives. System definition contains the

formulated problem, system characteristics, and system boundary.

Model Formulation is the process by which the conceptual model is envisioned to represent the system under study. The *Conceptual Model* [20] is "that model which exists in the mind of the modeler. The form of the conceptual model is influenced by the system, the perceptions of the system held by the modeler (which are affected by the modeler's background and experience and those external factors affecting the particular modeling task), and the objectives of the study." *Model Qualification* [32] is "determination of adequacy of the conceptual model to provide an acceptable level of agreement for the domain of intended application." *Domain of Intended Application* [32] is the prescribed conditions for which the model is intended to match the system under study. *Level of Agreement* [32] is the required correspondence between the model and the system under study, consistent with the domain of intended application and the study objectives.

Model Representation is the process of translating the conceptual model into a communicative model. *Communicative Model* [20] is "a model representation which can be communicated to other humans, can be judged or compared against the system and the study objectives by more than one human. Several communicative models could be constructed during a study, each derived from a preceding communicative model (following the first) or different conceptual models." Entity cycle diagrams, flow charts, pseudocodes, flow diagrams, block and logic diagrams, or activity charts are examples of communicative models. *Communicative Model Verification* is confirming the adequacy of the communicative model to provide an acceptable level of agreement for the domain of intended application.

Programming is the process of translating the communicative model into a programmed model. A *Programmed Model* [20] is a model representation that admits execution by a computer to produce results. GPSS, DYNAMO, C-SIMSCRIPT, and FMPS programs are examples of programmed models. *Programmed Model Verification* is substantiating that the programmed model represents the communicative model (and the system under study) within specified limits of accuracy.

Experimental Design is the process of defining a set of conditions under which the system (or the model) is to be observed or experimented with and determining how this observation or experimentation is to be carried out. An *Experimental Model* is the programmed model incorporating an executable description of the experimental design. *Experimental Design Verification* is substantiating that the experiments are correctly designed and their translation into an executable code is correctly done.

Data Validation is confirming that the data used is accurate, complete, unbiased, and appropriate in its original and transformed forms. *Model Validation* is substantiating that the experimental model, within its domain of applicability, behaves with satisfactory accuracy consistent with the study objectives. *Domain of Applicability* [32] is the prescribed conditions for which the experimental model has been tested, compared against the system to the extent possible, and judged suitable for use.

Experimentation is the process of experimenting with the model on a computer for a specific purpose. Some purposes of experimentation are: (1) comparison of different operating policies or procedures, (2) evaluation of system behavior, (3) sensitivity analysis, (4) predic-

tion, (5) optimization, and (6) determination of functional relations. *Model Results* [20] are "the outcome from a single execution of the experimental model or those results produced to satisfy a single test scenario, which might require several model executions with different input value specifications, structural changes, etc."

Redefinition is the process of (a) updating the experimental model so that it represents the current form of the system, (b) altering it for obtaining another set of results, (c) changing it for the purpose of maintenance, (d) modifying it for other use(s), or (e) redefining a new system to model for studying an alternative solution to the problem. The process of redefinition should follow the entire life cycle starting with the definition of the system and study objectives and culminating with the model results.

Quality Assurance (of the Experimental Model) is substantiating that the experimental model, within its domain of applicability, possesses satisfactory quality consistent with the study objectives. (Experimental) *Model Quality* is determined through the integration of the credibility assessment stages shown by solid arrows in Fig. 1.

A MDE should provide an integrated and complete collection of computer-based tools which offer continuous and cost-effective support to all phases, processes, and stages of the model development life cycle in Fig. 1. A MDE should implement a model development methodology. Automated tools should be provided to support the development methodology since these tools do not normally exist as ends in themselves, but rather as means to an end. The Conical Methodology [20] which comprises a top down model definition and a bottom up model specification approach is an extremely useful methodology for developing

discrete event simulation models and can be implemented by a discrete event MDE.

The overall objectives of a MDE are stated as follows:

(1) offer cost-effective integrated support continuously throughout the entire life cycle of model development,

(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.

The layered approach to the development of environments for the language ADA [1] is followed in structuring the MDEs. In this approach, the development environments are composed of four layers as depicted in Fig. 2 and described below:

Layer 0: Hardware and Operating System

Computer hardware and a host operating system constitute the basis upon which the environment is built. An operating system (i.e., UNIX [16], INTERLISP [35], VMS [6]) creates its own programming environment the tools of which are utilized in the construction of a MDE. Therefore, a MDE is dependent on the particular operating system chosen.

Layer 1: Kernel Model Development Environment (KMDE)

Primarily, this layer integrates all MDE tools into the programming environment. It provides databases, communication and run-time support functions, and a kernel interface, all of which need to be machine-independent so as to make the MDE portable to any computer system which runs the operating system of layer 0. In this case, the

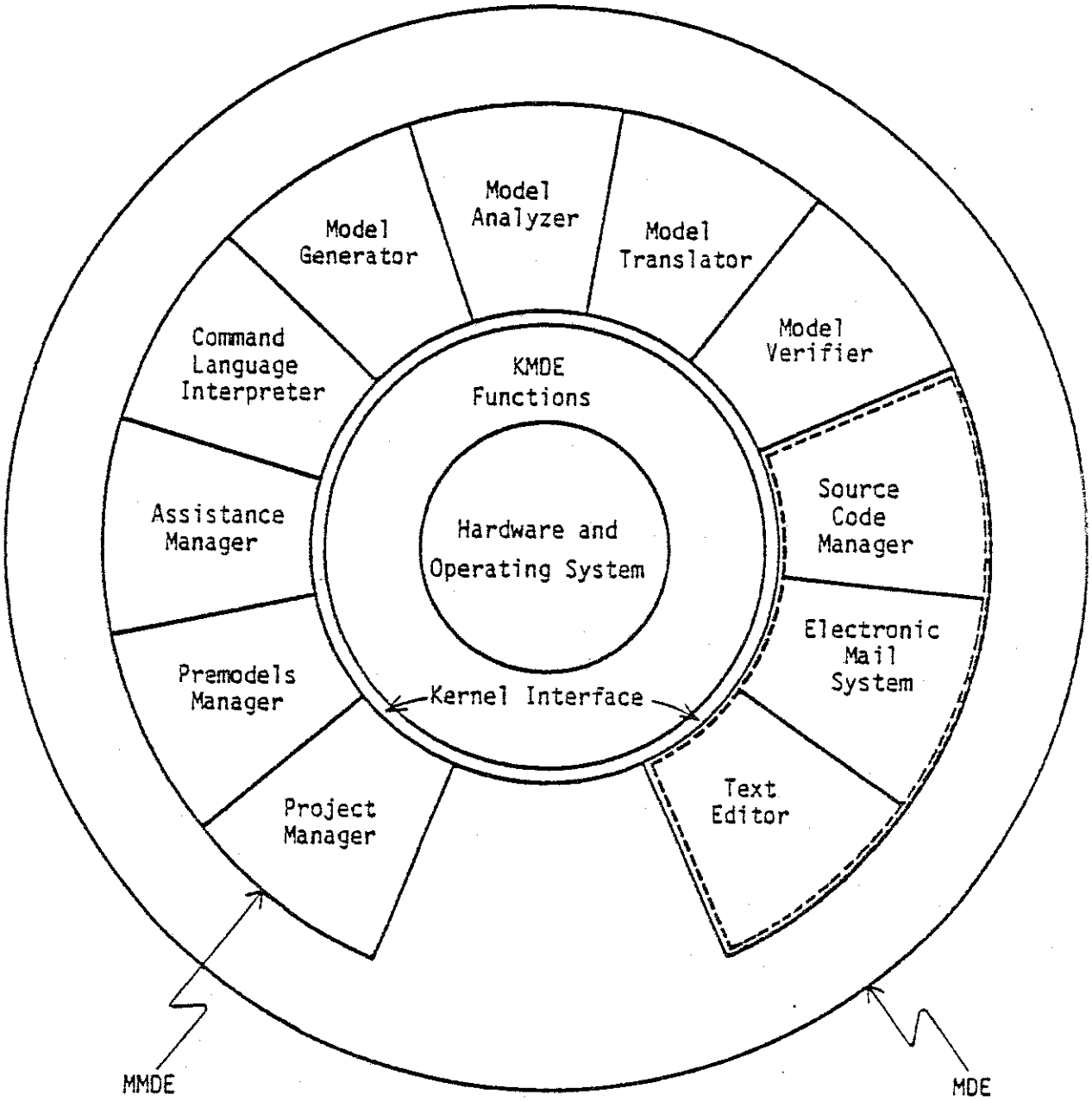

Fig. 2. The structure of model development environments.

MDE is as portable as the operating system (programming environment).

Layer 2: Minimal Model Development Environment (MMDE)

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 phases, processes, and stages of the life-cycle in Fig. 1. "Minimal" implies that the toolset is basic and general. It should be basic in the sense that the set of tools should enable modelers to work within the bounds of the MMDE without any significant inconvenience. It should be general in the sense that the toolset should be generically applicable to all types of abstract modeling tasks.

The MMDE tools are classified into two categories. The first category contains the tools which need to be constructed with respect to the type of modeling the environment will be built for. Project Manager, Premodels Manager, Assistance Manager, Command Language Interpreter, Model Generator, Model Analyzer, Model Translator, and Model Verifier fall in this category. If a MDE is to be built for discrete event simulation, these tools should be constructed accordingly. However, some of these tools such as the Command Language Interpreter, Assistance Manager, or Premodels Manager constructed for one type of modeling may well be used for another with some modifications.

The second category tools (also called assumed tools or library tools) are the ones which are expected to be provided by the programming environment. Source Code Manager, Electronic Mail System, and Text Editor fall in this category. Electronic Mail System and Text Editor can be used for each type of model development. Source Code Manager, however, may, for example, be the SIMSCRIPT, DYNAMO, or FMPS

source code manager depending upon the type of modeling.

Layer 3: Model Development Environments (MDEs)

This is the highest layer of the environment which is based upon a particular MMDE. In addition to the toolset of the MMDE, it incorporates tools that support specific applications and are of special interest only within a particular project or of interest only to an individual modeler. If no other tools were added to a MMDE toolset in the degenerate case, a MMDE would itself be a MDE.

The MDE tools are also classified into two categories. The first category tools are the ones which need to be constructed with respect to the specific area of application. These tools may need to be tailored for a specific project or other tools may need to be added 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 MDE tool is integrated with other tools and the programming environment through the kernel interface. The provision for this integration is indicated in Fig. 2 by the opening between Project Manager and Text Editor.

4. DEFINITION OF MDE REQUIREMENTS

Fundamental MDE requirements are defined in this section separately for each layer of the environment by taking the current modeling problems of Section 2 into consideration.

4.0 REQUIREMENTS FOR LAYER ZERO

4.0.1 Interactive (on-line) mode of processing is required to provide the necessary relationship between a modeler and the toolset supporting the modeling task. A computer system with such a processing mode can be used to build a MDE.

4.0.2 Alphanumeric video display terminals are required for on-line processing.

4.0.3 A high-speed line printer or a laser printer is required to produce hard copies and good quality documents.

4.0.4 An operating system is required with the following capabilities:

a) It should have a programming environment which provides the source code manager required, a mail system, and a powerful text editor.

b) It should be as portable as the MDE is required to be.

c) It should be as powerful as possible since the more powerful the operating system and its associated programming environment, the easier the construction of a MDE.

We have chosen the UNIX operating system in our research project. Evaluation of the UNIX host for a simulation MDE is presented in [23]. In addition to its being very powerful, UNIX (and thus the MDE) can be transported to a wide variety of processors, from main-frames to personal computers [16].

4.1 REQUIREMENTS FOR KMDEs

The KMDE provides databases, communication and run-time support functions, and a kernel interface. There are three databases at this level, namely, project, premodels, and assistance databases. The access functions of these databases are considered to be part of the KMDE because of their central importance to the MDE.

4.1.1 *KMDE Project Database Requirements*

4.1.1.1 The database should be constructed as the central repository of the MDE to provide the storage and retrieval of:

a) model (parts), being developed, in different representation forms (SMSDL - Simulation Model Specification and Documentation Language [20], primitive [25], graphical, SPL, MPS, etc.) and versions,

b) information regarding project management including project and work schedules, progress reports, due dates, telephone directories, "to do" lists for individuals, mailing lists, etc.

c) the system definition and study objectives,

d) information regarding configuration management,

e) project documentation,

f) personal documents of users,

g) model input and output data, and

h) any other data relevant to the modeling project.

4.1.1.2 Each object (a separately identifiable collection of information) in the database should be identified by a unique name [1].

4.1.1.3 An object in the database should consist of its content (the raw information it contains) and its attributes [1].

4.1.1.4 The number of attributes that an object has should not be restricted [1].

4.1.1.5 The database should not impose restrictions on the format of storage in an object [1].

4.1.1.6 Every object in the database should have a history (or derivation) attribute, a categorization attribute, and an attribute which indicates access rights to its content and to each of its attributes [1].

4.1.1.7 The database should allow the complete and accurate recording of relationships among its objects, even when these objects are created by modeler-supplied tools [1].

4.1.1.8 The database should allow the MDE tools to create and to access its objects [1].

4.1.1.9 The database should be capable of supporting many projects simultaneously.

4.1.1.10 The database should be machine-independent.

4.1.2 *KMDE Premodels Database Requirements*

The premodels database contains prefabricated models (submodels or model components) the quality of which is assured and which can be used in the construction of another model. Two types of prefabrication may occur. The first is due to an earlier project and the second is the prefabrication of a generic model purely for the purpose of general use. For example, a generic model of a G/G/s queueing system can be prefabricated for the purpose of using it in the construction of other models.

A modeler may retrieve prefabricated models from the database to learn or obtain more experience in modeling or the use of a particular programming language (i.e., GPSS, CSMP, MPSX). In this context, the premodels database can be viewed as a Learning Support System.

Requirements are stated as follows:

4.1.2.1 The database should be constructed in such a way that it can be searched using the Premodels Manager.

4.1.2.2 The database should allow the storage of a prefabricated model through the Premodels Manager.

4.1.2.3 The database should implement a standard terminology in describing a model (part) so that the search does not fail due to terminology mismatch.

4.1.2.4 The database should allow access to each meaningful and useful part of a prefabricated model and record the relationships among these parts for use by the searching mechanism.

4.1.2.5 Every accessible model (part) in the database should have a history (or derivation) attribute, a categorization attribute, and an attribute which indicates access rights to its content and to each of its attributes.

4.1.2.6 The database should allow the storage and retrieval of a model (part) in more than one form of representation (i.e., primitive, SMSDL, graphical, SPL, MPS, etc.).

4.1.2.7 The database should be machine-independent.

4.1.3 *KMDE Assistance Database Requirements*

Assistance database contains "help" and "tutorial" information for CLI commands, communication language terminology, MDE tools, and all techniques and procedures used in the MDE. Requirements follow:

4.1.3.1 The database should contain "help" and "tutorial" information about the syntax, semantics, and use of CLI commands and MDE tools.

4.1.3.2 The database should contain the definitions of the technical terms used in documentation and communication language.

4.1.3.3 The database should contain "tutorial" information to educate a modeler for applying the techniques and procedures available in the MDE effectively and correctly.

4.1.3.4 The database should be machine-independent.

4.1.4 *KMDE Function Requirements*

4.1.4.1 Basic run-time support facilities should be provided by the KMDE for all programs that execute within the MDE [1].

4.1.4.2 The KMDE should provide the necessary functions for accessing the project [1], premodels, and assistance databases through the managers.

4.1.4.3 The KMDE should provide the communication capability between the MDE tools [1].

4.1.4.4 The KMDE should provide a fixed set of terminal interface control functions [1] such as to

a) issue a request to terminate the current function or program,

b) suspend the current program and invoke the command language interpreter,

c) terminate the current command language interpreter invocation and resume the program suspended,

d) abort the current program and return to its invoker or to the nearest command language interpreter level.

4.1.4.5 The KMDE terminal interface control functions should be

human engineered.

4.1.4.6 All of the functional requirements stated above should be satisfied by machine-independent implementations.

4.1.5 *KMDE Interface Requirements*

4.1.5.1 The KMDE should implement interface definitions which should be available to the MDE tools [1].

4.1.5.2 The KMDE interface should be constructed in such a way that the MDE tools which communicate with or invoke each other should do so *only* through this interface.

4.1.5.3 The KMDE interface should be machine-independent [1].

4.1.5.4 The KMDE interface should be straightforward to understand and easy to use and modify [1].

4.1.5.5 The modeler should be able to access the interface from a variety of physical terminal devices [1].

4.1.5.6 The KMDE interface should allow the modeler to interact with the invoked tool and to exercise control over the tool [1].

4.1.5.7 Security protection should be imposed on the KMDE interface to prevent any unauthorized use of a tool or data.

4.2 REQUIREMENTS FOR MMDEs

The MMDE provides a comprehensive set of tools in two categories that are minimal for the development and execution of a model. Requirements for the first category tools that need to be constructed are stated as MMDE toolset requirements in Section 4.2.1. Requirements for the second category (assumed) tools that are expected to be provided by the programming environment are stated as MMDE library requirements in Section 4.2.2.

4.2.1 *MMDE Toolset Requirements*

4.2.1.1 *Project Manager*: This tool is required to

a) administer the storage and retrieval of items (a) through (h) in Section 4.1.1.1,

b) keep a recorded history of the progress of the project,

c) trigger messages and reminders (especially about due dates), and

d) respond to queries in a prescribed form concerning project status.

4.2.1.2 *Premodels Manager*: This should be an interactive tool implementing a language and is required to

a) administer the premodels database,

b) provide information on previous modeling projects, and

c) provide a stratified description and several representation forms of models or model components developed in the past.

4.2.1.3 *Assistance Manager*: This tool is required to

a) administer the assistance database,

b) provide information on how to use a MDE tool or a CLI command,

c) provide the definition of a technical term encountered in documentation or communication, and

d) provide tutorial assistance as appropriate.

4.2.1.4 *Command Language Interpreter*: This should

a) be capable of invoking all MDE tools,

b) be extensible,

c) be capable of executing a file of CLI commands,

d) provide escape to the operating system,

e) be human engineered, and

f) be menu driven where possible.

A CLI, satisfying the above requirements, has been developed based upon [19] in our research project.

4.2.1.5 *Model Generator*: This should be an interactive tool and is required to

a) create a specification of a model in a predetermined analyzable form which is independent of any SPL or MPS,

b) create multi-level (stratified) model documentation, and

c) effectively assist in model qualification.

We have developed two prototypes of the Model Generator implementing the Conical Methodology [20]. The first one [10] produces a primitive model specification [25, 27] that is general but very difficult to fully translate into an executable code. The second one produces a specification which is less general but which lends itself for 100% translation. Hence, our experience shows that the more general the model specification is, the more difficult the complete translation becomes.

4.2.1.6 *Model Analyzer*: This tool is required to

a) diagnose the model specification created by the model generator, and

b) effectively assist in communicative model verification.

Based upon the work described in [26, 27], the Model Analyzer is under development.

4.2.1.7 *Model Translator*: This tool is required to translate the model specification into an executable code after the quality of the specification is assured by the model analyzer. We agree with Henriksen [11] that the generation of complete executable code from *general* specifica-

tions will not be realized in the near future (may be in the late 1980's). However, full automation of the translation process can be realized for a model specification created under a less general conceptual framework. The objective should be to construct the model translator to produce an executable code as complete as possible from a sufficiently general model specification. The executable source code can be completed by using the text editor until the need for it is eliminated entirely.

4.2.1.8 *Model Verifier*: This tool is required for programmed model verification and should have at least the following capabilities:

a) It should assist in incorporating diagnostic measures within the source program.

b) It should provide a cross-reference map to identify where a particular variable is referenced and where its value is changed [1].

c) It should provide a chart of the programmed model control topology to indicate which routines are called from where in the program and identify inter-submodel communication calls [1].

d) It should provide dynamic analysis tools for snap shots, traces, breaks, statement execution monitoring, and timing analysis [1].

4.2.2 *MMDE Library Requirements*

4.2.2.1 *Source Code Manager*: This tool is required to translate a source code into a machine language and perform its execution. SIMSCRIPT, DYNAMO, and FMPS compilers and their corresponding linkers and loaders are examples of this tool. Execution of a particular language may require additional compilers. A GPSS program, for exam-

ple, may require a FORTRAN compiler in addition to its own when it calls FORTRAN subroutines. The source code manager, in this case, is required to compile both GPSS and FORTRAN source codes and perform their execution. The source code manager should interface to the KMDE by meeting interface requirements and thereby be cooperative with other tools.

4.2.2.2 *Electronic Mail System*: This tool is required to facilitate the necessary communication among people interested in the project. Primarily, it performs the task of sending and receiving of mail through (local or large) computer networks. The UNIX and VAX mail systems are examples of this tool.

4.2.2.3 *Text Editor*: This tool is required for general text processing, including the preparation of technical reports, user manuals, system documentation, correspondence, and personal documents. The editor should

a) be capable of editing general text and source programs,

b) be capable of full screen editing, and

c) provide a comprehensive set of human engineered functions.

4.3 REQUIREMENTS FOR MDEs

Since the highest layer (Layer 3) of the environment provides tools which are specific to a certain area of application, we need to choose such an area so that we can state the requirements accordingly. This area of interest is Discrete Event Digital Computer Simulation.

Tools are provided in two categories at the MDE layer. Global requirements are first stated for the MDE tools in Section 4.3.1.

Then, requirements for the first category tools that need to be constructed for discrete event simulation are stated as MDE toolset requirements in Section 4.3.2. Finally, requirements for the second category (assumed) tools that are expected to be available are stated as MDE library requirements in Section 4.3.3.

4.3.1 *MDE Global Requirements*

4.3.1.1 A MDE should provide a coordinated and complete set of tools which offer continuous and cost-effective support to the development of an experimental model.

4.3.1.2 The set of tools in a MDE should remain open-ended [1]. New tools should easily be added and old tools should easily be removed so as to protect a MDE against premature obsolescence.

4.3.1.3 The tool-tool, tool-user communications should be implemented *only* via the KMDE interface.

4.3.1.4 The protocols for communication between tools and the modeler should be straightforward and uniform throughout a MDE toolset [1].

4.3.1.5 A MDE tool should be capable of storing information in the KMDE project database for later use by other tools.

4.3.1.6 The MDE tools should be able to use the same file or data format.

4.3.1.7 The modeler should not need elaborate knowledge to be able to use a MDE tool. The MDE tools should be easy to use.

4.3.1.8 An inexperienced modeler should be able to gain a substantial portion of the benefits of a MDE tool with only a small subset of its available facilities, while an experienced modeler who uses the same tool should be able to gain additional benefits through its complete set of features [38].

4.3.1.9 A MDE tool should be assured to be a high quality tool. That is, it should be verified, validated, and reliable.

4.3.1.10 All MDE tools should be human engineered by taking the ergonomic aspects into consideration.

4.3.2 *MDE Toolset Requirements*

4.3.2.1 A tool is required for experimental design verification.

4.3.2.2 A tool is required for data validation.

4.3.2.3 A tool is required for model validation.

4.3.2.4 An experimental design system is required.

4.3.3 *MDE Library Requirements*

4.3.3.1 A powerful text formatting or typesetting tool is required to prepare reports, manuals, system documentation, correspondence, and personal documents. SCRIPT [37], Runoff [7], and Nroff/Troff [24] are examples of this tool.

4.3.3.2 A statistical analysis system is required for

a) data analysis,

b) distribution fitting and estimation of parameters,

c) testing randomness, univariate and multivariate normalities, equality of variances and covariances, and

d) random variate and random number generation.

IMSL [15], SAS [34], and UNIFIT [18] are example tools for this system.

4.3.3.3 A library of graphics routines, a graphics production system, and its associated hardware are required.

There may be other tools required for meeting special project requirements. Open-endedness feature of the MDE provides for easy integration of added tools into the environment.

5. THE MODELING PROBLEMS VERSUS THE MDE TOOLS

In this section, a brief explanation is given on how the MDE tools will be addressing the current problems in modeling. A summary of which MDE tool contributes to the solution of which problem is

given in Table 1.

The high cost of model development will be reduced significantly due to the integrated, automated support provided by each one of the MDE tools. The model generator, model analyzer, model verifier together with other verification and validation tools will contribute to the assurance of model quality. The automated support is expected to increase the model quality.

The MDE provides a well integrated, comprehensive, automated support of model development throughout its entire life cycle with most of its tools marked "X" in Table 1. The influence of a SPL or a MPS will be avoided with the use of a "meta-language" (i.e. SMSDL) in developing the conceptual and communicative models. This will be achieved by the use of the model generator and model analyzer.

The automated support provided by the MDE throughout the entire life cycle will facilitate the redefinition process to follow the entire life cycle. The project manager will be the key tool for improving model management during development phases. Management planning will be improved by the premodels manager providing prefabricated models and past experience. Using the CLI, a manager will be able to use the MDE tools. The stratified documentation created by the model generator will help the manager(s) understand the development and coordinate the project much better. The electronic mail system will help the manager(s) monitor and control the project. The text editor will be the basic tool for preparing documents necessary for project management.

The stratified (multi-level) documentation created by the model generator will help to resolve inadequate documentation problems. The

Table 1. The modeling problems versus the MDE tools.

MODELING PROBLEMS	Project Manager	Premodels Manager	Assistance Manager	CLI	Model Generator	Model Analyzer	Model Translator	Model Verifier	Source Code Mgr.	Elec. Mail System	Text Editor	Other MDE Tools
1. High Cost of Model Development	X	X	X	X	X	X	X	X	X	X	X	X
2. Model Quality Assurance					X	X		X				X
3. Lack of Full Life Cycle Support	X		X	X	X	X	X	X	X	X	X	
4. Influence of a SPL or a MPS					X	X						
5. Poor Redefinition	X	X	X	X	X	X	X	X	X	X	X	X
6. Inadequate Management	X	X		X	X					X	X	
7. Inadequate Documentation	X				X						X	
8. Poor Communication	X		X	X	X					X	X	
9. Delays in Completion	X	X	X	X	X	X	X	X	X	X	X	X
10. Inadequate User Participation	X			X	X					X	X	
11. Past Experience Not Utilized		X										

project manager will be in charge of preserving the project and model documentation throughout the development phases. The text editor will be extensively used for documentation. The automated support provided by the MDE will facilitate the updating of documentation and help to keep it up-to-date.

Communication among the interested parties will be enhanced by the assistance manager providing a glossary of terms and tutorial information on the conceptual framework of communication. A person involved in the project will be able to invoke the project manager through the CLI to retrieve the stratified model and project documentation. This documentation will help the person communicate with others much better. The electronic mail system will provide the means by which the communication will take place very effectively. The text editor will be used for message preparation.

The productivity will be significantly increased in developing large scale and complex models due to the integrated, automated support of model development throughout its entire life cycle. The user(s) or sponsor(s) will be able to invoke the project manager through the CLI to retrieve documentation and information. Using the text editor, they will be able to prepare messages and communicate with others by way of the electronic mail system. The MDE will facilitate the user participation quite effectively.

The premodels manager will provide information on prefabricated models and reveal past experience to avoid duplication of effort. An inexperienced modeler will learn from past experience without actually living it. The premodels manager provides an effective learning support system.

6. CONCLUSIONS

The fundamental requirements perceived in this paper characterize a comprehensive and integrated plan for the automated support of model development. This plan offers guidance for the designers and implementers of simulation (discrete event, continuous, combined), mathematical programming, econometric, and other types of MDEs.

We believe that the individual requirements do not possess a high-risk characteristic due to the fact that they are perceived within the state of the art at the present time. However, we recognize that the complete set of requirements poses a significant technical challenge to MDE designers and implementers especially for the model generator, model analyzer, model translator, and model verifier. Nevertheless, we are confident that the challenge can be met by way of an evolutionary development of MDE prototypes.

ACKNOWLEDGMENTS

I am indebted to Richard E. Nance for his constructive comments and suggestions which contributed to ideas developed in this paper. I am grateful for many useful discussions with John A.N. Lee, C. Michael Overstreet, Robert H. Hansen, and Robert L. Moose.

REFERENCES

1. Advanced Research Projects Agency, Requirements for ADA programming support environments - "STONEMAN". U.S. DoD, Arlington, Virginia (1980).

2. J. S. Annino and E. C. Russell, The ten most frequent causes of simulation analysis failure - and how to avoid them!. *Simulation* *32*(6), 137-140, (1979).

3. O. Balci and R. E. Nance, Introducing formulated problem verification as an explicit requirement of model credibility. *Simulation*, to appear (1985).

4. O. Balci and R. G. Sargent, A methodology for cost-risk analysis in the statistical validation of simulation models. *Commun. ACM 24*(4), 190-197, (1981).

5. C. M. Delfosse, *Continuous Simulation and Combined Simulation in SIMSCRIPT II.5*. CACI, Inc., Arlington, Virginia (1976).

6. DEC, *VAX/VMS User's Guide*. Order No. AA-D643A-TE, Maynard, Mass. (1978).

7. DEC, *VAX-11 DIGITAL Standard Runoff User's Guide*. Version 2, Maynard, Mass. (1982).

8. S. E. Elmaghraby, The role of modeling in IE design. *Industrial Engineering 19*(6), 292-305, (1968).

9. S. I. Gass, Decision-aiding models: validation, assessment, and related issues for policy analysis. *Op. Res. 31*(4), 603-631, (1983).

10. R. H. Hansen, The model generator: a crucial element of the model development environment. Technical Report CS84008-R, Department of Computer Science, Virginia Tech, Blacksburg, Virginia (1984).

11. J. O. Henriksen, The integrated simulation environment (simulation software of the 1990s). *Op. Res. 31*(6), 1053-1073, (1983).

12. J. O. Henriksen and R. C. Crain, *General Purpose Simulation System/H (GPSS/H) User's Manual*. Second Edition, Wolverine Software Corporation, Annandale, Virginia (1983).

13. IBM, *Continuous System Modeling Program III (CSMP-III) and Graphic Feature*. Program product number 5734-X59, Manual number GH 19-7000, White Plains, New York (1971).

14. IBM, *Mathematical Programming System Extended/370 (MPSX/370)*. Reference manual, New York, New York (1976).

15. IMSL Inc., *International Mathematical and Statistical Libraries (IMSL)*. Reference manuals, Volumes 1-4, Houston, Texas (1982).

16. B. W. Kernighan and R. Pike, *The UNIX Programming Environment*. Prentice-Hall, Englewood Cliffs, New Jersey (1984).

17. P. J. Kiviat, R. Villanueva and H. M. Markowitz, *SIMSCRIPT II.5 Programming Language*. (edited by E. C. Russell), CACI, Inc., Los Angeles, Calif. (1975).

18. A. M. Law and S. G. Vincent, *UNIFIT: An Interactive Computer Package for Fitting Probability Distributions to Observed Data*. User's Guide, Simulation Modeling and Analysis Company, Tucson, Arizona (1983).

19. R. L. Moose, Jr., Proposal for a model development environment command language interpreter. Technical Report CS83032-R, Department of Computer Science, Virginia Tech, Blacksburg, Virginia (1983).

20. R. E. Nance, Model representation in discrete event simulation: the conical methodology. Technical Report CS81003-R, Department of Computer Science, Virginia Tech, Blacksburg, Virginia (1981).

21. R. E. Nance and O. Balci, The objectives and requirements of model management. In: M. Singh (editor-in-chief), *Encyclopedia of Systems and Control*. Pergamon Press, Oxford, to appear (1985).

22. R. E. Nance, A. L. Mezaache and C. M. Overstreet, Simulation model management: resolving the technological gaps. *Proc. Winter Simulation Conf*. Atlanta, GA, pp. 173-179, (1981).

23. R. E. Nance, O. Balci and R. L. Moose, Jr., Evaluation of the UNIX host for a model development environment. *Proc. Winter Simulation Conf*. Dallas, TX, pp. 577-584, (1984).

24. J. F. Ossanna, NROFF/TROFF user's manual. Technical Report 54, Bell Laboratories, Murray Hill, New Jersey (1976).

25. C. M. Overstreet, Model specification and analysis for discrete event simulation. Ph.D. Dissertation, Virginia Tech, Blacksburg, Virginia (1982).

26. C. M. Overstreet and R. E. Nance, Graph-based diagnosis of discrete event model specifications. Technical Report CS83028-R, Department of Computer Science, Virginia Tech, Blacksburg, Virginia (1984).

27. C. M. Overstreet and R. E. Nance, A specification language to assist in analysis of discrete event simulation models. *Commun. ACM 28*(2), 190-201, (1985).

28. A. A. B. Pritsker and C. D. Pegden, *Introduction to Simulation and SLAM*. John Wiley and Sons, New York (1979).

29. A. L. Pugh, III, *DYNAMO II User's Manual*. MIT Press, Cambridge, (1970).

30. R. Richels, Building good models is not enough. *Interfaces 11*(4), 48-54, (1981).

31. R. F. Roth, S. I. Gass and A. J. Lemoine, Some considerations for improving federal modeling. *Proc. Winter Simulation Conf*. Miami Beach, FL, pp. 213-217, (1978).

32. S. I. Schlesinger, et al., Terminology for model credibility. *Simulation 32*(3), 103-104, (1979).

33. Sperry Rand Corporation, *Sperry UNIVAC 1100 Series Functional Mathematical Programming System (FMPS) Programmer Reference*. St. Paul, Minnesota (1975).

34. SAS Institute Inc., *Statistical Analysis System (SAS) User's Guide*. Cary, North Caroline (1982).

35. W. Teitelman and L. Masinter, The INTERLISP programming environment. *Computer 14*(4), 25-33, (1981).

36. U.S. General Accounting Office, Report to the congress: ways to improve management of federally funded computerized models. LCD-75-111, U.S. General Accounting Office, Washington, D.C. (1976).

37. University of Waterloo, *SCRIPT Reference Manual*. Version 3.7, Department of Computing Services, University of Waterloo, Ontario, Canada (1982).

38. A. I. Wasserman and S. Gutz, The future of programming. *Commun. ACM 25*(3), 196-206, (1982).

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CS83022-R	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) REQUIREMENTS FOR MODEL DEVELOPMENT ENVIRONMENTS		5. TYPE OF REPORT & PERIOD COVERED Final June 6 - Oct. 15, 1983
		6. PERFORMING ORG. REPORT NUMBER CS83022-R
7. AUTHOR(s) OSMAN BALCI		8. CONTRACT OR GRANT NUMBER(s) N60921-83-G-A165
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Sea Systems Command Office of Naval Research		12. REPORT DATE 6 February 1985 13. NUMBER OF PAGES 33
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Surface Weapons Center Dahlgren, Virginia 22448		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

A limited distribution of this report is being made for early dissemination of its contents for peer review and comments.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Discussions with Richard E. Nance, John A.N. Lee, C. Michael Overstreet, Robert H. Hansen, and Robert L. Moose are acknowledged as contributing to ideas developed in this report.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

automated support, mathematical programming, modeling, model development, model management, model quality assurance, simulation.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report presents the fundamental requirements for Model Development Environments (MDEs) and offers guidance for MDE designers and implementers. A MDE provides an integrated and

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

comprehensive collection of computer-based tools to (1) offer cost-effective, integrated, and automated support of model development throughout its entire 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. The structure of the MDEs is composed of four layers, namely, hardware and operating system, kernel MDE, minimal MDE, and MDEs. Although the requirements perceived for each layer of the environment are generically applicable for simulation (discrete event, continuous, combined) and mathematical programming modeling tasks, the focus of this report is on discrete event simulation model development. A scenario is included to illustrate the uses of minimal MDE tools and to provide a view of the operation of a MDE.