Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

A STANDARD SIMULATION ENVIRONMENT: A REVIEW OF PRELIMINARY REQUIREMENTS

Chair
Mary Ann Flanigan Wagner

SysTech Software Solutions
P.O. Box 413
Annandale, VA 22003, U.S.A

Panel Presentation

Suleyman Sevinc

Knowledge Systems Group Basser
University of Sydney
N.S.W. 2006, AUSTRALIA

Oryal Tanir

Bell Canada
Quality Engineering and Research
2265 Roland Therrien Blvd.
Longueuil, QC J4N 1C5, CANADA

Respondents

Peter L. Haigh

High Performance Software, Inc.
PO Box 292466
Dayton, OH 45449, USA

James D. Arthur
Richard E. Nance

Systems Research Center
Department of Computer Science
Virginia Polytechnic Institute & State University
Blacksburg, VA 24061-0251, USA

Herbert D. Schwetman

Mesquite Software, Inc.
8920 Business Park Drive
Austin, TX 78759, USA

A Standard Simulation Environment: A Review of Preliminary Requirements

Oryal Tanir
Suleyman Sevinc

ABSTRACT

The concept of a simulation environment combines different aspects of the simulation process into one complete powerful tool. The environment should provide the necessary state-of-the-art advances and concepts that a modeler may require during the simulation development and execution process. The "environment" requirements must address the end user needs while minimizing the necessity of learning a new language. The requirements must specify the useful features, independent of different platforms and implementation languages. Hence the opportunity exists for standardization of environments. This paper opens a discussion on challenges and rewards awaiting the simulation community with the advent of a standard environment.

1 INTRODUCTION

Simulation is accepted as an integral part of the product life cycle in many industries. The concept of a simulation environment encompasses the underlying simulation language and support tools. The real world of simulation is an amalgam of different environments striving to meet similar objectives. Although these environments provide similar features, they rarely use the same modelling language. A frustrating endeavour for many system modelers is to utilize a large model that was carried out by another group in the same company, using a different simulation language. Obstacles such as learning a new paradigm, incompatible models, and incompatible data formats prevent the exploitation of previous work.

This paper presents some ideas as to possible avenues that can be taken to ease model exchange, re-use of model components between simulation environments, and other amiable connotations of a standard simulation environment. The paper will first summarize the notion of a reference model for simulation environments introduced in Tanir and Sevinc (1994). The reference

model categorizes specific groups of requirements that an environment should possess, partitioning them into layers of functionality. The potential standardization of the different layers of the model is then discussed to promote some thought as to what a standard may provide to the community.

2 THE REFERENCE MODEL

The various constructs of simulation environments can be better understood by defining different layers that characterize their functional behavior. This representation is classified as the "reference model" for a simulation environment. The intent of the model is to permit a uniformity of features. Guided by this model, standards could be defined for different layers of the model that would permit the development of simulation independent tools.

The model consists of five distinct layers with all of them accessing the bottom most layer (layer 0). The top-most layer (the application layer) is allowed access to all layers. This enables application developers to add application specific constructs to their environment. Properties of lower layers make it possible to implement similar features at higher layers. The lowest layer, layer 0, provides basic language level support for the environment. Layer 1 defines the requirements for model specification. Layer 2 deals with model knowledge management. Layer 3 is the system design layer. A summary of the basic requirements of the layers follows below.

2.1 Layer 0: The Host Language

Layer 0 provides the basic facilities for supporting model specification. This layer can be any general purpose programming environment. However, to properly support the higher level activities of modelling and simulation, the layer needs to support high level data structures and operations.

The choice of the host language will affect the freedom of the modeler at higher levels of simulation. It is not our intention to determine one unique language for layer 0. It may well be a set of languages that are used as the host language. Even after the host language is chosen, one must still emphasize the specific implementation of the language so that it will not restrict the higher level activities of simulation.

2.2 Layer 1: Model Specification

This layer provides the basis for the specification of models and a model base. It is generally agreed that modelling is the most basic activity of a simulationist. Modelling is an activity in which real or imaginary worlds are expressed in a symbol space of a formal language.

The requirements for layer 1 specifies the functionality and features that can be accessed by and incorporated at the higher two layers. With this in mind, the following requirements constitute the minimum basis for flexible and powerful modelling capabilities. It is also to be noted that the requirements for the higher layers mostly rely upon the features defined in layer 1. However, for added flexibility to the modeler, the upper layers can also access layer 0.

THE MODEL ABSTRACTION: A model represents a segment of the domain being studied. It replicates the behavior of its real counterpart in symbol space by generating trajectories consistent with those of what it represents. A model is the most basic construct available to a simulationist—analogous to an object in the object-oriented paradigm. Therefore, support for defining a model in a simulation environment is fundamental.

A model consist of a set of operations, data and interfaces. A model can be described using a set of operations on data. A data structure, often referred to as state or representation, may take the typical form such as a stack, list or queue. Operations manipulate the data and need to be consistent with the structure. Although the operations are primarily intended for data manipulation, other operations for pre- and post-processing purposes such as window drawing and filtering may be desired.

Interfaces are intended to ensure that models can interact with their environments, including other models, in a predefined manner. Typically, communication is established by traditional message passing mechanisms between classes. This enhances modularity, which is both a powerful and necessary requirement in a simulation environment.

HIERARCHICAL MODEL COMPOSITION AND DECOMPOSITION: Building up models from simpler ones or decomposing a complex model into its sub-models is a powerful and necessary requirement. Decomposition facilitates the model verification and validation. A large model can potentially be decomposed into simpler sub-models. The problem of verification of the large model can then be reduced to verifying the simpler sub-models. Modularity and decomposition are complementary concepts. It is useful to separate models which are decomposed from those that are not. A model with no components is an atomic model; otherwise, it is a coupled model.

The major distinction of coupled models is the existence of a coupling scheme to connect the interfaces of its component models. Coupled models may have

operations for pre- and post-processing, as well as for manipulating their component models and outputs.

The modeler should be free from implementation details of high level constructs in the environment. That is, the environment should provide facilities for debugging which will recognize constructs such as atomic or coupled models, as well as simpler types of operations.

FORMAL BASIS: The validity of the modelling process for discrete event simulation can be established with a sound underlying basis. Many mathematically sound formalisms exist as options, such as DEVS (from System Theoretic foundations) or Petri-Nets (from graph theory). The formal basis that is chosen may well define the ease of validation or representation of a model and model base.

MODEL SPECIFICATION LANGUAGE: The language selected to represent the model will impose limitations, specific to its peculiarities, on the ease of representation. The specification language can be distinct from the implementation language, allowing the modeler to mostly work oblivious to the underlying language syntax.

MODULARITY: Models are distinct objects that can communicate with one another and the environment through their interface specifications. Hence, adherence to the concept and structure of "model" implies modularity. The level of communication between models will be defined by the attributes that are inherited from layer 0.

VERIFICATION: An important and often neglected aspect of a simulation environment is the ability to test and verify the behavior of each sub-model of a larger model. With the ability to test each model before building a larger and complex one, the modeler minimizes verification problems in the larger model. The interface specification of the model must permit viewing the behavior of a given model with respect to controllable stimuli. Decomposing a large model into its smaller constituents, generally facilitates the management of the verification process.

EXPERIMENTATION FACILITIES: Experimentation is the phase that follows modelling in a design or analysis task. The purpose of experimentation is to learn more about the system under study by subjecting its model to various input sequences selected from the legitimate inputs of the model.

Since developing experiments can be regarded as a modelling activity, experimentation specifications used in the modelling environment should be similarly applicable to design experiments as well. The key issues can be summarized as:

- Data or statistics gathering capabilities. Such capabilities should provide the ability to monitor specific points in a model, i.e. inputs and outputs.
- Observation of interactions between the components models. Facilities for intercepting data (messages) exchanged between sub-component models and means to relate the cause and effect of such exchanges.
- Facilities for redefining basic algorithmic components of the simulator. e.g., flexibility in choosing internal random number generators or utilizing an external algorithm or data file.
- Facilities for presentation of information related to experimentation. Some examples; check-pointing, resetting, restarting, stopping, etc.

ARCHIVE CAPABILITIES: The layer must provide provisions for storing a specific model as well as retrieving it.

MODEL REPLICATION: Models within a model base must be replicable. This implies that their respective attributes and integral components can be copied to a new model. This requirement enables models to be reused within a project.

MODEL REUSE: This requirement dictates the ability to inherit or reuse the whole or a part of a given model. Modularity alone implies reusability at the model level. However, more support is required to ensure reuse of parts of a complex model at higher levels.

EXTENSIBLE MODELS: This requirement refers to the ability to extend the definition (or specification) of a model. Therefore a new model can be generated from another by inheriting its specifications and incorporating additional features. The last two features are both required to support model composition.

2.3 Layer 2: Knowledge Management

Layer 2 provides the mechanisms for utilizing and manipulating the models specified at layer 1. The current state of the art in simulation provides ways of organizing and using the model knowledge at this layer. However, developing requirements at this level requires considerably more investigation and analysis as to avenues of interest to the simulation community.

Modelling is usually not a stand-alone activity. It is typically a part of a design or analysis project. Therefore the models may be used in many different ways. For example, a model may need to be stored and checked against other models for equivalence, etc. Consequently there must be higher level constructs to manage such functions in an efficient way which we call knowledge management.

A typical example is a mechanism for organizing models in a structured manner whereby model interconnection can be managed by a knowledge management and library support system. When a complex re-usable sub-model is requested from the library, the environment can determine and create the necessary model connections and resources to effectively configure the model to function with the parent model.

2.4 Layer 3: System Design

This layer is the level dealing with design oriented user issues. There is a significant amount of work to be performed in defining requirements and features at this layer. Typically the issues are more complex and not clearly defined. This implies support for the design methodology using the primitives of the lower layers. Important advances at this level have been reported. However, since there is not a general consensus for defining requirements at this level, specification of requirements for a reference model at this time is difficult.

2.5 Layer 4: Application Layer

At this layer the environment is influenced by the application domain. There is necessarily no restriction upon applicable domains, as they can vary from manufacturing to computer network simulation. What should be clear however is that this layer will be specific to a given application and we are not suggesting standardizing at this level. What is perceived for this layer is a set of domain specific extensions to the standard. For example, within this layer, the telecommunications domain may wish to make available features for easy modelling of transmission lines, digital switches or modulators. Most of these requirements would not be applicable to another domain such as flexible manufacturing systems. Hence separate requirements would need to be generated for the two domains.

Simulation technology has been evolving by developing upon concepts such as parallel computing and distributed simulation. The choice of language can be affected by the computer platform that will execute the simulation. An object-oriented language can be nicely implemented in a distributed simulation environment compared to a structured programming language. Hence requirements of different applications can demand additional ones. The environments endeavour to remain flexible in allowing future requirements to integrate into the various layers.

3 AVENUES FOR STANDARDIZATION

The reference model is a starting point for pursuing avenues of standardization of simulation environments. The level of standardization for each layer of the model is not the same and the potential benefits also vary. This section examines some of the issues evolving at the different layers, in the hope of promoting further discussion.

3.1 Layer 0

The implementation layer is simply a computer language, hence standardization within the scope of this paper is not applicable.

3.2 Layer 1

This layer presents most opportunities for standards to evolve. The requirements defined at this layer are necessities that current simulation practices demand. A standard firmly establishing the requirements can provide a common conceptual platform for the modelling capabilities of environments. Certainly environments will differ in the way in which they abstract data, however the capabilities can be recognized in a global manner.

For example, a definition of a formal basis for models can permit the verification of model properties across different simulation paradigms adhering to the same standard. This would imply that a model created in a particular language would be translatable into the common formal paradigm.

3.3 Layer 2

The knowledge management layer also presents strong standardization possibilities. This layer has the added potential of permitting model interchange or re-use within a limited scope. Mechanisms for organization of models are to be described in the layer, hence a possible equivalence of models can be contrived if models are archival in terms of specific properties attributed to each.

For example, if the knowledge management system permits the organization of processor models in terms of speed, architecture-type, data width etc., then different processor models adhering to the given attributes and are stored in a database could be transparently utilized by the knowledge management system.

3.4 Layer 3

The system design layer is difficult to fit into a single standard. The reason for this is that system design can take place within different abstraction levels of design,

each having different goals and requirements. For example, at an abstract architectural level of design, system requirement focus more message based communicating processes. At a lower level of design (for example circuit level hardware design), models may involve more detail on timing constraints and voltage levels.

It is envisioned that layer 3 could consist of a set of standards for the different system design abstractions employed throughout the design community.

3.5 Layer 4

The application layer is also amiable to a set of standards. This layer is application domain dependent and would require a sub-standard for potential domains. The manner in which a domain is partitioned is significant and successful only if achieved by consensus of the domain modelers. For example, a potential domain may be telecommunication systems. A potential starting point may be to utilize existing standards (such as the Specification Description Language - SDL) for specifying telecommunications constructs in conjunction with added simulation symbols and constructs to define a standard.

4 CONCLUSIONS

Issues that are likely to be involved in defining a standard simulation environment have been investigated. A multi-layered reference model is proposed as a means of systematically defining the requirements of such an environment. This paper has highlighted some possible avenues for standardization. We hope that the article will stimulate discussion and further development in this area. As a result the user would be provided with a richer set of tools to produce more reliable models, and suppliers would be provided with a potentially rewarding market to explore.

REFERENCES

Tanir, Oryal and S. Sevinc, "Defining Requirements for a Standard Simulation Environment," IEEE Computer Magazine, February 1994, pp. 28–34.

Defining Requirements for a Standard Simulation Environment:

An Examination of Environment Design and the Applicability of Software Engineering Requirements to the Standard Simulation Environment

James D. Arthur

An environment can be described loosely as those parts of a system that a user perceives. This perception extends beyond the immediate user interface to facilities that provide ancillary user support. It includes the psychological "feel" of the system as well as the details of its functionality. In essence, the environment defines how one expresses computations as well as what one can express. More specifically, an environment is the synergistic integration of software modules to provide a strong, cohesive framework for formulating and solving a given set of tasks.

As so appropriately stated by Tanir and Sevinc, "the world of simulation is really an amalgam of different environments striving to meet similar objectives." I applaud their efforts in taking the first step toward establishing consistency and structure in the development of a standard simulation environment.

The following critique is intended to provide constructive suggestions from two perspectives: the general design of environments and the promotion of fundamental software engineering concepts which are also applicable to simulation.

Based on their reference model the authors convincingly argue requirements for each individual layer. From an environment perspective, however, what has been overlooked (or at least under-emphasized) is the additional requirement that each layer must present a consistent and integrated interface to each other. All environment tools must work in harmony. The authors state such a requirement for tools within each layer, but this must be extended to encompass the interaction among tools between different layers. Moreover, because components defined at lower layers of the reference models support specification abstractions at successively higher layers, the definition of requirements promoting a unified, synergistic relationship among tools should (must) start at layer 0 and be propagated upward.

The authors also state that components developed at the application layer can directly address elements of the lower three layers. While this is certainly a desirable and beneficial arrangement, interface accommodations must be addressed if multiple layer 4 abstractions can include (or make visible), as part of

their own interfaces specification, those interface specifications attributable to tools defined at different lower layers. For example, tool A (defined at layer 0) and tool B (defined at layer 3) might produce structurally similar data elements, but which contain information encoded by different algorithms. If C and D (layer 4 tools) employ A and B, respectively, information exchange between C and D will produce erroneous results.

Because the user can (and will) interact directly with all four (4) layers during task definition, the requirement for one (or more) specification language(s) at each layer should also be considered. For any given layer, the resident specification language should embody orthogonal constructs that reflect abstractions appropriate to that layer, composition mechanisms for combining those constructs, and operations that permit such structures to be manipulated in a manner consistent with the objectives underlying the definition of each respective layer. Effectively, layers 2, 3 and 4 should support a model manipulation language, a system design language, and a user specification language, respectively. For layer 0, the base language itself will serve as its own specification language.

From a software engineering perspective, specific provisions (or requirements) should be introduced to support documentation at each level (or layer) and traceability among layers. Documentation requirements should focus on succinctly capturing the initial system design and subsequent evolution (or maintenance) thereof, as well as the rationale behind each design/maintenance decision. Traceability of system requirements has been a long-time (yet elusive) goal of software engineering. The tracing of high-level requirements to their lower-level embodiments is crucial to verification, validation, testing and maintenance. All four of these activities are inherent to simulation model development. A layered model development environment, like that suggested by the authors, underscores the need for a traceability requirement.

In support of system design the authors suggest the need for a methodological approach to dealing with the attendant complex and ill-defined issues. A methodology is defined by a collection of methods, and a set of rules for applying them. More specifically, a methodology (1) organizes and structures tasks comprising the effort to achieve a global objective, (2) defines methods for accomplishing individual tasks within the framework of the global objective, and (3) prescribes an order in which classes of decisions are made which lead to the overall desired objective. Present in each of these elements is the notion of a global objective. Realizing such an objective, however, requires the use of specific principles in the development (or specification) process. In identifying candidate requirements for a standard simulation environment, if a methodology is included (or encouraged) in its definition, then derived requirement(s) for tools to support the employment of principles to achieve the stated methodological objective should also be considered.

Standards for Simulation Environments

Peter L. Haigh

GENERAL

The following comments are offered from the perspective of a practitioner and a vendor in the field of computer system and network performance engineering, as opposed to that of a researcher in the field of simulation technology or methodology. Practitioners are usually pragmatists, and are concerned with the practical matter of cost effective and timely simulation projects. We are less concerned with the formalities of elegance, structure, object orientation and environment standards. These things, however, can make our work easier and improve our craft, which is an objective of researchers and tool developers.

INTERACTION

Probably the greatest practical advantage of a standard environment would be to be able to execute multiple models simultaneously and have them interact. Models implemented in different languages could then be integrated to form larger models. This demands that multiprocessing and inter-process communication features be inherent in the operating system. The environment would provide a convenient mechanism to accomplish this.

PROPRIETARY VENDOR ENVIRONMENTS

Vendors hope to attract users to their product offerings by providing a comprehensive tool set designed to facilitate ease of use of their core product or products. There may be a motivation for a vendor to produce a proprietary environment and tools and develop a familiarity of the user to the vendor's orientation. Users can develop a bias toward a particular world view and tool set. Another vendor's product, even if superior in several ways, may be rejected by the user because it doesn't look and feel like the familiar product. If there were a standard environment definition, users could judge which parts of the environment are provided by

various simulation products. This would be helpful in evaluating and selecting simulation products.

Vendors who do not embrace an open architecture do themselves and their users a disservice. As a user of a hypothetical simulation product, I would like to be able to integrate with it, for example, a different vendor's statistical analysis software or a graphics package. The current availability of such packages serves to illustrate that few, if any, 'environments' contain all the features a modeler might need. Indeed, the best situation from the modeler's viewpoint would be to be able to integrate all one's favorite tools produced by several different vendors into a single environment and have them all accessible and able to interface with each other.

EDUCATION

A set of standards for simulation environments could benefit both decision makers and modelers by having a common reference when discussing simulation projects. It might also facilitate easier movement of simulationists among projects and employers. As simulation is still a specialized (and under-appreciated) field, education of the decision making management, as well as modelers, in standards for simulation environments is desirable to help promote awareness of simulation technology. Simulation is still an under-applied discipline, and anything that would get business planners and managers talking knowledgeably with system designers about simulation early in a project cycle would be a step forward.

IMPLEMENTATION

The problem with defining and implementing standards in a discipline which has evolved without a standard 'environment' is that there are some excellent products in use, each with its own version of an environment. Converting these products to a standard environment may be difficult for vendors and yield minimal benefit for users.

REFLECTING ON THE REQUIREMENTS FOR A STANDARD SIMULATION ENVIRONMENT

Richard E. Nance

INTRODUCTORY REMARKS

We are indebted to Tanir and Sevinc for the focus paper and the earlier paper from which it is derived (Tanir and Sevinc 1994). In seeking to define requirements for a standard simulation environment, the authors have affirmed the degree to which simulation model development and analysis has transitioned from a "programming" to a "modeling" focus, noted some ten years ago (Nance 1983, p.326). Much of what the authors offer is perceptive, descriptive and accurate; however, the tenor of the respondents' remarks must focus on omissions and differences. That is the nature of this type of session, and the basis for progress toward the laudable goal of a standard simulation support environment.

CRITICAL ANALYSIS

My principal criticism of the authors' description of their approach is that a confusion is created between desirable characteristics, ergo requirements, of the **model** and those of the **environment**—the set of tools used to create the model. (The reference to the environment with which a model interacts provides an additional obfuscation.) Does "message passing between classes" (actually class objects) enhance the modularity of the environment or the model? The authors appear to make the claim for the former. While it may be "useful to separate models which are decomposed from those that are not," what requirements are placed on the environment to make this distinction?

The use of a reference model is understandable; however, justification of the layered partitioning is not clear. The characterization of System Design (Layer 3) is so sketchy that the intended functionality is indiscernible. Description of functionality at the detailed level of the NIST Reference Model (NIST 1993) is not necessary, but the authors are operating at a very gross level. Permitting upper layers to access Layer 0 directly for added flexibility sacrifices **environment modularity** in a major way.

Environment support for model archival, reuse, and extension are stipulated as requirements, but are decomposition and composition the only forms of abstraction to be provided? Are the principles of iterative refinement and progressive elaboration to be supported? What about inheritance—single or multiple? Should model documentation be supported—as a byproduct or addition to model specification? Verification is noted, but little is said about validation. In fact, several of the requirements stipulated in Balci (1986), cited by Tanir and Sevinc, are either relegated to layers 3 and 4 or disregarded, such as automated model diagnosis, on-line modeler assistance in using the environment, and seamless tool integration.

ADDED GUIDANCE IN REQUIREMENTS DEFINITION

Requirements definition is difficult. Our colleagues in software engineering have repeated this conclusion loudly and often. The efforts of Tanir and Sevinc are timely and well-intentioned, but in my opinion three fundamental precursors to requirements definition are missing from their approach as described:

1. the recognition of the problems that confront the development of and experimentation with large, complex simulation models,
2. the definition of a methodology intended to reduce or eliminate such problems, and
3. user feedback from experience with environment prototypes designed to support the defined methodology to overcome the recognized problems.

Some might challenge the necessity for a defined methodology, but without it, the result is likely to be a suite of tools with minimal or non-existent interfaces rather than an environment.

Fortunately, all three precursors are at hand, or nearly so, and stimulated by the efforts of Tanir and Sevinc we should continue with the arduous task of requirements definition. Clearly, we should use the results of related efforts, such as (PSESWG 1993; NIST 1993), while being mindful of the distinctive requirements for supporting simulation modeling and experimentation.

REFERENCES

Balci, O. 1986. Requirements for model development environments. *Computers and Operations Research 13*:53–67.

Nance, R.E. 1983. A tutorial view of simulation model development. In *Proceedings of the 1983 Winter Simulation Conference*, ed. S.D. Roberts, J. Banks, and B. Schmeiser, 325–331. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

NIST (National Institute of Standards and Technology). 1993. Reference model for frameworks of software engineering environments. Special Publication 500–211 (Technical Report FCMA TR/55, 3rd Edition), August.

PSESWG (Project Support Environment Standards Working Group). 1993. Reference model for project support environments, Version 1. NAWCADWAR-93023–70, February.

Tanir, O. and S. Sevinc. 1994. Defining requirements for a standard simulation environment. *IEEE Computer 43*:28–34.

PORTABLE SIMULATION MODELS

Herbert D. Schwetman

ABSTRACT

This presentation discusses some of the issues which limit the movement of models among different computer systems. The major factor which impact model portability include:

- Programming language
- Operating system
- File format
- Processor
- Database interface
- Graphics user interface

Experiences with two simulation software products highlight some of the successes and some of the difficulties found in moving simulation models between systems.

INTRODUCTION

Many users want to move a simulation model from one system to another. In many cases, an analyst has developed a model for his/her own use. Later, another organization (or group within the organization) discovers the model and wants to use it on another system. Many times, consultants have models which could be used by clients, if the model could execute on the client's systems. In universities, students would like to use models from a class on their home-based systems.

There are several approaches to dealing with these kinds of problems. This paper will discuss some of these approaches and give some insights into which approaches work in which situations.

LEVELS OF MODEL PORTABILITY

In this discussion, there needs to be a distinction between the **host** system and the **target** system. The host system is the one on which the model was originally developed. The target system is the system to which the model is to be moved.

There are also different kinds of uses of simulation models. In some cases, all that is required is that the model be able to execute on the target system. This assumes that all changes to the model have been anticipated and can be handled by the model at runtime. The model itself does not have to be changed; only the inputs to the model are changed as user needs dictate.

Some models need to be changed in ways which require the model to be recompiled. In these cases, the compiler for the language used to write the model has to

be available on the target system. In addition, some models may call on routines in different subroutine libraries. Copies of these routines have to either be provided with the model or have to be available on the target system.

Some models make use of a graphical user interface (GUI), either to visualize and modify the model and to view the output or the operation of the model (model animation). In these cases, the target system has to support the GUI used by the model on the host system.

Finally, some models may use a database management system (DBMS) to store model outputs for use in summary reports. Again, either the DBMS software has to accompany the model, or it has to be available on the target system.

EXECUTION LEVEL COMPATIBILITY

The most elemental level of model portability is when the host and target systems are very similar. Both systems have the same type of processor (instruction set) and the same operating system. Furthermore, both systems have all of the support software required by the model.

Today, many people have access to a IBM compatible PC with Windows 3.1. Such systems range from laptop or notebook systems up to powerful workstations and servers. Thus, a model which can runs on a PC with Windows is potentially portable to a large number of sites.

If the first kind of portability is assumed (model execution), then the model can by supplied to the target system as an executable file. With most widely used systems, such as PC's, Macintoshes and UNIX workstations, this is easy to do. However, it is crucial to make certain that not only do the host and target system have the compatible processors, but that the operating systems are also compatible.

LANGUAGE LEVEL COMPATIBILITY

It is possible to distribute software which can be modified and recompiled on different target systems. If the host and target are very similar and if they both have the same versions of the compiler and the same runtime libraries, etc., then this is fairly straightforward. If, on the other hand, the host and the target systems are different (such as different processors, different operating systems and/or different compilers), then the task of moving a model which must be recompiled becomes more difficult. The model developer can take care to use features of the language which are common (almost) to all compilers for the target systems. Another issue here is whether or not all parts of the model can be distributed in source form. In some cases, the underlying library of simulation routines cannot be distributed in source form.

Another solution to this problem is to distribute a compiler with the model. If the compiler for the language of the model runs on the target system, then modified versions of the model can be recompiled and executed on the target.

PORTING MODELS WITH GUI'S, ETC.

If the model makes use of many features of the host system, then moving the model to a different kind of target system becomes much more difficult. If the model (or the simulation environment used to develop the model) use machine language code or features which are unique to a particular system, the porting the model is almost impossible. Similarly, if the model has a GUI which is not supported on the target, moving the model is difficult, if not impossible. In some cases, it is literally cheaper to purchase a new platform which is similar to the host platform than to try to move the model and its support software.

These problems present special challenges to people developing, selling and supporting simulation products and simulation models.

WHAT WORKS

Moving a model which just needs to execute on the target system is easy, if the system is very similar to the host system. Most sophisticated simulation environments are developed for a limited number of platforms. The developers make guesses as to which platforms most of their customers will have and then target only those platforms.

Some developers support a wider range of platforms. CSIM17, a product from Mesquite Software, runs on most UNIX workstations, as well as PC's and Macintoshes (with floating point hardware). However, CSIM17 does not support a GUI.

In the final analysis, the modeler has to make choices about what levels of support they need (in terms of features in their simulation environment) and what degrees of model portability they need.