Proceedings of the 1983
Winter Simulation Conference
S. Roberts, J. Banks, B. Schmeiser (eds.)

MODELING SYSTEMS USING DISCRETE EVENT SIMULATION

by

Lee Schruben

Cornell University

1. INTRODUCTION

In this article, we present an introduction to discrete event modeling and discuss some of the important issues related to model development. We are not talking about simulation codes nor statistical models for the outputs from such programs. Rather we will focus on the interface between a real system and a simulation code where we describe the system.

Typically, our understanding of the system we wish to study is limited. In modeling, we attempt to describe the behavior of the system. Since simulations, like all computer codes, have an explicit structure our system description must explicitly reflect the behavior of the system elements and their relationships. We resort to simplification and abstraction in obtaining this system model.

SYSTEMS:

By a system we mean a set of entities that interact with a common purpose according to a collection of laws and policies. The system may be real or hypothesized. We are using a functional definition of a system, that is, we define a system by its purpose.

The entities in a system are the physical or symbolic components of the system. For example, the entities in a factory include the physical machines, workers, storage spaces, material handling equipment, product routings, production schedules, etc.

There are two general categories of entities which depend on their membership in the system. Some entities may be considered as always being part of a system. These entities, such as machines in a factory, we will refer to as resident entities. Many popular computer simulation languages have special features for handling resident entities where they are called "permanent entities" (SIMSCRIPT), "facilities" (GPSS), and such. Other entities, such as parts being produced by a factory, are only part of the system for an interval of time. We will refer to entities that join and leave a system as transient entities. SIMSCRIPT refers to such entities as "temporary entities" while they are usually modeled using "transactions" in GPSS. The distinction between resident and transient entities is important for two reasons. First they may be handled differently in the simulation computer code (storage allocated

to transient entities may be reused once an entity leaves the system). More importantly, quite different models of a system may be created depending on whether one adopts a transient or resident entity viewpoint. For example: we might model a factory by describing what happens to parts as they travel through the production process or we might model the same factory by describing what happens to the machines and inventories.

Entities are conveniently thought of as belonging to sets or owning sets of other entities. From a resident entity viewpoint, a machine may be considered as owning a set (queue) of parts waiting to be processed. From a transient entity viewpoint, we might think of a part as owning a set (routing) of machines that will work on it before it is finished. It makes sense when studying a system to consider models both from a transient entity and from a resident entity viewpoint. Models based on a transient entity viewpoint are particularly suitable for implementation using a simulation language that has a "process" orientation (eg., GPSS or the network part of SLAM). Models based on resident entities are most easily implemented using a language that has facilities for handling "activities" or "events" (eg., SIMSCRIPT or SLAM). We defer discussion of various simulation languages and their particular "world-views" to the language tutorial.

Entities are described by their attributes. These attributes may be dynamic or they may be static. Dynamic attributes will change as time passes whereas static attributes remain constant over the period of interest to the system study. Attributes may also be qualitative or quantitative. In a model of a factory from a resident entity viewpoint, a machine may have a dynamic-qualitative attribute of activity, a static-qualitative attribute of machine type, a dynamic-quantitative attribute of number of parts awaiting processing, and/ or a static-quantitative attribute of its location. From a transient entity viewpoint, a factory model may have parts with a dynamic-qualitative attribute of shape, a static-qualitative attribute of its material, a dynamic-quantitative attribute of location, and/or a static-quantitative attribute of due date.

In the simulation program, attributes are often kept in an array indexed by an identifier code for the entity which they describe. As a rough guide, the storage requirements for a simulation program will be an increasing function of the sum of the of number of entities times the number of attributes for each entity.

Laws are rules that govern the behavior of or relationships between entities in a system that are not under the control of the system designer, operator, or manager. Examples are the law of gravity, labor laws, etc. Laws are (strictly speaking) deterministic; however, we may express our ignorance of the exact mechanism by which a law operates using probability. Sets of similar laws are often indexed by parameters. For instance, the acceleration of gravity at different altitudes is a parameter for the law of gravity. Parameters generally refer to quantitative indices for a family of laws. Laws are often the result of theory or experience and are rarely absolute. The parameters for laws are usually estimated from the observed behavior of the system or of similar systems. Both data and experience go into describing the mechanism of a law.

Policies are rules that concern the behavior of and relationships between entities in a system

that are under the system designer, operator or manager's control. Examples are the priorities for processing work orders, break times for workers, etc. Policies often determine the existance of entities in a system. For instance, the policy of how many workers to hire or machines to buy determines the numbers of such entities in the system. Similar policies are indexed by factors. The re-order point of an inventory stocking policy is such a factor. Factors may be quanitative or qualitative.

A complete description of a system includes the values of entity attributes, parameters for the laws, and policy factors as well as what can be said about the system's past and future. We usually refer to such a description as the system's state. The state space for a system is the set of all possible system states. A process is an indexed sequence of system states. The index is quite often but not necessarily always relative time.

MODELS

There are many definitions of models. Here we think of a model as a system that is used as a surrogate for another system. A real system may have several somewhat vague purposes: a private hospital provides quality health care to a community, employment, training for health professionals, and may try to make a profit. Many models, particularily analytical models used in operations research, are systems that have a single numerically measurable objective.

The entities in a real system are usually physical whereas those in a model system are often symbolic. In a computer simulation the entity N_i may symbolize the quantity of part i in inventory. Real entities have many qualitative attributes. Mathematical model systems that are run on a computer express all attributes numerically; perhaps by employing coding systems. Real systems may have a general state space where model systems have a finite state space. In computer simulations the state space is over a multidimensional but finite subset of numbers. Real systems tend to change with time where many models are static or dynamic only in a limited manner. There may be real system variables that are continuous such as time. In model systems on a computer the finiteness of the computer dictates that all variables be discrete. If the computer has a large enough word size this may not be an important distinction between real systems and models. Besides, measurement devices applied to real systems may make continuous variables appear discrete.

The policies in real systems (since they often involve people) are typically flexible. In mathematical models the policies are explicit and rigid. Depending somewhat on one's prejudices, the laws in a real system may be regarded as deterministic. In models incomplete understanding of the behavior of entities often forces us to probability laws to express our lack of knowledge.

Real systems are open in that they are influenced by their environment. Model systems on a computer are closed while the computer run is in progress.

Real systems are adaptive in that they may change their behavior depending on the history they experience. Computer simulations are adaptive only to anticipated histories which makes them essentially not adaptive.

Real systems are not stable (there may be shocks to the system from which the system cannot recover). Many models are stable in that catastrophies are not permitted.

The responses of real systems to changes in policy factors or parameters for laws are usually not linear. (By our definitions, if a factor or anything else changes in a system it becomes a different system.) A large class of engineering and statistical models assume that the performance of the system is in some way linear.

Real systems are non-stationary; the state of the system depends on time. For example, the state of a restaurant is different during the dinner hour than it is in the middle of the afternoon. Many models assume that the system is stationary.

Finally, in real systems all entities are strictly speaking transient, eg., machines wear out or become obsolete. In model systems some entities are viewed as permanent residents in the system.

With all of the above differences between real systems and model systems, one can not expect that the model be in a strict sense valid. The only valid model is the identity, that is, the system itself. A "valid model" is a contradiction.

MODELING

Modeling, in particular mathematical modeling as done in simulation, involves abstraction and simplification. The amount of detail in a system description is influenced largely by the use of entity attributes. Consider a manufacturing corporation that has several divisions. Each division may have several factories. Each factory has several departments which in turn have several machines. One may formulate a highly detailed description of the corporation by including the individual machines as entities. A less detailed description of the system may view the departments as entities with attributes of the numbers and types of different machines in each department. A higher level (less detailed) model might view the factories as entities with the attributes of the number and types of departments, and so forth. The objectives of the systems study determine the level of detail needed in the system description.

In simulation studies there is a tendency by less experienced practitioners to include unnecessary detail simply because it can be included. It is good practice to justify the inclusion of detail rather than trying to justify the exclusion of detail. In computer simulations, detail costs money and tends to make a model less flexible. It is usually easier to enrich a simulation by adding detail than it is to simplify one that has excessive detail. Unless a particular aspect of a system can be seriously expected to alter the study results it should be excluded from the model by making reasonable assumptions or aggregation of several entities into sets of entities. For example, it may not be necessary to individually identify each part awaiting processing by a particular machine if the purpose of the study is to measure machine utilization. On the other hand, if the delay times spent waiting for a machine are important then it may be necessary to consider each part as an individual entity with the attribute of waiting time.

Mathematical modeling is essentially a conceptual way of thinking. People with experience in modeling approach problems differently than those who have never modeled a system. The models they create identify relationships between entities, laws, and alternative policies for

designing and operating a system. It is important to realize that the process of modeling a system is in itself worthwhile. Even if a model is never used or the simulated system is never run on a computer, a subject system is better understood because of the modeling.

Thirteen references were surveyed that give "steps" in the mathematical modeling process. The union of these (paraphrased) steps is presented in the following table with the frequency that the references included each step.

Table 1
STEPS IN THE MODELING PROCESS

Step	Frequency that the Step is recommended
1. Establish the system boundry	1/13
2. Formulate the Problem	13/13
3. Factor the Problem	1/13
4. State Modeler and User Prejudices	0/13
5. Seek Analogies	2/13
6. Try a simple example	1/13
7. Decide on the level of detail	2/13
8. Define some symbols	2/13
9. State the obvious	1/13
10. Choose a modeling technique	2/13
11. Choose a solution technique	2/13
12. Collect data	3/13
13. Estimate parameters	2/13
14. Evaluate estimates	2/13
15. Design experiments	2/13
16. Build a model	13/13
17. Analyze the model	13/13
18. Test the solutions	6/13
19. Implement results	3/13
20. Update the model	2/13

There seems to be no definitive statement of exactly how one does modeling; none will be attempted in this paper. The following is a list of some of the more important simulation modeling activities that we will discuss in the remainder of this paper.

ACTIVITIES IN A SIMULATION PROJECT

1. problem identification
2. identify system structure
3. inventory data sources
4. simple models and solutions
5. design experiments
6. design run strategies
7. code simulation
8. test code
9. identify data needs
10. collect and prepare data
11. validate simulation
12. run experiments
13. analyze output
14. prepare recommendations

A simulation project does not typically progress serially from step to step through the above list. Each activity may be repeated several times or not at all in a particular simulation modeling project. Furthermore, the activities often take place simultaneously. The ordering in the list represents an ideal progression through a project when nothing goes wrong. The remaining discussion is to a large extent academic; the author admits to having no experience with such "ideal" projects.

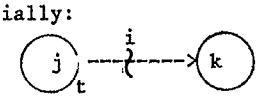
The first step, problem identification, is an activity that continues before, during, and after a

systems study. It is often not clear to the system designers, users, or analysts exactly what the problems are or even if there is a problem. One of the benefits of a systems study is that it provides the people involved with an opportunity to discuss some of the trade-offs between various system objectives. Either the people involved can reach an agreement on system priorities, or at least they know where they disagree.

In the second step, identifying the structure of the system, there are three general approaches using discrete event simulations. The three approaches, called world views, are event scheduling, process interaction, and activity scanning. The particular approach taken depends largely on the language chosen to implement the model.

The basic concept that describes the dynamics of a discrete event system is the notion of an event. The literature on simulation modeling contains several definitions of an event. Rather than adopt one of the formal definitions, one may think of an event as a potential change in the state or future of a system. There are two fundamental aspects of an event. One is the transformation of the state, the other is the influence of an event. Events may change the values of state variables and may cause further events (or maybe the same event) to be scheduled in the future. Events are usually implemented as subroutines or procedures in a discrete event simulation computer program.

A convenient way to describe the event structure of a system is to use an event graph. Basically, the state transformations associated with an event are represented by vertices or nodes on a directed graph. The events that may be scheduled (or cancelled) by a particular event are connected to that event with conditional edges. Pictorially:



means that upon occurrence of event j, if condition i is true, event k will be scheduled to occur in t time units. A similar event cancelling edge for the graph can be defined.

The analysis of such graphs can be used to identify events that must be initially scheduled in a simulation, determine the minimal set of state variables in a simulation, remove events that do not require separate procedures, and anticipate the possible problems that may arise when two or more events are scheduled to occur simultaneously. By associating paths in the event graph with processes and cycles in the graph with activities, it is possible to identify several potential equivalent system representations that may be developed into simulation programs. Several examples of event graph analysis will be presented during the tutorial.

The third step, taking inventory of the available data, is performed early in the study so that different modeling approaches may be considered that match the availablility of data. This also gets the systems analyst close to the real system at an early stage in the study. It is important that the search for data not be limited to data whose need is anticipated. Data on a system is collected because someone at sometime thought it was important. Knowing what records are kept on a system offers insight into how the system operates and what is considered important to the various persons involved with the system. Data collection can be one of the more expensive

activities in a simulation project. In this step one just tries to find out what data is available and in what form it is kept. The actual gathering of data should typically not begin until there is an explicit need for the data and a pretty good idea of how the data is going to be used.

The fourth step, development of analytical models (ie., mathematical expressions for the system relationships) and rough cut solutions, is important for two reasons. First, there may be no need to develop a computer simulation model once the structure of the system is examined using the discipline of mathematical modeling. Second, this activity will force the systems analyst to consider simple well structured models of the system. This may help prevent the study from becoming bogged down in excessive detail.

The fifth and sixth steps (designing experiments and strategies for the runs in each experiment) should begin before the seventh step (actual coding of the simulation program). In practice, step 7 is usually performed before steps 5 and 6. This can result in a simulation code that is cumbersome to use in the experiments that are to be run. One should have a pretty good idea of how a program is going to be used before coding.

The eight step, code testing or verification, is like the first step in that it is a continuing activity. In a large simulation program, it is often not possible to guarantee the correctness of the code in all possible situations. The eleventh step, customarily referred to as model validation (which seems to the author to be a contradiction in terms), means that the model system behaves like the system for which it is a surrogate in all the ways that are important to the study. A model will be easier to understand and hence more useful if it does not behave exactly like the system under study. The abstraction and simplification inherent in good modeling imply that the model will not in a literal sense be valid.

The ninth and tenth steps, data gathering and preparation, are (like the seventh step) often performed in a different sequence than in the activity list. It is not uncommon to worry about the unavailability of data even before there is an established need for the data. "Lack of necessary data" is probably one of the biggest reasons given for not starting a simulation project. This is nonsense, one does not know what data is really necessary until the project is well under way. Perhaps different modeling approaches can be used that have different data requirements.

The twelfth and thirteenth steps of experimentation and output analysis should use the best available statistical techniques. This is the subject of a later tutorial by Professor W. David Kelton at this conference.

Finally, the study recommendations should identify recommended solutions to the problems that led to the study and suggested strategies for implementation and monitoring the performance of these results. There should also be a summary that identifys problems discovered during the study. Like many rare things, a final report that identifiea identis errors made during a study is of great value.

There are too many references on simulation modeling to attempt a complete bibliography in the space permitted. The thoughts presented in this article are the author's own but were greatly influenced by others in the simulation community.