Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

ALTERNATIVE MODELING PERSPECTIVES: FINDING THE CREATIVE SPARK

James O. Henriksen
Wolverine Software Corporation
4115 Annandale Road
Annandale, VA 22003-2500, U.S.A.

ABSTRACT

This paper is a teaching piece. It is the latest of a series of papers by the author on the general subjects of modeling and problem solving (see Henriksen 1981, Henriksen 1986, Henriksen 1987, and Henriksen 1988). For the fourth year in a row, Dr. Alan Pritsker is also presenting a paper in a very similar vein (see Pritsker 1986, Pritsker 1987, Pritsker 1988, and Pritsker 1989). This paper examines nine tradeoffs which should be considered in developing or applying approaches to modeling. To illustrate these tradeoffs, this paper revisits a number of the examples used in previous papers and introduces some new examples.

1. INTRODUCTION

Previous papers by this author have presented a variety of modeling techniques. For the most part, those papers left unsaid where the motivations for the techniques had come from. In Sections 2.1 through 2.9, this paper sets forth a series of criteria that one should consider when developing a modeling approach. Each of these criteria gives consideration to tradeoffs in a particular aspect of modeling; hence the word "versus" appears in many section headings. These criteria are by no means orthogonal; there is a great deal of overlap and similarity among many of them. Furthermore, these criteria do not comprise a modeling methodology; they are merely a set of suggestions intended for students of modeling.

Section 3 sets forth a few brief thoughts on teaching of modeling, and Section 4 presents conclusions.

2. MODELING TRADEOFF CRITERIA

Each criterion is illustrated by example. In each example, the creative spark which occurred in the development of a modeling approach is identified, or the creative spark which must be found to select an appropriate approach is described.

2.1 Active vs. Passive World-Views

In Henriksen (1981), two alternative models of a machining facility were presented. The first model was written from the "active object, passive server" perspective. This perspective is very natural in network languages such as SLAM or SIMAN, and in transaction flow languages such as GPSS. Using this perspective, the operation of the machining facility was described as the flow of objects (parts) through a network, competing for scarce resources (machines). The resources are passive; i.e., they are acted upon by flowing objects; they do not have dynamic behavior of their own.

While the "active object, passive server" approach is often very natural, it works poorly when large numbers of objects compete for very scarce resources and/or when the rules by which resources are engaged and disengaged are very complex.

For these reasons, a second model was developed from the "active server, passive object" perspective. In this model, the behavior of the machining facility was described from the perspective of a machine. A machine's behavior pattern was easily described as follows:

1. Wait for a part to arrive in my input bin.
2. Process the part.
3. Wait for available space in my output bin.
4. Place the part in my output bin.
5. Go to step 1.

The second modeling approach offered several advantages:

1. The rules of operation for a machine were concisely stated in a single section of model code, rather than being spread throughout the model.
2. Performance was significantly enhanced.
3. Memory requirements were significantly reduced.

What was most interesting about this example was that both approaches could be easily implemented in network or transaction flow languages; however, the first approach was an obvious, knee-jerk approach, while the second was far less intuitive. The creative spark came in recognizing that active resources could be easily implemented in languages which otherwise encourage their users to think of resources as passive entities.

2.2 Time Domain vs. State Domain

In Henriksen (1986), alternative approaches to modeling accumulating conveyors were presented. One particularly difficult problem that was discussed was the issue of how to find free space of length N on a conveyor passing a given point, where the conveyor contains randomly placed objects of random length. A commonplace, but inferior approach to this problem is to divide the surface of the conveyor into a large number of small units of length, e.g., inches, and to model the operation of the conveyor as a sequence of small time advances (inch-by-inch operation).

The essence of this problem is the difficulty of handling complex state events. (A state event is an event conditioned on a specified state arising; the classic example of a state event is "don't fire until you see the whites of their eyes.")

An improved approach to this modeling problem was developed by viewing the surface of the conveyor as an alternating sequence of free and occupied spaces of variable length. Each space was characterized by its length and the time at which it was created. (Placing an object on the conveyor divides a free space into an occupied space and one or two smaller free spaces; removing an object creates or enlarges free space.) The improved data structure made it easier to scan the surface of the conveyor to find suitable free spaces, and most importantly, it made it possible to easily calculate the time at which a given free space would reach a given point along the conveyor. Using this approach, the arrival of a suitable space at a given point along the conveyor could be modeled as a scan of the space list and a time advance. A good representation of model state facilitated moving a complex state event from the state domain into the time domain. Of course, virtually all simulation languages are good at handling time advances, so the time domain approach works very well.

To summarize, model logic of the following form should always be regarded with great suspicion:

1. Does condition "X" exist now?
2. If so, go to step 5.
3. Wait for a tiny amount of time
4. Go to step 1.
5. ...

The creative spark occurs when (1) such conditions are recognized, and when (2) a means can be found to compute in advance (pun intended) the time it will take to get from step 1 to step 5. Another approach to this kind of problem is to introduce a synthetic process which "watches for" the required condition in some intelligent fashion. Synthetic processes are discussed in Section 2.9 below.

Favoring the time domain should not be regarded as a universally desirable approach to modeling. In the section which follows, an improved model was created by moving modeling actions out of the time domain and into the state domain.

2.3 Microscopic vs. Macroscopic Focus

In Henriksen (1988), a series of alternative models of a hypothetical battle between opposing infantry forces was presented. The first model presented was developed from the perspective of a soldier's eye view of the battle. Attention was focused on the behavior pattern of the individual soldier. The behavior pattern of a soldier was described as follows:

1. Select an enemy soldier to aim at. (Time elapse required)
2. Aim and fire. (Time elapse required)
3. Remove the enemy soldier from the model.
4. Repeat this behavior pattern as long as "I" (the individual soldier) am still alive and any enemy soldiers remain.

A second modeling approach was devised by considering the battle from the perspective of an imaginary observer looking down on the battle from above. From this perspective, the battle was described as follows:

1. Observe which side fires next.
2. Reduce the size of the opposing army by one soldier.
3. Continue this behavior pattern until one side has been annihilated.

At any given time, "which side fires next" is a uniformly distributed random variable, determined by the ratio of the remaining sizes of the opposing armies. For example, if 100 red soldiers and 50 blue soldiers remain, the probability that a red soldier is the next to fire is 2/3 (assuming equal equipment and marksmanship). This world-view enabled an important simplification of the simulation model: time became irrelevant. Using a uniform distribution to model "which side fires next," determines the sequence in which casualties occur, and this sequence is time-independent. This allows replacing the simulation model with a Monte Carlo model, i.e., a model which performs random sampling, but includes no time delays.

In this case, a significant simplification in the model was achieved by simply looking at a given system from a greater distance. The simplification did not require enlarging or reducing the scope of the model, nor did it require omission of any essential details. In fact, all it did was eliminate one non-essential detail, the need for a time advance mechanism. Of course, in the world of simulation, eliminating the need for time advance is rather remarkable.

Recognizing that simplifications can be achieved through a change in perspective requires a creative spark. When developing a complex model, one should continually seek abstractions which eliminate details, but retain the essence of system operation. One way to accomplish this is to continually ask the question "what would this system look like to me if I didn't know so much about it?"

My efforts to seek a more abstract view of the battle were rewarded by finding a perspective which eliminated only non-essential details and entirely preserved model authenticity.

2.4 Toy Models vs. Real Models

A number of years ago, I was called in to develop a model of the hardware and software systems for a new computer under development by a manufacturer. By the time I was called in, a prototype of the hardware was working, and a great deal of software had been written. The manufacturer wanted me to develop a model of the existing system, to calibrate the model against benchmarks run on a single-user prototype machine, and to then use the model to predict performance of the full, multi-user system. To the user, the hardware appeared to be a "source language" machine; i.e., it directly executed programs and commands without requiring the user to compile and link programs. The source language was interpreted by underlying firmware, invisible to the user. Because the firmware was large, and memory was expensive in those days, and hardware-assisted virtual memory was technically infeasible, the firmware had to be overlaid. A single copy of the overlaid firmware was shared by all users. Given the intensity of execution of the language interpreter, it remained to be demonstrated whether overlaying the firmware would actually work in a multi-user environment.

I was asked to model several applications running on the prototype system. Because of the manufacturer's concern over the efficacy of employing overlaid firmware, I was asked to model the existing overlay structure as accurately as possible. This entailed building into the model a representation of the then current overlay structure (strategy for which pieces of the program can or cannot reside in memory at the same time).

In due course, I was able to faithfully replicate known benchmark results with my model; however, the model was relatively ineffective for experimenting with alternative overlay structures. The reasons for this were twofold. First, it was very easy to redefine the real overlay structure on the prototype machine, using its linker. Second, the structure of the applications was built into the model, so experimenting with "new" applications required significant reprogramming. Despite these deficiencies, the model offered one distinct advantage over the real system: it collected statistics which gave insights into the components of performance.

This was a perfect situation in which to use (or at least start off with) a toy model. In a toy model, I would have characterized the application software and the overlay structure statistically. I would have expressed the operation of the system in terms of (1) the duration of time spent in each overlayable element of the firmware, and (2) the probabilities of transition to other overlayable elements. Durations and transition probabilities would have been randomly generated. In one or two days' time, I could have built a model which pretty well approximated the behavior of the prototype system, by broadly mimicking the characteristics of the current overlay structure. Using my toy model, I could have easily conducted meaningful experiments. What if my estimates of duration in a given state were off by a factor of two in either direction? What if a new overlay structure contained twice as many overlayable components? What if a typical element could invoke 4 or 5 randomly selected elements? By experimentally determining the answers to such questions, I would have learned a lot about how such systems really work. This knowledge would have been indispensable to designers of the real system.

In this story, the creative spark is missing. I can only look back and reasonably speculate that building a toy model at the outset of the modeling project would have greatly improved the results of the project.

2.5 Detailed vs. Abstract Models

In the project discussed in the previous section, I frequently encountered situations in which detailed descriptions of system operation were available, but abstract descriptions were not.

Programmers had access to a hardware timing device, so they could tell me literally to the microsecond how long any given routine would take to execute. At one point I needed information on the operation of a component of a database subsystem. In particular, I was interested in the operation of a modified B-tree algorithm. B-trees are a popular way of organizing keys used to access databases. As part of their operation, B-trees require occasional reorganization (shuffling). The frequency and duration of shuffles is critical to system performance, particularly when multiple users share a common database.

Although the programmer implementing the database code could provide very precise execution times for all of his routines, neither he nor anyone else could even hypothesize a statistical distribution for the frequency of their invocation. In the absence of this information, I had to hypothesize a distribution. By changing the parameters of the hypothesized distribution, I could get rough estimates of the extent to which system performance was sensitive to database algorithms.

In general, it has been my experience that models are far more often written with too much detail than they are written with too little. Models are written with too much detail for at least two reasons. First, as in this story, the only information available to the modeler is often very detailed. Second, modelers often think that details are inherently good; i.e., the more details that can be incorporated into a model, the more accurate the results will be. This belief arises from confusion between precision and accuracy. If I say that I put in an average work day of 10 hours, 23 minutes, and 1.793 seconds, I am making a very precise statement. If, in reality, I only work an 8-hour day, my statement is very inaccurate.

Accuracy of characterization is almost always more important than precision in a modeling project. When details abound and abstractions are hard to come by, take it as a giant red flag. Creative sparks occur rarely with respect to issues of detail versus abstraction; one should not expect flashes of brilliance which facilitate setting aside mountains of detail. On the other hand, if one ignores these issues, the potential for being buried by details looms large.

2.6 Discrete, Semicontinuous, and Continuous Models

In (Henriksen 1981), two alternative models of a spark plug packing line were presented. The packing line consisted of a single, rapidly moving conveyor, and a series of packing machines along the conveyor. The rate of flow of plugs was such that several packing machines operating at full speed were required handle the flow. Each packing machine was susceptible to jamming. When a machine jammed, plugs flowed past the machine, downstream. If enough machines were broken down simultaneously, plugs flowed off the end of the conveyor into a barrel.

In the first model, individual plugs were modeled as transactions flowing through a network. As a plug reached each machine in succession, if the machine was not busy and not jammed, the plug exited the conveyor to be packed by the machine. This model was straightforward to write, but it required large amounts of computer time and memory.

A second model was developed by regarding the system as a continuous system. If one imagined the flow of plugs to be a continuous, liquid stream rather than a flow of discrete objects, the operation of a packing machine could be described as follows:

1. Wait until my incoming rate of flow is greater than zero.
2. Start packing plugs at maximum speed (if the incoming rate meets or exceeds my maximum) or at a rate equal to the incoming rate (if the incoming rate is less than my maximum).
3. Set downstream flow equal to my input rate less my processing rate.
4. Continue processing plugs until my input rate changes, or I incur a jam.
5. If a jam is incurred, (1) set downstream flow equal to my incoming rate, (2) and repair the jam (time delay).
6. Go to step 1.

Missing from the above description is a mechanism for downstream propagation (with appropriate time delays) of changes in flow. By adding a surge propagation mechanism and a mechanism for integrating flow over time, a continuous model of the packing line could have been developed.

Rather, a semicontinuous model was developed. (A semicontinuous model is one which integrates rates of flow over time, but in which changes in rate are instantaneous.) In the semicontinuous model, flow changes were propagated using ordinary discrete events, and integration of flow over time was accomplished by simple algebra (rate * time = amount).

The semicontinuous model was more difficult to write than the straightforward discrete event model, but its performance was distinctly superior. This example is similar to that of Section 2.3, in that the creative spark occurred as a result of taking a more distant view of the problem. From a distance, what was initially regarded as a discrete problem looked more like a continuous problem. The ultimate semicontinuous formulation was selected as the appropriate modeling approach, because (1) is was fairly easy to do in a discrete event language, and (2) a truly continuous formulation (actually integrating differential equations) would have been wasteful overkill.

2.7 Variants and Invariants

In (Henriksen & Schriber 1986), alternative approaches to modeling conveyors were presented. One of the problems discussed was how to model a non-accumulating conveyor subject to frequent starting and stopping. (A non-accumulating conveyor is a conveyor which cannot slide under objects on the conveyor; i.e., once two objects are placed on the conveyor, the distance between them does not change.) The usual approach to this problem is to model the time it takes an object to get to its destination (usually the end of the conveyor) as a time advance. Unfortunately, if the conveyor is stopped, all pending time advances for objects on the conveyor must be suspended and rescheduled when the conveyor restarts. This is a form of preemptive scheduling, which is difficult to do in most simulation languages. (See (Henriksen 1987).)

An improved modeling approach was developed by noting a significant invariant relationship among objects: once placed on the conveyor, their relative positions do not change. A invariant is that at any given time, either the conveyor is empty, or there is one object (the "leader") which is the furthest downstream on the conveyor.

Using these two facts, the operation of the conveyor can be described by (1) identifying the leader, (2) modeling its movement as a time delay, (3) placing all other objects into a list structure in an order corresponding to their physical order on the conveyor, and (4) expressing the position of each object other than the leader as the time separation from its predecessor.

Using this description, modeling the operation of the conveyor over time is reduced to modeling the timing of the leader and replacing the leader when it exits the conveyor. If the conveyor is stopped and restarted, only a single object needs to be rescheduled, since the timing of all other objects with respect to the leader is invariant. This approach is called the follow-the-leader approach. It's easy to implement, and it works very well.

The creative spark in this case arose from recognition of a simple, time-invariant relationship which could be exploited. An example of a subtler time-invariant relationship is contained in the same paper referred to above.

2.8 Implicit vs. Explicit Representation

In every model some aspects of system operation are represented very explicitly, others are represented implicitly, and still others fall somewhere in between. For example, let us reconsider the machining line example of Section 2.1. For that system, an "improved" model was developed by taking an "active server, passive object" perspective. An unstated assumption of Section 2.1 was that the parts being machined were indistinguishable; i.e., the

machines didn't have to know anything about a given part to machine it. Let us suppose that this technique had been applied extensively in a model of a full-scale, real-world assembly line. Let us further suppose that after a complete model had been developed, someone came along and said "I'd like to extend the model to handle multiple part types, where each part type has unique machining characteristics."

What we have here is an issue of implicit versus explicit representation. Our modeling approach is in fact a degenerate case of implicit representation. Because all parts were identical, no representation of part types was required at all. (You can't get any more implicit than that.) The requirement to introduce explicit representation of part types may leave us in big trouble. If we're lucky, we may be able to wiggle off the hook. For example, if part types follow some random distribution, we may be able to model part type-specific actions by randomly sampling from a part type distribution. This works well if a part type must be generated at a single, independent point. However, if the part type must be carried along with the part as it flows through the system, we're still in trouble.

In (Henriksen 1984), some techniques for extending the "active server, passive object" approach are given. In an object-oriented language, these techniques would be easy to implement. In any case, the unplanned, after-the-fact imposition of the requirement for explicit representation remains a problem.

Thus the creative spark of Section 2.1 may become a downstream dud. In developing and utilizing modeling approaches, one must not get so carried away with what is being said as to lose sight of what is left unsaid.

2.9 Synthetic vs. Real Processes

Realism is a goal universally sought by modelers. This sometimes leads to the feeling that every component of a model must be the analog of some real component in the real system. As a counterexample, consider the following approach to modeling a one-line, single-server queueing system. Among the many ways this system can be modeled is as two cooperating processes, an arrivals process and a service process. The arrivals process generates arrivals into the system and places them into a queue for the server, as follows:

1. Wait for a randomly sampled interarrival time.
2. Create a customer.
3. Place the customer in the queue for the server.
4. Go to step 1.

The service process executes the following behavior pattern for as long as the system is in operation:

1. Wait for the queue to become non-empty.
2. Remove the first customer from the queue.
3. Provide service to the current customer for a randomly sampled time duration.
4. Go to step 1.

It is interesting to contrast the two processes outlined above. The service process is such a straightforward representation of operation of the server that we can easily mentally substitute the process for that which it represents. We find ourselves thinking that the process is the server. By contrast, the arrivals process has no analog in the real system; it is a purely synthetic process.

Synthetic processes can be used as a convenient repository for model logic that would otherwise be spread throughout a model. Suppose, for example, that we were building a hardware/software model of a personal computer. Furthermore, assume that at some point well into the modeling process, we decided that it would be very interesting to know what percentage of the time the CPU and the hard disk were simultaneously busy. (Productivity is improved to the extent that disk and computation can be overlapped.) The following synthetic process could easily accomplish the required statistics collection:

1. Wait until the CPU is busy.
2. Wait until the disk is busy.
3. If the CPU is no longer busy, go to step 1.
4. Record the onset of simultaneous activity.
5. Wait until the CPU or the disk is no longer busy.
6. Record the duration of the interval of simultaneous activity.
7. Go to step 1.

Note that implementing step 5 requires that the modeling tool be capable of recognizing a state event which is the OR of two conditions. If the modeling tool used is incapable of recognizing such state changes, the process above could create a clone of itself after step 4. The parent and its clone could look for cessation of CPU and disk activity, respectively. Whichever process first recognizes the condition it monitors must (1) record the duration of the interval, and (2) set some kind of a flag, to let its partner know that the end of simultaneous activity has already been recognized.

If the modeling tool used has excellent facilities for recognizing complex state changes, steps 1-3 above could be combined into a statement of the form "wait until the CPU and the disk are simultaneously active."

Introducing synthetic processes in a model is often a creative spark.

3. ALTERNATIVE MODELING APPROACHES IN THE CLASSROOM

The student of modeling must learn a collection of basic techniques, and (s)he must learn to consider the appropriateness of these techniques. Learning techniques is science; learning appropriateness is art. To an extent learning techniques can be accomplished by reading (as a starting point, consider the bibliography of this paper).

Polya (1973) points out that people remember far better that which they discover than that which they read or that which they are told. Hence, the teacher's challenge is to set before the student a series of challenges. These challenges must not be too easy, or there will be no element of discovery. Neither must these challenges be too difficult. Difficulty gives rise to frustration and the need for dispensing "hints." Hints are nails in the coffin of discovery.

Sections 2.1 - 2.9 presented a number of modeling tradeoffs to be considered. Well-conceived homework problems or course projects could be devised to illustrate several of these tradeoffs. For example, a course project might include the requirement to build a toy model and two real models, each from a different modeling perspective (with the particular perspective specified or left unspecified).

4. CONCLUSIONS

This paper has examined a variety of modeling perspectives, through reconsideration of a number of old examples and consideration of some new examples. Each example considered a particular dimension of modeling along which tradeoffs must be considered. Making proper tradeoffs requires a creative spark. The potential for finding that creative spark lies within each of us.

REFERENCES

Pritsker, A. Alan B. (1986). Model Evolution: A Rotary Index Table Case History. In: *Proceedings of the 1986 Winter Simulation Conference* (J. Wilson, S. Roberts, J. Henriksen, eds.). Institute of Electrical and Electronics Engineers, Washington, DC, 703-707.

Pritsker, A. Alan B. (1987). Model Evolution II: An FMS Design Problem. In: *Proceedings of the 1987 Winter Simulation Conference* (A. Thesen, H. Grant, and W. D. Kelton, eds.). Institute of Electrical and Electronics Engineers, Atlanta, Georgia, 567-574.

Pritsker, A. Alan B. (1989). A Model Purpose - Desired Results Paradigm. To be published in: *Proceedings of the 1989 Winter Simulation Conference* (E. MacNair, K. Musselman, and P. Heidelberger, eds.). Institute of Electrical and Electronics Engineers, Washington, DC.

Henriksen, James O. (1981). GPSS - Finding the Right World-View. In: *Proceedings of the 1981 Winter Simulation Conference* (T. I. Oren, C. M. Delfosse, and C. M. Shub, eds.). Institute of Electrical and Electronics Engineers, Atlanta, Georgia, 505-515.

Henriksen, James O. (1986). You Can't Beat the Clock: Studies in Problem Solving. In: *Proceedings of the 1986 Winter Simulation Conference* (J. Wilson, S. Roberts, J. Henriksen, eds.). Institute of Electrical and Electronics Engineers, Washington, DC, 713-726.

Henriksen, James O. (1987). Alternatives for Modeling of Preemptive Scheduling. In: *Proceedings of the 1987 Winter Simulation Conference* (A. Thesen, H. Grant, and W. D. Kelton, eds.). Institute of Electrical and Electronics Engineers, Atlanta, Georgia, 575-581.

Henriksen, James O. (1988). One System, Several Perspectives, and Many Models. In: *Proceedings of the 1988 Winter Simulation Conference* (M. Abrams, P. Haigh, and J. Comfort, eds.). Institute of Electrical and Electronics Engineers, San Diego, California, 352-356.

Polya, George. (1973). How to Solve It, Second Edition. Princeton University Press, Princeton, NJ.

AUTHOR'S BIOGRAPHY

JAMES O. HENRIKSEN is the president of Wolverine Software Corporation, which he founded in 1976 to develop and market GPSS/H, a state-of-the-art version of the GPSS language. Since its introduction in 1977, GPSS/H has gained wide acceptance in both industry and academia. From 1980-1985, Mr. Henriksen served as an Adjunct Professor in the Computer Science Department of the Virginia Polytechnic Institute and State University, where he taught courses in simulation and compiler construction at the university's Northern Virginia Graduate Center. Mr. Henriksen is a member of ACM, SIGSIM, SCS, the IEEE Computer Society, ORSA, and SME. A frequent contributor to the literature on simulation, Mr. Henriksen served as the Business Chairman of the 1981 Winter Simulation Conference and as the General Chairman of the 1986 Winter Simulation Conference. He presently serves as the ACM representative on the Board of Directors of the Winter Simulation Conference.

James O. Henriksen
Wolverine Software Corporation
4115 Annandale Road
Annandale, VA 22003-2500, U.S.A.
(703) 750-3910