Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

HOW SHOULD WE TEACH SIMULATION?

Ingolf Ståhl

Department of Managerial Economics
Stockholm School of Economics
Box 6501
SE-113 83 Stockholm, SWEDEN

ABSTRACT

This paper deals with the issue of how one can teach simulation in the most time-efficient way. We first distinguish between different types of student as regards their background and future needs. We next look at reasons for studying simulation at a business school. Next we compare animation oriented simulators with simulation languages. We then study a list of desirable criteria for simulation software, in particular simulation languages, that should be used in education. We finally answer the question if there is any system that fulfills all of these criteria.

1 INTRODUCTION

Simulation, in particular discrete-event simulation, is no doubt a very important tool that can be used in a very large area of applications. Simulation has proved to be a very powerful tool, not only in engineering, but also in business administration. Against this background it is surprising that most business schools and quite a few engineering schools do not give their students any substantial amount of teaching in simulation. In the opinion of many experts, simulation is far away from being as broadly used and taught, as it should rightfully be.

One reason seems to be the following "Catch 22": A widespread usage of simulation requires that many people have knowledge about simulation technology. However, there is a substantial cost of learning simulation and many potential students will set this in relation to the probability that the acquired simulation knowledge will be used in the future. If students find that simulation is not used much in practice, they believe that they are not likely to get their invesment in time of learning simulation paid back and they will then not learn simulation. There will then not be so many simulation experts to expand the usage of simulation.

It is therefore very important to cut down the cost, in particular in terms of time, of learning simulation. Student time is an increasingly scarce resource in a very crowded curriculum. Often only a handful of hours can be spent on simulation. How one can teach simulation in a time-efficient manner is hence the main topic of this paper.

2 MAIN DETERMINING FACTORS

When starting to analyze how simulation education can become more effective, it is important to take the following three factors into account:

1. The knowledge background, in particular with regard to programming and statistics. One must here distinguish between the following three main groups of university students that appear to be the main target groups of simulation courses:

    a. Computer science majors, who have a strong knowledge of programming and a fairly good background in statistics.
    b. Other types of engineering students, e.g. of production or transportation (logistics, material handling etc.). They have probably taken a programming course earlier, but are not experienced in programming. They have a reasonably good knowledge of statistics. With regard to the specificity of their simulation usage to be discussed below, I shall distinguish between two groups: "production students", focussing on manufacturing planning, in particular in engineering shops, like in automobile production, and "logistics students", aiming at a broader, more general, usage of simulation, in logistics, transport planning, supply chain management, inventory planning etc. There are of course other types of engineering students studying simulation, in areas like communication networks, but I think the distinction above will be enough for our later discussion of software.

   c. Business students, who generally have no knowledge of computer programming and are also fairly weak in statistics. (This is my experience from having taught 5000 business students in Sweden and the U.S.)

2. The future envisaged use of simulation by the students. Are they likely to work many months in the future doing simulation or will they rather be intelligent buyers of simulation services, at most doing some rapid, "quick and dirty", simulation prototyping?
3. The total teaching time that can be spent on simulation: Is it months or just a handful of hours?

There is a correlation between points 1 and 2. For business students, but also many logistics students, the focus will be on learning to make a rough model, a rapid prototype, to be the basis for discussions with the simulation specialist. The knowledge of simulation is also important for making reasonably realistic time assessments of the work of the simulation specialist. The focus is on creating informed buyers of simulation services. Knowledge of some rapid simulation prototyping is also important for being able to "sell" the idea of making a larger simulation project to top-management. The computer science major will on the other hand in the future be the specialist from whom the other parties mentioned above would buy the simulation modeling efforts.

There is probably also a strong correlation between points 1 and 3. At least when it comes to the teaching of simulation in more basic years, like in compulsory courses, or at least courses expected to be taken by a large part of all students, the time that can be spent by computer science majors is most likely considerably longer than the time that can be spent by business students and also by most types of engineering students. Simulation will for most business students most likely be part of a much more general course, including e.g. simulation in spreadsheets with @Risk or Crystal Ball, or be focused on a major project work in a corporation. Only a minor part of the course can be spent on learning a simulation language or package.

As regards business and logistics students, the time available in the curriculum can vary; sometimes only 2 - 4 classroom hours will be available, sometimes 10, sometimes a whole course of 25 - 30 hours. It is my strong belief that as long as at least four classroom hours are available, one should try to teach some basic simulation modeling. The alternative is a very broad overview, without allowing for any "hands on" experience. Most teachers with this approach believe that the learning of a simulation language or package would take too much time. If any computer is used, it is limited to the input of data into an already existing model. This "black-box" approach has several drawbacks compared to allowing the students to actually work with a simple simulation package. Only by doing some kind of simulation modeling, a student can get some idea about both the potentials and restrictions of simulation. As discussed below, students can learn to do quite interesting models within four hours.

If at least 10 classroom hours and at least 40 hours of individual work are available, I would recommend that the students get a chance to work their way through the whole simulation process as regards some concrete problem. In this way, the student can actively learn the whole process of doing simulation for a problem, from delimiting the actual problem, formulating the question to be answered by simulation, gathering data, outlining the simulation program graphically, coding the program, verifying, validating and documenting the program, running the program a sufficient number of times, doing a statistical analysis for drawing significant conclusions, and presenting the results in a form suitable for a potential user, with a focus on getting the results implemented.

If only the minimum times indicated above are available, one should allow the students to do a small simulation project on a system that they are familiar with, like "the hospital I worked in last summer". The students should make a reasonably valid simulation model of the present set-up. They should gather input data (on items like arrival and service times) from the real system and then compare the output data (e.g. on the length of waiting lines) from the tentative model with this real data. Finally, the students should provide and test a suggestion for an improvement of the system.

If more time is available, like in a full course of 25 - 30 classroom hours, the ambition of the project can be raised. I have in such courses had good experience of students in groups of two or three doing project work in different Swedish corporation, for example in banking, telecommunications and retailing. Quite a few projects have dealt with "sales support simulation models" where the simulation model is run on a laptop and the program is run interactively with a client, regarding e.g. the optimal configuration of a corporate telephone exchange system. Many of the project programs have had continued use in the corporations.

3 WHY SIMULATION AT A BUSINESS SCHOOL?

Before starting a discussion about what kind of software is suitable in the educational process, I must first answer a question that I am often asked, especially by people from other disciplines: Why do you teach discrete-event simulation at a business school? Why are you not content by just simulating financial flows in a spreadsheet? Discrete-event simulation, implying dynamic stochastic simulation, is of great importance for the following four reasons:

1. *Replacement of other areas in Management Science:* Many teachers, including myself, have

had simulation replace, or at least complement, other Management Science methods, such as queuing theory, inventory theory and PERT/CPM. The students have appreciated this, since this has implied a greater focus on solving problems, and fewer methods to be learnt (and forgotten).

2. *Importance of physical flows and of production economics:* At many business schools there has in the opinion of many leaders of industry been too great an emphasis on the financial aspects of the firm. The students have lacked an understanding of the physical flows that constitute the reality behind these financial flows and of the importance of manufacturing and the economics of production. Simulation is one way of giving business students an introductory understanding of some of the problems in the areas of production economics, material handling, inventory management, etc.
3. *Importance of the relationship between physical flows and financial flows:* Closely connected to this interest in production economics is a desire to demonstrate the connection between physical and financial flows in a company. With this I want to stress that we are here interested not only in manufacturing operations as such, but also in some kind of modeling that allows the student to see the **connection** between the physical activities in the firm and the consequential financial flows. For this kind of simulation, a general-purpose simulation system is of greater interest than a system focused entirely on manufacturing.
4. *Importance of stochastic dynamic simulation for financial planning:* Uncertainty is the core of financial theory. We can just think of how we want to answer the following questions: How much will we sell next year: 100,000 units for certain or 80,000 - 120,000 units? When will this customer pay: Within 30 days for certain or with 80 percent probability within 60 days? How many DM will buy a dollar a year from now: 2.00 for certain or between 1.60 and 2.40? In all cases, the last answer, indicating uncertainty, seems more reasonable.

In fact, if all future payments could be forecast with certainty, all corporate debt would be as safe as government bonds. There would then be no need for different types of financial instruments, such as convertibles and options, and hence no need for financial theory. Against this background, it seems strange that most simulation of the future financial position of a corporation, e.g. cash forecasts, is done using deterministic simulation, without any uncertainty, by ordinary spreadsheets. Instead most financial simulation should be stochastic.

The need for *dynamic* simulation, allowing us to follow each major payment, regardless of when it takes place, can be illustrated by two graphs of a cash forecast of a small corporation, where Figure 1 presents the cash position only at the end of the month.

These diagrams have been produced by a simple GPSS simulation program for cash forecasting, presented as program 50 on `<webgpss.hk-r.se>` and in Ståhl (1996). The program deals with an importer that buys and sells certain machines. It pays the foreign producer in cash directly for each unit, but provides the customers with credit. Orders arrive according to an exponential distribution, while customers' payment times vary according to an Erlang distribution. Our students can write this type of programs after about seven hours of study.

We see that the two graphs give completely different impressions. From Figure 1 it appears that there would be enough cash for the corporation and hence not any liquidity problems. The financial problems, with negative cash a great many times in the future, are clearly seen in Figure 2, where we can follow payment by payment. In Figure 1 these problems are not perceived at all, since it by chance happens that there is a cash surplus at each time-point that we regard as the end of the month. This clearly illustrates the need for a dynamic, discrete-events approach to corporate financial planning. As discussed in Ståhl (1993), the need for this type of dynamic simulation for cash flow forecasts is especially large, when a couple of hundred large payments constitutes more than half of the payments of the company. This is true for many smaller corporations in areas such as mechanical engineering and construction.

The question then arises why this has to be done in a language for discrete-event simulation and not in a spreadsheet, which business students are used to. The root of the problem is that while a simulation language like GPSS is "forward directed", a spreadsheet is "backward directed". In a simulation system we can schedule the payment of a sales transaction made on January 3 to come

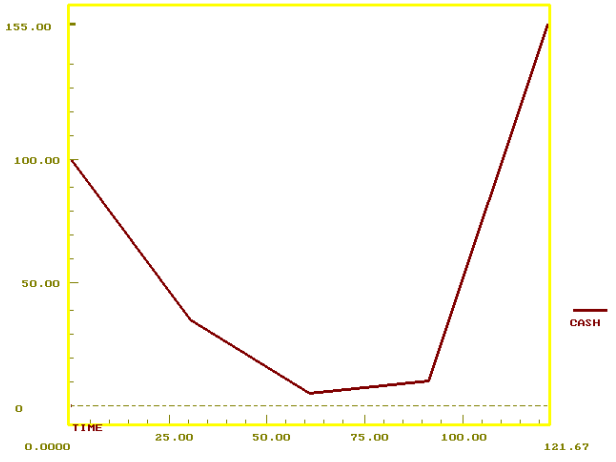

Figure 1: Static Cash Graph

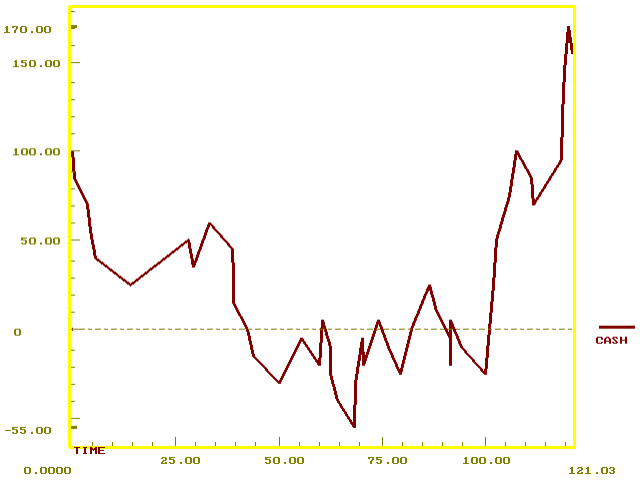

Figure 2: Dynamic Cash Graph

after a sampled time of e.g. 57 days, i.e. on March 1. In a spreadsheet we have to write in the cell, denoting the day of repayment, from which **earlier** day it shall take the sales transaction that on this later day leads to a payment. The correct scheduling of every payment for sales of random amounts when payment time is also a random variable is then impossible without using complicated macros. If we have the payment of each day point at some stochastic earlier date, it might very well happen that some sales are never paid for and some sales paid for several times. Without going into the details, which are presented in Ståhl (1993), it can be mentioned that for the simple case when all sales on a specific day are supposed to be paid on the same future day, but payments possibly can be delayed for up to a year and we want to follow payments day-by-day, simulation in a spreadsheet requires at least a 365 x 365 matrix.

4 SOFTWARE FOR TEACHING SIMULATION

The type of software to be learnt will also no doubt differ for the four types of students discussed above.

For the simulation specialist, a general purpose computer language, like C++ or Java, to be combined with special add-on packages for simulation, like e.g. Yansl or Silk, might be the best software. Closely connected to such GPLs, but with the simulation more integrated, there are text-based simulation languages, like MODSIM III, SLX or Simula. The computer science student, who most likely has a considerable experience with C++ and/or Java before starting in simulation, can hence proceed fairly rapidly with learning the details of a simulation system, which are generally hidden in the systems that are discussed below. Since I have little experience in teaching to this kind of students, I shall here leave this group for others to comment on.

For the production engineering students, the teaching is often focused on a specific special purpose system, often called an **animation oriented simulator,** like e.g. WITNESS. We shall below call this an AOS. Here one focuses on one specific type of usage. The model is built up graphically by choosing building blocks consisting of icons representing e.g. machines or conveyors and locating them on an area representing the system to be modeled. By clicking on an icon, a menu is provided by which one can input the specific characteristics of the process of the machine. The learning time of the system is limited, since the area of application is narrow. Most of these systems also come with animation, in particular when it comes to production planning. The major limitation is that the learning and modeling effort increases rapidly when one starts to get outside of the area of application for which the software is intended. For students narrowly focussed on a specific type of application, like production planning in the automobile industry, this limitation is not so serious and the teaching of an animation oriented simulator is a natural choice.

For the business students, but to some extent also for the logistic students, a more general-purpose system is often preferred. However, the general type of programming languages discussed for the computer science majors above would imply too long a learning time in relation to the value, both due to the background of the student and the more limited type of future work. For these students, a preferred choice has for many years been a simulation package in the form of a **simulation language,** below referred to as a SL, like GPSS, SIMAN and SLAM. These languages have reached a high degree of maturity and are applicable in a wide area of applications. The language elements are on a higher level of abstraction than those of the AOS. The SLs are thereby general purpose and make simulation programming easy for the user by providing automatic management of events, updating of the simulation clock and gathering of statistics. The user can concentrate on the actual model. For the visualization of results, many simulation languages provide special interfaces to animation systems. A SL requires some amount of learning, but generally much less than a GPL requires. Up to recently, simulation languages have had the disadvantage of being mainly text based, requiring the students to work with an editor for the input of the programs. For students used to working in the GUI environment of Windows this has been a problem.

While the choice of a GPL or GPL-based system is the natural starting point for computer science students and the AOS for the production planning students, the choice of a SL for the business and logistics students would need to be further discussed. Many people would claim that a modern AOS is always preferable when starting to learn simulation, since one could in a very short time get started with a very simple simulation model of a production system that would allow also for some form of simple animation.

In order to compare the SL with the AOS, it is important to give some further characteristics of the two types of system.

In the SL, the world consists of temporary entities, like customers, being served by permanent service stations, like a barber. In most AOS systems the worldview is similar, but with the permanent servers being more in focus. There is a difference between the two types of systems as regards with which type of entities you start the detailed modeling. In the AOS you usually start placing the permanent servers, while you in the SL start with the temporary entities, in particular by deciding on how and when they come into the system.

The difference of the greatest importance is, however, that in the AOS each permanent server is in principle only represented once, since it in the animation work space, representing e.g. the factory floor, must be in only one place. In the SL a permanent server can be represented in many different places, since we here follow the temporary entities and, if different entities use the same machine, this usage of the machine can take place in different parts of the program.

This difference is important when it comes to establishing what kind of models students will be able to write on their own after a certain amount of learning time. For the AOS certain simple systems are very easy to model, namely when each server only serves one type of temporary entity and only does so once. Hence a system where each product has its own machines, each visited once, is easy to model. You just place the machines in the work area and draw the paths from the entry source through the machines to the exit. For each machine you just input the processing times. If one machine, however, processes more than one product, modeling becomes considerably more complicated. You must then have rules for determining which processing time applies to which product and which path, leading from the machine, each product should take. It becomes even more complicated if a product comes to a machine several times.

This can be illustrated by the following problem, "the Boris vodka shop", where we instead of machines have humans and instead of products have customers: "At a store customers arrive at rate of 7 ± 3 minutes (assume a uniform distribution for all time data). In the store there are two people working, Boris and Naina. Customers first go to Boris, choose the good and find out how much they have to pay. This takes between 3 and 7 minutes. Next they go to Naina to pay for the goods and obtain a receipt. This also takes between 3 and 7 minutes. Finally, they return to Boris to pick up their goods after presenting the receipt, which is then stamped. This takes between 1 and 3 minutes. They then leave the store. There is one waiting line in front of Boris and one in front of Naina. Customers returning to Boris to pick up the goods have to start at the end of this line again. The program should be written so that times spent by customers in the store can be easily measured. Assume that the store is closed after eight hours and that the mentioned statistics refer to customers having left the store at closing time. Calculate by repeated runs whether there is any significant risk (e.g. happening in one out of ten cases) that a customer has to spend more than an hour in the store".

This example has been used in two experiments for comparisons between AOS and SL systems.

The first experiment was carried out in September 1996 with a class of Latvian students at the Riga Technical University, with no prior experience of simulation. First they had four full hours of an AOS (WITNESS), then four hours of a SL (micro-GPSS, a streamlined, easy-to-learn version of GPSS, see Ståhl 1996). At the end of each of these sessions they were asked to solve the Boris problem. While none of the AOS students could write a program solving this problem, all the SL students could do so.

The second experiment was carried out with a number of vendors at the Winter Simulation Conferences of 1995 and 1996. The vendors of different systems were asked to solve the Boris problem using their own system. While the vendors of SLs could solve this problem in around five minutes, all of the vendors of different AOSs required more than 30 minutes to do so. It is hence not surprising that the AOS students could not solve this problem.

Since one does not want to restrict all modeling to very simple problems of the type "each machine has its own products visiting it only once", one will in case of a limited learning time allow much more flexibility when it comes to project work if one chooses to teach a SL rather than an AOS. It should also be mentioned that also other factors imply that an AOS allows less flexibility than a SL. The building blocks of the AOS allow for a lower degree of abstraction than is possible with a SL. Presently available building blocks like machines, conveyors, etc., can only with difficulty be used for general service systems. To be suitable for the education discussed here, these simulators would require new types of building blocks (Herper and Ståhl, 1999).

For business students there will furthermore be a greater need than for the production students to handle repeated runs of the model and have an automatic statistical analysis, as well as a good graphical representation of the results of these multiple runs. In particular, when it comes to simulations involving financial streams, such as cash flows, the handling of uncertainty is very important and many runs are needed. This also poses a stronger requirement on the execution speed of the software than is the case with many production systems, for which a few runs is often sufficient. In an AOS the focus is often more on qualitative understanding and less on numbers. An animation is, also for pure time reasons, seldom run several times for the same set up of decision variables. In many cases only one run is done. It is then also more natural that fewer stochastic variables are introduced into the model. In many cases, one is in reality limiting the use of animation to problems where stochastic factors are of relative minor importance, as can e.g. be the case of factory layout problems. SLs are, in contrast, more often used for problems where random variations are important and one wants to

know within which limits the "universal" average of a result variable lies with e.g. 95 percent probability. Due to the importance of this, several SLs have superior facilities for making repeated runs and carrying out a statistical analysis of the results of these runs.

Another factor speaking for using a SL rather than an AOS is the possibility of good documentation. This is very important for the teacher when debugging, correcting and marking the student program. As regards documentation, some SLs have the advantage of providing both a compact and readable text version of the program as well as an easy-to-read block diagram presenting the logic of the model. One can start by looking at the main structure of the block diagram, before looking at the details of the program syntax. When it comes to AOS, the documentation is generally not as clearly coupled to the way in which the model was originally constructed. In some AOS systems a document is obtained that contains code that will often appear completely unfamiliar to the student who has built the model by making selections in a great number of different windows.

5 ANIMATION

The main sacrifice involved in a choice of a SL instead of an AOS is that one will get inferior animation possibilities. Some SLs allow for animation, but generally of lower quality or with more effort than in the case of an AOS. It is in this context of interest to determine what the purpose of the animation is. We can here distinguish between four main types of purpose:

1. Verification and debugging of the program, i.e. allowing the model producer to control that the model is functioning as intended and, if not, pinpointing where the error is occurring.
2. Validation of the model, implying that one uses the animation to do an "ocular check" that the simulation model seems to be a reasonable representation of the real system being modeled. This validation can be done by the modeler, the user of the simulation model or a third party, e.g. the sponsor of the simulation project.
3. Demonstration of "the message of the simulation", often implying that one with the animation demonstrates the benefits of a certain set-up or procedure that one wants to implement in the real system. An interesting example of this is given in (Savén 1995), on how one in ABB used simulation with WITNESS to persuade the labor union to accept certain new production methods.
4. Teaching of simulation principles. Animation can be used to show simulation students how different simulation constructs work and thus make the simulation package less of a black box than otherwise.

What can be regarded as the most suitable form of animation will vary depending on which of these different purposes is given priority. For the purpose of validation, in particular when someone else than the modeler does this, a fair amount of face resemblance between the animation and the actual system is required. Here the AOS is superior. For the demonstration purpose, a more fancy animation is often required. It is in this case often of importance to impress the viewers of the animation. Also here the AOS is superior. For verification and debugging, especially when done by the modeler herself, the focus is more on closeness between the animation and the simulation program and the need for "picture closeness" to the simulated system is much smaller. Here the SL can match the AOS. For the purpose of teaching simulation principles, it is also important that the animation is close to the simulation model, in particular with regard to how the model is executed step by step. There is also a need for simplicity, so that the student can clearly comprehend the animation. Also with regard to this goal a SL can give the AOS a match.

It should also be noted that for many types of problems in business, like service systems, the animation is not as interesting as for manufacturing systems. In the animation of manufacturing systems there is a constant time compression factor c, i.e. the animated time t is always cT, where T is the real time. In the animation of the systems of services, business processes, inventories, cash flows, etc., i.e. simulation systems more typical for business students, such constant time compression is seldom possible. Take a simple animation of a doctor's office, where the average service time is 10 minutes per patient and the time of the movement of the patient from the door to the doctor is 5 seconds. A system with constant time compression would either be very jumpy (e.g. if c=0.01) or very boring (e.g. if c=1). For this reason, "fancy" animation systems is often of less interest for business students. The main interest of animation here lies in the verification of the model. For this type of purpose, much simpler forms of animation, e.g. block diagram based ones, can be of greater interest.

6 CRITERIA FOR SIMULATION SOFTWARE TO BE USED IN EDUCATION

I shall below give a number of criteria for what I regard as a suitable software to be used for teaching simulation to the last two types of students discussed, the logistics and the business students. I shall, based on the discussion in Section 4 above, focus on a SL, i.e. a simulation language, although several of the criteria outlined below would also have helped me in deciding on a SL instead of an AOS. The criteria are meant to be helpful on the following three

time-levels of teaching, defined according to the approximate number of available classroom hours:

1. A 4-hour rapid introduction to simulation modeling, leading to models on the level of the Boris example.
2. An 8 - 10 hour part of a course, leading to the level of the cash flow example above and a little beyond (see e.g. the 52 program examples at `<webgpss.hk-r.se>`).
3. A 25 - 30 hour course, going slightly beyond the material under point 2 above, but involving a substantial simulation project in a corporation, aimed at being implemented, and covering all the aspects of simulation. A great amount of time should hence be left for the issues of collection and evaluation of input data, the principles of experimental design, statistical analysis of the output, aspects of implementation, etc., i.e. issues that are left out in many courses where all time is spent on the mechanics of a difficult-to-learn simulation language.

6.1 (A) Ease of Learning

A1. The learning should not presuppose any pre-knowledge of programming, except possibly some very elementary (Visual) BASIC.

A2. As we want the students to focus on modeling (and experimentation), and not on syntax detail, the language should be such that one in the course does not have to learn a new concept every time that a new and different thing shall be done. It is from a pedagogical point of view often preferable that the new aspects can be handled using already known concepts, even if the program thereby becomes slightly longer. One should very carefully restrict the number of concepts used in the language. The motto is: "Less rather than more"**.**

A3. The simulation language should be fun to learn. It should in itself provide incentives for learning. It is here important that the language provides a possibility for the students to do interesting things after only a very short period of learning. Preferably the students should already after one or two classroom hours be able to write some non-trivial simulation programs, i.e. students should be able to do simple things in a very simple fashion. One should not sacrifice the ease of introduction for the sake of having sophisticated features for the advanced user; for example one should **not** have separate modules for the model and the experiment (i.e. contrary to Zeigler 1976), since this has proved to be confusing to the novice. It is important to "encourage users to forge ahead and experiment rather than present barriers that lead to discouragement" (Banks 1995). Such a positive aspect of learning will, according to my experience, give the simulation course, and its teacher, favorable student ratings.

A4. One should furthermore restrict unnecessary details, e.g. avoid commas that are not absolutely essential.

A5. When students frequently make the same mistake, one must always consider the alternative of changing the language instead of forcing them to learn strange features which, for example, might depend on hardware limitations of computers in the 60's or on pure mistakes made by early developers. The language should not be bound to compatibility with earlier versions of the software. Such compatibility is of interest to old users who have already done a lot of programming in earlier versions of the language, but it is of no interest to the novice who desires to have as easy a learning process as possible with regard to the goal of being able to do a certain kind of simulation.

A6. The system must provide most necessary statistics automatically, since the novice does not know what kind of statistics is of interest and should not have to spend time in the beginning of the simulation course on learning different print commands.

A7. The language should be such that it can be completely covered in a pedagogical manner, with many examples etc., in a book of reasonable size (a maximum of 400 pages) and hence with a moderate price. The student should not have any need for an unpedagogic manual in order to find features not covered in the textbook or in class. According to my experience, many students have run into great difficulties in their project work when they have attempted to use features that are not covered in the textbook and in class. It is therefore of utmost importance that the textbook covers every aspect of the language.

A8. The simulation language should facilitate the teaching in computer labs as well as self-studies in front of the computer, instead of being mainly aimed at having lectures in ordinary classrooms or textbook studies as the prime teaching mode. In this connection it is important that the system allows several programs to be run in a stream, with both program listing and different types of output presented one screen at a time, without the student having to leave the simulation system.

A9. To facilitate learning, in particular self-studies, the system itself must be supplemented with program examples, tutorial lessons and help pages.

A10. When being projected on the screen by a LCD projector, e.g. in a PC lab, it is important that all important aspects on the projected screen picture is readable by the students. This, in turn, implies that

one must avoid having a lot of small details on the computer screen picture.

A11. The language should make it easy to learn to use different kind of functions, not the least to allow for the easy definition of an empirical random distribution by a number of value pairs. For discrete random functions, in each such pair, one value should be the function value and the other value just the number of observations of this function value. The system should then translate this into a cumulative function of the traditional type, something which students appear to find very difficult to learn to do.

A12. In order that interesting modeling, e.g. on the effect of uncertainty on queuing behavior, should be possible already for the novice, e.g. within a few class room hours, it should be very easy to learn to do some simple modeling of different degrees of uncertainty. Is should also be easy to stop the simulation both after a certain time or a certain number of customers or a combination of these factors.

6.2 (B) Ease of Input

B1. It should be very easy to input the program. The main form of input should be in form of a Graphical Users Interface, where one from a menu of symbols can choose the (building) blocks of the program. One should, whenever this is a reasonably efficient method, work using a mouse. One should also allow for "short cuts" using keystrokes when this can be more efficient.

B2. The choice of symbols should be done using either a "drag-and-drop" or a "point-and-click" method. In drag-and-drop, one first clicks on the symbol that one wants to move to the block diagram. One then drags this symbol to the desired place in the block diagram. At this place, one clicks again to release the symbol. In point-and-click, one will just click on a symbol and it will then immediately appear in the block diagram at the place desired by the user. An "insert marker", usually determined automatically, denotes the location where the next block is to be placed.

The "point-and-click" approach appears to be the best one when a block can be placed in only a limited number of possible positions. Furthermore, drag-and-drop is unnecessarily time-consuming. For each symbol to be input, one has to move back and forth between the symbol menu and the block diagram part of the screen. This time loss is large, if one follows what we consider the preferable method when modeling a simulation model in block symbols, namely a "top-down" approach. This implies that we first draw the general structure of the model, relying only on the block symbols. First, when the full structure has been drawn, it is time to give the values to the operands of the blocks. With this approach, one can with point-and-clock in most cases outline the whole model without the cursor leaving the symbol menu.

B3. The number of symbols in the symbol menu should be strictly limited so that the student can directly find a block symbol without having to do any scrolling.

B4. For inputting the operands of a block, one should be able to click on an individual block in the block diagram to open a dialog for inputting the operands of this block. In order to diminish the need of a manual, this dialog should reveal the syntax of the block operands. It should also be possible to write the values of the operand directly into the block. After having used a block several times, many students are likely to prefer this faster direct input.

B5. It should also be possible to input the program as text, by using a simple editor of the Notepad type in a text edit window of the GUI. This editor should translate a block diagram to text, and *vice versa*. There are several reasons for such a text editor. Advanced students might find it faster to input the whole program directly in text format. All students will save time by using the text editor for smaller changes in a program originally built in the GUI. In text format, the language must have a **completely** free format so that students need not worry about starting certain words in a certain column. No distinction should be made between upper and lower case letters.

B6. It is important that the length of the program does not become unnecessarily long. A small block diagram and a short program are generally preferable. One should, for example, not have to define the capacity of servers that can serve only one transaction of a time. This saving might lead to around 7 percent shorter programs (Ståhl 1993b).

6.3 (C) Ease of Reading Output

C1. It must be easy to read the output. The system should not provide a lot of advanced output that the novice does not know how to read and would find confusing.

C2. It should also be easy to read the extended program listing provided by the system, with an automatic line-up of operators and operands, so that a neat and easy-to-read program listing is obtained regardless of the appearance of the original code. A clear, readable and compact program listing, with short comments, is essential for making it easy for the teacher to correct and mark the student programs.

C3. One should also be able to complement the program listing with the block diagram, which should also be directly obtainable from the program in text format. Block diagrams also make it easy for students to study, discuss and document the logic of a program. Block symbols should be clearly distinguishable from each other, but block types that are related to each other should be similar, e.g. "mirror pictures", to facilitate the understanding of the program logic. Because of the importance of block diagrams, it is essential that every block type has a corresponding block symbol.

C4. The output should contain graphs and histograms that are clear and easy to understand.

C5. A simple form of animation, facilitating program verification as well as the understanding of how the simulation program works, is essential. This animation can for these short student programs be limited to post-processing animation, allowing the student to see symbols for various transaction types, such as customers, move through the block diagram.

6.4 (D) Ease of Doing Replications and Experiments

D1. Since it is very important that the students understand that the simulation programs should be run several times with different random streams, it is essential that it is very easy to make replications of the runs by just one command, easily available in the GUI.

D2. It is also desirable that the simulation system can automatically carry out a statistical analysis of these repeated runs, e.g. to calculate, using Student's *t*-distribution, the limits within which the universal average lies with e.g. 95 percent probability.

D3. It is also desirable to have some form of very simple optimization, even if it is only done in one dimension for a finite number of alternatives.

6.5 (E) Safe Programming

E1. Closely related to ease of learning, but also to ease of use, is the principle of **safe programming.** We want to minimize the risk of logical errors, i.e. that the program produces unwanted and erroneous output. In this way a great amount of student time spent on debugging can be saved. If the simulation language is made as safe as possible with regard to logical errors, this also reduces the need for an extensive debugging system which, in turn, requires a lot of student time to learn. To secure safe programming, we want to stress the "Lead us not into temptation" principle, implying that the simulation language should not be excessively permissive, allowing constructions that with a significant probability lead to logical errors. It is better that an unsuitable construction leads to a syntax error message and execution stops right away than have it lead to a difficult-to-find logical error.

E2. Closely connected with the idea of E1 is the aim that students should not run into surprises and unexpected logical errors due to not having learnt the full language. It is of special importance that the language does not have any reserved words, in particular reserved words that lead to strange logical errors. (We must avoid problems such as that SEIZE XID1 and SEIZE XJD1 in GPSS/H lead to completely different results. The student who does not know that XID1 is a reserved word with special meaning in GPSS/H can make a serious logical error.)

E3. It is also important that the simulation language has an extensive error trapping system with as clear error codes as possible. The best way to develop such a system is to have all students report on errors with no, or an unclear, error message.

E4. Even if points E1 - E3 are fulfilled, the language must have some simple, very easy-to-learn, system for debugging and program verification, e.g. in the form of the block based animation mentioned under C5, where one, moving forwards in time, event by event, can see which transactions move, where and when, in the system. One can also have a simple text based tracing system.

E5. It is important that the simulation system itself is thoroughly debugged and that the internal mechanisms of the system can be subject to scientific scrutiny (like in Schriber and Brunner 1998).

6.6 (F) Efficiency

Although efficiency is not of primary importance, execution times must not become excessively long so that students get discouraged. Execution time is important not the least to encourage sufficiently many replications from a statistical point of view.

6.7 (G) Availability

G1. It is desirable that the simulation language is available on many computers and that it works in the same way on these computers so that programs developed on one computer also runs on other kind of computers. Because of the wide availability of the Macintosh and Linux, it is important to also have versions for these systems.

G2. It is highly desirable that a system to be used in education is available at a very low cost, so that students can afford to buy their own copy of the

software. Of course, availability of some version free of charge is ideal.

G3. It is also desirable that the educational simulation software is available over the Web for the following three reasons:

1. The students can always be assured of using the latest version of the software. The school need not worry about constantly updating the software.
2. The students can after leaving university be sure of getting access to the software wherever they are later going to work. The future employer might not allow the software to be loaded on the hard disk of a computer on the company's network.
3. In many cases, a student or a teacher might want to have a first look at a software without having to go to the risk and troubles connected with downloading it.

G4. It is desirable that there are many textbooks and many program examples for the system.

6.8 (H) Advancement Potential

It is sometimes desirable that the student, after having worked some time with the educational system, can very rapidly move on to some similar system used more widely.

7 DOES SUCH A SYSTEM EXIST?

We have above given a list of desirable features of a system for teaching simulation to students of business and logistics. It is quite a long list. The reader probably wonders if there is any system that fulfils all of these features. The answer is that there is at least one such system and that it is available free on the Web, namely WebGPSS at the site <webgpss.hk-r.se>. Virtually the same system is available as a stand-alone version in Windows (WinGPSS) and systems for Linux and the Mac (LinGPSS and MacGPSS) are in the pipeline. They are all based on the same simulation engine, micro-GPSS, available for many years on DOS and several other systems (Ståhl 1990). It should be mentioned that the animation system mentioned under C5 and presented in Ståhl (2000) is at present only available on the PC, but its transfer to the Web is under way.

Micro-GPSS is a streamlined, easy-to-learn version of GPSS (the General Purpose Simulation System), which, at least five years ago, was still the most widely used simulation software (McHaney 1996). Micro-GPSS is based on the feedback from teaching GPSS for more than 20 years to more than 5000 students. In the process many complicated and redundant syntax features have been eliminated. Thus micro-GPSS has only 22 block types, compared to the 70+ block types of other GPSS versions. Thanks to these simplifications, we have now in ten hours been able to cover the same material that required 22 hours when using traditional GPSS. Yet micro-GPSS is almost as powerful. We have been able to rewrite 99 percent of the programs in leading GPSS textbooks with virtually the same amount of code. For example, for the 29 programs in Schriber's "red book" from 1974 the average number of blocks used is virtually the same (18.6 in Standard GPSS and 18.8 in micro-GPSS).

In order to give just one idea of what micro-GPSS looks like, I shall present the program that solves the Boris problem presented in Section 4.

```
simulate   10
qtable     store,0,10,7
generate   7,3
arrive     store
seize      boris
advance    5,2
release    boris
seize      naina
advance    5,2
release    naina
seize      boris
advance    2,1
release    boris
depart     store
terminate
generate   480
terminate  1
start      1
end
```

Figure 3: Micro-GPSS Program for Boris Vodka Shop

In order to give an idea of the WebGPSS GUI, I present it for the case of the famous Joe's barbershop.

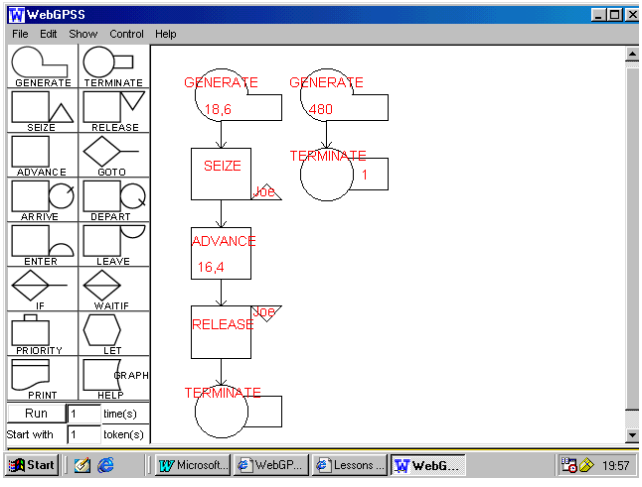

Figure 4: WebGPSS GUI with Joe's Barbershop

It should finally be mentioned that every WebGPSS program can be translated into a GPSS/H program. WebGPSS can hence provide a GUI for producing

GPSS/H models. Thus one can e.g. in the case of very long repeated runs utilize the superior speed of GPSS/H.

The whole of micro-GPSS is presented in Ståhl (1990). The easiest way to start learning GPSS is to turn to the site <webgpss.hk-r.se>. Here GPSS is supplemented with 52 program examples, a score of tutorial lessons and a large set of help pages.

REFERENCES

Banks, J. 1995. Semantics of simulation software. *OR/MS Today,* December 1995, pp. 38 - 40.

Herper, H. and I. Ståhl. 1999. Micro-GPSS on the Web and for Windows: A tool for introduction to simulation in high schools. In *Proceedings of the 1999 Winter Simulation Conference,* eds. P. Farrington, H. Nembhard, D. Sturrock and G. Evans. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

McHaney, R. 1996. *Simulation project success and failure: Some survey findings*. Working paper, Dept. of Management, Kansas State University, Manhattan.

Savén, B. 1995. *Verksamhetsmodeller för beslutsstöd och lärande - En studie av diskret produktionssimulering vid Asea/ABB 1968-1993*. Linköping Studies in Science and Technology. Dissertation No. 371. Linköping.

Schriber, T. 1974. *Simulation Using GPSS*. N.Y.: Wiley.

Schriber, T. J. and D. T. Brunner. 1998. How discrete-event simulation software works. In J. Banks (ed.) *Handbook of Simulation*. New York: Wiley-Interscience.

Ståhl, I. 1990. *Introduction to Simulation with GPSS: On the PC, Macintosh and VAX*. Hemel Hempstead, U.K.: Prentice Hall International.

Ståhl, I. 1993. Discrete-event simulation for corporate financial planning. In *Proceedings of the 1993 Winter Simulation Conference,* eds. G. Evans, M. Mollaghasemi, E. Russell and W. Biles. Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Ståhl, I. 1993b. GPSS will prevail - Some reasons for the resilience of the GPSS simulation ideas. In *GPSS-Users' Group Europe -Gruendungsveranstaltung*. Magdeburg: ASIM.

Ståhl, I. 1996. *Simulation Made Simple with micro-GPSS: A Short Tutorial with Eight Lessons*. Stockholm: Stockholm School of Economics.

Ståhl, I. 2000. Automatic animation with a block based simulation language. In T. Schulze, P. Lorenz und V. Hinz (Hrsg) *Simulation und Visualisierung 2000*. Erlangen: SCS - ASIM.

Zeigler, B.P. 1976. *Theory of Modeling and Simulation*. New York: Wiley.

AUTHOR BIOGRAPHY

INGOLF STÅHL is a Professor at the Stockholm School of Economics, Stockholm, and has a chair in Computer Based Applications of Economic Theory. He was visiting Professor, Hofstra University, N.Y., 1983-1985 and leader of a research project on inter-active simulation at the International Institute for Applied Systems Analysis, Vienna, 1979-1982. He has taught GPSS for twenty years at universities and colleges in Sweden and the USA. Based on this experience, he has led the development of the micro-GPSS and WebGPSS systems. He is also consultant in simulation to Swedish banks and industry. His email address is <ingolf.stahl@hhs.se> and the web address for WebGPSS is <webgpss.hk-r.se>.