Proceedings of the 1999 Winter Simulation Conference
P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds.

MICRO-GPSS ON THE WEB AND FOR WINDOWS: A TOOL FOR INTRODUCTION TO SIMULATION IN HIGH SCHOOLS

Henry Herper

Institute for Simulation and Graphics
Otto-von-Guericke University
D-39016 Magdeburg, GERMANY

Ingolf Ståhl

Department of Managerial Economics
Stockholm School of Economics
S-11383 Stockholm, SWEDEN

ABSTRACT

There is a demand in European high schools for learning discrete-event simulation. The best system for this appears to be micro-GPSS. Up to now, micro-GPSS has lacked a Graphical User Interface for the build up of models. In this paper, two projects for such a GUI system are presented: WebGPSS, where the GUI is developed as a Java-Applet and the system runs over the Web, and WinGPSS, developed in Delphi to allow for a complete interface with other Window programs.

1 INTRODUCTION

In high school education, models are playing an increasingly important role, not the least in Continental Europe. (Here the last year of high school, with students graduating at the age of 19, is more like the freshman year at US colleges.) Earlier, models have mainly been connected with the sciences, in particular physics. In the last decade, computer-based models have found their way into high schools, sometimes in the form of "Virtual worlds". Students have, for example, with computer models been able to construct models of their school environment and discuss improvements of it. Teaching modeling also incorporates a critical attitude to the limitations of modeling, e.g. in what respects a model will differ from the real system. The basic methodology of modeling has thus spread from the traditional sciences, like physics, to computer science or "informatics". Here, simulation is playing an increasingly important role.

There are mainly three classes of simulation models used in the teaching of informatics in Central Europe.

1. Functional models which represent technical systems with various components. This can e.g. be models of computers, networks or robots. The purpose is to show the process of the technical system. These models are often similar to the earlier mentioned models in physics.
2. Models of biologic and environmental systems as well as general models of development of society or the world. These models are mainly implemented as continuous models using a System Dynamics approach. The models require that the students have some knowledge of mathematics. Since the systems studied are not meant to be changed, the models have a forecasting character. This class of models at present constitutes the focus in the modeling/simulation area of high school informatics in some European countries.
3. Discrete simulation models. In practice, such models of e.g. production, transport and health service systems play a very important role. Although the development of such complex models is out of the reach for schools, there are many less complex systems of interest for teaching in high schools. This can be models of smaller service systems, like a small shop, a school cafeteria, etc. In the development of these models, the students can use their own personal experience of these systems. Discrete-event simulation models have, however, up to now, been very little used in schools. One reason is the lack of suitable software for this type of educational simulation. We shall proceed to discuss this issue.

2 SIMULATION SOFTWARE FOR EDUCATION

In order to determine what constitutes suitable simulation software for the high school, we must first discuss the goal of this educational simulation activity. It should be stressed that the main purpose is not to teach the use of a particular simulation tool. The focus is rather on the learning of

methods for model construction connected with simulation and of the general principles of using simulation tools (Lorenz & Schriber 1996). Furthermore, simulation can be seen as an efficient way of learning general principles of informatics.

This model building and simulation activity should not be limited to only program implementation (e.g. coding) and running of the simulation experiments. It should include all phases of a simulation study. A description of the different steps in such an education-centered study of a service system is given in Herper & Ståhl 1997. One starts with an abstract model before proceeding to implement the model in the form of a program on the computer. For this step, a software tool is required which will allow the student to build on the modeling skills just acquired. In order to determine what software is suitable in this situation, we first look at the main types of tools generally available for this type of simulation.

2.1 Classification of Simulation Tools

The first, and oldest, class of simulation tools consists of the **General Purpose Languages**, like FORTRAN, C++, VB or Pascal. They allow for the highest degree of universality and abstraction. In practice, GPLs are still used to a fairly large extent for simulation, in particular in non-ordinary applications (see table 1 below). When it comes to simulation education in high schools, however, GPLs do not seem suitable, since already for small problems there is a high demand on programming experience and knowledge of simulation mechanics. One can, however, by showing the program of a simple one-server, one-type-of-customer, system in a GPL make the students happy that they do not have to program larger systems in a GPL, but that they can use other tools.

The second class of simulation tools is the **simulation languages**. They have reached a high degree of maturity and are applicable in a wide area of applications. The language elements are, like in the GPLs, on a high level of abstraction. They 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 simulation language requires some amount of learning, but generally much less than a GPL requires. For the use in high schools, the simulation language has the advantage that students, without having to bother about the internal mechanism of the simulation system, can develop short simulation programs for small and medium size service systems.

Although software licenses for full scale versions of the simulation languages in general are much too expensive for high schools, during recent years student versions of several simulations languages, with prices of $ 20 - $ 40, have appeared with enough power to do the modeling of the smaller service systems mentioned above (e.g. by allowing 100 - 200 blocks).

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.

The third class of simulation tools consists of **animation oriented simulators**. 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. These building blocks allow for a lower degree of abstraction than is possible with a simulation language. 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 blocks (Herper & Ståhl 1997).

Full versions of these simulators are prohibitively expensive for schools. Some vendors of animators provide student versions, with a limited size of the model. In comparison to student versions of some simulation languages, the size of the models allowed is much smaller. Another disadvantage with these animators is that the execution times can become unacceptable for the repeated runs that might be necessary in order to draw statistically significant conclusions

The fact that the three mentioned types of tool in their past form have had limited usability for educational purposes has, in turn, contributed to the fact that discrete-event simulation has up to now not had any significant usage within high schools. The question then arises as how this situation can be changed.

2.2 Demands on Tools for Simulation Education

Of the three types of systems discussed in the preceding section, the second class, the simulation languages, appear to have the least draw backs, in particular since they, unlike the simulators, allow a more general simulation of the systems of interest in the student environment, without putting a very great demand on the student time or pre-knowledge, which would be the case of a GPL. However, it does not appear that any now generally commercially available simulation language is ideal for this educational purpose.

The demands that one can put on a tool to be used in the education of simulation is in several regards different from the demands that one can put on a simulation tool to be used in business for the solution of practical problems

often leading to savings of large amounts. When the commercial use of a simulation tool has big economic benefits, one can not only allow the simulation programmer to spend a long time building the simulation model, but one can also allow him/her a long time to learn the simulation tool. It is also important that the system, often quite complex, can be modeled with the precision that a commercial client requires. Commercial simulation tools hence have a very large syntax, allowing for very complex and reality-close modeling, but they require a long learning process.

As regards the simulation tools for high school education, there are, on the other hand, less complex systems and less demand on the accuracy of the simulation model. Instead, the time that can be spent learning the tool is much more restricted. Often there will be at most ten classroom hours available for learning the tool, since time must also be spent on discussing general simulation methodology. An educational simulation tool must therefore rely on much fewer language elements than a commercial simulation tool. Hence, there are good reasons for having special simulation tools for educational purposes. It is, however, desirable that these educational tools are related to commercial tools so that the time spent on the educational tool can be utilized by persons later migrating to the commercial tools.

Secondly, as mentioned, students demand a Graphical User Interface of a Windows similar type for the simulation language. This is especially important at the start of the learning process. It is possible that some more advanced students might prefer to work with a text editor for inputting, or correcting, the code of the simulation program, but for the novice a GUI will serve also as a tutorial, making it easier to learn the language syntax. The optimum is to allow both a GUI and a text editor.

The GUI should be largely self-instructing, providing information about the language syntax in different easy-to-reach menus. It should also be complemented by a system with HELP-texts both to further facilitate the understanding of the syntax and to serve as a tutorial. In this way, the system could also be used for self-studies.

It is furthermore desirable that the system allows some simple animation by which one can see how the different entities, like customers, move through the system, e.g. a cafeteria. This would facilitate both debugging, program verification and model validation. It is also desirable that there is an easy interface to external systems, both for input/output of data in the form of an interface to Excel and to a more advanced type of animation, e.g. to Proof Animation.

Furthermore, because of limited school budgets, forcing some schools to stay with several years' old computers, an educational simulation system should not make very high demand on the computer equipment, as regards processor speed, memory size and the size of monitors. The GUI text should be easy to read also on a 14-inch screen. Finally, for these schools with a limited budget, it is also important that the license fees for a software, capable of running even the largest student projects, should be at a low cost and that this license should allow students to make free copies of this software in order to work on their own computers at home.

3 MICRO-GPSS

Against the background of the requirements discussed above, we want to present what we believe is the solution that best corresponds to this demand. This is a Web- and Windows-based micro-GPSS system. While the simulation language micro-GPSS, run under DOS, has been used in teaching, in particular in Sweden, Germany and the US, for the last decade, the movement to a Web- and Windows-based GUI is a recent development.

Micro-GPSS belongs to the GPSS language family. (GPSS = General Purpose Simulation System). It is a streamlined version of GPSS. GPSS has the advantage of being the software for discrete-events simulation that has been most widely used of all and is still one of the most widely used simulation tools, as shown by table 1 (McHaney 1996).

Table 1: Software Used in Simulation Projects 1994-95

1. GPSS-type: GPSS (12.2%), SIMAN, SLAM	30.9%
2. Animators: e.g. AutoMod, ProModel	22.0%
3. GPL based: SIMSCIPT, MODSIM, Simula	10.5%
4. GPLs: e.g. FORTRAN, Pascal, PL/I	21.3%
5. Remaining 15 systems, incl. Excel and Stella	15.3%

Micro-GPSS is the result of an educational development process based on 20 years of teaching GPSS to over 5000 students. Micro-GPSS is **much** easier to learn than traditional GPSS. With micro-GPSS one has been able to cover the same material in 10 hours that required 22 hours when using traditional GPSS. The main reason for this is that we have reduced the number of blocks from the 50 - 65 of other GPSS dialects to 22, mainly by cutting down redundancies. For example, LET in micro-GPSS replaces the ASSIGN, SAVEVALUE, INITIAL, BLET and LET statements of GPSS/H. We have also simplified the syntax as well as replaced some blocks like GATE and TEST with blocks that have a name and syntax more in line with common GPLs. For the case that we go to NEXT if the simulation clock is less than 100, IF Cl<100,NEXT of micro-GPSS is easier to understand than TEST GE Cl,100,NEXT of other GPSS versions.

On the other hand, micro-GPSS is almost as powerful as traditional GPSS. 99 percent of the programs in leading GPSS textbooks have been rewritten in micro-GPSS with around the same amount of code. Micro-GPSS is intended

to be the starting point in learning simulation, making sure that sufficient time would be left for learning all other aspects of simulation methodology, like data gathering, experimental design and output analysis. In courses that devote a maximum of twenty class room hours to simulation, micro-GPSS is certain to surpass every other simulation system with regard to the power and relevance of the programs that students, with no computer training, can write at the end of the course.

Up to now, there have not been any generally available GUIs for micro-GPSS. Some attempts have been made earlier (Nywall 1992), but first in 1998 an intensive development on a GUI for micro-GPSS has started. In fact, there does not seem to be any generally available Windows-based GUI for any GPSS dialect. Other GPSS GUI attempts have been limited to the Macintosh (GPSS/VI, Ball 1992), DOS (Gramos-GPSS, Diedenhofen 1993) and OS/2 (GPSS World).

The development of a GUI for micro-GPSS now follows two, at present separate, lines: A Web-based version and a Windows-based version. We shall in the next two sections present each of these two projects separately and then make a comparison of them. The two development projects run in parallel. There are many similarities as regards the appearance and the functioning of the two GUIs, but there are at present also several differences as regards important design topics. The idea is that there should in the long run be at least so much convergence that the two GUIs could be described in the same text book without having to distinguish between them. However, in many cases it is not yet clear which is the best design. We believe this can only be established by testing the GUIs with many different high school students. We might then find that there are even better designs. At this stage, it is not reasonable to commit ourselves to one specific design, but we rather see it as beneficial that we can test two different designs.

It should be mentioned that both systems deal with the GUI only. The underlying simulation engine GPSS.EXE is the same engine as runs under DOS, although some changes have been made to it in order to run better on the Web and under Windows. This change of GPSS.EXE involves mainly the following items: 1. New interactive input. 2. Interruption of programs running too long. 3. A new simple way of running programs several times. 4. A new method for making experiments with automatic calculation of the limits of the universal average based on Student's *t*-distribution. 5. A new simple definition of functions. 6. New files for graphs and for histograms. 7. Generation of data for a "GPSS/PC" type of animation, where the movement of the transactions is displayed in the block diagram. 8. A new way of handling error codes. 9. All output and error codes will be supplied also in Swedish, German and, in the future, other languages (Ståhl 1999).

4 WEBGPSS

When the Swedish Foundation for Knowledge and Competence Development (the KK Foundation) was looking for software developed at Swedish Universities that could possibly be used also in Swedish high schools, micro-GPSS was one of the few software products chosen. A requirement for support was that micro-GPSS should be put on the Web with a very simple-to-use Graphical User Interface.

The main aim of this project is to have Swedish students use micro-GPSS in the project work of about one month that most Swedish students do during their last year at high school. The idea is here to allow the students to do a small simulation project on a system that they are familiar with, like the school's cafeteria, "Uncle's gas station", "the hospital clinic I worked last summer", etc. Our hope is that students in the simulation study can make a reasonably valid simulation model of the present set up. They should gather input data (on items like arrival patterns and service times) from the real system and then compare the output data (e.g. on length of waiting lines) from the tentative model with this real data. As a final step, the students shall then provide and test a suggestion for an improvement of the system. Experience with similar projects done by US college students who were only a year older indicates that this is a realistic aim.

This WebGPSS project is a joint venture between the Department of Managerial Economics at the Stockholm School of Economics (SSE), Flux Software Engineering, Ronneby Soft Center, the University College of Karlskrona/Ronneby (HKR) and the city of Ronneby. A basic version with the following three main components of WebGPSS came into operation in June of 1999:

1. A GUI in the form of a Java-based Applet that runs on the student's computer and will be used by the student for constructing the micro-GPSS programs. It is built in such a way that the student by constructing a number of programs of increased complexity will learn a significant part of micro-GPSS in a short time. The GUI module can to some extent be seen as a tutorial with, among other things, number of Help or Tutorial texts made available to the user in a convenient manner. This component is a joint venture between the SSE and the HKR.
2. The adjustment of the original micro-GPSS software, GPSS.EXE, to make it run on the Web. This component is the task of the SSE.
3. The system for making it possible to transfer the programs constructed with the GUI on the student's client computer to a server, then run the program using GPSS.EXE on the server

and finally transmit the simulation results back to the client computer. In this way, the student will first construct an ordinary micro-GPSS program using the GUI. This program will then be sent from the client to the server where it will be processed by the "micro-GPSS engine" GPSS.EXE. This, in turn, will provide output in the form of files that will be sent back to the client. Some of these files will be presented as text. Other files will be used by the WebGPSS Applet to provide graphs, histograms and very simple "animations" showing the movement of the transactions through the block diagram. This component, dealing mainly with the server, is the task of the HKR.

It should be noted that we have chosen to have the basic simulation engine, GPSS.EXE, run on the server, limiting the task of the Applet on the client computers to allow the users to construct the simulation programs here. Another approach is to have the whole simulation run on the clients using an Applet provided by the server. One reason for our choice of the first approach is the great advantage of speed that this alternative implies. A simulation model, in particular when run a substantial number of times, will require rapid execution, which is not possible, at least not within the next few years, due to the slow speed of execution of a Java based Applet, compared to that of a program like GPSS.EXE, written in FORTRAN and compiled for high execution efficiency. Another reason is the cost of rewriting around 20,000 lines of code, developed over a period of two decades, from FORTRAN to Java.

There are several advantages of providing this kind of educational simulation software on the Web:

1. The users can always be assured of using the **latest version** of the software. This is an advantage for both the school, which need not worry about constantly updating the software, and for the student as a later user of the software in business. This assurance of always being able to run the latest version is of particular importance for software that is in the process of rapid development. This is often the case with simulation software with a heavy emphasis on graphics and on an easy-to-use interface.
2. The students can after leaving school or college be sure of getting **access to the software** wherever they are later going to work. Even if they have a student version on a diskette, it is not sure that the future employer is going to allow the software to be loaded on a 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. First, downloading and then running any executable file is always risky from the virus point of view. Running a Java-based Applet is, however, risk free from this point of view. When downloading a program, one often has to go through a time consuming installation and later, possibly, a de-installation procedure. For PCs with a limited amount of hard disk space left, users might be reluctant to commit disk space to a demonstration version of a software for which they are uncertain about whether they are going to use it or not.
4. Getting a chance to run a program on the Web also increases the possibility of **connecting** the running of the software **with other net activities**, such as "chatting" with other users of this software.
5. In case the Web-system is implemented in Java, an additional advantage is that one can then, as discussed below, with a minor extra effort obtain a **platform independent** stand-alone software.

5 WINGPSS

The work on the Windows-based GUI for micro-GPSS has been initiated by Henry Herper. The programming of the system is being done in Delphi, with the aid of A. Krüger and H. Schliefke at the Institute for Graphics and Simulation, Otto-von-Guericke University, Magdeburg. The initiative stems from Herper's experience from teaching simulation to German teachers of informatics. Here the demand for this kind of simple-to-use Windows-based simulation system became clear.

In the preceding section, we mentioned the advantages of a Web solution. There are, however, several disadvantages of a Web-based based system compared to a stand-alone system. One is that the cost of connecting schools to a remote server on a reasonably fast line are quite high in many European countries, like Germany. Some students might also find the download time for the Applet too slow. Students might also want to have a version to work with at home or on a laptop without Internet access.

Furthermore, when it comes to saving a program, or opening an old program file, there is a substantial difference between a Web-based system and a stand-alone system. The Java-Applet is, for well-justified reasons, not

allowed to save a file on the client's hard- disk. For administrative and cost reasons, WebGPSS will not either allow programs to be saved on the server. The saving of files in WebGPSS must hence be done by copying the program from the WebGPSS Edit window, (see below), then pasting this text in e.g. Notepad and then saving it in the ordinary Notepad way. In WinGPSS the saving process is of course much simpler.

Even if we foresee that Web-GPSS, written in platform-independent Java, can be transferred to a stand-alone system fairly easily, the main interest of this is probably for other operating systems than Windows. A GUI, directly developed for Windows in Delphi, will have several benefits over a Java-based stand-alone system. There will be some advantage of speed, although probably not so large. It will furthermore be easier to provide efficient and safe interfaces to Excel and to the new Windows-based version of Proof Animation. Finally, it will be easier to provide easy-to-read block diagrams. This is because the GPSSDIA system, which provides clear and easy-to-read block diagrams based on the text of a micro-GPSS program, was developed in Turbo-Pascal (Ståhl 1993). The critical parts of this program, dealing with the intricate logic for placing the blocks on the right spot, can be incorporated without major changes into Delphi-based WinGPSS, since Delphi includes Turbo-Pascal.

6 CHOICES REGARDING BLOCKS

Both WinGPSS and WebGPSS follow the principle that one should, whenever this is a reasonably efficient method, work by using a mouse, although one should also allow for "short cuts", using keystrokes, when this can be even more efficient. There should furthermore be a main menu bar, allowing for pull down menus, with the leftmost menu dealing with File handling.

The main window will, below this main menu bar, have a menu of GPSS block symbols. In line with other systems, this menu has been placed to the left in the window. By clicking on some of these block symbols in this symbol menu, the user can construct the "core" of the simulation program, i.e. the structure of the model in the form of a block diagram that will be placed to the right of this symbol menu in the main window.

In this block diagram, the blocks should be displayed in straight columns, with the symbols below each others, as has been done in all GPSS block diagrams. While the size of the block symbols in the symbol menu is fixed, the size of these symbols in the block diagram can vary due to zooming. The size determines the number of block symbols that placed in one column that can fit into the main Web-window. In principle, we allow for variable zooming, but for easy use some standard alternatives are given. The form of the blocks is the GPSS standard (Schriber 1974), except for those of two special micro-GPSS blocks (Ståhl 1990).

As regards the number of blocks in the block menu there is a difference between the two systems. In WinGPSS the block menu contains all 22 block symbols of micro-GPSS. In WebGPSS we have in the initial version of WebGPSS included only the 16 blocks that are used in the tutorial booklet (Ståhl 1996; Herper&Ståhl 1998) that is the basis for the tutorial part of WebGPSS. The difference in blocks can be seen by comparing the work space of WebGPSS (with Swedish text) in Figure 1 with that of WinGPSS (with German text) in Figure 2. For most Swedish high schools projects, these 16 blocks seem sufficient. If we had included all 22 blocks in the block menu, students would constantly have to look at unknown blocks on the screen. Another argument for limiting the number of blocks to 16, was that even for these 16 blocks WebGPSS does not present the full syntax. Presenting all blocks in the block menu might then give the false impression that the operand menus cover the full syntax.

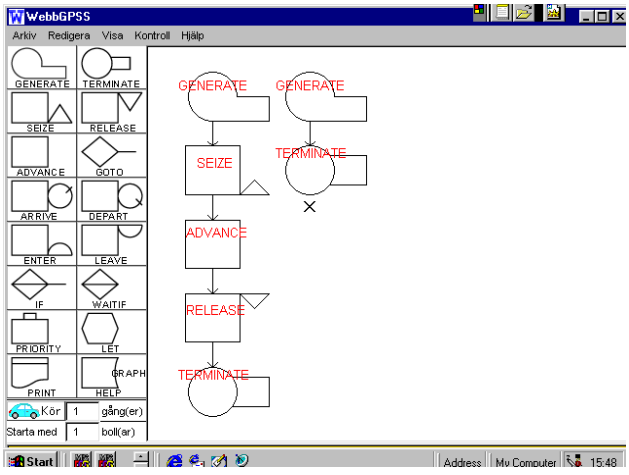

Figure 1: Block Menu and Block Diagram in WebGPSS

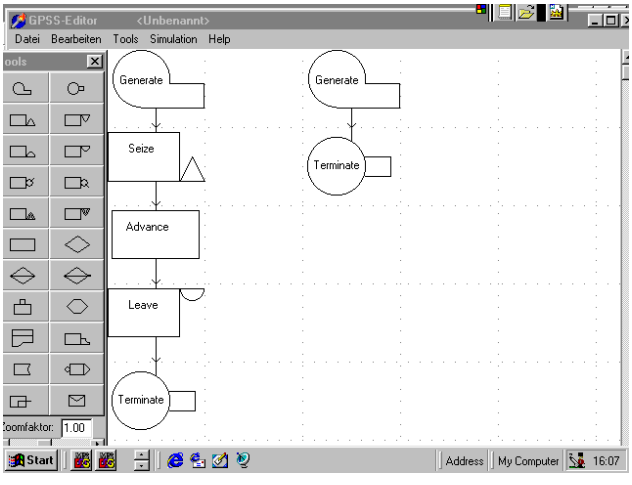

Figure 2: Block Menu and Block Diagram in WinGPSS

With only 16 block symbols, we can also use larger block symbols making it possible to write the name of the block next to the symbol.

In WinGPSS with all 22 blocks, but with smaller symbols, the block names are in the block menu only shown in component captions when moving the mouse to the block. All 22 block symbols of the block menu are encapsulated into a Toolbox, which can be moved to any location on the screen. Once a symbol is dragged from the toolbox to the block diagram workspace, the symbol is shown with its full name in the block diagram.

When it comes to the building of this block diagram, we have another difference between Win-GPSS and Web-GPSS as regards the initial strategy. WinGPSS uses a "drag and drop" approach. One will first click 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 part in the right hand part of the window. When one has reached this place, one then clicks again to "release" the symbol. WebGPSS on the other hand uses a "click only" approach. One will just click on a symbol and it will then immediately appear in the block diagram at the place desired by the user. In this case, there is an "insert marker" denoting the location where the next block is automatically going to be placed. The rules for this insert marker are quite simple. The next block will be just below the earlier block, except for a GENERATE block, which is put at the top of the next "column" to the right. If one wants another position, one will have to move the insert marker to that position.

Both systems cater to what we regard as the preferable method when modeling a GPSS model in block symbols, namely a "top-down" approach. This implies that one will first draw the general structure of the model, relying only on the block symbols. Only when the full general structure has been drawn up, it is time to give the detailed values to the operands of the different blocks. In order for this "top-down" approach to be possible, we had to decide that the input of the operands of a block should be made separate from the selection of a block. Our approach is here quite different from a system like GPSS-Edit (Nywall 1992), where one has to specify the operands of a block before one can proceed to bring another block into the block diagram. Figures 1 and 2 show a block diagram without operands for a simple single-server simulation, "Joe's barbershop".

For editing the block diagram, one can, besides the insert activity described above, delete blocks by "painting" them with the mouse and then clicking on e.g. Edit Delete. One can copy a single or set of blocks by "painting" them with the mouse and then click on Edit Copy and finally, after having moved the mouse to the desired location for the copy, click on Paste. In WebGPSS one can also, if one wants to copy all blocks of a column, i.e. a whole segment, instead of "painting" all blocks, click on a Edit Mark Segment.

For inputting the operands, one will, after preferably having constructed the outline of the block diagram, without operands, in both systems move the mouse to the right part of the screen. By next double clicking on an individual block in the block diagram, a dialogue box or small window is provided for inputting the operands of this block. In this dialogue, one will see the basic syntax of the block operands, i.e. what the different operands mean. Besides the operands, there is in every operand dialogue a field for the address or label of the block as well as a field for a comment on the block. Once the operand box is closed, the operands can be seen in the block.

In WebGPSS it is also possible to write the values of the operand directly into the block. When using a block the first couple of times, the operand menu is a valuable tool for learning the syntax and avoiding syntax errors. After having used a specific block several times, some users are likely to prefer the faster direct input.

Both systems also provide a simple editor of the Notepad type in a Text Edit window. There are several reasons for this text editor. Some more advanced students are likely to find it faster to input whole programs directly in text format. Less advanced students will save time by using the text editor with a search function for making smaller changes in a program originally built in the GUI. In WebGPSS the text editor is, as mentioned above, also used in connection with saving and opening program files.

7 CONTROL STATEMENTS

In WinGPSS all control statements will be handled from the pull down menus at the top menu. In WebGPSS all but the two will be handled in this way. By limiting the number of blocks to 16 in WebGPSS, there is in the symbol menu also room for two control symbols that are used extensively in the first lessons of the tutorial, namely START and SIMULATE (= the VW in Figure 1). The START value determines the ending of the simulation; the SIMULATE value determines the number of runs to be made. Since it is very important to have students understand the importance of making several runs, it should be very easy to do so.

Also in this case, WebGPSS will in its initial stage limit the control statements to those used in Ståhl 1996, implying that we shall in our basic system use only eight out of the 13 possible control statement types.

We shall as regards these control statements only mention how the FUNCTION control statements are handled in WebGPSS. From the top-menu under Control Function, one can reach the **FUNCTION** definition dialogue. Here one can input a new function, based on an empirical distribution, and delete or edit one of those already listed. When inputting a new function, we reach a

new dialogue in which we write the name of the function. By clicking on one of two radio-buttons, we choose whether the function is of random type or of a normal type. The random type is a new type, special to micro-GPSS, allowing for the easy definition of a discrete empirical random distribution. This new way of defining such a function, with consequences also in the program text, is a product of our work with WebGPSS.

Having chosen the type of function, random or normal, we reach another dialogue, where we input a number of value pairs defining the empirical distribution. We exemplify this with a random function as shown in Figure 3. For this type of random function, the first value of the pair is the output value, i.e. the value returned by the function, and the second value is the frequency, i.e. the number of empirical observations of this value. GPSS.EXE will translate this distribution automatically into a cumulative function of the traditional GPSS type.

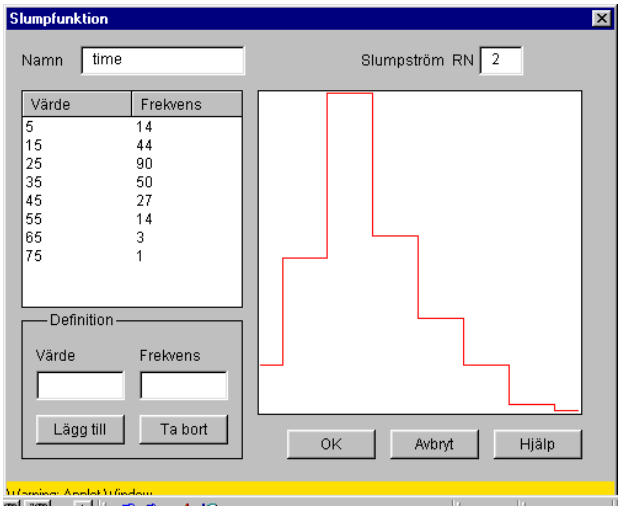

Figure 3: Definition of a Random Function in WebGPSS

The function definition function in Figure 3 (with text in Swedish) will produce a function definition, which in the micro-GPSS program text reads as follows:

```
TIME   FUNCTION  RN2
5  14
15 44
25 90
35 50
45 27
55 14
65  3
75  1
```

This is clearly much easier for students than calculating and writing the following lines in standard GPSS:

```
TIME   FUNCTION RN2,D8
0.05761,5/0.23868,15/0.60905,25/0.81481,35
0.92593,45/0.98354,55/0.99588,65/1.00000,75
```

8 OUTPUT

When it comes to output, both WebGPSS and WinGPSS use a different approach than the old text-based micro-GPSS. In "old" micro-GPSS, the output is presented in one single unit, either on the screen, 23 lines at a time, with the user moving on to a new screen by pressing a key, or in one single result file. In WebGPSS and WinGPSS, all output is placed by GPSS.EXE in a number of different files, one for each group of output that the particular program produces, e.g. program listing, station statistics, queue statistics, etc. Each of the two systems then places the data of these files into a separate window, with tabs to separate sub-windows, with one tab for each particular group of output.

We shall illustrate this with the output of a very simple barbershop program. The program is presented in Table 2 and the output in WingGPSS in Figure 4.

Table 2: Barbershop Program in Micro-GPSS

```
SIMULATE
QTABLE     SAL,0,10,20,G
GENERATE   20,10
SEIZE      SAL,Q
ADVANCE    20,18
RELEASE    SAL
TERMINATE
GENERATE   600
TERMINATE  1
START      1
END
```

In the simple micro-GPSS barbershop program in Table 2, we notice two important differences as compared to GPSS/H. The single block SEIZE SAL,Q does the same job as the three blocks QUEUE SAL, SEIZE SAL and DEPART SAL of GPSS/H. In the QTABLE statement (which does not require a label), we see an new operand G (for Graphics) at the end, which causes the histogram in figure 4 to be produced.

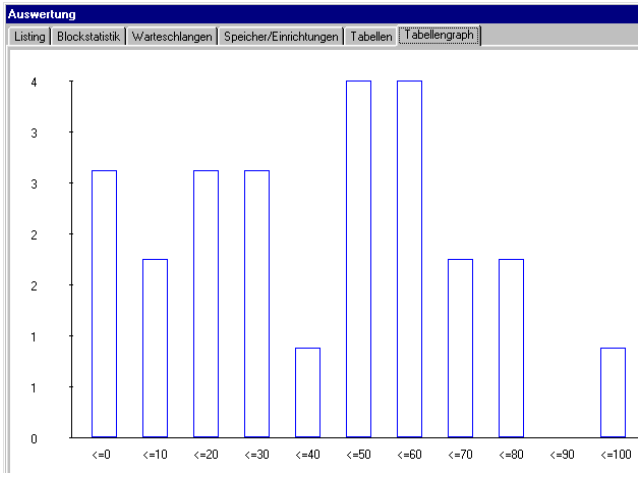

Figure 4: The Output Window in WinGPSS

In this figure we see that the program in Table 2 produces six output tabs, namely for program listing, block statistics, queue statistics, station statistics, a table over queuing as regards who waited at least 10 minutes, 20 minutes, etc, and, finally under the activated tab, the histogram of this table. It should be mentioned that the output window in WebGPSS is very similar.

9 CONCLUDING REMARKS

When the two presented GUIs for micro-GPSS, WebGPSS and WinGPSS, after tests with student groups, e.g. in Germany and Sweden, have become not only stable and error-free, but also as easy to learn and use as possible, we believe that we have a simulation software that would allow beginning students, whether in high school or college, to learn a simulation tool in such a short time that there would be ample time left for learning the usually more critical aspects of simulation methodology such as problem analysis, data gathering, experiment design, output analysis and model validation.

ACKNOWLEDGEMENTS

The design of WebGPSS and WinGPSS have been favorably influenced by F. Hall, Flux Software Engineering, Ronneby, and by A. Krüger and H. Schliefke, Otto-von-Guericke University, Magdeburg.

REFERENCES

Ball, D. 1992. GPSS/VI. In *Proceedings of the 1992 Winter Simulation Conference*, ed. J. Swain, D. Goldsman, R. Crain and J. Wilson. Arlington, VA: SCS.

Diedenhofen, H. 1993. Gramos-GPSS. In *GPSS-Users' Group Europe, Gründungs-veranstaltung*. Magdeburg: ASIM Heft Nr. 36.

Herper, H. and I. Ståhl. 1997. Diskrete Modellierung und Simulation - Methoden und Werkzeuge, für den Informatikunterricht. In *Tagungsband: 7. GI-Fachtagung Informatik und Schule - INFOS'97*, 139 -51. Berlin, Heidelberg, NY: Springer Verlag.

Lorenz, P. and T. Schriber. 1996. Teaching Introductory Simulation. In *Proceedings of the 1996 Winter Simulation Conference*, ed. J. Charnes, D. Morrice, D.Brunner & J. Swain. Coronado: SCS.

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

Nywall, J. 1992. *GPSSEDIT Manual*. Mimeo. Karlstad College, Karlstad, Sweden.

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

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

Ståhl, I. 1993. Recent Developments of micro-GPSS. In *GPSS-Users' Group Europe, Gründungs-veranstaltung*. Magdeburg: ASIM Heft Nr. 36.

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

Ståhl, I. 1999. Changes of a simulation software in order to make it run better on the Web. In *Simulation und Visualisierung '99*, ed. O. Deussen, V. Hinz and P. Lorenz. Ghent: SCS.

Ståhl, I. and H. Herper. 1998. *Einführung in die Simulation mit Micro-GPSS - Ein Grundkurs in acht Lektionen, Vorlesungsskript.* Handelshochschule Stockholm/ Universität Magdeburg.

WebGPSS can be found at the site **webbgpss.hk-r.se.** At this site there will also be links to the English version of WebGPSS, planned to be released in the fall of 1999.

AUTHOR BIOGRAPHIES

HENRY HERPER is an Assistant Professor at the Institute for Simulation and Graphics at the Otto-von-Guericke University, Magdeburg. His areas of research are the modeling of logistical and manufacturing systems and their dispositions. He is also teaching simulation to informatics teachers. He is a member of ASIM and the GPSS-Users' Group Europe.

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 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 system. He is also consultant in simulation to Swedish banks and industry.