Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

INDUSTRIAL STRENGTH SIMULATION USING GPSS/H

Robert C. Crain
Douglas S. Smith

Wolverine Software Corporation
7617 Little River Turnpike
Annandale, Virginia 22003-2603, U.S.A.

ABSTRACT

GPSS/H is a tried-and-true simulation tool whose user base, both commercial and academic, continues to grow despite the presence of so many "new" simulation technology trends. The *process-interaction* world view combines with the advanced features available in GPSS/H to make it one of the most powerful and flexible tools available, *capable of handling the largest simulation projects with ease.*

In the following sections, we provide an overview of GPSS/H and its process-interaction world view. We then discuss model-building interfaces and the advantages and pitfalls of graphical modeling environments. Advanced and newly-added features of GPSS/H Release 3 are detailed. Then we describe the *special-purpose simulator* and features of GPSS/H which make it ideal for use as a simulator engine, including a summary of the recent developments in the run-time product line.

1 INTRODUCTION

The widespread success of GPSS/H stems both from the superiority of its original design and from years of improvements and enhancements. At its introduction, GPSS/H was many years ahead of its time and opened new horizons in flexibility and ease-of-use. GPSS/H requires some programming-style effort but does so within a natural modeling framework that can be readily used by individuals without extensive programming experience.

Although many new simulation tools have been introduced over the past decade, they are often designed for a limited set of applications. In strong contrast, GPSS/H continues to be one of the most *general*, *flexible*, and *powerful* simulation environments currently available. GPSS/H is presently used worldwide for modeling manufacturing, transportation, distribution, telecommunications, hospitals, computers, and many other types of queueing systems, both simple and complex.

2 PRODUCT OVERVIEW

GPSS/H is a discrete-event simulation language. Models are conveniently developed in a text-based environment, and subsequently *compiled directly into memory* and executed. Rapid prototyping and iterative model development are encouraged by *exceptionally* fast compilation and execution.

Two primary modeling approaches, or world-views, are applied in simulation modeling. GPSS/H uses the intuitive and natural process-interaction approach. In this world view, the modeler specifies the sequence of events, separated by lapses in time, which describes the manner in which "objects" flow through a system. A GPSS/H model thus resembles the structure of a flowchart of the system being modeled. This intuitive modeling approach contributes greatly to the ease and speed with which simulation models can be built.

After the model has been built, the process representation is executed by GPSS/H, and the activities of "objects" are *automatically* controlled and monitored.

2.1 GPSS/H Process Representation

An "object" in a GPSS/H model might be a patient, a telephone call, or any other type of discrete entity. The representations of these entities in GPSS/H are called *transactions*. As the model executes, many transactions may be flowing through the model simultaneously—just as many "objects" would be moving through the real-world system. In addition, multiple transactions can execute GPSS/H model statements at the same instant in time *without any special action required of the modeler*. The execution of a process-interaction simulation model is thus akin to a multi-threaded computer program. This differs greatly from the single-threaded, sequential execution of most general-purpose programming languages.

Many simulation projects focus on the use of system resources such as people, machines, conveyors, computers, physical space, and so on. In a GPSS/H simulation model, transactions ("objects") compete for the use of these system resources: as transactions flow through the process representation, they *automatically* queue up when unable to gain control of a necessary resource. The modeler does not need to specify the transaction's waiting time or its queueing behavior. Hence, the passage of time in a GPSS/H model can be represented *implicitly*, as in the case of a part waiting for a machine to be free, as well as *explicitly*, as in the case of a part being processed by a machine.

As is the case in most real-world systems, a GPSS/H model may consist of multiple processes operating simultaneously. Furthermore, each process may in some way affect the other processes in the system. For example, two parallel manufacturing processes may converge to a single inspection point where they are competing for a single resource—the inspector. GPSS/H provides the capability for multiple parallel processes to interact with each other *automatically*. Transactions ("objects") may be sent between processes; they may control or share common resources; or they may influence the (global) operation of all processes.

3 GRAPHICAL MODELING — GOOD & BAD

Often, power and ease-of-use of a simulation environment are confused with the *model building interface* provided within the tool. The model building interface is that part of the software used to enter the model specification into the software. This may be comprised of menus and data forms, or—as in the case of GPSS/H—it may consist of text entry, or any combination of the two.

The latest *"fashion"* in simulation technology is directed at trying to model systems visually. This takes the form of placing icons on the computer screen to represent system components, and then specifying the operating characteristics of each component by moving through a series of menus and data forms. One inherent advantage of this approach is that even novices can build simple models quickly—*although not necessarily accurately*. Building models of complicated systems, however, requires more than simply placing icons on the screen. To model many processes, a limited programming environment must be provided. For example, a part routing based on a time-dependent math equation cannot be represented visually. As a result, models of complex (real world) systems created using the visual approach often require the modeler to create *substantial amounts of programming code* in addition to the visual representation.

3.1 Developing and Editing Models

Many graphical modeling tools force their users to make the model fit within a rigid framework. The advantage of such a framework is that it tends to steer even a beginning modeler through the model-building process. The disadvantage is that currently available frameworks are rarely versatile enough to *accurately* model complicated systems.

Additionally, large visual models can become very cumbersome to view, edit, and document. Large models can be comprised of many "screens" of icons, many of them with associated program code reachable only by going through multiple levels of menus. Editing—or even just browsing—these models forces the user to navigate through a labyrinth of icons, menus, click-buttons, data fields, and code segments.

3.2 What Defines an Easy-to-Use Simulation Tool?

A tool's ease-of-use cannot be evaluated on the presence or absence of a single characteristic. Whether a tool is "easy-to-use" is determined by the *combination* of general characteristics and specific features that are *frequently* used in model development. The selection of simulation software should be based on how well it is suited to the *detail* and *complexity* of the specific type of model to be developed.

The ease-of-use associated with a simulation tool can mean—among other things—easy to learn, easy to use repetitively, easy to modify models, easy to use when building simple models, or easy to use when building large, complex models. Tools that are claimed to be "easy-to-use" often fall short when modeling complex, real-world systems.

Graphical model building, often touted as a *breakthrough* in ease-of-use, springs from attempts to apply to simulation recent trends in the design of computer interfaces used primarily for word-processing, spreadsheets, database access, and the like. Although a graphical user interface is well suited for many kinds of tasks, it is not always practical for developing simulation models—especially in circumstances where programming is necessary to define the operations of complex processes.

There is a point at which a graphical environment presents its user with more barriers than advantages.

For example, proponents of graphical modeling techniques frequently claim shorter model development time, but this is primarily because users of graphical tools tend to build simpler models. As was discussed in section 3.1, creating and editing complex models with graphical tools often requires more time than creating and editing such models in a text-based environment.

4 IMPORTANT FEATURES OF GPSS/H

Several unique characteristics make Wolverine's GPSS/H an ideal choice for a general simulation environment. GPSS/H's key feature is the *conceptual flexibility* to model a wide range of different types of systems: any system that can be described as a process flow, with objects and resources acting upon each other, can be modeled. This may include people on a mass transit system, tasks in an office environment, or data flow within a computer network.

Definition flexibility is also provided within the language: complex math formulas, expressions, and constants can be used virtually anywhere in the model. To promote model readability, elements and entities may be specified by names instead of numbers. Basic simulation output data, such as queueing and service statistics, are *automatically* provided without any programming.

GPSS/H also allows *flexibility in the selection of hardware platforms:* GPSS/H runs on PCs, SUN SPARC workstations, VAX/VMS computers, and mainframes. On the PC, GPSS/H Professional runs as a true 32-bit application under DOS, Windows, OS/2, or Windows NT, providing tremendous speed as well as model size that is limited only by the computer's available memory. Running under Windows, OS/2, and Windows NT, virtual memory is used, which allows model size to exceed the amount of physical memory installed in the machine.

4.1 GPSS/H File and Screen I/O

The file and screen I/O built into GPSS/H provide a variety of ways to get data into a model and to produce custom output. GPSS/H can read directly from the keyboard or from text files, and it can write directly to the screen or to text files. The GETLIST statement and the BGETLIST block read integer, character, and double-precision floating-point data. Data files are free-format (values on each line are simply separated by blanks), and special actions may be specified for error and end-of-file conditions.

Customized output is generated using the PUTPIC statement and the BPUTPIC block. These use a very intuitive "picture" type of format specification, which follows the "what you see is what you get" convention. Special provisions are also included to allow easily formatted tabular output. Character strings can also be manipulated using built-in capabilities.

4.2 Experiment Control

The results produced by a *single run* of a simulation model can only provide *single estimates* of random variables that may be subject to wide variations. Careful experimental design, using multiple runs, is *essential* to accurately predict the behavior of the model outputs. GPSS/H provides the tools to build a complete experimental framework.

A complete *scripting language* is available to construct experiments and control model execution. Experiments can be automated with DO loops and IF-THEN-ELSE structures. Statistics collection may be totally or selectively reset, and/or data values assigned, both during a model run and before or after each run in a series of runs. The experimental specifications and parameters, like any other model data, can be read in from a data file or from the keyboard if desired.

The need to provide multiple *independent* streams of random numbers for use in different parts of the model (or in the same parts for different runs) is very important, particularly after a model is largely complete and the modeler is concentrating on validation and the running of experiments. The *indexed Lehmer* random number generator provided in GPSS/H was designed and implemented specifically to provide exceptionally *simple, straightforward* control of the random number streams used in a model. Modelers can easily specify any number of streams and guarantee that they will be independent (that they will not be autocorrelated due to overlap). GPSS/H also *automatically* detects any accidental overlap, providing an extra measure of protection to users.

4.3 Debugging

The GPSS/H Interactive Debugger conveniently provides for rapid model development and verification. Simple commands are provided by the debugger to control a model's execution and examine its status. Functions are provided to "step" through the model, to set breakpoints and traps that interrupt model execution, and to return to a previously saved state of the model. Almost all data values can be examined, including local data, global data, transaction attributes, entity statistics, and array data values.

The debugger provides a "windowing" mode that displays source code, model status, and interactive user input as the model runs.

GPSS/H provides *"just in time"* debugging. Even when the debugger was not invoked at the beginning of a run, if an error occurs during model execution, the debugger automatically "pops up" to allow the modeler to explore the cause of the error. The modeler can also interrupt a long-running model at any time and use the debugging features to make sure that everything is running correctly before resuming execution.

The GPSS/H debugger has almost no effect on execution speed. Because of this, many modelers use the debugger as their everyday run-time environment for GPSS/H.

5 FEATURES OF GPSS/H RELEASE 3

GPSS/H is continually improving and evolving. Some of the more significant recent additions to the widely-used GPSS/H Professional version are:

- The BLET Block and the LET Statement can now be used to assign a value to ***any*** GPSS/H data item. Unless you need the rarely used range-type assignments, *there is no longer any reason to use* the ASSIGN, SAVEVALUE, and MSAVEVALUE Blocks. Similarly, the INITIAL Statement is now only necessary for setting the states of Logic Switches (unless you need range-mode assignment for other data).

  Besides providing a single, straightforward syntax for assigning values to all GPSS/H data items, the newly-extended BLET Block prevents confusing a Transaction Parameter's name with its value, which could happen occasionally when using the ASSIGN block. For example, consider the following two old-style Blocks, which have different meanings but look very similar:

  ```
  ASSIGN    PF$ALEX,1,PF
  ASSIGN    ALEX,1,PF
  ```

  A modeler might write the first Block (which assigns a value of 1 to the Parameter whose number is given in PF$ALEX), when the intent was to assign a value of 1 to PF$ALEX itself (as is done in the second Block).

  Using BLET, assigning a value to PF$ALEX is quite intuitive:

  ```
  BLET      PF$ALEX=1
  ```
  or
  ```
  BLET      PF(ALEX)=1 (preferred notation)
  ```

  Assigning a value to the Parameter whose number is given in PF(ALEX) is similarly intuitive, yet is not likely to be written by accident:

  ```
  BLET      PF(PF(ALEX))=1
  ```

- GPSS/H now supports built-in random-variate generators for 23 additional statistical distributions (26 in all), and GPSS/H Professional now comes bundled with UniFit II, the highly-regarded distribution-fitting software from Averill M. Law and Associates.

- Release 3 of GPSS/H Professional supports user-written external routines in both C and FORTRAN. Although it is rarely necessary to go "outside" GPSS/H when developing a model, it can be helpful in special situations. For example, it might be desirable to use scheduling software from the real system as a component of the simulation model. A modeler might also want to use pre-existing computational code, or need to write extremely complex computational routines that can become somewhat cumbersome as GPSS/H Blocks. Other special situations might involve the need to interface with non-ASCII data files, or to develop a specialized user-interface.

- GPSS/H built-in I/O now supports opening an existing file for output so that new data are appended at the end of the file.

- CHECKPOINT and RESTORE statements allow a model to save its state at a predetermined point during execution, then make repeated runs using that state as the starting point. In many cases, CHECKPOINT and RESTORE can be much easier to use than the traditional READ and SAVE statements.

- The SYSCALL statement and the BSYSCALL Block, which take an operating system command line as an operand, allow a running GPSS/H model to *shell out* to the operating system to perform the specified command. SYSCALL and BSYSCALL are especially useful when using existing programs to perform data analysis during model execution or between simulation runs. The models can communicate with the external programs through data files. The ability to shell out to the host operating system has also been implemented in the GPSS/H Interactive Debugger. In order to use this feature, one merely types a "$" followed by the operating system command at the debugger command line prompt.

- The INSERT compiler directive allows model code to be read from multiple files during compilation. The INSERT directive acts much like the *#include* statement in the C language. At compilation, the INSERT command line is effectively replaced by the text in the file specified by the INSERT directive. This directive is especially useful for developing and inserting libraries of GPSS/H Macros into a model. It can also be convenient to have GPSS/H functions reside in external files which are generated by other programs.

- The operations that can be performed on Transactions in a User Chain have been extended. New SCANUCH and ALTERUCH Blocks allow examining and changing the Parameters of such Transactions without having to UNLINK and reLINK them. They operate on User Chains in exactly the same way as SCAN and ALTER operate on Groups.
- Floating-point Parameters can be examined and/or modified during operations on User Chains and Groups.

Other enhancements, under development as of this writing, will be discussed in the tutorial session. Persons unable to attend may obtain the latest information by contacting Wolverine Software Corporation.

6 BUILDING A SIMULATOR USING GPSS/H

Earlier in this paper, we contrasted the capabilities of visual-based modeling tools with those of languages. Regardless of which approach is used, the modeler must still build from scratch a model that represents the physical system of interest. Modeling complex systems *correctly* requires intimate knowledge of both the simulation software and the system under study. However, not everyone who can benefit from using simulation has the time or the training necessary to build simulation models.

As a result, a third type of modeling-tool, the *special-purpose simulator*, has emerged as a way of providing simulation capabilities to users with little or no modeling experience. Special-purpose simulators are most commonly developed under circumstances where:

- a single model development effort can benefit multiple users
- modeling expertise can only be obtained from indirect sources such as external consultants

In these cases, an experienced modeler develops the model, freeing the end-user from learning modeling and simulation-software skills.

The *special-purpose simulator* is a custom-built analysis tool designed by an experienced simulation-model builder. At its heart is a data-driven model of a specific system or set of similar systems. The simulator provides its user with a method to easily modify model parameters, define experiments, run tests, and get results. A simulator is usually comprised of a data-entry front end, a simulation engine, and an output browser. The simulation engine runs a *parameterized* model which accepts user-specified data at execution time. Combining these tools brings the power of simulation analysis into the hands of the non-simulationist.

6.1 Data-Entry Front End

The front end is the means by which the user of a *special-purpose simulator* modifies the run parameters without changing the underlying model. This may take several forms, the most basic and rarely used of which involves manually editing a text file. In another approach, the model itself prompts the user for input from the keyboard as the model executes. Still other designs require modifying data by using an external spreadsheet or database program. No matter which approach is used, the purpose of the front end is to conveniently produce a data file which can be used by the simulation model as it executes.

A more advanced approach integrates a customized front-end data-entry program, a simulation engine, and an output browser under a single outer *shell* (Figure 1). Typically created using a general-purpose programming language or a tool such as Visual Basic, the shell may be menu-driven. Data-entry "windows" and dialog boxes guide the user through the process of specifying parameters, running the model, and viewing the output. The shell may also provide built-in help facilities and data "range-checking" (*e.g.* verifying that all operation times are non-negative before executing).

6.2 Simulation Model

The most important component of the special-purpose simulator is the underlying model. Since the end user is generally prevented from modifying the model, this component determines the maximum flexibility offered by the simulator. It must be generic enough to accept a broad range of inputs, and it must be updated periodically to insure that the model *remains* valid.

A static simulation model can be produced and its design frozen when the simulator is initially created, or model code can be generated "on-the-fly" every time that the model parameters are modified by a user. In either case, user input is not limited to operating-parameter values — it can also alter logic embedded deeply within the model. For example, based on a user-specified value, the model could select one of three different order-picking algorithms that have been pre-coded into the model.

6.3 Simulation Engine

The simulation engine is usually a simulation language used to run the model and generate output. There are several features to look for when selecting the engine.

Most importantly, the language used for the engine must be flexible enough to handle the demands that a generalized model places on the software. Flexibility is crucial in the areas of file input, file output, and control logic within the model. Execution speed is also a primary concern. The faster a model executes, the better—time executing a model is often down-time for the user. GPSS/H's speed and built-in flexibility make it the ideal simulation engine for a special-purpose simulator.

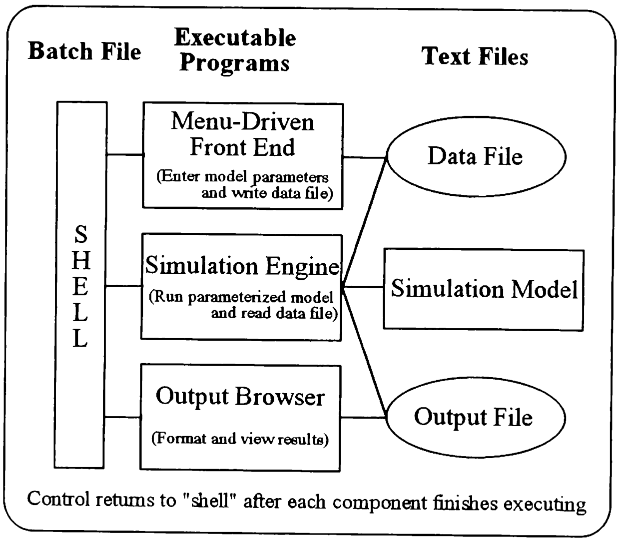

Figure 1: Components of a Special Purpose Simulator

6.4 Output Browser

The output browser displays the data generated by the model in an easy, understandable form. Assuming that the simulator's user has limited experience in simulation-modeling, the standard-style output reports provided by the engine may be difficult to decipher. Custom-formatted output, which includes summary statistics, should *always* be used to present simulator results.

Statistical analysis of the output can be performed directly by the shell program, or by a specialized statistical software product. For *special-purpose simulators* running under Microsoft Windows, SIMSTAT, from *MC*^{2} *Analysis Systems,* reads and analyzes standard output data generated by GPSS/H.

Animation is yet another form of simulation output. Animating a generalized model can sometimes present obstacles. Accounting for variations in resource numbers and capacities, flow and routing-patterns, and physical layout dimensions makes animating a generic model more difficult than animating a specific model. However, a basic animation helps confirm model validity to the non-simulationist. High quality animations can be generated by coupling a GPSS/H model with Proof Animation, a general-purpose animation tool.

6.5 Run-Time Versions Provide an Economical Simulation Engine

A simulator is generally developed for a single application, where it is intended to be used by many people. However, each user must have a copy of the simulation software in order to execute the model. For a simulator used by dozens or even hundreds of users, the cost of the simulation software may render a project economically infeasible. Wolverine's Run-time GPSS/H offers a solution to this problem.

Run-time GPSS/H is identical to Wolverine's 32-bit GPSS/H Professional, except that it can only run models which have been previously compiled with the regular Professional version. The run-time version allows economical distribution of high-performance GPSS/H-based simulators.

Security is another important feature provided by the run-time version. Since only *pre-compiled* models can be run, the end user cannot view or edit the model "source" code. The user has access only to the data files used by the front-end and the output browser; hence, confidential models can be safely distributed. *Even further security* can be obtained by producing special "project-specific" pre-compiled models that can only be run by a specially designated group of users.

A Run-time version of Proof Animation is available for distributing animated simulation models. It provides all of the viewing capabilities of the full version of Proof. The user can zoom-in, fast-forward, or jump to any point in time in the animation. However, the user cannot modify the Proof layout file, so this file must be generated in advance. Ordinarily the model developer would develop the layout with a full version of Proof Animation.

SUMMARY

GPSS/H has a strong history of success in both commercial and academic environments. The product continues to evolve in functionality and to grow in use. Although GPSS/H uses a more traditional text-based model definition, it continues to break new ground in ease-of-use and modeling flexibility.

Wolverine's future product offerings can be expected to include the use of graphical interface techniques for model building, but not just for the sake of appearance. Careful consideration is being given to just how this technology can be used to bring *real* advantages to modelers of diverse and complicated real-world systems. After all, technology for the sake of technology sometimes produces more costs than benefits.

REFERENCES

Banks, J. 1991. Selecting simulation software. In *Proceedings of the 1991 Winter Simulation Conference,* ed. B.L. Nelson, W.D. Kelton, and G.M. Clark, 15-20. Institute of Electrical and Electronics Engineers, Phoenix, Arizona.

Banks, J, J.S. Carson II, and J.N. Sy. 1989. *Getting started with GPSS/H.* Annandale, Virginia: Wolverine Software Corporation.

Henriksen, J.O., and R.C. Crain. 1989. *GPSS/H reference manual,* Third Edition. Annandale, Virginia: Wolverine Software Corporation.

Law, A.M., and W.D. Kelton. 1982. *Simulation modeling and analysis*. New York: McGraw-Hill Book Company.

Schriber, T.J. 1991. *An introduction to simulation using GPSS/H.* New York: John Wiley & Sons.

Smith, D.S., D.T. Brunner, and R.C. Crain. 1992. Building a simulator with GPSS/H. In *Proceedings of the 1992 Winter Simulation Conference*, ed. J.J. Swain, D. Goldsman, R.C. Crain, and J.R. Wilson. 357-360. Institute of Electrical and Electronics Engineers, Arlington, Virginia

Wolverine Software Corporation. 1992. *Using proof animation*. Annandale, Virginia: Wolverine Software Corporation.

AUTHOR BIOGRAPHIES

ROBERT C. CRAIN joined Wolverine Software Corporation in 1981. He received a B.S. in Political Science from Arizona State University in 1971, and an M.A. in Political Science from The Ohio State University in 1975. Among his many Wolverine responsibilities is that of lead software developer for all PC and workstation implementations of GPSS/H. Mr. Crain is a Member of IEEE/CS, SIGSIM, and ACM. He served as Business Chair of the 1986 Winter Simulation Conference and General Chair of the 1992 Winter Simulation Conference.

DOUGLAS S. SMITH received a B.S. in Industrial Engineering and Operations Research from Virginia Tech in 1987, and an M.S. in Manufacturing Systems from Georgia Institute of Technology in 1988. He joined Wolverine as an Industrial Engineer in 1992 where his responsibilities include sales, support, and consulting. Mr. Smith was formerly employed as a Manufacturing Engineer with Hewlett Packard and later as an independent simulation consultant. He is a senior member of IIE.