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

GPSS/H: A 23-YEAR RETROSPECTIVE VIEW

James O. Henriksen
Robert C. Crain

Wolverine Software Corporation
2111 Eisenhower Avenue, Suite 404
Alexandria, VA 22314-4679, U.S.A.

ABSTRACT

GPSS/H was first placed into commercial service in November, 1977. Over the ensuing 23 years, it has become well known for its speed and dependability. In GPSS/H, the process-interaction world view has been combined with many advanced features to make an extremely powerful and flexible tool, capable of handling large, complicated models with ease, yet still providing exceptionally high performance. The sections that follow provide an overview of GPSS/H and its process-interaction world view, a discussion of model-building interfaces including tradeoffs associated with graphical modeling environments, and a summary of advanced and recently-added GPSS/H features. Finally, the use of special-purpose simulators is discussed, along with features of GPSS/H which make it very well-suited for use as the engine in such simulators. This paper provides a retrospective look at the history of GPSS/H.

1 INTRODUCTION

The first version of the GPSS language was introduced by IBM in October, 1962. GPSS introduced the transaction-flow modeling paradigm. Under this paradigm, active objects, called transactions, travel through a block diagram representing a system, competing for the use of passive objects, e.g., single- and multiple-servers. This modeling paradigm has been adopted in quite a number of simulation languages. Terminology differs from language to language. Transactions may be called items, entities, tokens, etc., but the basic architecture is the same: active objects compete for passive resources while traveling through a diagrammatic representation of a system.

GPSS was designed by Geoffrey Gordon as a tool to be used by non-programmers. It is interesting to note that 38 years later, in the year 2000, "no programming required" is still featured as a virtue in advertisements for simulation software. GPSS models were developed by drawing block diagrams and manually typing character-based representations of blocks. Over time, the sizes of GPSS models grew well beyond what Geoffrey Gordon had ever anticipated. While drawing a 100-block diagram is relatively easy, drawing a 5000-block diagram is difficult. As users began developing larger and larger models, and as they became more comfortable with typing in blocks without first having drawn them, GPSS came to be viewed as a true programming language.

The widespread success of GPSS/H has resulted from both the superiority of its original design and the subsequent years of improvement and enhancement. Most of the enhancements developed for GPSS/H have been improvements to it as a programming language. The landmark modeling concepts developed by Geoffrey Gordon remain largely intact, although improvements and additions have been made to them, as well.

Since it is a simulation *language*, GPSS/H requires some programming-style effort, but it does so within the intuitive modeling framework Gordon designed to be readily used without extensive programming experience. It is well suited for modeling both simple systems and large, complex systems.

Although many new simulation tools have been introduced over the past decade, they have often been designed for specific classes 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 in use around the globe modeling manufacturing, transportation, distribution, telecommunications, hospitals, computers, logistics, mining, and many other types of queuing systems.

2 PRODUCT OVERVIEW

GPSS/H is a discrete-event simulation language. Models are developed with an editor and saved in ordinary text files. With GPSS/H, the text files are subsequently compiled directly into memory and executed. Exceptionally fast compilation and execution encourage rapid prototyping and iterative model development.

GPSS/H uses the transaction flow modeling world-view. A modeler specifies the manner in which "objects" flow through a system. As a result, a GPSS/H model *reads* like a flowchart of the system being modeled, and more often than not, no explicit, graphical representation of the flowchart is generated. This 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

A transaction in a GPSS/H model might represent a patient, a telephone call, a truck, a data packet, or any other type of individually identifiable entity. As the model executes, many transactions can be flowing through the model simultaneously—just as many "objects" would be moving through the real-world system being modeled. In addition, multiple transactions can, while flowing through the model, execute the same GPSS/H model statements at the same instant in simulated time without any intervention by the modeler. The execution of a process-interaction simulation model is thus similar to a multi-threaded computer program. This differs greatly from the single-threaded, sequential execution of most general-purpose programming languages, and is a good reason why such languages are usually not good tools for writing simulation models.

Many simulation projects focus on the optimal use of system resources such as people, machines, conveyors, computers, physical space, and so on. In a GPSS/H simulation model, transactions compete for the use of these system resources. As transactions flow through the process representation, they automatically queue up when they can't gain control of a necessary resource. The modeler does not have to specify the transaction's waiting time or its queuing behavior for this to occur. 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 become free, or *explicitly*, as in the case of a part being processed by a machine.

A GPSS/H model, like most real-world systems, may consist of multiple processes operating at the same time. Furthermore, each such process may 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 MODELING A SYSTEM: TEXT VS. PICTURES

Often, the power and ease-of-use of a simulation tool are confused with the model-building interface provided by the tool. That interface may be comprised of icons, menus and data forms; or—as with GPSS/H—it may consist of text entry; or it may be a combination of the two.

3.1 Developing and Editing Models

Many simulation tools try to build models "visually". Icons are placed on the computer screen to represent system components, links between them are drawn, and then the operating characteristics of each component are specified by moving through a series of predefined menus and data forms. One advantage of this approach is that even novices can build simple models quickly–*but not necessarily accurately*.

Building models of complicated systems requires more than simply placing and connecting icons on the screen. To model many processes, an explicit programming environment must be provided. For example, the complicated operating logic of a microprocessor-based equipment-controller often needs a procedural specification—it is simply too cumbersome to represent such logic visually. As a result, models of complex (real world) systems built using the iconic approach often still require the modeler to create substantial amounts of programming code, using a *procedural and text-based* representation, to supplement the "visual" model.

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

In contrast, a text-based interface allows browsing a model simply by scrolling and reading. There is no need to try to remember what details are associated with an icon, or to shift back and forth between icons and forms at one level, and text-based procedural logic at another. Providing a more detailed representation in part of a model simply requires continuing to work in the same style as before, but adding detail to the logic.

Graphical modeling tools can also force their users to make the model fit within a rigid framework bounded by the available icons, menus and forms. An obvious advantage of such a rigid framework is that it tends to steer even a beginning modeler through the model-building process. The not-so-obvious disadvantage is that the framework may not be versatile enough to *accurately* model complicated systems, so the modeler may be forced to choose between an inaccurate model and starting over with a new simulation tool. For an excellent discussion of this situation, see Banks and Gibson (1997).

It is interesting to note that GPSS, which emphasized graphically-based model representations in its earliest implementations, has, in GPSS/H, evolved to regarded as a programming *language.*

3.2 What Does Ease-of-Use Mean?

The presence or absence of a single characteristic—such as a "visual" interface for building models—is not sufficient to determine a tool's ease-of-use. Whether a tool is "easy to use" depends upon the combination of *general characteristics* and *specific features* that are *heavily used* when real-world models are built.

Moreover, the ease-of-use that is claimed for almost all simulation tools can mean many things—that the tool is easy to learn, easy to use once learned, easy to use when modifying models, easy to use when building simple models, or easy to use when building large, complex models. Rarely is a product equally good at all of these, but equally rarely is the prospective user informed about which "ease-of-use" is being claimed.

Another meaning, which is often overlooked yet very important when modeling complex, real-world systems, is that of ease in verifying that the tool's inner workings, when running the model, are accurately mimicking the operation of the actual system. If such verification is difficult, it may not be done. The risks of not doing it have been pointed out by Schriber and Brunner (1997).

A simulation language's ease-of-use is rarely constant over the duration of a simulation project. The ease-of-use of a tool that is optimized for developing small models may diminish as a model gets larger and larger, and more details of system operation have to be incorporated into a model. Conversely, a tool that requires a bit more study to master may be harder to use in the early stages of a project, but offer downstream rewards when it is able to meet modeling requirements for a large, complex system. Scalability of technology is an important contributor to ease-of-use. GPSS/H has excellent scalability. It is able to handle very large models (10's of thousands of blocks) and programmatic specification of complex operating rules.

4 IMPORTANT FEATURES OF GPSS/H

Several characteristics make Wolverine's GPSS/H an excellent choice for a general simulation environment. A key feature of GPSS/H 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.

Specification 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 queuing and service statistics, are *automatically* provided each time a model is run, which greatly aids incremental model development. In addition, with very little effort virtually any of the automatically gathered statistics can be written to a file for use with other software.

GPSS/H is available for PCs and SUN SPARC workstations. On the PC, GPSS/H Professional runs as a true 32-bit application under Windows 98, Windows 95, Windows NT, Windows 3.x, OS/2, or even—if necessary—plain DOS, providing tremendous speed as well as model size that is limited only by the computer's available memory.

4.1 GPSS/H Input and Output Capabilities

The file and screen I/O built into GPSS/H provide a variety of ways to get data into a model and get results out. GPSS/H can read directly from the keyboard or from standard 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. Input data files are free-format (values on each line are simply separated by blanks or tabs), 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 included to make it easy to get tabular output. Character strings can also be manipulated using built-in capabilities. Writing output files using the comma-separated-value (.CSV) format, for easy input into a spreadsheet, is very straightforward with GPSS/H.

4.2 Scripting Language for Experiment Control

One run proves nothing. The results from a single run of a simulation are only single observations 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 may be 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.

4.3 Statistically Robust Random-Number Streams

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 *automatically* detects any accidental overlap, providing an extra measure of protection to users.

4.4 Validation and Debugging

The GPSS/H Interactive Debugger provides invaluable assistance for rapid model development and verification. Prior to the advent of GPSS/H in 1977, most simulation tools featured batch-oriented, non-interactive debugging. To debug a model, one typically generated a large trace file and very carefully proofread the trace file to verify model correctness.

In GPSS/H, simple debugger commands are used to control a model's execution and to examine its status. Functions are provided to "step" through the model, to set breakpoints and traps that interrupt model execution based on multiple criteria, 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.
- A modeler can 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

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

- The BLET Block and the LET Statement can 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. The BLET Block provides a single, straightforward syntax for assigning values to all GPSS/H data items. For example, using BLET to assign a value of 1 to the Transaction parameter named ALEX is quite intuitive:

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

Using indirect addressing, such as assigning a value to the Parameter specified by the number given in PF(ALEX), is similarly intuitive, yet is not likely to be written by accident:

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

- GPSS/H supports convenient built-in random-variate generators for 26 statistical distributions.
- 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 prompt.
- The operations that can be performed on Transactions in a User Chain were extended. The 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 both User Chains and Groups.

6 BUILDING A SIMULATOR USING GPSS/H

Earlier in this paper, the capabilities of visual-style modeling tools were contrasted 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 (Schriber and Brunner 1997). However, not everyone who can benefit from using simulation has the time or the training necessary to build simulation models.

Consequently, 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 often developed under circumstances where:

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

In these cases, it makes sense to have an experienced simulationist develop the model, freeing the end-user from learning modeling and simulation-software skills.

The special-purpose simulator is thus 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 tool such as Visual Basic, or through use of macros within spreadsheet software such as Excel, the shell may be menu- or button-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).

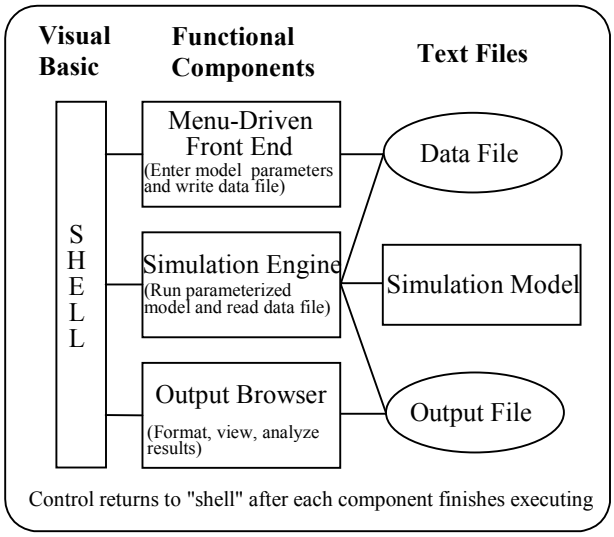

Figure 1: Components of a Special Purpose Simulator

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 ensure 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—a user 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 runs the model and generates 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 an ideal simulation engine for a special-purpose simulator.

An excellent example of the use of GPSS/H as a simulation engine is given in Coughlan and Nolan (1995).

6.4 Output Browser

The output browser displays the data generated by the model in an easy-to-understand form. If the simulator's user has limited experience in simulation modeling, the standard-style statistical reports provided by the engine may be totally unsuitable. Custom-formatted output, including summary statistics, should *always* be used to present simulator results. Statistical analysis of the output can be performed directly by the shell program, by a spreadsheet or similar program, or by a specialized statistical software product.

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 Wolverine's 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 too expensive. 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 previously have been specially 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.

SUMMARY

GPSS/H has a strong 23-year 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 forge a reputation for the robustness, modeling flexibility, ease-of-use and very high performance that experienced modelers demand for their projects.

REFERENCES

Banks, J. and R. Gibson. 1997. Simulation modeling: some programming required. *IIE Solutions* February 1997: 26-31.

Coughlan, K.L., and Paul J. Nolan. 1995. Developing special purpose simulators under Microsoft Windows. In *Proceedings of the 1995 Winter Simulation Conference,* ed. C. Alexopoulos, K. Kang, W.R. Lilegdon, and D. Goldsman, 969-976. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Schriber, T.J., and D.T. Brunner 1997. Inside simulation software: how it works and why it matters. In *Proceedings of the 1997 Winter Simulation Conference,* ed. S. Andradottir, K.J. Healy, D.H. Withers, and B.L. Nelson, 14-22. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

AUTHOR BIOGRAPHIES

JAMES O. HENRIKSEN is the president of Wolverine Software Corporation. He was the chief developer of the first version of GPSS/H, of Proof Animation, and of SLX. He is a frequent contributor to the literature on simulation and has presented many papers at the Winter Simulation Conference. Mr. Henriksen has served as the Business Chair and General Chair of past Winter Simulation Conferences. He has also served on the Board of Directors of the conference as the ACM/SIGSIM representative. He can be reached via e-mail at: <mail@wolverinesoftware.com>.

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 Wolverine responsibilities is that of chief developer for PC and workstation implementations of GPSS/H. Mr. Crain is a Member of ACM. He served as Business Chair of the 1986 Winter Simulation Conference and General Chair of the Twenty-Fifth Anniversary Winter Simulation Conference in 1992. He can be reached via e-mail at: <mail@wolverinesoftware.com>.