Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

BUILDING A SIMULATOR WITH GPSS/H

Douglas S. Smith
Daniel T. Brunner
Robert C. Crain

Wolverine Software Corporation
4115 Annandale Road
Annandale, Virginia 22003, U.S.A.

ABSTRACT

A *simulator* is a data-driven simulation tool designed for a specific system or type of system. Combining a data-entry front-end with a parameterized simulation model produces a tool which a "non-simulationist" can use to analyze complex systems. This paper provides an overview of the classes of simulation tools, describes the components used to build a custom simulator, details relevant features of GPSS/H, and discusses Wolverine's new GPSS/H run-time version.

1 CLASSES OF SIMULATION TOOLS

Over the past decade, numerous discrete-event simulation systems have emerged. However, "off-the-shelf" modeling software can still be classified into two general categories: *packages* and *languages*. The primary differences between simulation packages and simulation languages are the model-building interface and the level of detail allowed by the software. The tradeoff is between ease-of-use and modeling flexibility. Packages tend to cover a broad range of specific applications, while languages are usually general purpose.

A third type of tool, the simulator, has emerged as a means to provide simulation capabilities to users with little or no simulation modeling experience. Simulators are custom-built analysis tools designed by experienced simulation model builders. They are used to analyze a single system or a narrow range of specific systems.

Many of the software products that have been referred to in the past as simulators are more appropriately called packages. Packages include a model-building interface which may expedite the modeling process but still requires the end user to know how to *design* simulation models. Simulators, on the other hand, require *no* model building by the end user. The parameterized model that is used in a simulator is designed and built by an expert and is driven only by data supplied by the end-user.

1.1 Packages

The scope of a package is often limited to a specific class of applications such as manufacturing, communications, or transportation. Packages are characterized by software interfaces that conceal most of the underlying engine, and sometimes by promotional claims that "no programming is required."

The goal of packages is to provide the modeler with a user-friendly interface; however, the interface can impose some restrictions on the serious modeler. In an attempt to make a package easier to learn and use, its developer may implement hidden assumptions. Those assumptions can render models invalid or unrealistic without the modeler's suspecting it. The assumptions also sacrifice flexibility. This can result in added effort when programming complex control constructs. The model must remain within the framework of the package, as it is very difficult to model beyond the scope provided (Banks 1991). Models developed with packages are also sometimes difficult to validate due to a lack of built-in debugging tools.

1.2 Languages

Discrete-event simulation languages have been in use for over thirty years and are used extensively today to model a wide range of systems. The built-in flexibility in many languages makes them capable of modeling complex systems in great detail. Virtually any realistic system can be modeled using a simulation language. However, the amount of detail allowed often implies that more time is needed to insure that all of the details have been captured accurately (Banks 1991).

As simulation products evolve, the distinction between packages and languages is fading. Languages

are offering more features previously associated with packages while many packages now provide some programming capabilities. Ultimately, packages and languages will be grouped into a single class of tools characterized as *model builders*. Yet, there will always be a strong distinction between model builders and simulators.

2 THE SPECIAL PURPOSE SIMULATOR

When using either a package or a language, the user must build from scratch a model that represents the physical system. Modeling complex systems requires intimate knowledge of both the 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.

A recent addition to the family of simulation tools, the *simulator*, is growing in use and popularity. The heart of the special-purpose simulator is a data-driven model of a specific system or set of similar systems. The end user is provided with a method to easily modify model parameters, design experiments, and run tests. This can be accomplished by combining a data-entry front end, a simulation engine, and an output browser. The engine runs a *parameterized* model which accepts user-specified data at execution time. This combination of tools brings the power of simulation analysis into the hands of the non-simulationist.

Simulators are most commonly developed under circumstances where: 1) a single model development effort can benefit multiple users; and 2) 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 software-specific modeling skills.

2.1 Data-Entry Front End

The front-end is the means by which the user modifies the run parameters without changing the underlying model. This may take several forms, the most basic and rarely used of which has the user manually edit a text file. In another approach, the software prompts the user for input from the keyboard as the model executes. Other designs require the user to modify data in an external spreadsheet or database program. No matter which approach is used, the purpose of the front end is to produce a data file which can be read by the simulation model as it executes.

A more refined technique integrates a customized front end program, a simulation engine, and an output browser under a single shell, (Figure 1). Typically created using a general-purpose programming language, 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).

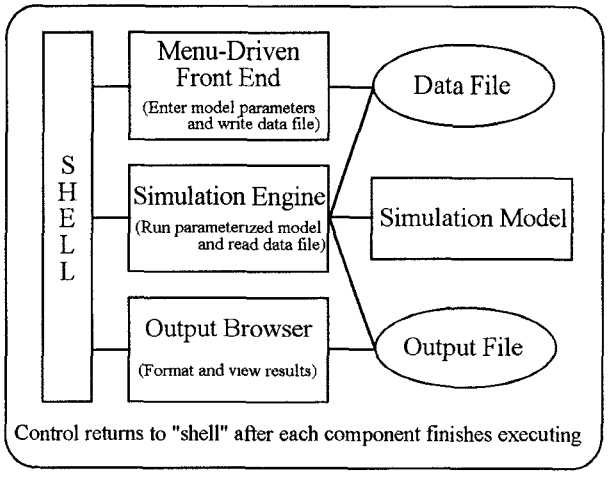

Figure 1: Components of a Special Purpose Simulator

2.2 Simulation Model

The primary component of the simulator is the underlying model. Since the end user is generally prevented from modifying the model, this component ultimately defines the maximum flexibility offered to the end user. It must be generic enough to accept a broad range of inputs and it must be maintained periodically to insure that the model remains valid.

The simulation model can be generated 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 the user. In either case, the user input not only consists of operating parameter values, but it can also contain data used to alter logic embedded deeply within the model. For example, based on a user-specified value, the model can select one of three different order-picking algorithms that have been pre-coded into the model.

2.3 Simulation Engine

The simulation engine is usually an "off-the-shelf" simulation language used to run the model and generate the output file. There are several features to look for when selecting the simulation engine.

Most importantly, the language used for the engine must be flexible enough to handle the demands that a general model places on the software. This flexibility is crucial in the areas of file input, file output, and control logic within the model. Execution speed should also be a primary concern. The faster, the better; time executing a model is often down-time for the end user. If possible, the simulation engine should be invisible to the user. There are few reasons for the simulation engine to interrupt the user while it is executing; any error conditions should be directed to the user from the shell program.

2.4 Output Browser

The output generated by the simulation can be custom formatted by the model or by the simulator shell. The function of the output browser is to display the data generated by the model to the end-user. Assuming that the end user has limited experience in simulation modeling, standard output reports may be difficult to decipher by anyone not intimately familiar with the underlying model. Custom formatted output, which includes summary statistics, should be used to present simulation results.

Experiment design may be partially defined by the model builder, but at least some flexibility should be given to the end user. Access to random number streams, number of replications, and warm-up period are essential for sound statistical experimentation. If these options are unavailable, the end user has no way to compare systems using common random numbers, to control run-lengths, or to throw out transient data on a run-by-run basis.

Statistical analysis of the output can be performed directly by the shell program or by a specialized statistical software product. For simulators running under Windows, SIMSTAT from *MC^2 Analysis Systems* reads and analyzes standard output data generated by several major simulation software products.

Animation is yet another element of the 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.

3 GPSS/H FEATURES

Wolverine Software Corporation's GPSS/H is an excellent engine for use in a simulator. Among the most flexible simulation languages, GPSS/H can be used to model a wide range of applications including transportation systems, communications networks, manufacturing and material handling systems, computer systems, and service-based operations. GPSS/H runs on a number of platforms ranging from personal computers to VAX/VMS machines, UNIX workstations, and IBM mainframes.

GPSS/H Professional is a 32-bit, protected-mode application which is capable of running large models at exceptionally fast speeds on the 80386/486 platform. GPSS/H Professional has no built-in model size limitations. For very large models, it can access all of the physical RAM in the machine. In addition, the same version of GPSS/H can run as a 32-bit DOS application under Windows 3.x or OS/2 2.0 with full access to virtual memory capabilities. A customized shell program can run seamlessly under any of these environments. To suppress the default screen output and conceal GPSS/H from the user while it is executing, a capability is provided which enables GPSS/H to run in "quiet mode."

3.1 GPSS/H File and Screen I/O

The file and screen I/O built into the GPSS/H language provide the modeler and the end user with a variety of ways to get data into the model and to write custom output files. GPSS/H can read directly from the keyboard or from text files, and it can write directly to the screen or text files. The GETLIST statement and the BGETLIST Block read integer, double precision floating point, and character data. Values in the data file are separated by blanks, and special actions may be taken for error and end-of-file conditions.

Customized output is generated using the PUTPIC statement and the BPUTPIC Block. These use a "picture" type of format specification, which follows the "what you see is what you get" convention. It is also possible to read and write character strings using a similar group of statements. This flexible family of input and output capabilities provides the essential interfacing necessary to create a special purpose simulator.

3.2 Experiment Control

The results produced by a single run of a simulation model provide only point estimates of random variables. Experimental design is essential to accurately predict the statistical behavior of the model outputs. GPSS/H provides the necessary tools to build a complete experimental framework into the model.

GPSS/H contains a complete run control *language*. Experiments can be automated within the model file

using DO loops, IF-THEN-ELSE structures, and other branching constructs. Statistics collected by the model may be selectively CLEARed, RESET or INITIALized during execution or between subsequent runs. All of the experimental specifications can be read into the model from a data file just like the other model parameters.

3.3 Run-Time Version

A simulator is generally developed for a single application, but it is frequently intended to be used by many people. However, each user must have a licensed 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 new product, Run-time GPSS/H, offers a solution.

The run-time version is identical to GPSS/H Professional except that it can only run models which have been previously compiled with the Professional version. The other fundamental difference is the price. The run-time version allows economical distribution of GPSS/H based simulators.

The security provided by the run-time version is another feature. Since it runs pre-compiled models, the end user cannot change the model "source" code. The user has access only to the data file; hence, confidential models can be safely distributed.

SUMMARY

Historically, significant time and training were necessary to perform a detailed simulation analysis. The special-purpose simulator allows a non-simulationist to use simulation without building models. A simulator consists of four basic components running under a single shell program: the data-entry front end, the simulation model, the simulation engine, and the output browser.

GPSS/H contains many features that are necessary and convenient for use as a simulator engine. Fast execution, operating system independence, and extended file I/O are just a few. Additionally, a run-time version is available which allows the model developer to distribute compiled models at reasonable costs.

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.

Blaisdell, W. 1991. *SIMSTAT for Windows 3.0 User's Manual.* Troy, New York: MC^2 Analysis Systems

Brunner, D.T., and R.C. Crain. 1991. GPSS/H in the 1990s. In *Proceedings of the 1991 Winter Simulation Conference,* ed. B.L. Nelson, W.D. Kelton, and G.M. Clark, 81-85. Institute of Electrical and Electronics Engineers, Phoenix, Arizona.

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

Schriber, T.J. 1991. *An Introduction to Simulation Using GPSS/H.* New York: John Wiley & Sons.

Seppanen, M.S. 1990. Special Purpose Simulator Development. In *Proceedings of the 1990 Winter Simulation Conference,* ed. O. Balci, R.P. Sadowski, and R.E. Nance, 67-71. Institute of Electrical and Electronics Engineers, New Orleans, Louisiana

AUTHOR BIOGRAPHIES

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 and consulting. Mr. Smith was formerly employed as a Manufacturing Engineer with Hewlett Packard and as an independent Simulation Consultant. He is a senior member of IIE.

DANIEL T. BRUNNER received a B.S. in Electrical Engineering from Purdue University in 1980, and an M.B.A. from The University of Michigan in 1986. He has been with Wolverine since 1986, where his responsibilities include product marketing, product development support, and simulation consulting. Mr. Brunner served as Publicity Chair for the 1988 Winter Simulation Conference and Business Chair for the 1992 conference, and is General Chair for the 1996 conference.

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 is General Chair of the 1992 Winter Simulation Conference.