Proceedings of the 1984
Winter Simulation Conference
S. Sheppard, U. Pooch, D. Pegden (eds.)

THE USER INTERFACE OF GPSS/PC™

Springer Cox
Minuteman Software
P.O. Box 171
Stow, Massachusetts 01775-0171

Abstract

GPSS/PC is a new implementation of GPSS, the General Purpose Simulation System, with several interactive extensions to the user interface. Its overall design integrates a syntax directed statement parser with the run time system. This results in a number of advantages in the controllability of running simulations.

Unlike traditional simulation language implementations, GPSS/PC was designed with the simulation primitives on the user interface. This means that functions previously accessible only to transactions within the simulation at run time can now be called manually from the keyboard. This results in a high level of control over the actions of entities within the simulation. Any GPSS block statement can be applied in this manner and the results of the interaction can be explored immediately.

GPSS/PC also provides for numeric and structural modifications to be made at run time. Simulations can be stopped or interrupted and any named value can be modified without requiring reassembly of the model. Further, with relatively mild restrictions, blocks in the GPSS model can be added, deleted, or replaced without the need for reassembly.

Several additional aids to usability have been implemented in the user interface. A syntax directed statement parser refuses to accept keystrokes which cannot possibly lead to a syntactically valid statement. A command recognition feature allows partial specification of statement verbs. Other aids include cursor prompting, online help, and an integrated statement editor. Nearly all statements, including GPSS block statements, can be assigned to function keys and recalled with single keystrokes.

Keywords

GPSS, discrete event simulation, syntax directed parsers, user interface.

The Scope of this Paper

This paper does not cover all the design principles behind GPSS/PC™ and it does not fully describe all the features of the user interface. Instead, it treats a few of the highest principles and follows their effect on the user interface of GPSS/PC.

The Need for Immediacy

Present day simulation languages are characterized by their heritage as descendants of programming languages. From the earliest days of the computer industry, running a program meant preparing a text file of source code and then using it as input to another program, the compiler. Even after the compilation, the task is far from done. The next step is to combine existing compiled programs so that large programs can take advantage of modular designs. The output of this, the link step, was a computer program that could be executed. However, notice that the time, when the user was concerned with the details of the program, has long since passed. His/her mental context has been shifted toward problems imposed by an unwieldy computer system, away from the simulation problem he/she is attempting to solve. In fact, the situation is even worse than this. In order to find, debug, and correct the inevitable programming errors, the user must repeatedly run the incorrect program hoping for decent diagnostic messages, making corrections to the original text file, and repeat the whole process for each error.

This edit-compile-link-run-debug-edit cycle puts unnecessary distance between the analyst and the simulation. In such an environment the user's mental context has changed considerably between the time when a simulation primitive is asserted and the time when it is tested. This increases the risk of new errors caused by the correction of old ones. Clearly, it is desirable to detect errors as soon as possible. This is one of the highest principles behind the design of GPSS/PC™.

On the constructive side, it is extremely desirable for the user to see effects of his/her actions as soon as possible. When the closeness between the user and the simulation allows feedback in short time periods, the user develops a "feel" of the environment in which he is working. It is the immediacy and responsiveness of close working environments that best serve the exploratory and intuition-building activities of the simulation analyst. Just as a hammer is the extension of the hand of the carpenter, the simulation environment should be the extension of the mind of the analyst. It should be immediately responsive to inquiry and to manipulation. It should allow the immediate correction of errors, and it should provide for a function which can monitor for exceptional conditions. This immediacy is the highest design principle of GPSS/PC™.

Immediacy of Error Detection - Levels of Error Defense

The major design goal with respect to user errors is to detect them and treat them as soon as possible, preferably before the user has lost the mental context in which the statement is asserted. In GPSS/PC there are several lines of defense with respect to errors. We start with the earliest and proceed to the most removed error handling features.

The first line of defense carries the principle of early detection into the mind of the user. It is the attempt to prevent the user from committing the mental error in the first place. In a sense this line of defense seems trivial and need not be stated. However, it is useful to include it in the overall user interface description. The user interacts with the simulation syste according to a mental picture he/she has developed of the product. Errors develop when the picture of the interface is inaccurate, or when conclusions on it are drawn incorrectly. There is little we can do about the latter type, however the mental picture presented to the user can be controlled somewhat by product architecture and documentation. To attack the problem, we must make the product architecture consistent and predictable. We must remove restrictions and exceptional conditions. Allocation limits, unnecessary data typing, and unnecessary exceptions in the operation of the product must be removed. And most important, the product documentation must build the mental picture of the product as efficiently as possible.

The second line of defense comes into play after the mental error has been committed. If the error is formulated in the mind of the user we must prevent it from entering the simulation environment, and we must provide immediate negative feedback before the mental context of the user begins to dissolve. When a mental error is committed, we must not allow the user to pass to the next operand or statement. GPSS/PC™ does this by a feature called "keystroke error prevention". By checking each keystroke for invalid syntax, GPSS/PC™ refuses to allow syntax errors into the simulation environment. Online help is then available to present valid alternatives. We will discuss this in more detail below.

The third line of defense treats those errors which have not been stopped by keystroke error prevention. In general, errors related to block statement interactions and some remaining restrictions are detected only after the simulation has begun. To maximize the closeness of the error to the mental context in which it was conceived, GPSS/PC combines all major phases of the edit-compile-link-run-debug cycle into a single phase called a session. When an error causes a simulation to stop abnormally (an "error stop"), the user can correct the offending statement(s) and proceed immediately. At this time the mental context of the user is as fresh as possible. Included in the third line of defense is a mechanism for detecting unusual conditions during the simulation. A PLOT is available to observe state variables during the simulation When a run time error is detected, exploratory commands can be issued to examine the state variables of the simulation.

The Fourth line of defense is required by the third. When run time error detected, we must provide for an immediate fix. GPSS/PC™ allows the structure of the model to be modified with each statement scanned. Not only is it unnecessary to leave the run time environment, but the simulation does not even have to be restarted. In order to make this work, the user must be able to make corrections to the structure of the model and to manipulate entities in great detail. In GPSS/PC™, we allow blocks to be inserted, deleted, and replaced in the current model without even requiring that the simulation be restarted. In addition, the design of GPSS/PC brings all the simulation primitives up to the user interface. By doing this, the user can apply the same operations to transactions manually as can be done deep within the simulation. We therefore refer to this feature as "manual simulation". Block definition statements become action commands permissible through keyboard entry allowing a whole new level of controllability in the simulation environment. This is discussed below.

A final, less directed line of defense includes all functions designed to relieve the tedium of various man-machine interactions. In this group we place automatic spacing, which relieves the tedium of alligning fields in statement formats, and command recognition, which allows a user to use his/her own abbreviations for reserved words. Assignable function keys allow any statement to be repeated with a single keystroke. This is quite useful in stepping through a simulation during the debugging period.

Consistent and Predictable Architecture

Considering the difficulty of mastering a large user manual, sometimes it is far more important to maximize what does not have to be said, than it is to say everything. It is the unpredictability of a specification that demands its inclusion in the documentation.

GPSS/PC has several features intended to reduce the "knowledge load" of the user. First, in GPSS/PC all integers enjoy unlimited precision. The allocation for each numeric value expands to fit the number. This implementation makes GPSS integer arithmetic superior to floating point arithmetic in several important respects. First, with a floating point system clock, there is a danger that after a long simulation, system clock updates will underflow, thereby invalidation simulation results. Second, accumulations of large values, most notably of sum-of-squares in variance calculations, is in danger of overflowing, again, yielding invalid results. Neither of these problems occur in GPSS/PC. In a sense, the user interface has been cleaned of the need to be concerned with these exceptional conditions.

"Unlimited precision" integers have several other effects on the user interface. First, data typing for GPSS numerical elements is reduced to a single type. It is no longer necessary to specify a type for each numeric item. Second, the need to predict the maximum value of data items is removed.

With the introduction of "unlimited" precision integer values, GPSS/PC™ eliminates many of the worries about data types while providing internal calculations numerically superior even to double precision floating point arithmetic. If all the memory in the computer has not been used, there is no limit to the size of the integer values. The dangers of clock underflow and overflow in the statistics accumulators, exist in systems with floating point system clocks, but not in GPSS/PC™. For any choice of time granularity there is no possibility of underflow or overflow in the system clock.

To reduce the level of system knowledge needed to access sophisticated mathematical algorithms, GPSS/PC has integrated a rich mathematical library into the GPSS language. Complex expressions can include SNAs (System Numeric Attributes), exponentials, logarithms, trigonmetric functions, and logical operators. This allows probability functions in closed form to be included as GPSS variable entities. Similarly, GPSS/PC can be used to simulate continuous state systems using any of the commonly used integration techniques.

Some other unnecessary aspects of traditional GPSS dialects have been removed. Dynamic allocation of entities and transaction parameters removes most of the individual entity count limits from the user interface. The older, more uniform method of indirect addressing through a parameter is retained in GPSS/PC. Also, the unnecessary distinction between System Numeric Attributes and System Logic Attributes is removed.

Finally, an operand-specific help screen is available for each operand in the language. The correct syntax and the valid forms for any operand can be called up on the screen by a single [?] keystroke.

Uniphasic Simulation Environment, the "Session"

GPSS/PC combines all phases, except report printing, into a single phase called a session. The processing required by the compilation-link phase is completed incrementally as each statement is scanned. Most sessions start with the scanning of a program file, which is the ASCII representation of a set of line numbered statements. By the time the program file has been completely read, the data structures representing the model have all been built in the main memory of the personal computer.

Let us take a moment to review the multiphasic operation depicted in figure 1. By following the course of action required for development of a simulation in the upper diagram, we see how unnecessary tedium is introduced when an error is detected, analyzed, and repaired. This task, which should be as immediate as possible, requires that several phases be entered and exited. Not only does each phase transition require several seconds, but the phases themselves may requires several minutes as is the case with a long compilation. In many cases, the user must direct these actions, thereby losing much of the original mental context he/she had when the offending statement was created.

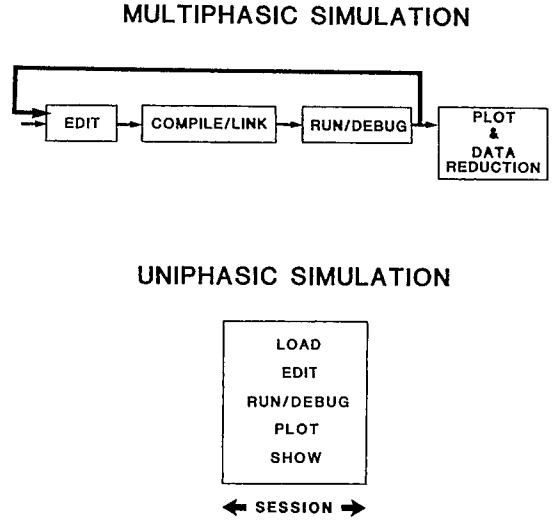

Figure 1

Simulation Phases

In the top half of figure 1 we see the sequence required of a multiphasic simulation environment. This sequence is our heritage from the first programming languages. In order to even begin a simulation, one or more preliminary phases must be entered. Generally, the creative activities of model building are done in phase 1, but the corrections to errors

must be conceived much later, out of the original context. Even if the original error is corrected, it is easy to introduce new errors because the mental context under which the the original model was composed has had time to deteriorate. In addition, the time required to correct each problem is extremely long, each requiring a passage through several phases.

Contrast that to the uniphasic design of GPSS/PC depicted in the bottom half of figure 1. Remember that syntax errors are not even accepted. Errors that do get into the environment are detected at run time, during a simulation. In the uniphasic design, simulations can be started immediately. When an error is detected, the simulation stops and a diagnostic message is printed on the screen. At that instant, the analyst can explore the state of the simulation, manipulate transactions in manual simulation mode, or make structural changes to the model. He/she may then immediately resume the simulation using the corrected model.

Error Prevention

When GPSS/PC™ scans a statement, a syntax directed statement parser courses through a state diagram as keystrokes are being incorporated into the current statement. Invalid syntax is not admitted into the simulation environment. If a keystroke is made which cannot possibly lead to valid syntax, GPSS/PC sounds an audio signal, places an error pointer on the screen, and rejects the keystroke by neither echoing nor incorporating the character.

The first step in screening keystrokes is to classify the keystroke into alphabetic, numeric, special, delimiter, and [CR] ("enter" or "return"). Only key classes represented in at least one valid operand are admitted past the screening test. Other keystrokes are rejected immediately without further testing.

If the class of the character includes valid possibilities, the keystroke is tested according to state-specific criteria. To do this the GPSS/PC statement parser is at all times in exactly one state of a complex finite state diagram. Each keystroke of the user is tested to see if it calls for a valid state change. This process is depicted in figure 2. In this figure, we see a detail of the state diagram which represents the valid grammar for a GPSS block statement operand.

The validity of each keystroke is determined by calculating the union of two flag fields which have as elements flags which represent GPSS operand forms. The first flag field contains truth values representing valid operand forms for the block (or statement) type and operand number. For example, the A operand field for ADVANCE blocks has flags set for the flag for names, for nonzero positive integers, for indirect addressing forms, and so on. Those forms which are not valid in operand A of ADVANCE have an associated flag which is not set. This first flag field is the "Permissible Set".

The second flag field used in the calculation contains flags which represent which operand forms can still be attained by the user, assuming there is no backspacing. As the user presses more keys, valid transitions in the state diagram causes flags in the "Achievable Set" to be effectively turned off. The represents a decrease in the number of forms still possible. By the time a valid operand is specified, all possibilities but one have been ruled out.

The validity of state transitions is tested by calculating the intersection of the Permissible Set and the Achievable Set. Before a keystroke is accepted as valid, if a state transition is triggered by the keystroke, the intersection of Permissible Set of the operand and the Achievable Set of the new state must not be null. When it is null, it means that there are no valid operand forms that can be reached by additional keystrokes. Therefore, this condition is not allowed, and any keystroke leading to it is rejected.

For example, let us consider two cases in which the user is attempting to enter the valid form

*45

In GPSS, this is a form of indirect addressing which takes the value of parameter number 45 of the active transaction. Let us consider the case when the operand is entered correctly. We assume that the verb (e.g. GPSS block name) of the statement has already been identified and that the Permissible Set for this operand has been identified. The operand read routine starts in state 1, as is depicted in figure 2. When the [*] key is pressed, it is classified as a special character and admitted to the state-specific testing. A reference to the state diagram determines that if valid, the keystroke will lead to transition to state 2. State 2, as are all states, is associated with an Achievable Set, which represents all operand forms distal to state 2 in the grammar tree. Each leaf in the state diagram which can be reached from state 2 is a possibility for the operand under construction. The intersection of the

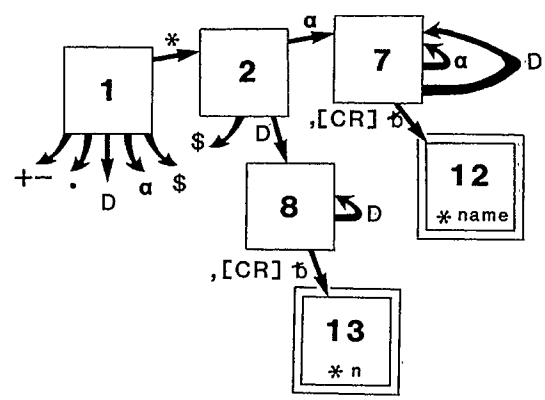

Figure 2

Detail of GPSS/PC Grammar

permissible set for the current operand and the Achievable Set for state 2 is calculated. If it is null, there is no valid operand that can be reached if the current keystroke is accepted. However in the present case the form represented by state 13 is in both sets. The [*] keystroke is therefore accepted and incorporated into the statement, and the parser enters state 2.

Similarly a limited sequence of digits are accepted, promoting the parser to state 8. When a delimiter key is pressed, indicating the user's desire to complete the operand, a final test of the terminal state number 13 leads to acceptance of both keystroke and operand.

Now let us consider the case where the user incorrectly presses the key sequence

***4B**

In this case the parser begins in state 1, as before. The valid keystroke [*] promotes the parser to state 2, and the keystroke [4] promotes to state 8. Both keystrokes are accepted and incorporated into the statement. However we now incorrectly press the invalid [B] key. In this case, no valid transition (including back to state 8) is associated with alphabetic keys when the parser is in state 8. The keystroke is therefore rejected.

If this statement operand was associated with a Permissible Set which did not allow indirect addressing forms, the [*] keystroke would be rejected when the intersection of the Permissible Set and the Achievable Set was calculated just before the transition from state 1 to state 2.

Mutability of the Environment

The simulation environment must respond to the whim of its master. Ideally, single actions should allow the analyst to explore and modify the simulation environment. However, we are just beginning to see these design principles enforced in commercially available simulation systems.

The earliest appearance of interactive simulation controls, was that of debugging commands. These generally provided the user with the ability to stop the simulation at specific points, and to trace details of the simulation, and to cause the simulation to proceed in a step-wise fashion. GPSS/PC includes these older methods as well as several new ones.

The first level of modification available to users of GPSS/PC is the ability to change named values. At any time during the session, named values except block locations, can be changed by interactively entering an EQU statement. This flexibility is a strong motivating force for using only symbolic, not literal, numeric constants in a GPSS program. Then, the dynamic effects of numeric changes can be viewed easily during a running simulation.

The second level of modification involves the structure, itself, of the model. GPSS/PC™ allows GPSS blocks to be deleted, replaced, or inserted in the middle of a simulation. This is a much higher level of mutability than has been available before, and it carries with it the responsibility of understanding what the effects of the changes will be. For those users who venture into it, the debugging time of model development will be drastically reduced.

The last new level of modification is given the name "manual simulation". In a truly interactive environment, the user must have full control. To this end, GPSS/PC brings all the simulation primitives (block statements) accessible to transactions deep in the simulation, up to the user interface. The statements may now be entered as keyboard commands. Manual simulation is depicted in figure 3. The automatic, non-manual mode of simulation is depicted in the upper half of the figure. In this case, after the structure of the model has been described by line-numbered block statements, a START, STEP, or CONTINUE command causes block entries to occur until the stopping condition is detected. Control of each block entry resides within the transaction scheduler.

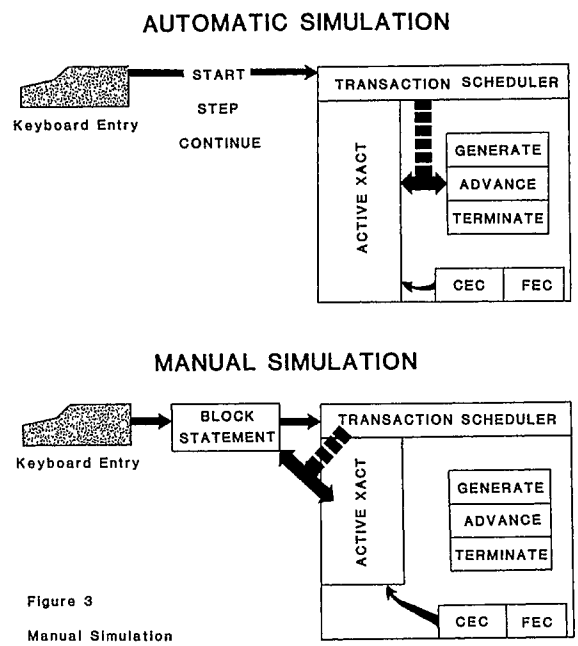

Figure 3

Manual Simulation

Manual simulation is depicted in the bottom half of figure 3. Any time after a simulation has been begun, it may be stopped or interrupted. Manual simulation requires that an "active transaction" has been previously selected, and set up on the CEC (Currents Event Chain) by the transaction scheduler. In the most trivial case, a STOP condition may be set for the very first transaction. When the simulation stops (or is interrupted), manual simulation statements may be entered in the form of unnumbered block statements. Each manual simulation statement causes a tempo-

rary block to be created. The active transaction then attempts to enter this block. In this process, the old block for which the active transaction was scheduled is saved. If the manual simulation does not displace the transaction to a new destination, the original "next block" is restored to the active transaction. A long sequence of blocks can be introduced in this manner. However, if the active transaction is placed on a delay chain or is otherwise removed from the CEC, the transaction scheduler chooses a new transaction as the active transaction.

Manual simulation presents a great variety of new possibilities to simulation analysts. In real world applications, these possibilities remain largely unexplored. From the point of view of control, it enriches the user interface. From the point fo view of exploration, it places a magnifying glass on the active transaction.

Figures 4 & 5 show the richness and immediacy of the user interface when the simulation primitives are accessible during the run phase. In figure 4 the remoteness of the multiphasic design is depicted by the compartmentalization of functions in each phase. Parts of the user interface expressed in one phase are inaccessible in other phases. Also, a mental modality is induced on the user who must always know which phase he/she is in. Figure 5 depicts some of the advantages of the uniphasic design. All functions are always accessible, and the user must be familiar only with a single, highly-integrated environment.

PHASE 1	PHASE 2	PHASE 3
> EDIT	> COMPILE/LINK	> RUN
GENERATE		STOP
ADVANCE		STEP
TERMINATE		
START		

Figure 4

Primitives on Phase 1 Interface

Manipulation of a simulation serves to strengthen the intuition of the analyst. Not only can the dynamics of the simulation be observed, but alternate designs can be partially explored without having to leave the run time environment. Since the overall value of a simulation study improves with the richness of the set of design alternatives, the intuitive side of the simulation process must not be neglected.

There are several points that must be kept in mind, while modifying running simulations. First, when one manipulates a running simulation, the homogeneous conditions required for statistical treatment of the results may no longer exist. Further, changes to the structure of the model may change the standard output reports. It will usually be necessary to know

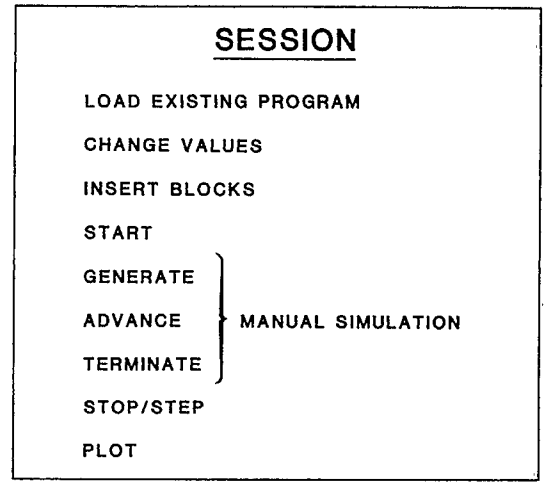

Figure 5

Simulation Primitives on the Session Interface

what changes were made in order to correctly interpret the simulation output.

Conclusions.

The highest design goal of GPSS/PC™ is that of immediacy. In application, this means that user errors are detected and remedied as soon as possible and that the simulation environment is immediately responsive to inquiry and manipulation.

The user interface of GPSS/PC offers a new level of interaction between the analyst and the simulation. The single phase session squeezes much wasted time out of the development cycle and reduces the probability of new errors caused by fixes to old problems. This is possible because GPSS/PC provides for alteration of named values, and for alteration of the model's structure in the middle of a simulation without the need for recompilation.

The new level of controllability of the simulation made available by the manual simulation mode of GPSS/PC gives the analyst the power to manipulate transactions with the same primitives that exist deep within the simulation. In effect, the simulation primitives have been brought up to the user interface. This level of interaction readily reveals the dynamics of the simulation, but it also demands a clear understanding of the simulation in order to be used without error. There are many open design questions related to this new interactive environment. It is the users of the simulation environment who will lead the way.

Reference

Henriksen, James O. 1983. The Integrated Simulation Enviornment (Simulation Software of the 1990s). Operations Research Vol. 31, No. 6, pp. 1053-1073.