DISCRETE EVENT SIMULATION LANGUAGES

Charles M. Shub
Computer Science Department
University of Vermont
Burlington, VT

The current state of the art of discrete simulation languages is reviewed. The available language orientations are delineated, compared, and related. The state of availability of major features including such items as data structures and storage management features, efficient event set management, random variate generation, computational abilities, statistics display analysis, diagnosic facilities, efficiency, interacting, semantic content, graphics, optimization, and other features is chronicled. Recent developments and directions for future developments in several major languages are reported. A bibliography and lists of references in various sub-areas is provided.

1. INTRODUCTION

The state of the art of discrete event simulation languages is quite healthy. There is a wide range of language orientations. There are clear delineations of the future directions of the languages. There is a substantial body of work on existing and useful features. There are comparative evaluations of techniques for handling event set management. There is a body of research on random variate generation. Efforts towards interactive simulations have been reported. Annotated bibliographies exist.

This summary will first briefly describe the major orientations of discrete event simulation languages to indicate the variety of tools available to the discrete modeler. This initial section will also provide the reader with references to the papers charting the developments of the approaches, important works describing the major simulation languages, a brief description of the relationships among the approaches and languages, and some comparisons.

Following this orientation and background portion will be a section describing general features and facilities of simulation languages, the extent to which they have been implemented, and possible mechanisms for implementation of desired features. First, the list of features to be considered will be delineated along with a brief justification for inclusion. Then each feature will be described in greater depth along with a presentation of the current state of development in that particular area.

Having described the state of the art from a feature and facility perspective, this summary will examine several languages as to the directions in which the languages appear to be moving.

Throughout this review, there will be many references to previous works. The abundance of related research and development works provides a strong basis for much of what is reported here. The one unfortunate thing about the previous work is that all too little of it involves comparisons. Thus the bibliography contains a significant number of citations. Subsequent to the listing of the citations themselves is an index by category to the referenced works.

2. ORIENTATIONS OF LANGUAGES

2.1 Introduction

TH0079-4/80/0000-0107$00.75 C 1980 IEEE.
Simulation with Discrete Models: A State-of-the-Art View
T.I. Oren, C.M. Shub, P.F. Roth (eds.)

Discrete event languages can be stratified, according to Shannon [SHAN75], into four major categories. He lists these categories as Transaction Flow, Process Oriented, Event Oriented, and Activity Oriented. These different categories provide different mechanisms or frameworks for the modeler to describe a system. The earlier historical development of the orientations is chronicled by Kiviat [KIVI67] who indicates the initial dichotomy which was that of the earliest block diagram languages versus the statement oriented languages. Jean Sammet also describes many simulation languages in her book on programming languages [SAMM69]. More recently, Adam and Dogramacı [ADAM79] address the comparisons with their own extensive descriptions followed by in depth reports by the developers of the major languages. Oren provides an annotated bibliography [OREN74] which includes additional material.

The following will serve to introduce the major orientations as given in Shannon [SHAN75]:

A) The Transaction flow languages, GPSS and BOSS for example, involve specifying a sequence or several alternative sequences of actions for units in the system by means of a block flow diagram.
B) The process oriented languages, SIMULA for example, describe a system as a set of processes competing for resources, somewhat akin to, though apparently developed independently from, the current process view of computer operating systems [SHAW74, DIJK68].
C) The event oriented languages, SIMSCRIPT, the GASP packages and the new PASCAL -Based languages for example, describe systems by delineating event algorithms which occur instantaneously (in zero simulated time) at various points in time during the period of system modeling. These algorithms describe how the status of the system changes at that point in time.
D) The Activity Oriented languages, HOCUS, CSL and ESP for example, provide for description of activities which can occur and the conditions under which they can occur.

Others indicate different classification. SLAM (via an activity network similar to PERT) accomodates transaction flow.

Roberts [ROBE80b] observes, "The distinction between transaction languages and process languages is not clear and many, many confuse the two. Transaction languages have a very rigid specific structure where entities graphically flow through blocks which activate predefined processes. On the other hand, process languages allow for a variety of processes to be modeled and do not have the same rigid formation...[of] transaction languages." He goes on to observe "Activity languages are not especially different from transaction oriented Languages. The difference is that activity languages require closed loop models, whereas transaction languages can be open models. Furthermore, the transaction languages usually permit a secondary entity such as a facility/resource/server whose flow is not explicitly modeled in the block diagram. This secondary entity can interrupt and require synchronization. The approach is different in the activity languages where all active entities must be represented explicitly by transactions and incorporated in the block diagram." Proponents of the various approaches, and even specific languages, will often argue that their modeling tool is better than others. Shannon [SHAN75] provides language selection criteria which transcend these arguments.

2.2 Transaction Languages

By far and away the most popular and widely used transaction flow language is GPSS. There are over 30 dialects of the language and a number of substantial treatises on the use of the language in modeling exist. This brief article could not hope to do justice to GPSS. Major text books on GPSS include (in alphabetical order) Bobillier, et.al. [BOBI76], Gordon [GORD75] and Schriber [SCHR74]. Furthermore, Gordon [GORD79] provides a review of the development of GPSS.

The transaction flow languages relate quite nicely to the other types of languages. Intrinsically they are process languages in the sense that the block diagram is a flow chart type description of a process. However, they do differ from the statement languages in the sense that they (at least in the majority of versions) lack constructs (blocks) for detailed examination of and communication with other independent units in the system. This is related to the simplified syntax of the available statements in these languages.

2.3 Process Languages

SIMULA is one of the most popular process oriented languages. The best treatments of SIMULA are the book by Franta [FRAN77a] and the original paper by Dahl and Nygaard [DAHL66]. Consolidated Analysis Centers, Inc has developed capabilities for handling processes and resources which allows SIMSCRIPT to also be considered as a process oriented language [RUSS76b]. Also, SLAM [PRIT79a] provides a process capability. SOL, another process oriented language developed by Knuth [KNUT74], has been extended to the system 370 [GUFF75]. The treatments in the literature indicate that even though the description of the model is in terms of cooperating processes, the simulation software or run time support of the language does in fact convert to a next event oriented implementation. In fact, the documentation of the SIMSCRIPT implementation describes the system generated events and waiting lines used in the conversion.

2.4 Event Languages

The class of event languages is quite large. It includes SIMSCRIPT, a number of brand new PASCAL-based

languages, and many, many subroutine packages of which the GASP/SLAM/Q-GERT series are most widely available and best documented. Kiviat's original paper [KIVI68] and his book with Villenueva and Markowitz [KIVI73] provide the details of the newest version of SIMSCRIPT. Additional information can be found in Chao [CHAO71]. Markowitz [MARK79] provides a summary of the various versions of SIMSCRIPT. Major developments in the PASCAL-based languages which appear in recent literature include the work of Johnson [JOHN79] and Bryant [BRYA80]. Both have been concerned with making event languages available on small Minis and Micros. Pritsker has written several excellent books on the GASP packages [PRIT69, PRIT74, PRIT75,PRIT79] which along with the thesis by Hurst [HURS73], the work with Pegden, [PEG79, PRIT79a] and the status report [PRIT79c], summarize recent developments in that area. It should be noted that while the other languages (with the exception of earlier versions of the PASCAL-like languages) are full blown languages which are either compiled directly or sifted into some embedding language for translation, the GASP and SLAM packages are in fact a set of subroutines called from within some procedure oriented language such as FORTRAN or PL/1 i.e. the user is really using FORTRAN or PL/1.

2.5 Activity Languages

Much of the work in activity languages has come from Great Britain at the University of Birmingham which has developed the Extended Control and Simulation language. The techniques are described by Hutchinson [HUTC75a, HUTC75b], Buxton [BUXT66] and Clementson [CLEM73a, CLEM73b]. The work of Hutchinson includes a description of CAPS which is a software package which assists in programming the simulation.

2.6 Comparisons

Pragmatically, the orientation or category of the language or subroutine package serves more to give a convenient framework or viewpoint for the simulation modeler to describe the system he wishes to investigate than any other purpose. Despite this, a number of papers and books discuss the relationships and make comparisons among language types. Normally, these comparisons are based upon specific problems or classes of problems and the conclusions drawn do not always generalize. Those papers which attempt to explain difference among languages without making relative merit type of judgements are usually more informative and/or useful. The earliest comparisons are by Krasnow and Merkallio [KRAS65] who look at general simulation, and by Tocher [TOCH65]. Chronologically next, Kiviat [KIVI67] looked at this in his chronicaling of language development from birth to 1967. Krasnow [KRAS69] did a comparison of GPSS, SIMULA, and SIMSCRIPT in 1969 emphasizing the different orientations as being different world views. Earlier, Teicherow and Lubin [TEIC66] provide a technique based comparison. Kay has written several papers inventorying and comparing languages [KAYI71, KAYI72, KAYI75]. More recent comparisons in addition to those found in most of the major simulation texts [FISH73, FRAN77a, GORD78, GRAY80, MAIS72, NAYL66, SCHM70, SHAN75] are the methodology comparison of Shub [SHUB78], the application directed comparison restricted to SIMULA and GPSS of Atkins [ATKI80] and the performance comparison by Scher [SCHE78]. Some authors of these comparisons can not resist the temptation to pontificate and make value judgements of relative merit. Any such judgement necessarily reflects the author's biases and should be viewed as equivalent to saying that Baldwin Apples are better than Mackintoshes. The important point, which Sammett [SAMM69] makes so well, is that the orientations are different and the choice that needs to be made is one of selecting the best tool for a particular problem rather than adapting an inappropriate modeling orientation to a given problem.

3. LANGUAGE FEATURES

3.1 Introduction

At the 1976 Winter Simulation Conference Professors Miller and Morgan of the Wharton School chaired a Panel Session on the scene in Simulation Languages in 1976 [MILL76]. The panel included A. Alan B. Pritsker [PRIT76], Ed Russell [RUSS76a] and Julian Reitman [REIT76].

Miller and Morgan delineated the following features which they indicated were expected by language users as early as 1976.

1. Data Structures and Memory Management

   A) Representation
   B) Ordering and Grouping
   C) Creation and destruction of data

2. Time management including event sequencing

3. Sampling from probability distributions

4. Computational capabilities

5. Data collection and statistics

6. Debugging or verification facilities

7. File access systems

8. Partial compilations

Further, they cited the need for improvements and further developments in:

1. Debugging interactively

2. Statistics

3. Interfacing with an existant data base

4. Graphics

The anthology edited by Adam and Dogramaci [ADAM79] has an introduction which echoes the concerns of Miller and Morgan. Furthermore, it suggests as additional concerns those of optimization (in the sense of seeking and optimal solution) and efficiency, both from a programming and a processing view. Markowitz, in a paper in the anthology [MARK79], augments the wish list with continuously changing attributes, a more wholesome process view, style and its relation to content (thus including documentation efforts).

The summary here will consider the data structures and storage management features, the simulation time management function including the type of clock and algorithms for event set management, the current state of random number generation within the languages, the computational abilities within the languages, the language facilities for statistics display and analysis, the user friendliness and data input, the notion of efficiency as it relates to processing programs not only in translation but also in execution, interacting with simulation programs during execution, the semantic content of the programs, and other areas.

These topics reflect an assimilation of the subjects outlined above. The assimilation was made to allow cohesive treatments of related areas. The topics were deliberately restricted to exclude the issue of ease of modeling with a language and the modeling philosophy.The philosophy of modeling is embedded within and is the orientation of the language. The relative merits of the languages and approaches will continue to be debated.

3.2 Data Structures

Most of the major simulation languages have data structuring capabilities. The major development in the area of data structures is the general awareness within the simulation community of the issues of complete declarations and strong typing. Strong typing first became prevalent in PASCAL [WIRT71] and has been commented upon in the simulation literature by Markowitz [MARK79], Bryant [BRYA80], and Johnson [JOHN79].

First, and perhaps not only less severe in its constraints but also more useful, is a requirement that each identifier be explicitly defined or declared. The arguments for and against this notion have received wide attention in the programming language literature and need not be delineated here. Such a requirement in the context of simulation can, according to Bryant [BRYA80], be quite useful. He gives examples of misspelling errors which can be detected at program translation time by such a requirement. SIMULA, the PL/1 based GASP, and the PASCAL-based languages do require the explicit declaration of all identifiers while SIMSCRIPT, GPSS and the FORTRAN based GASP packages do not.

The major constraint of Strong Typing is that any identifier can normally be used in at most one context. If the program has two distinct types of data structures, then two distinct sets of identifiers must be used to refer to the structures. The argument in favor of this is that such a requirement allows detection during the program translation of the "comparing apples with oranges" types of error. Again, Bryant [BRYA80] and Markowitz [MARK79] provide examples. The only argument which can be made against requiring this is that it prohibits the use of an identifier such as FRUIT when one wishes to refer to some common attribute of either APPLES or ORANGES. This can tend to make the programming somewhat more complex in such cases. At present, the second strong typing constraint is available only in the PASCAL-based simulation languages.

3.3 Time Management

In the area of simulation time management there are two issues. The first is that of an integer versus real valued simulation clock, and the second is the management of the future event set. Gordon, the developer of GPSS, indicates that the motivation for the integer clock in GPSS came from the desire to assure reproducibility of results despite round off errors and minor changes in the model [GORD79]. Gordon's admission that he would advocate use of a real clock in a revised GPSS [GORD79] is justified quite well in terms of better understanding in the simulation community of the requirements for and meaning of reproducibility.

The other time issue is far more complex. McCormack and Sargent [MCCO79b] delineate many algorithms for event set management and report on comparisons under a variety of circumstances. The topic has received substantial attention. Comfort [COMF79] also provides a taxonomy and analysis which is independent of McCormack and Sargent. Comfort makes different assumptions. Both comparisons evaluate the algorithms of Franta and Maly [FRAN77b,FRAN78], Gonnett [GONN76], Henrikson [HENR77a], and Wyman [WYMA75]. Also they both draw upon the earlier comparisons of Vaucher and Duval [VAUC75]. Comfort includes the work of Ulrich [ULRI78] while McCormack and Sargent consider not only the Steady State analyses of Davey and Vaucher [DAVE76] but also the time indexed list method [ENGL78] as influenced by Laughlin [LAUG75], the Priority Queue method [JONA75], the subcalenders of SIMSCRIPT [KIVI73, TANE76], the distribution of times given by Vaucher [VAUC76] the classic texts of Knuth [KNUT68, KNUT73], and the Doctoral Dissertation of McCormack [MCCO79a] which apparently provided much of the basis for their work.

In essence, the event set management problem can be abstracted to developing a generalized data structure for efficient insertions into priority ordered lists and efficient removal of the highest priority item. The distribution of the priority values and the sequences of insertion and removal vary from one application problem to another. McCormack and Sargent conclude, as one would expect, that no single algorithm is always the best. Despite Vaucher's algorithm doing best overall in their study, they point out that they did not, and could not, test all possibilities. They express concern about the sensitivity of Vaucher's algorithm to interval sizes and the necessity to specify them. They suggest maintaining a pointer to the middle of event list and after the single halving of the list at the middle pointer, a linear search from the front of the appropriate half of the event list as a good compromise which will usually be relatively efficient.

Both summaries make suggestions for future research in event set management, though neither suggests pursuit of a truly adaptive algorithm which could analyze the event set and automatically select or change algorithms in mid-simulation.

3.4 Generating Random Numbers

The state of the use of random number generation in discrete languages is robust. Published research generally addresses two issues:

1. Generating random variates from diverse distributions
2. Quality of randon number sequences generated.

The October 1976 issue of Simuletter [HIGH76] provides a summary of many generation techniques. Tadikamilla [TADI78] delineates the generation of Gamma distributed variates, and with Schreiber [SCHR77] considers the Weibull distribution as well. Cheng [CHEN78] gives algorithms for Beta Variates with non-integral shape parameters. Burford [BURF78] not only considers Erlang Variates but also relates the quality of the variates to the underlying uniform generation scheme. Schmeisser [SCHM78] looks at the general problem of generation and Kisco [KISC76] provides an automatic procedure for producing piecewise linear cumulative distribution functions for empirical observations. Kisco's computer program actually generates the GPSS function description statements.

In the area of random variate quality there is the already mentioned work of Burford [BURF78], Babadim and Stohr's work [BABA75] on comparing the generation in different implementations of GPSS and noting that for shorter sequences some seed values behave better than others and the more recent presentations by Saxon and Schreiber [SAXO79] delineating which initial seed values produced higher quality streams.

The need for diverse random distributions is recognized in SIMSCRIPT [KIVI73] which provides ten built in distributions and a convenient form for specifying others. In GPSS a few distributions are built in, and the literature contains additional forms [HIGH76, SCHR77]. The inverse transform method, the rejection method, and composite methods for generation are described in the simulation texts [FISH73, FRAN77a, GORD78, GRAY80, MAIS72, NAYL66, SCHM70, SHAN75]. Fishman's new text [FISH78] is a cut above the norm in this area. A bit more detail on the mechanics of generation and testing on random numbers than is found in the texts is provided by Knuth [KNUT69].

3.5 Computional Ability

This need is met in most all the statement oriented languages. GPSS, however, has no facilities for major computations though more recent versions do have the added feature of floating point variables. The two cures for this involve allowing GPSS the ability to link to other languages through HELP blocks or embedding GPSS constructs into a general purpose language. Gordon [GORD79] points this out clearly and delineates both approaches. He remarks that initially he had expected the HELP block not to be extensively used, but events proved otherwise. He suggests that abandonment of the interpretive mode of execution for GPSS in favor of a translate and execute scheme would allow for a much cleaner HELP block implementation and permit extensions in both directions. Wimmert [WIMM80] has published a paper describing the details of one such HELP block callable FORTRAN Subroutine package which performs a range of tasks. The most valuable portion of his paper is the delineation of the mappings from GPSS Standard Attributes to FORTRAN Array Subscripts. Use of such packages allows for at least the following:

1. Generating additional distributions
2. Providing additional random number streams
3. Initializing save values at run time rather than translation time
4. Formatting of output
5. Statistical analysis

Closely related to this area, though the major discussion will occur in the efficiency section is the work of Henrikson [REIT80] in the developing of a Translate and Execute version of GPSS which provides the cleaner implementation of HELP blocks.

In terms of embedding GPSS within a language, Rubin [RUBI80] discusses the design concepts involved in doing the embedding with a language building utility program. The successful embedding of GPSS within APL has received some attention as well [IBMC76].

3.6 Statistics as related to Simulation

From a language point of view, the statistical tools needed include facilities to collect and display statistics. Packages and procedures for the statistical analysis of simulation output have received considerable attention elsewhere and will not be discussed in this context other than to reference the papers of Hurwitz and Quirk [HURW76] and that of Lewis [LEWI57] which can be of considerable assistance and do indeed provide a fresh insight.

Most languages do have facilities for the automatic collection and display of certain statistics. Palme [PALM75] describes putting statistics into SIMULA programs. Markowitz, et.al. [KIVI73, MARK62] describe the evolution of the techniques in SIMSCRIPT. Perhaps the most important development (from a programming language point of view) is the use of monitoring functions and left handed functions as a language feature to build the statistics gathering facilities into the SIMSCRIPT language. Conceptually, a monitored variable is declared as such in the source program. The compiler, upon recognizing the use of that variable, rather than generating a memory reference generates a call to a function which can take any action it desires at every execution time reference to that variable. Moreover, the particular function called can be specified as being dependent upon whether the monitored variable is being assigned a value or accessed.

SIMSCRIPT uses this feature to allow automatic collection of statistics (time weighted or value weighted). The programmer merely declares which statistics he wants collected on which variables and the compiler takes care of everything. The compiler generates the monitoring functions for that variable and generates the calls to the monitoring function at each reference to the variable. The programmer merely uses the variable in the ordinary sense and the compiler handles the rest of the details. However, the compiler does not cause automatic display of the statistical data upon termination of execution.

The approach in GPSS is rather different. Certain tabular data is routinely automatically collected and then displayed upon termination of execution. The GPSS programmer also has the facility to declare statistical tables whose entries will be routinely displayed upon termination of the program. However, the GPSS programmer must specifically provide for entries to be made into the table at each position in his model where that is desired.

Ideally, a merging of the two techniques would be useful. For example, it would be convenient to be able to globally declare not only that you want statistics accumulated on a particular variable, but also that you want display of those statistics every so many units of time, or more importantly, whenever certain conditions are met. The current state of the languages allow this to be accomplished indirectly by explicit design by the programmer of monitoring routines to perform that function. However, the languages could certainly be extended to include such specifications. The development in SIMSCRIPT was to use the concept of a monitored variable to make the collection of statistics declarable rather than executable, and the natural corresponding extension is to make the same transformation in terms of display of statistical data. Newer versions of the other languages can have similar features built in. When SIMPAS [BRYA80] is implemented as a language rather than a preprocessor, that language could include these extensions.

GPDS [XERO72] and GPSS10 [MART78] include a SSTATE card which allows the programmer to terminate simulation when a particular Standard Numerical Attribute value remains within a delineated range for a specified period of time. A more general similar feature, provided through a HELP block interface, is described by Andrews and Schriber [ANDR78] and will be discussed more fully in the section on interactive simulation. Other than this, the only tools for doing any sort of statistical analysis are those developed by specific users of the languages.

3.7 User Friendliness

Within the general category of User Friendliness fall a number of related issues including:

1. Verification or debugging,
2. Translation and to some extent run time efficiency,
3. A subset of the interactive facilities which can be used as debugging tools, and
4. Ease of input to the model.

The tools and techniques one is concerned with are quite extensive and include:

1. Compiler diagnostics,
2. Execution diagnostics,
3. Interactive debuggers, and
4. Format-free I/O routines.

Simulation languages stand up quite nicely in terms of compiler diagnostics. Generally one finds capabilities for cross referencing names and a variety of consistency checks. Bryant [BRYA80] argues persuasively for the additional compile time checking as in the PASCAL family. These facilities are described in more detail by Wirth [WIRT71]. Markowitz, one of the developers of SIMSCRIPT, also suggests these extensions as a possible direction for that language [MARK79]. Unfortunately, the inclusion of additional compile time diagnostic capability makes the translation a much more complex process. The increased capability for compile time consistency checking comes with the constraint that all the parts of the program which are being checked for consistancy must be compiled together. For example, how can a compiler assure that a subroutine or a procedure is being called with the right number and types of arguments if the compiler does not have the subroutine text to scan so that it can know what arguments the subroutine expects? Such difficulties have plagued the FORTRAN type programs for quite some time. Markowitz also calls for partial translations ala the style of FORTRAN though the general trend in programming languages seems to be the reverse. Any person who has had to recompile an entire 4000 statement program because of one errant comma can easily agree with that view. Unfortunately, provisions for partial compilation in any programming language with concepts such as block structure, differing scopes of variables, declarations which generate performable code, and the like, are just plain difficult.

The crucial point is that the compile time diagnostic enhancements must come at the expense of a less efficient translation process. You can have one or the other but not both. A partial solution might be to borrow from certain translators for other languages and allow the amount of compile time checking to be a compiler option. This extension is non-trivial and raises a host of research issues that are being pursued in the programming language field. The major issue is one of compatibility, usually between those portions of the program translated with one set of options and those portions translated with another.

The run time diagnostic facilities vary widely and run from none at all to trace information display to true interactive debuggers which are, or can be, a subset of an interactive simulation capability. At the worst, the run time debugging is at the core dump level assisted by manual insertion of WRITE statements within the program. The next level of sophistication includes trace data provided within the language such as the transaction flow traces in GPSS or the reverse history of procedure call nesting available in many procedure type languages. The GPSS facility is a little bit more sophisticated in that it gives a sequence of flow rather than static display of where one is and how one got there. With access to the timing routine, the statement oriented languages can be extended to provide this sort of capability as well. One of the SIMSCRIPT texts [KIVI73] describes the built in facility to allow a user to construct his own dynamic trace, but the tracing of the event sequence is not part of the language. The highest and most sophisticated level of debugging is the true interactive debugger. Simulation languages (with the exception of GASP) are not particularly nice in this area. The notable exceptions are Jim Henrikson's work on an interactive debugger for GPSS [HENR78, HENR77b, REIT80], GPSS10 [MART78], Norden GPSS [NORD78], and the interactive debugger on Xerox GPDS [XERO72]. All allow setting breakpoints and examining program variables and the like. Apparently none interfaces on a source statement level, but rather on a block number and transactions number level. However, the GPSS10 debugger also allows specifying the debugging arguments by name rather than numeric equivalent. The GASP packages do nicely in terms of interactive debugging because the existent features of the debugger for the embedding language can be used.

Many versions of procedure oriented languages do have true interactive debuggers working at the source statement level. The TERAK computer has software developed at Cornell University that provides an excellent such program development tool for Cornell's PL/1 student subset. Several machines allow interactive debugging of FORTRAN at the source statement level. Many operating systems do have machine instruction level interactive debuggers. However, use of these tools generally requires a

detailed knowledge of not only the machine's instruction set, but also the implementation details of the simulation language on that machine.

Judging from the progress made in programming languages one would suspect that these bells and whistles and conveniences will be available for simulation languages in the not too distant future. At the conceptual level, most of the problems are solved, and provision of the better debugging tools, and/or the associate friendlier compilers is a matter of implementation rather than solving basic problems.

Finally, just about all of the statement type languages have some form of format free input. The GASP packages are currently having this feature added. [PRIT79c]

3.8 Efficiency

Closely related to the user friendliness issue is the issue of efficiency in terms of the translation and execution speed of simulation programs. The trade off between fast compilation and compile time error checking has been touched upon already. The insertion of run time efficiency into simulation languages can be done in two areas. Frequently used algorithms can be analyzed and made more efficient as has been the case for such areas as event set management and random variate generation. More importantly, the compilers can be rewritten to produce more efficient code. Again, these capabilities have been developed and are available in many procedure oriented languages. The code optimization techniques of unfolding, common subexpressions, reducing operations, and removing instructions from loops can be inserted into the compilers for simulation languages quite easily. This should be done as a compiler option. Should the simulation language also be a system development language, then algorithms developed in the simulation of a computer system could be embedded within the operating system without rewriting.

The other obvious change is to provide not only a translate and execute version (for efficiency), but also an interpretive version (for debugging). Traditionally GPSS has been interpretive. Henrikson's translate and execute version has been reported to have speeded up the execution by a factor of up to ten in certain applications [REIT80].

In this area, and the user friendliness area the direction is clear. The modifications need not be so much to the languages themselves as they need to be made to the compilers. The languages are pretty much current technology languages. The enhancements will be made by changing the language processors.

3.9 Man in the Middle

This section is concerned with the capabilities of a person to interact with a simulation program to be able to make decisions, either at a specified epoch, or on an interrupt driven basis.

To some extent, the ability to interface with the simulation at prespecified epochs is generally available on systems which truly support interactive execution via having the simulation program prompt for inputs. An excellent example of this is the output analyzer for GPSS [ANDR78]. This HELP block interface prompts for interactions and allows the user to either stop or collect more data dependent upon statistical information provided by the routine. Most simulation languages do not have the facility to respond to an interrupt signal from an interactive terminal. It would appear that the interface to provide such a capability would be rather simple to design. A possibility would include a set of declarations regarding interrupt driven events and the keyboard interrupt to which they correspond. One would also need to make modifications to the event selection routine to check the interrupt variable values at each cycle. This could be done at the expense of one instruction per allowable interrupt. Such a modification would allow the interrupt driven event to occur on detection of an interrupt signal. The only additional modifications would be the interface between the operating system and the program to set the interrupt variable.

There are many applications for such a capability which present strong arguments for inclusion for such a facility. For example, a user of a simulator of a supermarket could be watching updated values of line lengths and based upon that, interrupt the progress of the simulation to open an additional check-out line. The most prevalent context for such usage, however, is in terms of interactive graphics capabilities. Early mention of the use of graphics in simulation dates back to 1971 [RAHE71]. Roberts and Hodges [ROBE80a] describe existant systems not written in a simulation language. More recently, Favreau [FAVR79] reports on the April, 1979 meeting of the Western Simulation Council devoted to that topic.

The future will see simulation languages augmented to allow for inclusion of such facilities. The technology for adding an interrupt based event exists and it can and will be done.

3.10 Semantic Content

In this category is the general notion of meaning and understandings of Simulation models. A computer program has a precise formal meaning to the computer. Humans rely on their intuitions and common

sense and by nature are less formal than machines. Thus that which is complete formal, exact, and precise to a machine may not always be well understood by people. This is why semantic information in programs is so important. In the area of semantic content, the efforts are being directed towards better understanding of the computer programs, not only by programmers, but by managers and users as well. Efforts are in two general areas, though they both relate to documentation as an aid to understanding what a computer program does. Schriber [SCHR74] gives a documentation protocol for GPSS. Cooley [COOL77] describes documentation for management. Highland [HIGH77, HIGH79] describes a taxonomy approach. Nance and Roth [NANC77] surveyed the prospects and problems in 1977. The following year saw Sibley [SIBL78] addressing documentation for management. Gass [GASS78] provides a detailed description of the stages of and reasons for documentation. He makes detailed suggestions as to an approach. Cooley [COOL79] reiterated her concerns. Metcalfe [METC79] echoes the need for documentation. Most recently, Nance [NANC79] summarizes efforts in this area and forecasts the prospects for developing documentation standards. All of these efforts have been in addition to the program itself.

Within the languages, the efforts have mostly been towards using more meaningful names by programmers. SIMSCRIPT has gone one step further and has adopted a language syntax which expresses the concepts within the syntax. Markowitz [MARK79] addresses the topic of relating style to content and the use of both to make programs more readable. He argues for additional syntactic structures within SIMSCRIPT to make statements even more meaningful.

3.11 Other Areas

Most of the work in optimum seeking simulations has been in continuous or combined simulation. The major recent work in optimal seeking facilities for discrete languages has been in the GASP [PEGD77] and SLAM [PEGD80] packages. This body of work represents implementation of pioneering work of Smith [SMIT73,SMIT76]. The notion of putting simulations in a loop has also been explored for GPSS by Lefkowits and Schriber [LEFK71]. Items such as run length and how many runs to make are controlled by the optimizing packages. Farrell [FARR77] gives a more thorough description on the general topic of optimization.

The interface with the database functions of an organization can be done with current technology, however it is a technology of interface rather than a feature of a language. That languages do not incorporate this is rather surprizing in light of the similarity between the various data structures used in both simulation and databases. Despite the persuasive arguments by Markowitz [MARK79] there has been no reported progress on implementation of truly continuously changing attribute values though several languages do provide for differential changes.

4. LANGUAGES (in Alphabetical Order)

4.1 GASP/SLAM/Q-GERT

The status of GASP was chronicled by Pritsker in 1979 [PRIT79c]. He describes the modeling philosophy being that of a subroutine package to provide common simulation problem software support in the context of FORTRAN or PL/1. As such, GASP has available all the nice debugging features of those languages. However, it suffers from the drawbacks of these languages. Since it is a subroutine package, it is quite flexible, and has a wide variety of applications. Pritsker indicates that the major recent developments are in the user interface portion of the language and includes format free input, control of output for either paper or terminals, graphics equipment interfaces, and the like. Pritsker indicates that the true interrupt driven simulation facilities as described in section 3.9 of this report have been incorporated in his packages.

4.2 GPSS

The major developments in GPSS are indicated by Gordon [GORD79] and Reitman [REIT76]. They include the translate and execute version and their resultant efficiency gains [REIT80], the packages to allow better use of the HELP blocks [ANDR78, WIMM80], and the interactive debuggers [HENR77b, HENR78, LEFK71, NORD78].

4.3 PASCAL Based Languages

The PASCAL based languages [BRYA80, JOHN79] are quite new and still in the fairly early development stage. They bring the advances in programming language design technology to simulation languages. To be sure, they retain the block structure and recursive capabilities of the ALGOL like languages. They bring the strong typing constraints and additional structuring developments of the past decade to the simulation programmer. They are, or will be available on small systems, thus bringing simulation capabilities to the microprocessor family of equipment. They lack some of the nice monitoring facilities of SIMSCRIPT, but that will come with time.

While the PASCAL based languages do not have the widespread use of the other languages and a portion of their glamor is certainly attributable to the parent language rather than simulation, they must be considered for two reasons. First, they represent a true "third generation" simulation language. Secondly, simulation research dollars are being used to further their development. A portion of

the funding of the two reported efforts has been in the form of Development grants by the Annual Simulation Symposium.

4.4 SIMSCRIPT

Markowitz describes SIMSCRIPT past, present, and future quite extensively [MARK79]. The global statistics gathering exists, there are the elements of structured programming, and he has set out the future directions. He crystallized the directions and touches on their solution. Most of his suggestions have been addressed in a fair amount of detail in Section 3 of this report.

5. CONCLUSIONS

The state of discrete event languages has been charted with respect to a number of existent and desired features. The author has attempted to hide his own prejudices about what he feels is desirable or undesirable and to summarize and echo the view of others. With the variety of orientations available, and the variety of choices within each orientation, it would be impossible to have chronicled all features of all languages. The languages selected for major emphasis were selected either because of their wide use or because of their conceptual newness or elegance. No one language has been described in detail, but the features have been covered. Sufficient references have been cited to allow the interested reader to pursue any area in further depth.

6.1 BIBLIOGRAPHY

The following abbreviations are used.

AW Addison Wesley Publishing Company
Reading, MA 01867

CACI Consolidated Analysis Centers Inc.
12011 San Vicente Blvd.
Los Angeles, CA 90049

CACM Communications of The Association for Computing Machinery
1133 Avenue of the Americas
New York, NY 10036

MS Management Science
The Institute of Management Sciences
146 Westminister Street
Providence, RI 02903

OR Operations Research Society of America
428 E. Preston Street
Baltimore, MD 21200

PH Prentice Hall Publishing Company
Englewood Cliffs, NJ 07632

SCSC Proceedings of the Summer Computer Simulation Conference
(available through SCS)

SIM Simulation-Technical Journalfor the Society for Computer Simulation
Box 2228
La Jolla, CA 92037

TAMPA Proceedings of the Annual Simulation Symposium
Box 22621
Tampa, FL 33622

WILEY John Wiley & Sons, Inc.
605 Third Avenue
New York, NY 10001

WSC Proceedings of the Winter Simulation Conference
(Available through ACM)

6.2 ALPHABETICAL LISTING

ADAM79 Adam, N.R. and Dogramaci, A. (Eds)
"Current Issue in Computer Simulation"
Academic Press 1979

ANDR78 Andrews, R.W. and Schriber, T.J.
"Interactive Analysis of Output from GPSS-Based Simulations"
WSC 1978, pp. 267-279

ATKI80 Atkins, M.S.
"A Comparison of SIMULA and GPSS for Simulating Sparse Traffic"
SIM Volume 34, #3, March 1980, pp. 93-98

BABA75 Babad, J.M. and Stohr, E.A.
"The Effect of Different GPSS Random Number Generators on Simulation Results"
SL Volume 6, #4, July 1975, pp. 55-73

BOBI76 Bobillier, P.A., Kahan, B.C. and Probst, A.R.
"Simulation With GPSS and GPSSV"
PH 1976

BRYA80 Bryant, R.M.
"SIMPAS: A Simulation Language Based on PASCAL"
WSC 1980

BURF78 Burford, R.L.
"Additive Multiplicative Uniform Pseudo Random Number Generators in the Generation of Erlang Variates"
WSC 1978, pp. 909-918

BUXT66 BUXTON, S.N.
"Writing Simulations in C.S.L."
Computing Journal, Volume 9, August 1966

CHAO71 CHAO, Y.W.
"Simulation with SIMSCRIPT"
WCS 1971, pp. 268-269

CHEN78 Cheng, R.C.H.
"Generating Beta Variates with Nonintegral Shape Parameters"
CACM Volume 21, #4, April 1978

CLEM73a Clementson, A.T.
"Computer Aided Programming for Simulation"
University of Birmingham 1973

CLEM73b Clementson, A.T.
"Extended Control and Simulation Language"
University of Birmingham 1973

COMF79 Comfort, J.C.
"A Taxonomy and Analysis of Event Set Management Algorithms for Discrete Event Simulation"
TAMPA 12,1979, pp. 115-146

COOL77 Cooley, B.
"Documenting Simulation Studies for Management Use"
WSC 1977, pp. 742-747

COOL79 Cooley, B.
"Documenting Simulation Studies for Management"
SL Volume 10, #3, Spring 1979, pp. 24-28

DAHL66 Dahl, O.J. and Nygaard, K.
"Simula - An Algol-Based Simulation Language"
CACM Volume 9, #9, 1966,pp. 671-687

DAVE76 Davey, D. and Vaucher, J.
"An Analysis of the Steady State Behavior of Simulation Sequencing Set Algorithms"
University of Montreal 1976

DIJK68 Dijkstra, E.A.
"The Structure of T.H.E. Multiprogramming System"
CACM Volume 11, #5 May 1968

ENGL78 Englebrecht-Wiggins, R. and Maxwell, W.L.
"Analysis of the Time Indexed List Procedure for Synchronization of Discrete Event Simulations:
MS Volume 24, 1978, pp. 1417 - 1427

FARR77 Farrell, W.
"Literature Review and Bibliography of Simulation Optimization" WSC 1977, pp. 116-125

FAVR79 Favreau, R.R.
"Computer Graphics in Simulation" SIM Volume 33 #1, July 1979, pp. 34-36

FISH73 Fishman, G.
"Concepts and Methods in Discrete Event Digital Simulation" Wiley 1973

FISH78 Fishman, G.
"Principles of Discrete Event Simulation" Wiley 1973

FRAN77a FRANTA, W.R.
"The Process View of Simulation" North Holland 1977

FRAN77b Franta, W.R. and Maly, K.
"An Efficient Data Structure for the Simulation Event Set"
CACM Volume 20, #8, August 1977, pp. 596-602

FRAN78 Franta, W.R. and Maly, K.
"A Comparison of Heaps and the TL Structure for the Simulation Event Set"
CACM Volume 21, #10, October 1978, pp. 873-875

GASS78 Gass, S.I.
"Computer Model Documentation"
WSC 1978, pp. 281-288

GONN76 Gonnett, G.H.
"Heaps Applied to Event Driven Mechanisms"
CACM Volume 19 #7, July 1976, pp. 417-418

GORD75 Gordon, G.
"The Application of GPSS to Discrete System Simulation"
PH 1975

GORD78 Gordon, G.
"System Simulation"
PH 1978

GORD79 Gordon, G.
"The Design of the GPSS Language"
Appears in [ADAM79]

GRAY80 Graybeal, W. and Pooch, U.W.
"Simulation: Principles and Methods"
Wintrop 1980

GUFF75 Guffee, C.O. and Ulfers, H.E.
"SOL 370"
SCSC 1975, pp. 1-11

HENR77a Henrikson, J.O.
"An Improved Events List Algorithm"
WSC 1977, pp. 546-557

HENR77b Henrikson, J.O.
"An Interactive Debugging Faciltiy for GPSS"
WSC 1977, pp. 330-339

HENR78 Henrikson, J.O.
"An Interactive Debugging Facility for GPSS"
SL Volume 10 #1, Fall 1978, pp. 60-67

HIGH76 Highland, H.J. (Ed)
"Randomness and Random Number Generators"
SL Volume 8, #1, October 1976

HIGH77 Highland, H.J.
"A Taxonomy Approach to Simulation Model Documentation"
WSC 1977, pp. 724-728

HIGH79 Highland, H.J.
"A Taxonomy Approach to Simulation Model Documentation"
SL Volume 10, #3, Spring 1969, pp. 19-23

HURS73 Hurst, N.R.
"GASP IV: A Combined Continuous/Discrete FORTRAN-Based Simulation Language"
Doctoral Dissertation, Purdue University 1973, 263 pages

HURW76 Hurwitz, J.A. and Quirk, D.A.
"Quirks Iota and Hurwitz' a Nu Statistical Measures of I-N-Significance"
SL Volume 8, #1, October 1976, pp.76-77

HUTC75a Hutchinson, G.K.
"Introduction to the Use of Activity Cycles as a Basis for System's Decomposition and Simulation"
SL Volume 7, #1, October 1975

HUTC75b Hutchinson, G.K.
"An Introduction to CAPS - Computer Added Programming and Simulation"
SL Volume 7, #1, October 1975

IBMC76 IBM Corporation
"APL GPSS Form SH20-1942"
White Plains, NY 1976

JOHN79 Johnson, G.R.
"A Portable Discrete Event Simulation Package for Microcomputers"
TAMPA 12, 1979, pp. 27-50

JONA75 Jonassen, A. and Dahl, O.J.
"Analysis of an Algorithm for Priority Queue Administration"
BIT Volume 15, #4, 1975, pp. 409-422

KAYI71 Kay, I.M.
"Digital Discrete Simulation Languages: A Discussion and an Inventory"
TAMPA 4, 1971

KAYI72 Kay, I.M.
"An Over the Shoulder Look at Discrete Simulation Languages"
Proc AFIPS S.J.C.C. 1972

KAYI75 Kay, I.M., Kisko, T.M. and Van Houweling, D.E.
"GPSS/SIMSCRIPT - The Dominant Simulation Languages"
TAMPA 8, 1975, pp. 141-154

KISK76 Kisko, T.M.
"An Automated Method of Creating Piecewise Linear Cumulative Probability Distributions"
WSC 1976, pp. 487-496

KIVI67 Kiviat, P.J.
"Development of Discrete Simulation Languages"
SIM, Volume 8, #2, February 1967, pp.65-70

KIVI68 Kiviat, P.J.
"Introduction to the SIMSCRIPT II Programming Languages"
Digest on 2nd Conference on Application of Simulation, NY December 1968, pp. 32-36

KIVI73 Kiviat, P.J., Villaneuva, R. and Markowitz, H.M.
"SIMSCRIPT II.5 Programming Language"
CACI 1973

KNUT68 Knuth, D.E.
"The Art of Computer Programming"
Volume 1, Fundamental Algorithms
AW 1968

KNUT69 Knuth, D.E.
"The Art of Computer Programming"
Volume 2, Semi-Numerical Algorithms
AW 1969

KNUT73 Knuth, D.E.
"The Art of Computer Programming"
Volume 3, Sorting and Searching
AW 1973

KNUT74 Knuth, D.E. and McNeley, J.L.
"SOL - A Symbolic Language for General Purpose Simulation"
IEEE Transactions on Computers, Volume EC-13
#2 August 1974

KRAS65 Krasnow, H. and Merkallio, R.
"The Past, Present, and Future of General Simulation Languages"
MS Volume XI, #2, 1965, pp. 236-267

KRAS69 Krasnow, H.S.
"Simulation Languages"
in "The Design of Computer Simulation Experiments"
Duke University Press, Durham, NC 1969, pp. 320-346

LAUG75 Laughlin, G.W.
"Reducing the Run Ration of a Multiprocessor Software System Simulator"
TAMPA 8, 1975, pp. 115-134

LEFK71 Lefkowitz, R.M. and Schriber, T.J.
"Use of an External Optimizing Algorithm with a GPSS model"
WCS 1971

LEWI57 Lewis, H.R.
"The Data Enrichment Method"
OR Volume 5, 1957, page 551

MAIS72 Maisel, H. and Gnugnoli, G.
"Simulation of Discrete Stochastic Systems"
SRA 1972

MARK62 Markowitz, H.M., Hausner, B. and Karr, H.W.
"SIMSCRIPT: a Simulation Programming Language"
PH 1962

MARK79 Markowitz, H.M.
"SIMSCRIPT: Past, Present, and Some Thoughts about the Future"
appears in [ADAM79]

MART78 Martin, M.D.
"General Purpose Simulation System for the DECsystem-10"
University of Western Ontario, London, Ontario, Canada 1978

MCCO79a McCormack, W.M.
"Analysis of Future Set Algorithms for Discrete Event Simulation"
Doctoral Dissertation, Syracuse University 1979

MCCO79b McCormack, W.M. and Sargent R.G.
"Comparison of Future Event Set Algorithms for Simulations of Closed Queueing Systems"
appears in [ADAM79]

METC79 Metcalfe, M.A.
"Documentation: A Growing Need, A New Tool"
SL Volume 10, #2, Spring 1979, pp 35-46

MILL76 Miller, L.W. and Morgan, H.L.
"Simulation Language Features in 1976: Existing and Needed"
WSC 1976, pp. 75-78

NANC77 Nance, R. and Roth, P.
"Documentation of Computerized Models: Prospects and Problems"
WSC 1977 page 722

NANC79 Nance, R.E.
"Model Representation in Discrete Event Simulation: Prospects for Developing Documentation Standards"
appears in [ADAM79]

NAYL66 Naylor, T.H., Balintfy, J.L., Burdick, D.S. and Chu, K.
"Computer Simulation Techniques"
Wiley, 1966

NORD78 Anon.
"GPSS/Norden Simulation Language"
National CSS Inc., 1978 Norwalk, CT

OREN74 Oren, T.I. (Ed)
"Annotated Bibliographies of Simulation"
Proceedings of the Simulation Councils
Volume 4, #1, June 1974, SCS La Jolla, CA

PALM75 Palme, J.
"Putting Statistics into a SIMULA Program"
SL Volume 6, #4, July 1975, pp. 39-43

PEGD77 Pegden, C.D. and Gately, M.P.
"Decision Optimization for GASP IV, Simulation Models"
WSC 1977, pp. 126-133

PEGD79 Pegden, C.D. and Pritsker, A.A.B.
"SLAM: Simulation Language for Alternative Modeling"
SIM Volume 33, #5, November 1979, pp. 145-157

PEDG80 Pegden, C.D. and Gately, M.P.
"A Decision Optimization Module for SLAM"
SIM Volume 34, #1, January 1980, pp. 18-25

PRIT69 Pritsker, A.A.B. and Kiviat, P.J.
"Simulation with GASP II"
PH 1969

PRIT74 Pritsker, A.A.B.
"The GASP IV Simulation Language"
Wiley 1974

PRIT75 Pritsker, A.A.B. and Young, R.E.
"Simulation with GASP-PL/1"
Wiley 1975

PRIT76 Pritsker, A. A. B.
"Ongoing Developments in GASP"
WSC 1976, pp. 81-83

PRIT79a Pritsker, A.A.B. and Pegden, C.D.
"Introduction to Simulation and SLAM"
Halsted Press 1979

PRIT79b Pritsker, A.A.B.
"Modeling and Analysis Using Q-GERT Networks"
Halsted Press 1979

PRIT79c Pritsker, A.A.B.
"GASP: Present Status and Future Prospects"
appears in [ADAM79]

RAHE71 Rahe, G.A.
"Simulation and Computer Graphics"
SIM Volume 16, #1, January 1971

REIT76 Reitman, J.
"Ongoing Developments in GPSS"
WSC 1976 page 87

REIT80 Reitman, J.
Private Communication, June 1980

ROBE80a Roberts, D.L. and Hodges, J.M.
"Interactive Graphics for Enhancement of Simulation Systems"
TAMPA 13, 1980, pp. 191-206

ROBE80b Roberts, Stephen D.
Private Communication, July 1980

RUBI80 Rubin, J.
"Imbedding GPSS in a General Purpose Programming Language"
Abstract submitted for WSC, 1980

RUSS76a Russell, E.C.
"SIMSCRIPT II.5 New Directions"
WSC 1976 page 86

RUSS76b Russell, E.C.
"Simulating with Processes and Resources in SIMSCRIPT II.5"
CACI 1976

SAMM69 Sammet, J.E.
"Programming Languages: History and Fundamentals"
PH 1969

SAXO79 Saxon, C.S. and Schriber, T.J.
"Statistically Suitable Initial Multiplier Values for IBM's GPSS V Random Number Generators"
WSC 1979, page 252

SCHE78 Scher, J.M.
"Structural and Performance Comparisons Between SIMSCRIPT II.5 and GPSS V"
Proceedings of the 9th Annual Pittsburgh Conference on Modeling and Simulation, April 1978, pp. 1267-1272

SCHM78 Schmeisser, B.
"Methods of Modelling and Generating Probabilistic Components in Digital Computer Simulation When the Standard Distributions are not Adequate"
SL Volume 10, #1, Fall 1978, pp. 38-43

SCHM70 Schmidt, J.W. and Taylor, R.E.
"Simulation and Analysis of Industrial Systems"
Irwin 1970

SCHR74 Schriber, T.J.
"Simulation using GPSS"
Wiley 1974

SCHR77 Schriber, T.J. and Tadikamalla, P.R.
"Sampling from Weibull and Gamma Distributions in GPSS'
SL Volume 9, #1, Fall 1977, pp. 39-47

SHAN75 Shannon, R.E.
"Systems Simulation, The Art and Science"
PH 1975, 387 pages

SHAW74 Shaw, A.C.
"The Logical Design of Operating Systems"
PH 1974

SHUB78 Shub, C.M.
"On the Relative Merits of Two Major Methodologies for Simulation Model Construction"
WSC 1978, pp. 257-266

SIBL78 Sibley, V.
"Management Oriented Documentation of Simulation"
WSC 1978, pp. 289-296

SMIT73 Smith, D.E.
"An Empirical Investigation of Optimum Seeking in the Computer Simulation Situation"
OR Volume 21, 1973, pp. 475-497

SMIT76 Smith, D.E.
"Automatic Optimum Seeking Program for Digital Simulation"
SIM Volume 27, #1, July 1976, pp. 27-31

STEW73 Stewart, J.P.
"Simulation Language", SIM PL/1
SL Volume 4, #2, January 1973

TAKI78 Tadikamalla, P.R.
"Computer Generation of Gamma Random Variables"
CACM, Volume 21, #5, May 1978

TANE76 Taneri, D.
"The Use of Subcalendars in Event Driven Simulation"
SCSC 1976, pp. 63-66

TEIC66 Teichrowe, D. and Lubin, J.
"Computer Simulation - Discussion of the Technique and Comparison of Languages"
CACM, Volume 9, #10, 1966, pp. 723-741

TOCH65 Tocher, K.D.
"Review of Simulation Languages"
OR, Volume 16, #2, 1965

ULRI78 Ulrich, E.G.
"Event Manipulation for Discrete Simulation Requiring Large Numbers of Events"
CACM, Volume 20, #9, September 1978, pp. 777-785

VAUC75 Vaucher, J.G. and Duval, P.
"A Comparison of Simulation Event List Algorithms:
CACM, Volume 18, #4, 1975

VAUC76 Vaucher, J.G.
"On the Distribution of Event Times for the Notices in a Simulation Event List"
INFOR, Volume 15, #2, 1976, pp. 171-182

WIMM80 Wimmert, R.J.
"GAP - a GPSS/Fortran Package"
TAMPA 13, 1980, pp. 111-126

WIRT71 Wirth, N.
"The Programming Language PASCAL"
Acta Informatica Volume 1, 1971, pp. 35-63

WYMA75 Wyman, F.P.
"Improved Event Scanning Mechanisms for Discrete Event Simulation"
CACM, Volume 18, #6, 1975

XERO72 Xerox Corporation
"Xerox General Purpose Discrete Simulation"
File 1x13, Doc. 901758B Xerox 1972

6.3 SUBJECT INDEX

6.3.1 Books

6.3.1.1 Language Books

GASP/SLAM/Q-GERT: HURS73, PRIT69, PRIT74
PRIT75, PRIT79a, PRIT79b

GPSS: BOBI76, GORD75, MART78
NORD78, SCHR74, XERO72

SIMSCRIPT: KIVI73, MARK62, RUSS67b

SURVEYS: SAMM69

6.3.1.2 Text books

FISH73, GORD78, GRAY80, MAIS72, NAYL66, SCHM70, SHAN75

6.3.2 Bibliographies and Summaries

ADAM79, FARR77, OREN74

6.2.3 Articles

6.3.3.1 Documentation

COOL77, COOL79, GASS78, HIGH77, HIGH79, METC79, NANC77, NANC79, SIBL78

6.3.3.2 Efficiency

LAUGH75, MARK79

6.3.3.3 Event Set Management

COMF79, DAVE76, ENGL78, FRAN77b, FRAN78, GONN76, HENR77a, JONA75, KNUT73, MCCO79a MCCO79b, TANE76, ULRI78, VAUC75, VAUC76, WYMA75

6.3.3.4 Graphics

FAUR79, RAHE71, ROBE80a

6.3.3.5 Interactive Simulation

ANDR78, HENR77b, HENR78

6.3.3.6 Languages

CSL: BUXT66, CLEM73a, CLEM73b, HUTC75a, HUTC75b

GASP: PRIT76, PRIT79c

GPSS: GORD79, IBMC76, REIT76, RUBI80, WIMM80

PASCAL-like: BRYA80, JOHN79

SIMPL/1: STEW73

SIMSCRIPT: CHAU71, KIVI68, MARK79, RUSS76a, RUSS76b

SIMULA: DAHL66

SOL: GUFF75, KNUT74

6.3.3.7 Language Comparisons

ATKI80, KAYI71, KAYI72, KAYI75, KIVI67, KRAS65, KRAS69, MILL76, SCHE78, SHUB78, TEIC66, TOCH65

6.3.3.8 Optimization

FARR77, LEFK71, PEDG77, PEDG80, SMIT73, SMIT76

6.3.3.9 Random Numbers

BABA75, BURF78, CHEN78, HIGH76, KISK76, KNUT69, SADO79, SCHM78, SCHR77, TADI78

6.3.3.10 Statistics

ANDR78, BABA75, HURW76, LEWI57, PALM75

6.3.3.11 Summaries and Surveys

ADAM79, FARR77, KAYI71, KAYI72, KIVI67, KRAS65, KRAS69, MILL76, TEIC66, TOCH65