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

SIMULATION PROGRAMMING LANGUAGES: AN ABRIDGED HISTORY

Richard E. Nance

Systems Research Center and Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061-0251, U.S.A.

ABSTRACT

Knowing history can be protective; we have all heard that those who do not are doomed to repeat it. Considering one well regarded expert's estimate of 137 simulation programming languages (SPLs) created by 1981, many perhaps have already duplicated the numerous mistakes of their predecessors. History can also be informative, instructive and entertaining as hopefully this abridged and differently focused approach can illustrate. Questions concerning the causes for so many SPLs, the remarkably similar parallel developments, and the role of the SPLs versus programming languages in general might admit to historical answers. At the least, sharing speculations could prove enlightening and amusing.

1 INTRODUCTION

This abridged history of discrete event simulation programming languages is taken from the more complete version presented at the History of Programming Languages II Conference in Cambridge, Massachusetts 20-23 April 1993. A preprint appeared in *ACM SIGPLAN Notices 28* (3): March 1993, pp. 149-175.

Discrete event simulation programming languages (SPLs) must meet a minimum of six requirements: (1) generation of random numbers to represent uncertainty, (2) process transformers, to permit other than uniform random variates to be used, (3) list processing capability, so that objects can be created, manipulated, and deleted, (4) statistical analysis routines, to provide the descriptive summary of model behavior, (5) report generation, to provide the presentation of potentially large reams of data in an effective way for decision making, and (6) a timing executive or time flow mechanism. Clearly these requirements could be met by a general purpose language (GPL), but developments of SPLs progressed with the conviction that model complexity could be better accommodated and modeling conveniences better provided with a language reflecting the problem-solving domain. Nevertheless, packages of simulation-specific routines in numerous GPLs are significant; some have emerged to a very popular form in use historically and today.

Figure 1 shows the chronological development of simulation programming languages from both a vertical and horizontal perspective. The vertical perspective follows the division among world views, the "WELTANSICHT" of Lackner (1962, p.3). The "event scheduling," "activity scan," and "process interaction" categorizations are now termed the "classical" because they were used very early by Lackner (1964) and Kiviat (1967; 1969) as the basis for distinguishing and contrasting SPLs. Overstreet (1986, p. 171) captured the distinctions best by using the concept of locality:

> *Event scheduling* provides locality of time: each event routine describes related actions that may all occur in a single instant.
>
> *Activity scanning* provides locality of state: each activity routine describes all actions that must occur because a particular model state is reached.
>
> *Process interaction* provides locality of object: each process routine describes the entire action sequence of a particular model object.

The horizontal perspective in the SPL chronology describes the organization of the remainder of this paper. The five periods chosen, although somewhat arbitrarily, reflect remarkably similar developments occurring in the SPLs of that period. In fact, the earliest SPLs have continued, either in extended or modified forms or as generational successors. These five periods are labeled:

1955-60:	The Period of Search,
1961-65:	The Advent,
1966-70:	The Formative Period,
1971-78:	The Expansion Period,
1979-86:	The Period of Consolidation and Regeneration.

Since some time is needed to develop a historical perspective, no attempt is made to include developments beyond 1986.

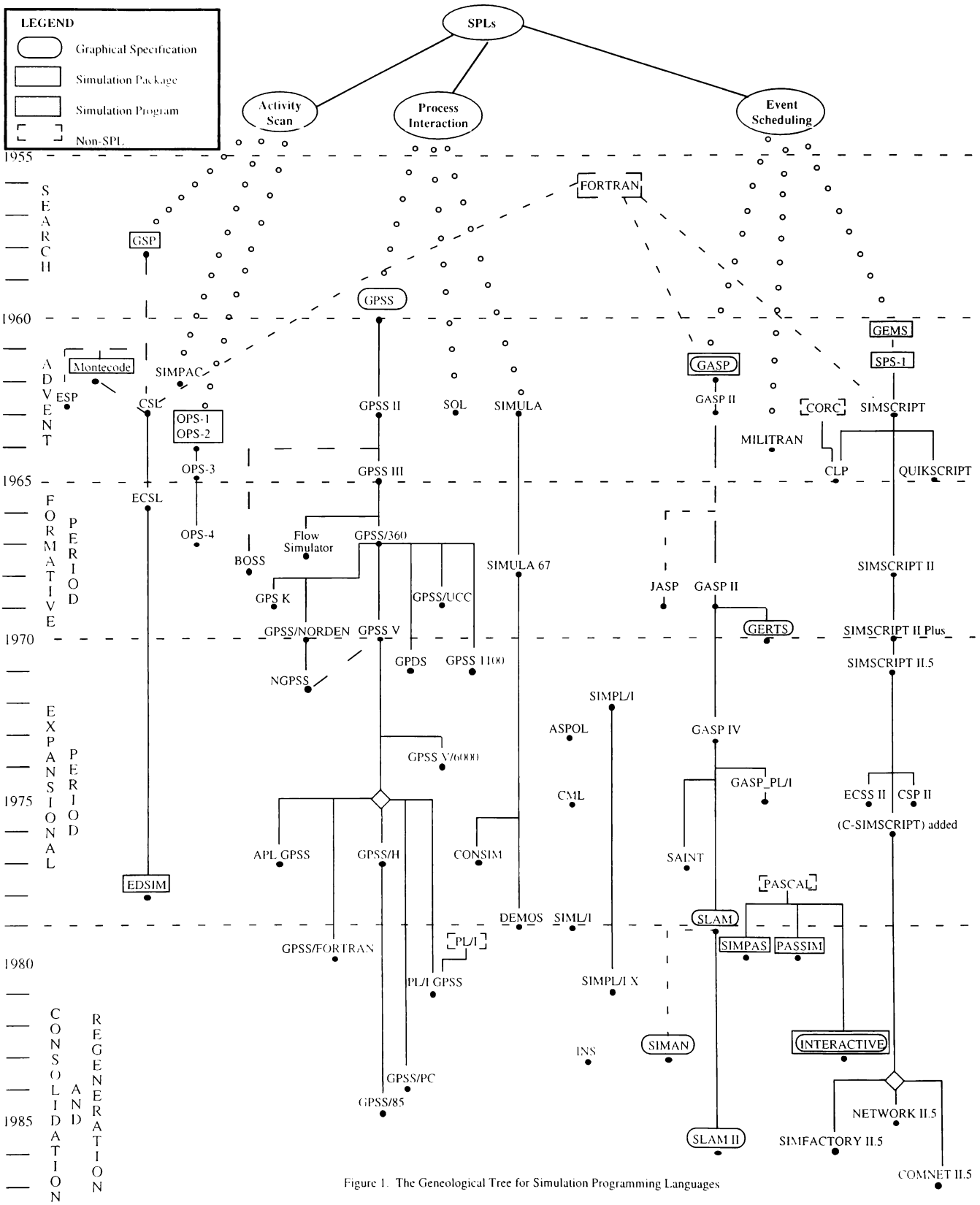

Figure 1. The Geneological Tree for Simulation Programming Languages

2 THE PERIOD OF SEARCH (1955-1960)

This period is marked by efforts to discover not only concepts of model representation but the representational needs for doing simulation analysis. Early groups at General Electric, UCLA, and elsewhere were cited in Conway (1967, p. 219). The General Simulation Program (GSP) of K.D. Tocher and D.G. Owen (Tocher, 1960) is considered the first "language effort." Tocher describes his identification and reuse of routines needed for each simulation application as a "simulation structure." The claim that such a structure would enable "automatic programming of simulations" (Tocher 1960, p. 51) seems a little inflated; however, the same claim was made for assemblers about five years earlier.

3 THE ADVENT (1961-1965)

The forerunners of SPLs currently in use appeared during the period of 1961-1965. GPSS was developed on various IBM computers and the name changed to the General Purpose Simulation System (from General Purpose System Simulator). Originating in the problem domain of communication systems, the original block semantics were ideally suited for queuing models. An interesting exercise demonstrating the importance of graphical output for on-line validation occurred in 1965 with an IBM 2250 interactive display terminal tied to a GPSS model (Reitman 1992).

At the Norwegian Computing Center the language SIMULA was begun in 1963 by Ole-Johan Dahl and Kristen Nygaard. The language was developed for the Univac 1107 and represented an extension to Algol 60, the most popular GPL in Europe. The early language version proceeded through four stages: (1) a discrete event network concept, (2) basing of the language on Algol 60, (3) modifications and extensions of the Univac Algol 60 compiler as the process concept is introduced, and (4) the implementation of the SIMULA I compiler. More follows concerning the impact of SIMULA, particularly the successor version, on programming languages in general.

The event scheduling SPL SIMSCRIPT appeared in 1963. Harry Markowitz, later to receive a Nobel Prize for his work in portfolio theory, provided the major conceptual guidance. The RAND corporation developed the language under sponsorship by the US Air Force, and Bernard Hausner was the sole programmer. Herbert Karr authored the SIMSCRIPT manual (Markowitz 1963, p.iii). Both syntactically and organizationally, SIMSCRIPT was heavily influenced by FORTRAN. The entire FORTRAN language was included as a subset. Later SIMSCRIPT I.5 was developed as a proprietary version by CACI (Karr 1965). A major difference was that SIMSCRIPT I.5 was compiled into assembly language rather than FORTRAN.

The Control and Simulation Language (CSL), representing the activity scan world view, appeared in 1963 as a joint venture of Esso Petroleum Company Ltd. and IBM United Kingdom, Ltd. The creators, John Buxton and John Laski, readily acknowledged the contributions of Tocher and his colleagues at United Steel Companies (Buxton and Laski 1962, p. 198). A second version of CSL, labeled C.S.L. 2 by Clementson (1966), is attributed to P. Blunden, P. Grant, and G. Parncutt of IBM UK (IBMUK 1965). CSL draws heavily on FORTRAN for the implementation, but the conceptual basis is clearly influenced by GSP. A simpler two-phase executive is used in CSL, in contrast with the three-phase executive adopted in GSP. The language became a favorite for program generation techniques based on the entity cycle or activity cycle diagrams, which were successors to the wheel charts originally developed by Tocher (Tocher 1966).

GASP (General Activity Simulation Program), developed by Philip J. Kiviat at the Applied Research Laboratory of the United States Steel Corporation, was begun in 1961, originally in ALGOL. Early on the decision was made to base the GASP simulator on FORTRAN II. Originally designed to bridge the gap between operating personnel and computer programmers, GASP, like GPSS, utilized flow chart symbols considered familiar to engineers. The set of FORTRAN II subroutines that made up GASP included a random number generator from the resident library. The GASP executive controlled the timing following an event scheduling view, and debugging help was provided in a special subroutine.

OPS-3, while relatively unknown to current day users, was an SPL ahead of its time. It represented an innovative technical effort at MIT to build simulation capability on the time sharing system CTSS. Principal developers were Greenberger, Jones, Morris, and Ness; however many others contributed in various ways (Greenberger 1965). The OPS-3 *system* was intended to be a multipurpose, open-ended, modular, compatible support for creative researchers that were not necessarily computing experts. Additions to OPS-3 could be made in a variety of GPLs. Clearly, the influence of SOL and SIMSCRIPT are present in the time flow mechanism, but the power of interactive model execution was unique to the language. The manifestation of the timing control was more akin to the three-phase method in Tocher's GSP than to any other form at the time.

Numerous other SPLs appeared during this advent period. Notable, although not a discrete event language, was DYNAMO. Developed at MIT for systems dynamic modeling, DYNAMO influenced several of the developers of discrete event SPLs (Kiviat 1991). Significant in the effect of DYNAMO was graphical output, strong typing, and extensive error detection capability. Other languages of note were SIMPAC, with a fixed-time-increment timing routine, developed at the System Development Corporation (Bennett 1962); SOL (Simulation Oriented Language), created by Knuth and McNeley as an extension to ALGOL and structured

much like GPSS; and MILITRAN, produced by the Systems Research Group of the Office of Naval Research (Systems Research Group, Inc. 1964).

4 THE FORMATIVE PERIOD (1966-1970)

The period from 1966-1970 reflected the conceptual clarification of languages. Concepts, possibly considered as secondary during the trials of bringing a SPL to implemented form, were reviewed and refined to promote a more consistent representation of a world view and to clarify its presentation to users. Rapid hardware advancements forced some languages, notably GPSS, to undergo major revisions. GPSS II and III, both of which emerged during the advent period, were replaced by GPSS/360, an extended version of the language. Other vendors also began to produce look-a-likes: RCA with its flow simulator (Greenberg 1972, p. 8), from Honeywell in 1969 (GPSK 1969) and GPSS/UCC. As a measure of the change in the language from GPSS III to GPSS/360, the number of block types increased from 36 to 44 and set operations were expanded by the introduction of groups. A HELP block opening the simulator to routines in other languages was also provided.

Major changes occurred in SIMULA, and the version SIMULA 67 emerged as a principal player at least on the conceptual level among programming languages. The concept of "classes of objects was added; subclassing and class concatenation to enable inheritance were provided; and direct, qualified references for object manipulation was created. Standardization emerged with the SIMULA 67 Common Base Language (Nuggard and Dahl 1981).

SIMSCRIPT II, also dependent somewhat on SIMSCRIPT I.5 for its basic concepts of entity, attribute, and set, clearly represented a major advancement in SPLs. An expressed goal of SIMSCRIPT II was to be self-documenting. In fact, the language was written to be used on five levels, and with its free-form, English-like mode of communication, and "forgiving" compiler, the user received major considerations in the language design. Markowitz (1979) indicated that the language was initially intended to be used for two additional levels, one that dealt explicitly with database entities and yet another to provide a language useful for writing other languages. (Markowitz 1979, p. 29).

ECSL, developed for Courtaulds Ltd. by the originators of CSL, became a popular language in the UK. ECSL departed from the heavy emphasis on FORTRAN of CSL, and Clementson (1966, p. 215) with the expressed intent to embrace users who were non-programmers relied heavily on the entity-cycle diagrams for model input. ECSL was the target language for CAPS (Computer Aided Programming System), the first *interactive program generator* (Mathewson 1975).

The careful reader has perhaps noted that GASP II appears twice in the genealogical tree of Figure 1. A preliminary description of the revised version appears in manual form in 1967 (Pritsker 1967) while a listing of FORTRAN subprograms designated as a GASP II compilation is dated "3/13/63." This early revision is due to Kiviat alone.

OPS-4 is primarily the doctoral research of Malcolm M. Jones (1967). Based on PL/I, OPS-4 encouraged incremental construction in test of model components. All three world views were supported in the language. Extensive debugging and tracing capabilities and on-line diagnostic explanations were available. Model execution could be interrupted and attribute values redefined.

New languages introduced during this period included BOSS (Burroughs Operational Systems Simulator) in 1967, Q-GERT in the early 1970s, and a number of others described in the hallmark workshop on simulation programming languages chaired by Buxton (1968).

5 THE EXPANSION PERIOD (1971-1978)

Major advances in GPSS during this period came from outside IBM as the success of the language caused others to extend either its capabilities or the host hardware environment. GPSS/NORDEN was an interactive, visual, on-line environment programmed in COBOL. This version permitted user interaction through a CRT terminal and interruption by the user to redefine certain standard numerical attributes and then resume execution. A second version NGPSS (NORDEN GPSS) is described as a superset of GPSS/360 and GPSS V for both batch and interactive execution. Limited database capability was provided.

GPSS V/6000, developed by Northwestern University for Control Data Corporation, was completed in 1975. Complete compatibility with the IBM product was claimed. GPDS (General Purpose Discrete Simulator) was developed as a program product of Xerox Data Systems for Sigma Computers. Similarly, a UNIVAC product labeled GPSS 1100 is referenced with a 1971 date (UNIVAC 1971a, 1971b). A major addition was GPSS/H in 1975, created by James O. Henriksen and produced in a compiled version. Released by Wolverine Software Corporation, GPSS/H has become the principal version of the language since that time.

This period of expansion for SIMULA took the form of a pure system description language called DELTA (Holbaek-Hanssen 1977). The DELTA project sought to implement system specification for simulation execution through a series of transformation from a high level user perspective through an intermediate form called BETA, culminating with an executable language called GAMMA.

A major departure from its roots was taken by SIMSCRIPT with the adoption of a process interaction world view in the mid-1970s. During this period, the capability for combined (discrete event and continuous) modeling was added to the language. This expansion period marked the beginnings of the breakdown in conceptual distinctions among SPLs.

Notable changes occurred to GASP during this period, primarily through the work of Pritsker and his graduate students at Purdue. GASP IV became available in 1974 (Pritsker 1974). GASP IV provided combined modeling, and differentiated between state events and time events as a means for representing both the condition concept of activity scan and the temporal change concept of event scheduling. A transaction flow world view was incorporated into GASP during 1975-1976 (Washam 1976a; 1976b). Pritsker and Young (Pritsker 1975) produced a PL/I version of the language, a phenomenon that occurred repeatedly during this period (the mapping of an SPL onto PL/I).

Interactive program generators were a major domain of activity in the U.K. Although initially beginning with Programming By Questionnaire in the U.S., CAPS-ECSL, DRAFT/FORTRAN, DRAFT/GASP and others sought to simplify the modeling task through the entity cycle (or activity cycle) diagram as a means for constructing the basic outline of the model. Mappings to different languages enabled a user to select the particular implementation with which he or she was most comfortable. A related, but very different approach, was taken by George Heidorn in the early 1970s in his work to build a natural language interface to GPSS (Heidorn 1976). In addition to GASP noted above, other mappings to PL/I included SIMPL/I, a PL/I preprocessor (SIMPL/I 1972), PL/I GPSS (Metz 1981) and SIMPL, a descendant of OPS-4 (Jones 1971a; 1971b). Slightly later SIML/I appeared in 1979, as a language for computer system modeling (MacDougall 1979).

6 CONSOLIDATION AND REGENERATION (1979-1986)

This final period in the chronology might be characterized as one where predominant SPLs extended their implementation to many computers and microprocessors while keeping the basic language capabilities relatively static. However, two major descendants of GASP appeared to play major roles: SLAM II and SIMAN. The Simulation Language for Alternative Modeling (SLAM), produced by Pritsker and Associates, Inc., sought to provide multiple modeling perspectives and combined modeling capabilities (Pritsker 1979). SLAM is a FORTRAN preprocessor, whereas its predecessors were packages. The major structure of the SLAM design still follows that of GASP; in fact, many of the identical subroutine and function names in GASP IV are repeated in SLAM.

SIMAN is derived from SIMulation ANalysis. Originally couched in a manufacturing systems application domain, SIMAN possesses a general modeling capability found in SPLs such as GASP IV. C. Dennis Pegden developed SIMAN as a one person faculty project in about a two year period. SIMAN is claimed to be the first major simulation language executable on the IBM PC and designed to run under MS-DOS constraints (Pegden 1991). The marketing of SIMAN has recognized the major advantage of output animation with a companion product called CINEMA.

Almost as history repeating itself, the emergence of PASCAL, yet another popular GPL, stimulated the appearance of a number of simulation packages based on the language. SIMPAS (Bryant 1980; 1981) is an event scheduling package that captures all the requirements identified for simulation modeling. SIMPAS, implemented as a preprocessor, was designed to be highly portable yet complete in its provision of services. In contrast, PASSIM (Uyeno 1980) provided less services, requiring more knowledge of PASCAL by the modeler. Yet another example was INTERACTIVE, described as a network simulation language using graphical symbols for model representation (Lakshmanan 1983).

INSIGHT was developed by Roberts (1983a) to model health care problems and for general simulation use. INSIGHT adopts the transaction flow world view and offers a graphical model representation that must be translated manually into INSIGHT statements. A FORTRAN preprocessor, INSIGHT provides the usual process transformers and some assistance in output analysis.

7 CONCLUDING SUMMARY

Why do the simulation programming languages exhibit remarkably similar periods of development? Perhaps it is the intense commercial competition, more intense than that for GPLs, which explains this fact. Perhaps many of the ideas and concepts, recognized eventually as important for general purpose use, in having their birth in SPLs created the early conceptual clashes which are inevitable for new ideas. Consider the numerous concepts and techniques that either originated with SPLs or gained visibility through their usage in a language:

- the process concept,
- object definition as a record data type,
- definition and manipulation of sets of objects,
- implementation of the abstract data type concept,
- quasi-parallel processing using the co-routine concept,
- delayed binding with run-time value assignment,
- English-like syntactic statements to promote self-documentation,
- error detection and correction in compilation,
- dynamic storage allocation and reclaim, and
- tailored report generation capabilities.

The importance of simulation in its influence on programming language development and concepts cannot be denied. The claim of Sammet (1969, p.650) that her justification for categorizing simulation languages as a specialization warranting limiting description is, "their usage is unique and presently does not appear to represent or supply much carry-over into other fields." Clearly the above list serves to dispel that misperception.

Now as we see simulation models at the core of analysis and training research, described as "virtual environments" or in entertainment applications using "virtual reality" are we entering a new period of change? Is the significance of modeling and simulation on its effect in sciences and business about to be universally recognized? Is Licklider's (19670 prediction in 1967 about to be realized:

> *In their static form, computer-program models are documents. They preserve and carry information just as documents printed in natural language do, and they can be read and understood by recipients who know the modeling language. In their dynamic form, however, computer-program models appeal to the recipient's understanding directly through his perception of dynamic behavior. That model of appeal is beyond the reach of ordinary documents. When we have learned how to take good advantage of it, it may - indeed, I believe it will - be the greatest boon to scientific and technical communication, and to the teaching and learning of science and technology, since the invention of writing on a flat surface.*

REFERENCES

Bennett, R. P., P. R. Cooley, S. W. Hovey, C. A. Kribs, and M. R. Lackner. 1962. *Simpac User's Manual*, Report #TM-602/000/00, System Development Corporation, Santa Monica, California.

Buxton, J. N., ed. 1968. Simulation programming languages. In *Proceedings of the IFIP Working Conference on Simulation Programming Languages*. North-Holland Publishing Company.

Buxton, J. N. and J. G. Laski. 1962. Control and simulation language. *The Computer Journal*, 5:194-199.

Clementson, A. T. 1966. Extended control and simulation language. *The Computer Journal*, 9:215-220.

Conway, R. W., W. L. Maxwell, and L. W. Miller. 1967. *Theory of Scheduling*. New York: Addison-Wesley Publishing Company.

Delfosse, C. M. 1976. *Continuous Simulation and Combined Simulation in SIMSCRIPT II.5*. CACI. Arlington, Virginia.

Greenberger, M., M. M. Jones, J. H. Morris, and D. N. Ness. 1965. *On Dash Line Computation and Simulation: The OPS-3 System*. Massachusetts: MIT Press.

Heidorn. 1976. Automatic programming through natural language dialogue, a survey. *IBM Journal of Research and Development*, 20:302-313.

Henriksen, J. O. 1976. Building a better GPSS: a 3:1 enhancement. In *Proceedings of the 1975 Winter Simulation Conference*, 465-469. New Jersey: AFIPS Press.

Holback-Hanssen, E., P. Händlykken, and K. Nygaard. 1977. System description and the DELTA language. *DELTA Project Report No.* 4. Second Printing. Norwegian Computing Center.

IBMUK. 1965. *CSL Reference Manual*. IBM United Kingdom Ltd.

Jones, M. M. 1967. Incremental simulation on a time-shared computer. Unpublished Ph.D. dissertation, Alfred P. Sloane School of Management, Massachusetts Institute of Technology.

Jones, M. M. and R. C. Thurber. 1971a. The SIMPL Primer.

Jones, M. M. and R. C. Thurber. 1971b. SIMPL Reference Manual.

Kiviat, P. J. 1967. Digital Computer Simulation: Modeling Concepts, RAND Memo RM-5378-PR, RAND Corporation. Santa Monica, California.

Kiviat, P. J. 1969. Digital Computer Simulation: Computer Programming Languages, RAND Memo RM-5883-PR. RAND Corporation. Santa Monica, California.

Kiviat, P. J. 1991b. Personal Communication.

Knuth, D. E. and J. L. McNeley. 1964a. SOL - a symbolic language for general purpose system simulation. *IEEE Transactions on Electronic Computers* EC-13:401-408.

Knuth, D. E. and J. L. McNeley. 1964b. A formal definition of SOL. *IEEE Transactions on Electronic Computers* EC-13:4, 409-414.

Lackner, M. R. 1962. Toward a general simulation capability. In *Proceedings of the SJCC*, 1-14. San Francisco, California.

Lackner, M. R. 1964. Digital simulation and system theory. System Development Corporation, SDC SP-12. Santa Monica, California.

Lakshmanan, R. 1983. Design and implementation of a PASCAL based interactive network simulation language for microcomputers. Unpublished Ph.D. dissertation, Oakland University, Rochester, Michigan.

Licklider, J. C. R. 1967. Interactive dynamic modeling. In *Prospects for Simulation and Simulators of Dynamic Systems*, eds. G. Shapiro and M. Rogers. New York: Spartan Books.

MacDougall, M. H. 1979. The simulation language SIML/I. In *Proceedings of the National Computer Conference*, 39-44.

Markowitz, H. M. 1979. SIMSCRIPT: past, present, and some thoughts about the future. *Current Issues in Computer Simulation*, eds. N. R. Adam and Ali Dogramaci, 27-60. New York: Academic Press.

Markowitz, H. M., B. Hausner, and H. W. Karr. 1963. *SIMSCRIPT: A Simulation programming language*. The RAND Corporation. Santa Monica, California.

Mathewson, S. C. 1975. Interactive simulation program generators. In *Proceedings of the European Computing Conference on Interactive Systems*, 423-439. Brunel University, U.K.

Metz, W. C. 1981. Discrete event simulation using PL/I based general and special purpose simulation lan-

guages. In *Proceedings of the Winter Simulation Conference*, eds. T. I. Oren, C. M. Delfosse, C. M. Shub, 45-52.

Overstreet, C. M. and R. E. Nance. 1986. World view based discrete event model simplification. *Modeling and Simulation Methodology in the Artificial Intelligence Era*, eds. M. S. Elzas, T. É. Oren, and B. P. Zeigler, 165-179. North Holland.

Pegden, C. D. 1991. Personal Communication.

Pegden, C. D., R. E. Shannon, and R. P. Sadowski. 1990. *Introduction to Simulation Using SIMAN*. McGraw-Hill.

Pritsker, A. A. B. 1967. *GASP II User's Manual*. Arizona State University.

Pritsker, A. A. B. 1974. *The GASP IV Simulation Language*. New York: John Wiley and Sons.

Pritsker, A. A. B. and C. D. Pegden. 1979. *Introduction to Simulation and SLAM*. New York: John Wiley and Sons.

Pritsker, A. A. B. 1990. *Papers, Experiences, Perspectives*. Indiana: Systems Publishing Company.

Pugh, A. L., III. 1963. *DYNAMO User's Manual*. The MIT Press, 2nd Ed.

RCA. 1967. *Flow Simulator*. RCA publication #70-05-008.

RCA. 1967. *Flow Simulator Information Manual*. RCA publication #70-35-503.

Reifer, D. J. 1973. Simulation Language Survey. Hughes Aircraft Corporation, Interdepartmental Correspondence, Ref. 2726.54/109.

Reitman, J. 1992. How the hardware and software world of 1967 conspired (interacted?) to produce the first in the series of winter simulation conferences. In *Proceedings of the 1992 Winter Simulation Conference*, eds. J. J. Swain, R. C. Crain, and J. R. Wilson, 52-58.

SIMPL/I. 1972. SIMPL/I (Simulation Language Based on PL/I). *Program Reference Manual*. IBM publication #SH19-5060-0.

Systems Research Group, Inc. 1964. *Systems Research Group, Inc. MILITRAN Programming Manual*. Prepared for the Office of Naval Research, Washington, DC.

Tocher, K. D. 1966. Some techniques of model building. In *Proceedings IBM Scientific Computing Symposium on Simulation Models and Gaming*, 119-155. White Plains, New York.

Tocher, K. D. and D. G. Owen. 1960. The automatic programming of simulations. In *Proceedings of the Second International Conference on Operational Research*, eds., J. Banbury and J. Maitland, 50-68.

UNIVAC 1100. 1971a. *UNIVAC 1100 Series General Purpose Simulator (GPSS 1100) Programmer Reference*. UP-7883.

UNIVAC 1100. 1971b. *UNIVAC 1100 General Purpose Systems Simulator II Reference Manual*. UP 4129.

Uyeno, D. H. and W. Vaessen. 1980. PASSIM: a discrete-event simulation package for PASCAL. *Simulation* 35:183-190.

Washam, W. B. 1976a. GASPPI: GASP IV with process interaction capabilities. Unpublished Master's thesis, Purdue University, West Lafayette, Indiana.

Washam, W. B. and A. A. B. Pritsker. 1976b. Putting process interaction capability into GASP IV, ORSA/TIMS Joint National Meeting, Philadelphia.

AUTHOR BIOGRAPHY

RICHARD E. NANCE is the RADM John Adolphus Dahlgren Professor of Computer Science and the Director of the Systems Research Center at Virginia Polytechnic Institute and State University. He received B.S. and M.S. degrees from N.C. State University in 1962 and 1966, and the Ph.D. degree from Purdue University in 1968. He has served on the faculties of Southern Methodist University and Virginia Tech, where he was Department Head of Computer Science, 1973-79. Dr. Nance has held research appointments at the Naval Surface Weapons Center and at the Imperial College of Science and Technology (UK). Within ACM, he has chaired two special interest groups: Information Retrieval (SIGIR), 1970-71 and Simulation (SIGSIM), 1983-85. He has served as Chair of the External Activities Board and several ACM committees. The author of over 100 papers on discrete event simulation, performance modeling and evaluation, computer networks, and software engineering. Dr. Nance has served on the Editorial Panel of *Communications ACM* for research contributions in simulation and statistical computing, 1985-89, as Area Editor for Computational Structures and Techniques of *Operations Research*, 1978-82, and as Department Editor for Simulation, Automation, and Information Systems of *IIE Transactions*, 1976-81. He served as Area Editor for Simulation, 1987-89 and as a member of the Advisory Board, 1989-92, *ORSA Journal on Computing*. He is the founding Editor-in-Chief of the ACM *Transactions on Modeling and Computer Simulation*. He served as Program Chair for the 1990 Winter Simulation Conference. Dr. Nance received a Distinguished Service Award from the TIMS College on Simulation in 1987. He is a member of Sigma Xi, Alpha Pi Mu, Upsilon Pi Epsilon, ACM, IIE, ORSA, and TIMS.