A History of Discrete Event Simulation Programming Languages*

Richard E. Nance

TR 93-21

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

June 11, 1993

**Cross-listed as Systems Research Center report SRC 93-003.*

ABSTRACT

The history of simulation programming languages is organized as a progression in periods of similar developments. The five periods, spanning 1955-1986, are labeled: The Period of Search (1955-1960); The Advent (1961-1965); The Formative Period (1966-1970); The Expansional Period (1971-1978), and The Period of Consolidation and Regeneration (1979-1986). The focus is on recognizing the people and places that have made important contributions in addition to the nature of the contribution. A balance between comprehensive and in-depth treatment has been reached by providing more detailed description of those languages which have or have had major use. Over 30 languages are mentioned, and numerous variations are described in the major contributors. A concluding summary notes the concepts and techniques either originating with simulation programming languages or given significant visibility by them.

CR Categories and Subject Descriptors: I.6.2 [**Simulation and Modeling**]: Simulation Languages; I.6.8 [**Simulation and Modeling**]: Types of Simulation – Discrete event; K.2 [**History of Computing**]: Software

Additional Key Words and Phrases: History of simulation languages

TABLE OF CONTENTS

1. INTRODUCTION

This introductory section is intended to explain the different types of computer simulation and to identify the key issues in simulation programming languages (SPLs). The approach to this survey is the subject of a concluding subsection.

1.1 Computer Simulation

Analog (or analogue) simulation, i.e., simulation on analog computers, is not addressed in this survey since no major "language" or language issues are associated with this early form. However, analog simulation was much in evidence during the 1950s and earlier. Digital simulation, i.e., simulation on digital computers, is considered synonymous with "computer simulation" for this history of SPLs.

1.1.1 Taxonomy and Terminology

Computer simulation is an application domain of programming languages that permits further division into three partitions:

(1) discrete event simulation,
(2) continuous simulation, and
(3) Monte Carlo simulation.

Discrete event simulation utilizes a mathematical/logical model of a physical system that portrays state changes at precise points in simulated time. Both the nature of the state change and the time at which the change occurs mandate precise description. Customers waiting for service, the management of parts inventories, or military combat are typical application domains for discrete event simulation.

Continuous simulation uses equational models, often of physical systems, which do not portray precise time and state relationships that result in discontinuities. The objectives of studies using such models do not require the explicit representation of state and time relationships. Examples of such systems are found in ecological modeling, ballistic reentry, or large-scale economic modeling.

Monte Carlo simulation, the name given by John von Neumann and Stanislaw M. Ulam [Morgenthaler 1961, p. 368] to reflect its gambling similarity, utilizes models of uncertainty where representation of time is unnecessary. The term is originally attributed to "a situation in which a difficult non-probablistic problem is solved through the invention of a stochastic process that satisfies the relations of the deterministic problem" [Morgenthaler, 1961]. A more recent characterization is that Monte Carlo is "the method of repetitive trials" [Shreider 1966, p.1]. Typical of Monte Carlo simulation is the approximation of a definite integral by circumscribing the region with a known geometric shape, then generating random points to estimate the area of the region through the proportion of points falling within the region boundaries.

Simulation as a problem-solving technique precedes the appearance of digital computers by many years. However, the emergence of digital technology exerted an influence far more than adding the term "computer" as an adjectival descriptor of this method of problem solving. The oppressive manual computational burden was now off-loaded to a much faster, more tolerant processor – the digital computer.

Three related forms of simulation are commonly noted in the literature. *Combined simulation* refers generally to a model that has both discrete event and continuous components (see [Law 1991, p. 112]). Typically, a discrete event submodel runs within an encapsulating continuous model. *Hybrid simulation* refers to the use of an analytical submodel within a discrete event model

(see [Shantikumar, 1983]). Finally, *gaming* or *computer gaming* can have discrete event, continuous, and/or Monte Carlo modeling components.

1.1.2 Language Requirements for Discrete Event Simulation

In her description of programming languages, Sammet [1969, p. 650] justifies the categorization of simulation languages as a specialization warranting limited description with the statement that, "their usage is unique and presently does not appear to represent or supply much carry-over into other fields." The impact of the object-oriented paradigm, and the subsequent clamor over object-oriented programming languages, first represented by SIMULA, would appear in hindsight to contradict this opinion, which accurately represented the attitude of the programming language research community at that time.

A simulation language must provide a model representation that permits analysis of execution behavior. This provision entails six requirements:

(1) generation of random numbers, so as to represent the uncertainty associated with an inherently stochastic model,

(2) process transformers, to permit uniform random variates obtained through the generation of random numbers to to be transformed to a variety of statistical distributions,

(3) list processing capability, so that objects can be created, deleted, and manipulated as sets or as members, added to and removed from sets,

(4) statistical analysis routines, to provide the descriptive summary of model behavior so as to permit comparison with system behavior for validation purposes and the experimental analysis for both understanding and improving system operation,

(5) report generation, to furnish an effective presentation of potentially large reams of data to assist in the decision making that initially stimulates the use of simulation, and

(6) timing executive or a time flow mechanism, to provide an explicit representation of time.

Every simulation programming language provides these components to some degree.

Many simulation applications are undertaken in general purpose languages. Strongly influenced by this fact, simulation "languages" have taken three forms: (1) package, (2) preprocessor, and (3) conventional programming language. A package is a set of routines in language X that can be called or invoked to meet the six requirements listed above. The user (programmer) might develop additional routines in X to provide needed capabilities or a tailored treatment. While such packages are not truly a language, some have had major influence, perhaps leading to descendants that take one of the other two forms. For example, the evolution of GASP, a package, produces SLAM, a preprocessor. All three "language" forms are included because examples of each have played significant roles in shaping the simulation scene.

1.1.3 Conceptual Frameworks for Discrete Event Modeling

Very early, Lackner [1962, p. 3] noted the importance of a modeling perspective as a "WELTANSICHT" or world view that "must be implicitly established to permit the construction of a simulation language." In a subsequent work, he attempted to lay the groundwork for a theory of discrete event simulation and used the differing world views to categorize SPLs [Lackner, 1964]. Lackner's categorization differentiates between event-oriented and activity-oriented SPLs. Kiviat

[1967; 1969] expanded the categories to identify three world views: event-oriented, activity-oriented, and process-oriented.

Fishman [1973] helped to clarify the distinction among the categories, which he labeled "event scheduling," "activity scan," and "process interaction." These labels have become more or less the conventions in the literature. A recent thesis by Derrick [1988] expanded the three to 13, doing so in a manner that encompassed more than SPL discrimination.

The differentiation among world views characterized by languages is captured best using the concept of locality [Overstreet 1986, p. 171].

> *Event scheduling* provides locality of time: each event routine in a model specification describes related actions that may all occur in a single instant.
>
> *Activity scanning* provides locality of state: each activity routine in a model specification describes all actions that must occur due to the model assuming a particular state (that is, due to a particular condition becoming true).
>
> *Process interaction* provides locality of object: each process routine in a model specification describes the entire action sequence of a particular model object.

1.2 Historical Approach

Although fundamental in distinguishing the modeling differences among SPLs, the world views do not suggest a basis for historical description for two reasons:

(1) the origins and initial development of the language(s) promoting a particular view took place concurrently without a clear progression of conceptual convergence, and

(2) remarkably parallel development patterns are evident throughout the major SPLs, irrespective of world view ties.

However, the extent and range of activity in SPLs mandates an organizational thesis, and a chronological presentation is used.

A chronological depiction of SPL development could proceed in several ways. One could provide a vertical trace of developments within each SPL. The tack taken in this paper is more of a horizontal cut across periods of time, which are noted in the genealogical tree of SPLs, shown in Figure 1:

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

Each major language originating in a period is described in terms of its background and rationale for language content. Description of that language in subsequent time periods is limited to changes in that language, particularly those that mark the designated time period. Descriptions of GPSS and SIMULA are limited also; since both languages are treated extensively in the first conference devoted to programming language history (see [Wexelblatt, 1981]). No attempt is made to describe language developments since 1986; sufficient time has yet to elapse to permit an historical perspective.

Figure 1 is organized vertically to distinguish languages according to world view. This categorization cannot be precise , for the language developer in most instances was unaware of the distinguishing characteristic. Crookes [1982, p. 1] places the number of SPLs as 137, although giving no source, and attributes this overpopulation to a lack of standardization in general purpose

computer languages. More likely, the excess in SPLs can be attributed to a combination of root causes:

(1) perceived differences among application domains, requiring language features not provided in extant SPLs;

(2) lack of awareness of SPLs or the reliance on the general purpose language "crutch" leading to package development;

(3) educational use of a well known language in which components are provided so that learning time is reduced and the cost of an additional translator is avoided; and

(4) competition among language vendors to exploit particular features or selected application domains.

The degree to which each of these causes has proved influential is quite dependent on time. The first and second above tend to mark the early periods; the fourth, the later periods.

One further point concerns the unfortunate but all too common instances of incomplete references, primarily missing dates. In the attempt to provide as much historical information as possible, dates are given if suggested by other sources, added notes (time stamps), or personal recollection. In those cases where the data is uncertain, a "?" suffix is placed in the citation, e.g., [Meyerhoff, 1968?] indicates that the recollection of Paul Roth is that this undated manual was completed in 1968 [Roth, 1992].

2. THE PERIOD OF SEARCH (1955-1960)

The title of this subsection is derived from the efforts during this period to discover not only concepts of model representation but to facilitate the representational needs in simulation modeling. Simulation languages came into being with the recognition of the power of the technique in solving problems that proved intractible to closed-form mathematical solution. The most prominent examples of such problems are the early manufacturing simulations by a group at General Electric; Jackson, Nelson, and Rowe at UCLA; and Baker and Dzielinski cited in [Conway 1967, p. 219].

K.D. Tocher is generally credited with the first simulator described in a 1960 publication jointly with D.G. Owen [Tocher, 1960]. The simulator, later called GSP (General Simulation Program), introduced the three-phase method of timing control generally adopted by activity scan languages. Significantly, Tocher characterized his work as "searching for a simulation structure," rather than merely providing a "simulation language," [Tocher 1960, p.51]. Excerpts from the GSP example given in the appendix of the 1960 paper are shown in Figure 2. While lacking the format regularity of assembly language programs, the syntactic structure of GSP seems to fall far short of the descriptor "automatic programming of simulations," [Tocher 1960, p.51]. However, this seemingly inflated claim was typically applied to languages other than basic machine coding, including assemblers (see [Rosen, 1967] for examples).

In the United States interest in simulation was emerging rapidly in the late 1950s; however, the recognition of concepts and the assistance of language in dealing with them was less recognized. Gordon [1981] describes his own efforts prior to 1960 as the writing of simulation programs for message switching systems. The Sequence Diagram Simulator, described in a conference in 1960, indicates some appreciation of language support for the application task [Gordon 1981, p. 405].

The treatment given to computer simulation in the 1961 issue of *Progress in Operations Research* [Ackoff, 1961] demonstrates the perceived importance of the technique. However, manual versus computer execution remains a matter of consideration [Morgenthaler 1961, pp. 406-407], and little consideration of language needs is indicated.

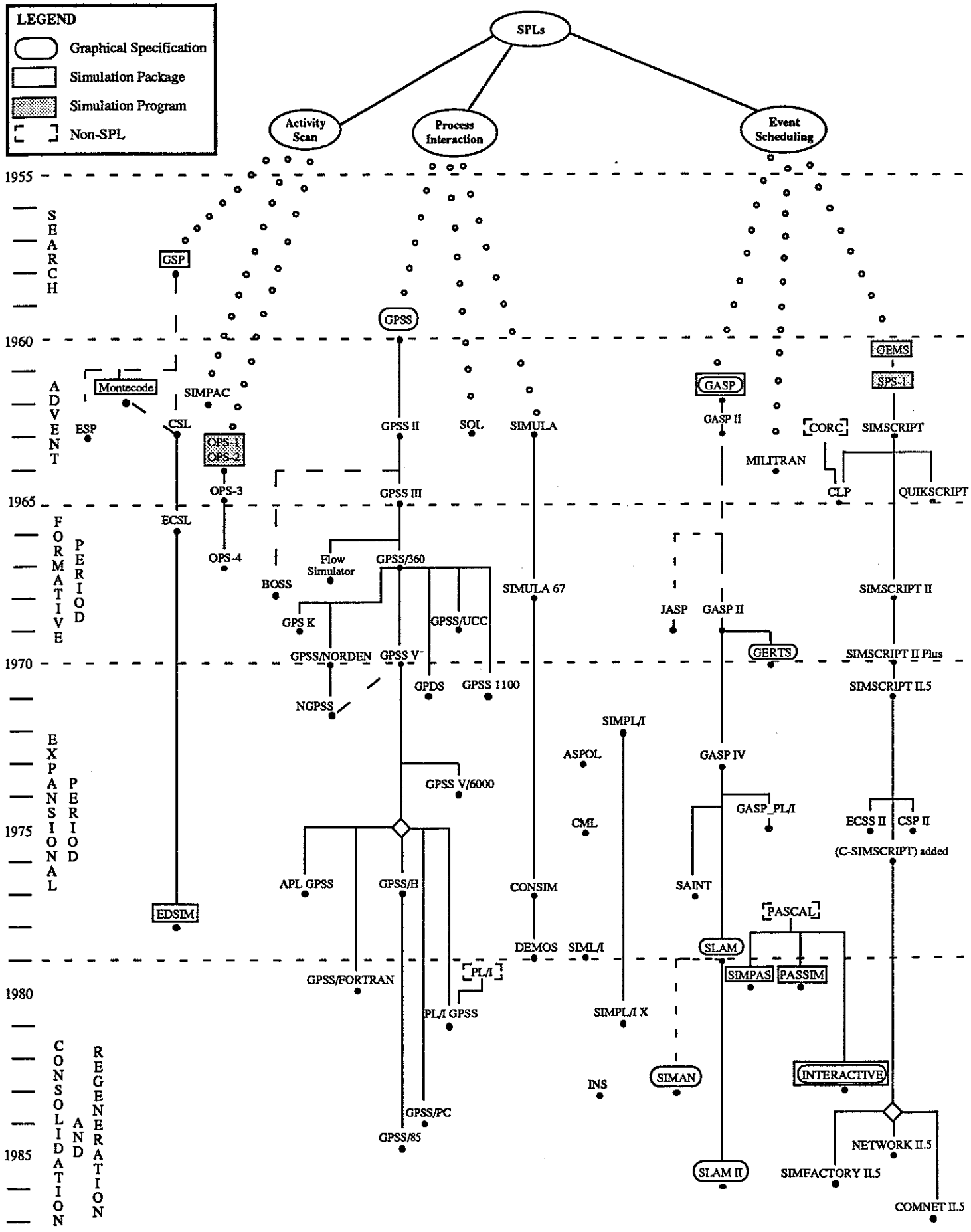


Figure 1. The Geneological Tree for Simulation Programming Languages

```
EXAMPLE FOR I.F.O.R.S. PAPER                    Title of job.
R1-5  B1  0  0  0   B3
T1-5   1  0  0  0  1440                         Initial values
U5                 1440                         for time-dependent
Z                                               machines, 1-5.

PARAMETERS
R-U11  +.462    +1000   +.576      +.4
L
R-W12  +.529   +46913  +75859  +129459  -5612  +3095
L
R-W13  +.371  +228219  -98655  +260848  -9995  +9162
L
R-W14  +.293  +120440  -30872   +45456  -4360  +2879
L
R-W15  +.801   +76869  +18834  +176332  -7907  +2895
L
R-W16  +.427  +228218  -98632  +228080  -9993  +9161
L
R-W17  +.516   +28675  +61077   +66404  -3897  +3295
L
Z                                               Parameters
                                                for routines
                                                SAMPLE and
                                                NEGEXP,
                                                defining
                                                distributions,
                                                etc.

DURATION  14400                                 Simulation to run for 10 days at a time.

TRANSLATE                                       (End of Initial Conditions.)

C1  PROCESSING                                          Activity C1

x ≠ 0                                           If there is a non-zero queue, and an available
n/2, A(n + 2), U = 0.                           unloaded machine, select process time for
Q(n + 2) → XYZ T11                              m/c and quality, record which quality this
v → SAMPLE R(n + Q(n + 2) + 12)                 item is, up-date idle time, start processing,
V(n + 2) - T + TIME, U → 1, D                   and reduce queue by one.
x - 1
K                                               ('K' marks end of Activity.)

C2  UNLOADING                                           Activity C2

A4                                              If the crane is available, and if there is an
n/2, A(n + 2), U = 1.                           available loaded machine, set (constant) un-
v → 3                                           loading time, up-date waiting time, start
W - T + TIME, U → 0, D                          unloading, record quality, and commit the
Q4 → Q(n + 2), D                                crane.
THEN B2
K
Z                                               (End of C-Activities.)

B1  ARRIVALS                                            Activity B1

v → NEGEXP R11/S                                Select next arrival interval, up-date no. of
Pn + 1, D                                       items in system, add 1 to queue, and if it now
x + 1, x > W,                                   exceeds previous maximum, record new
W → X                                           maximum.
K

B2  TRANSPORT TO STOCK                                  Activity B2

v → SAMPLE R(14 + Qn)                           Select transport time for given quality,
Rn → 0, D                                       cancel B2, and start transporting. Reduce
H, P1 - 1                                       items in system by 1, and check for consistency
H, P ≠ U2 + U3 + X,                             with m/c-states. (Note: 'H' allows operator
H, STOP                                         to suppress these statements.)
K

B3  REPORT RESULTS                                      Activity B3

PRINT.C   W1                                    Print:
PRINT.A   V2
PRINT.E   W2                                    TIME, MAX. QUEUE TODAY
PRINT.A   V3                                    IDLE TIME, WAITING TIME (m/c 1)
PRINT.E   W3                                    IDLE TIME, WAITING TIME (m/c 2)
Wn → (V2 + W2 + V3 + W3)                        PER CENT LOST TIME SO FAR
Wn → MULTIPLY Wn/50
Wn → DIVIDE.B Wn/T                              Reset reporting m/c to operate again after
PRINT.A   Wn                                    one day.
v → Un                                          Reset max. queue for tomorrow.
Dn
W1 → x
K                                               (End of B-Activities.)
Z

COMPILE                                         (End of Activities.)

XYZ
5.6   700
6     100
0     0717
0.1   1037
0     600
5.1   162
1.4   600
5.1   060
3     -000.
0
+0
+0
+0
14    600
5.1   610
5.6   710
0.0   7007
1.6   720
7     650
      F                                                 Subroutine XYZ

                                                Here, the text of the subroutine is supplied in
                                                Pegasus order-code. It will be incorporated
                                                in the specification of the model.
                                                (The routine produces one of two results: the
                                                quality '3' or the quality 'zero'. The former
                                                appears with probability given by element
                                                U11. A pseudo-random number stored in T11
                                                is used.)
```

Figure 2. Excerpts from a GSP Program

The treatment given to computer simulation in the 1961 issue of *Progress in Operations Research* [Ackoff, 1961] demonstrates the perceived importance of the technique. However, manual versus computer execution remains a matter of consideration [Morgenthaler 1961, pp. 406-407], and little consideration of language needs is indicated.

3. THE ADVENT (1961-1965)

The period of 1961-1965 marks the appearance of the forerunners of the major simulation programming languages currently in use. This claim is documented to some extent in the genealogical tree for SPLs (Figure 1).

The historical development of GPSS and SIMULA are described in detail in [Wexelblatt, 1981]. Herein, each is described sufficiently to furnish a trace of the evolution of the language through the subsequent time periods. More detailed description is provided for other languages emerging during this period, although none can be described to the extent permitted in the first conference.

3.1 GPSS

Beginning with the Gordon simulator in 1960, the General Purpose System Simulator (GPSS) was developed on various IBM computers, e.g., 704, 709, and 7090, during 1960-1961. With the later name change to the General Purpose Simulation System, GPSS developments continued under the version labeled GPSS II, which transitioned from table-oriented to list processing techniques. The GPSS III version released in 1965 was made available primarily for the large IBM systems (7090/94 and 7040/44) [IBM, 1965]. A major issue was the lack of compatibility between GPSS II and GPSS III.

GPSS has the distinction as the most popular SPL. Uses and users of the language outnumbered all others during the early years (prior to 1975). No doubt, this popularity was due in part to the position of IBM as the premier marketing force in the computing industry, and GPSS during this era of unbundled software was the discrete event SPL for IBM customers. That fact alone, however, does not explain the appeal of GPSS. The language readily gained proponents in the educational community because of the speed with which non-computer-oriented students could construct a model of a queueing system and obtain numerical results. Originating in the problem domain of communication systems, the GPSS block semantics were ideally suited for queueing models (although logic switches might have puzzled many business students).

The importance of graphical output for on-line validation was demonstrated in 1965 using an IBM 2250 interactive display terminal tied to a GPSS model [Reitman, 1992]. Boeing (Huntsville) were the first to employ this device to enable the observation of model behavior and the interruption and restart during execution.

3.2 SIMULA I

The selection of GPSS and SIMULA by the program committee for the first conference on the History of Programming Languages (HOPL I) is notable in that the two represent opposite poles in several measurement scales that could be proposed for languages. Within the simulation community in the United States (US), GPSS was viewed as easy to learn, user friendly (with the symbolic interface), highly used, limited (in model size and complexity), inflexible (especially so prior to the introduction of the HELP block), and expensive to run (the interpretive implementation). On the contrary, SIMULA was considered difficult to learn (lack of Algol knowledge in the US), lacking in its human interface (with input/output being an implementation decision following Algol 60), little used (in the US), but conceptually innovative and broadly applicable. The discovery of SIMULA by the programming language research community in the 1970s was probably the influencing factor in its selection for HOPL I rather than the regard held

for the language by simulation practitioners (notwithstanding its excellent reputation among a small cadre of simulation researchers in the US and a larger group in Europe).

Originating as an idea in the spring of 1961, SIMULA I represented the work primarily of Ole-Johan Dahl and Kristen Nygaard. The language was developed for the Univac 1107 as an extension of Univac's Algol 60 compiler. Nygaard and Dahl [1981] described the development of the language during this period as progressing through four main stages: (1) a discrete event network concept, (2) basing of the language on Algol 60, (3) modification and extensions of the Univac Algol 60 compiler, and the introduction of the "process concept," and (4) the implementation of the SIMULA I compiler.

The technical contributions of SIMULA are impressive, almost awesome. Dahl and Nygaard, in their attempt to create a language where objects in the real world could be accurately and naturally described, introduced conceptual advances that would be heralded as a paradigmatic shift almost two decades later. Almost lost in the clamor over the implementation of abstract data types, the class concept, inheritance, the co-routine concept, and quasi-parallel execution were the solution of significant problems in the extension of Algol 60. The creation, deletion, and set manipulation operations on objects are but one example. A second is that the sharing of attribute values by interacting processes required a means for breaking down the scoping restrictions of Algol 60. During the luncheon with language presenters at HOPL I in 1978, Nygaard remarked that only those who had programmed simulations really understood the power of the language (see [Wexelblat 1981, p. 485]).

3.3 SIMSCRIPT

SIMSCRIPT, in its original form was an event scheduling SPL. Two ancestors contributing to SIMSCRIPT are identified in the Preface to Markowitz [1963]: (1) GEMS (General Electric Manufacturing Simulator) developed by Markowitz at the General Electric Manufacturing Services and (2) SPS-1 developed at RAND. Morton Allen was identified as a major contributor to GEMS, and contributors to SPS-1 included Jack Little of RAND and Richard W. Conway of Cornell University.

3.3.1 Background

The RAND Corporation developed SIMSCRIPT under the auspices of the U.S. Air Force. The IBM 709/7090 computer was the target machine, and copies of SIMSCRIPT were distributed through SHARE (the IBM user's group). Harry Markowitz, whose broad contributions in several areas were to earn him a Nobel Prize, is generally attributed with the major design, and Bernard Hausner was the sole programmer, actually for both SPS-1 and the SIMSCRIPT translators. Herbert Karr authored the SIMSCRIPT manual [Markowitz 1963, p. iii].

SIMSCRIPT was considered a general programming system that could treat non-simulation problems, but the major purpose of the language was to reduce the model and program development times that were common in large efforts. Users were intended to be non-computing experts, and the descriptive medium for model development was a set of forms: (1) a SIMSCRIPT definition form in which variables, sets, and functions are declared, (2) an initialization form to define the initial system state, and (3) a report generation layout, for prescribing the format and content of simulation output. Accompanying the forms in the model definition is a routine for each endogenous and exogenous event and for any other decisions that prescribe the model logic. The applications context was machine or job shop scheduling and the initial examples reflected this focus.

3.3.2 Rationale for Language Content

3.3.2.1 Influencing Factors

The first version of SIMSCRIPT provided a set of commands that included FORTRAN statements as a subset. The syntax of the language and program organization were influenced considerably by FORTRAN. Basically, the language was transformed by a preprocessor into FORTRAN before compilation into the executable object program version. Insertion of a SIMULATION or NON-SIMULATION card in the compile deck differentiated between the intended uses, permitting omission of the events list and timing routines for non-simulation purposes. A control routine designated as MAIN replaced the timing routine.

SIMSCRIPT I.5, a version proprietary to CACI [Karr, 1965], is described by Markowitz [1979, pp. 27-28] as a "slightly cleaner version of SIMSCRIPT I" that did not appear much different externally. However, the SIMSCRIPT I.5 translator did not generate FORTRAN statements but produced assembly language. SIMSCRIPT I.5 is also described as germinating many of the ideas that later appeared in SIMSCRIPT II.

3.3.2.2 Language Design and Definition

SIMSCRIPT I (and SIMSCRIPT I.5) describe the world in terms of entities, attributes, and sets. Entities can be either permanent (available throughout the simulation duration) or temporary (created and destroyed during the simulation). Attributes define the properties of entities, and are distinguished by their *values*. Sets provide a convenience to the modeler for description of a class of entities (permanent entities), and set ownership and membership are defined relationships among entities.

The model structure is described in terms of a listing of events to prescribe the dynamic relationships, coupled with the entity and attribute definitions that provide a static description. Figure 3 taken from Markowitz [1963, p. 21] shows the declaration of exogenous and endogenous events for a job shop model. Figure 4 shows one exogenous input –an event with the name ORDRIN, also taken from Markowitz [1963, p. 22]. The event declarations create event notices that enable the timing executive to manipulate the events list, advancing time to the next imminent event. System-defined variables and functions such as TIME, RANDM, and PAGE are accessible by the modeler.

3.3.2.3 Non-technical Influences

Shortly after SIMSCRIPT I appeared, Karr left RAND to form Consolidated Analysis Centers, Incorporated (CACI). CACI was the major force behind SIMSCRIPT I.5, and implementations were done by Claude M. Delfosse, Glen Johnson, and Henry Kleine (see the Preface to [CACI, 1983]).

Referring to the language requirements described in Section 1 above, Figure 5 gives examples of each of the six language requirements. The terminology used in the middle column of Figure 5 follows that of the first version of SIMSCRIPT. Upper case in the examples shows the required syntax.

STATEMENT NUMBER		STATEMENT
1 2 5	6	7 72
		EVENTS
		3 EXOGENOUS
		ORDRIN (1)
		ANALYZ (2)
		ENDSIM (3)
		1 ENDOGENOUS
		EPROC
		END

Figure 3. Events List for Example – Job Shop Simulation in SIMSCRIPT

STATEMENT NUMBER		STATEMENT
1 2 5	6	7 72
		EXOGENOUS EVENT ORDRIN
		SAVE EVENT CARD
		CREATE ORDER
		LET DATE(ORDER) = TIME
		READ N
		FORMAT (I4)
		DO TO 10, FOR I = (1) (N)
		CREATE DESTN
		READ MGDST(DESTN), PTIME(DESTN)
		FORMAT (I4, H3.2)
		FILE DESTN IN ROUT(ORDER)
	10	LOOP
		CALL ARRVL(ORDER)
		RETURN
		END

Figure 4. Exogenous Event Routine Describing the Receipt of an Order

Language Requirement	Basis for Provision	Examples
Random Number Generation	Automatically generated functions Integers uniformly distributed from I to J	RANDI (I,J)
	Restart random number stream (odd integer)	RANDR(17319)
	Uniform variates from 0.0 to 1.0	RANDM
Process Transformers	Automatically generated functions Table look up with step functions (discrete) and linear interpolation (continuous)	
List Processing Capability	Entity Operations	CREATE temporary entity DESTROY temporary entity
	Control Constructs	FOR EACH OF set
Statistical Analysis	Miscellaneous Commands	ACCUMULATE COMPUTE
Report Generation	Form produced reports with specification of lines, text headings, spacing, etc.	
Timing Executive	Events Scheduling with Events List and Event Notices	EXOG EVENT ARRVL

Figure 5. The SIMSCRIPT Capabilities for Meeting the Six Requirements

3.4 CSL

Control and Simulation Language (CSL) was a joint venture of Esso Petroleum Company, Ltd. and IBM United Kingdom, Ltd. The purpose of the language was to aid in the solution of complex logical and decision making problems in the control of industrial and commercial undertakings [Esso, 1963].

3.4.1 Background

[Buxton, 1962] describe the simulation capabilities of CSL as based on the General Simulation Program [Tocher, 1960] and Montecode [Kelly, 1962]. The language is described as experimental in nature with an intentional provision of redundancy. This experimental approach also influences the design of the compiler, emphasizing modifiability over other factors such as reliability.

Buxton and Laski [1962, p. 198] attribute contributions to CSL from Tocher and his colleagues at United Steel Companies, Ltd., D.F. Hartley of Cambridge University, and I.J. Cromar of IBM, who had major responsibility in writing the compiler. They also acknowledge becoming aware of SIMSCRIPT only on the completion of the CSL definition.

No estimates are given for the language design effort, but the compiler is estimated to have required about nine man-months. The object program produced is described as "fairly efficient," both in terms of storage and execution time [Buxton 1962, p. 198]. A second version of CSL, described by Buxton [1966] and labeled C.S.L. 2 by Clementson [1966], attributed to P. Blunden, P. Grant, and G. Parncutt of IBM UK the major credit for the compiler design that resulted in the product described by [IBMUK 1965]. This version, developed for the IBM 7094, provided

extensive dynamic checking to aid program verification and offered some capability for parameter redefinition without forcing recompilation [Buxton 1966, p. 140].

3.4.2 Rationale for Language Content

3.4.2.1 Influencing Factors

Three factors appear to have influenced CSL:

(1) the intended use for decision making beyond simulation models prompted the adoption of a predicate calculus approach [Buxton 1962, p. 194],

(2) the reliability on FORTRAN for intermediate translation, and

(3) the dependence on techniques advanced by Tocher in GSP promoting the focus on activity as the basic descriptive unit.

The terminology of CSL resembles that of SIMSCRIPT in that entities, attributes, and sets are primary static model descriptors. The T-cells of CSL provide the basis for dynamic description, i.e., representation of time associated with the initiation and completion of activities.

3.4.2.2 Language Design and Definition

To the FORTRAN arithmetic and matrix manipulation capability, the developers of CSL added set manipulation operations, statistical utilities, and some logical operations to support the predicate calculus requirements intended for activity tests. The term ACTIVITIES both signals the use of CSL for simulation purposes (adding of timing requirements) as well as marking the beginning of activity description.

With reference to the six language requirements for discrete event simulation, CSL provided all, but some in limited forms as shown in Figure 6. The CSL manual contains no explicit identification of the random number generation technique although a linear congrential method is implied by the requirement of an odd integer initialization. Only the normal and negative exponential transformation techniques were supplied in addition to a user defined specification for table look up. Class definition of entities gave a major descriptive advantage over programming in the host language (FORTRAN II or FAP). The class definition example in Figure 6 shows a driver class with two attributes where entities belong to one of two sets, either local or long distance (LD). The use of TIME prescribes a T-cell associated with each driver. Set manipulation is illustrated in the FIND statement which shows that a customer entity has two attributes and the search is intended to find the first in the queue that has attribute 1 value greater than or equal to 10 and attribute 2 value greater than or equal to 35. Basically, conditions were tested to determine if an activity could initiate and the termination was usually prescribed through the assignment of T-cell value. The timing executive was very similar to that used by Tocher in GSP [Buxton 1962, p. 197].

Activities had a test and an action section. The test could be prescribed in rather complex logical statements comprising a chain. An evaluation of the chain would lead to the proper actions within the activity specification.

Translation was accomplished in four phases. An intermediate phase produced as output on an IBM 1401 (serving as input tape creator for an IBM 7090) a FORTRAN II program. Despite the multiphase translation with repetitive passes, the authors felt the process to be efficient, easy to modify, and to extend [Buxton 1962, p. 198]. A ratio of CSL to FORTRAN statements is "of the order of one to five." The authors also claim that an equivalent CSL program can be written in approximately 20% of the time required for a FORTRAN program.

The problem statement and example of CSL code shown in Figure 7 are taken from [Buxton 1962, p. 199].

Language Requirement	Basis for Provision	Examples
Random Number Generator	Automatically Generated Function Rectangular: variable = RANDOM (stream,range) (Type of generator not specified)	TM1=RANDOM(STREAMB,2)
Process Transformers	Automatically Generated Functions Normal: variable = DEVATE (stream,stddev,mean) Negative Exponential: variable = NEGEXP (stream, mean) User Defined: variable = SAMPLE (cell name,dist name,stream)	SERV.T = DEVIATE(STRMC,3,30) X = NEGEXP(BASE,MEAN(2)) T.MACH.K = SAMPLE(1, POISSON, STRM)
List Processing Capability	Class definition of objects having the same attributes that can be referenced as a set or sets with the use of TIME assigning a clock (T.cell) to each entity. Additions to (GAINS) and deletions from (LOSES) set, and searching based on FIRST, LAST, MAX (expression), MIN (expression), ANY	CLASS TIME DRIVER.10(2) SET LOCAL,LD FIND CUST QUEUE FIRST &120
Statistical	Provides only a HIST statement for accumulating data during program execution. FORTRAN II or FAP subroutines can be called	
Report Generation	FORTRAN input/output and formatting	
Timing Executive	Conditions specified which are tested. Time cells provide the link between time and state. Based primarily on GSP [Tocher, 1960]	WAIT.CUST(1) GE 10 WAIT.CUST(2) GE 35

Figure 6. The CSL Capabilities for Meeting the Six Requirements

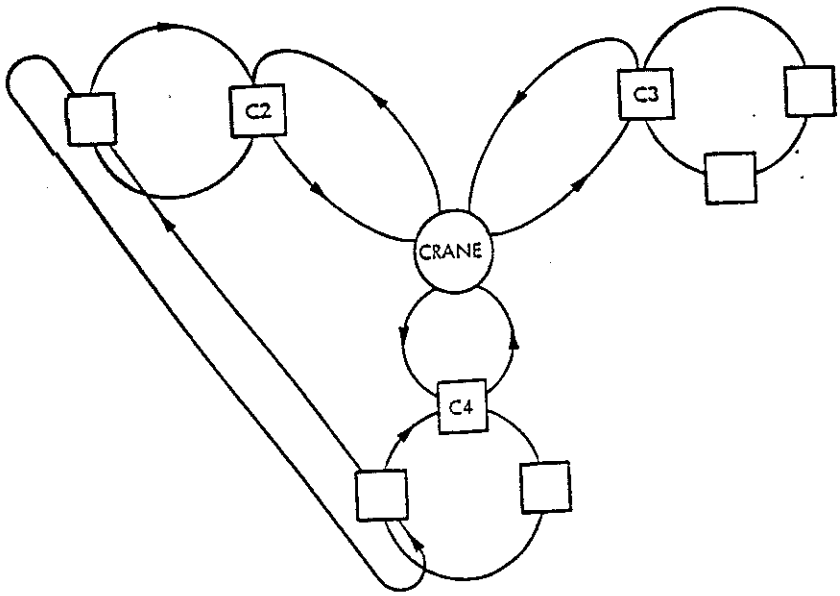
Initial statements in the program read in a two-dimensional data array of mileages between all airports in a given part of the world, and establish three sets which hold subgroups of airports. The first of these, AIRPORTS, holds the names of those airports between which possible transfer routings involving one change of aeroplane are required. The second set, TRANSFERPORTS, holds the names of the major airports where transfer facilities are possible. The third set, USED, is used during the program as a working-space set. A transfer routing is permissible provided that the total mileage flown does not exceed the direct mileage by more than 15%. The following program establishes valid transfer routings. The initial statements are omitted and the output statements are stylized to avoid the introduction of detail which has not been fully described in the paper. The transfer airports for each airport pair are written out in the order of increasing total mileage.

```
   FOR A AIRPORTS
      FOR B AIRPORTS
         A LT B  & 2
         WRITE A, B
         ZERO USED
1        FIND X TRANSFERPORTS MIN
            (MILEAGE (A, X)+MILEAGE (X, B))
            & 2
          X NE A
          X NE B
          100* (MILEAGE(A, X)+MILEAGE (X, B))
            LE 115*MILEAGE (A, B)
          AIRPORT.X NOTIN USED
         WRITE X
         AIRPORT. X HEAD USED
         GO TO 1
2        DUMMY
   EXIT
```

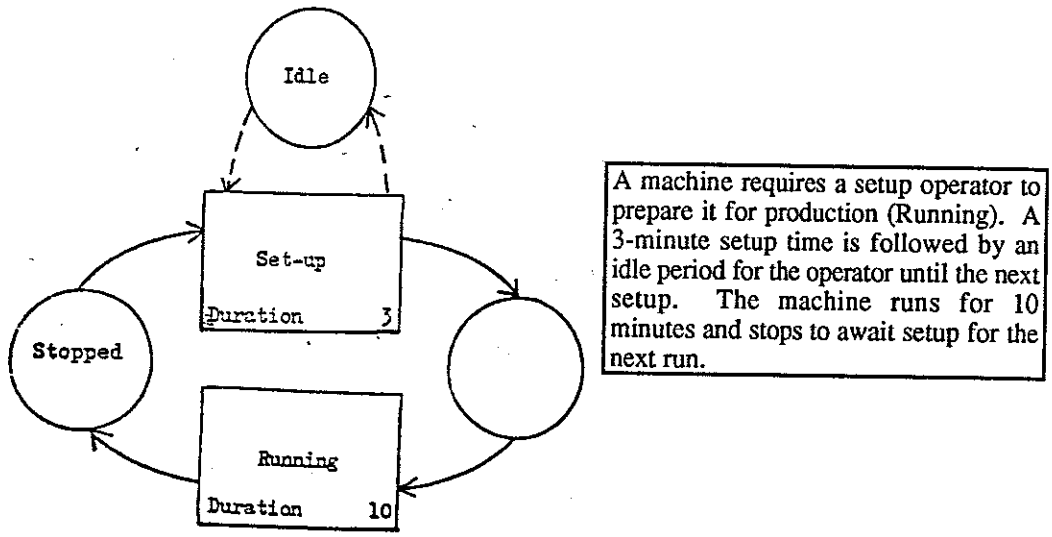
Figure 7. Problem Statement and Example of CSL Code [Buxton 1962, p. 199]

3.4.2.3 Non-technical Influences

CSL continued the GSP organization of a model through activities. It also continued the use of activity scan but provided a simpler two-phase as opposed to a three-phase executive, probably because the application domain stressed contingent (state conditioned) events more than determined (time dependent) events. Following both precedents kept CSL in the mainstream of British simulation efforts, and it soon became a favorite for program generation techniques based on the entity-cycle or activity cycle diagrams, the successors to the wheel charts. Examples of these early graphical modeling aids are shown in Figure 8.



(a) The Wheel Chart Example (from [Tocher 1966, p. 129])



(b) The Activity Cycle Example (from [ECSL-CAPS Reference Manual 1978, p. 9])

Figure 8. Wheel Chart and Activity Cycle Diagrams

3.5 GASP

GASP (General Activity Simulation Program) was developed by Philip J. Kiviat at the Applied Research Laboratory of the United States Steel Corporation. Kiviat came to U.S. Steel in June of 1961 and began work on the steel mill simulation project shortly thereafter [Kiviat, 1991a].

3.5.1 Background

The initial draft of GASP and a large part of the coding occurred during 1961, originally in ALGOL. Early on (presumably in 1961) the decision was made to base the GASP simulator on FORTRAN II. By February 1962, the design of the language was completed sufficiently so that Kiviat received approval for a seminar presentation entitled, "A Simulation Approach to the Analysis of a Complex Large Scale Industrial System," in the Department of Industrial and Engineering Administration at Cornell University. The invitation was extended by Professors Richard W. Conway and William L. Maxwell, both significant contributors to simulation conceptual development.

Developed for internal use within United States Steel, GASP was intended to run on several machines, e.g., IBM 1410, IBM 1620, Control Data 1604, and Bendix G-20. A preliminary version of the Programmer's Manual, dated January 16, 1963, also showed intended execution on IBM 7070 and 7090 with erasures of pencil inclusions of the IBM 7074 and 7094. Editorial markings by Kiviat noted that an update of the target machines would be necessary at final typing [Kiviat, 1963a].

A report during the same time period [Kiviat, 1963c] clearly casts GASP as the key ingredient in a method for utilizing computer simulation in the analysis of steel industry operations. The intended report was to educate potential users as to the benefits derived from simulation studies and to serve as an introduction to the use of the technique with a GASP "simulation-programming system that greatly reduces the time required for problem analysis, computer programming, and computer use," [Kiviat 1963c, Abstract].

3.5.2 Rationale for Language Content

Exposed to the subject in his graduate study at Cornell, Kiviat was both knowledgeable in simulation techniques and aware of other language developments. The completed version of the report describing GASP [Kiviat, 1963b] (a revision of the earlier titled "Programmers' Manual") contained references to SIMSCRIPT, Tocher's GSP, SIMPAC, and industrial dynamics [Forrester, 1961]. An incomplete reference contains the name "Gordon, Geoffery [sic]," but nothing more.

3.5.2.1 Influencing Factors

GASP was designed to "bridge the gap" between two groups: (1) operating or engineering personnel, unfamiliar with computer programming, and (2) computer programmers unknowledgeable of the application domain. Like GPSS, flow-chart symbols were intended to be used by the operational personnel in defining the system. The symbols followed the conventions for general purpose use. An example, taken from [Kiviat, 1963a], is shown in Figure 9. The sketch of language capabilities, provided in Figure 10, is supplemented by the following explanation.

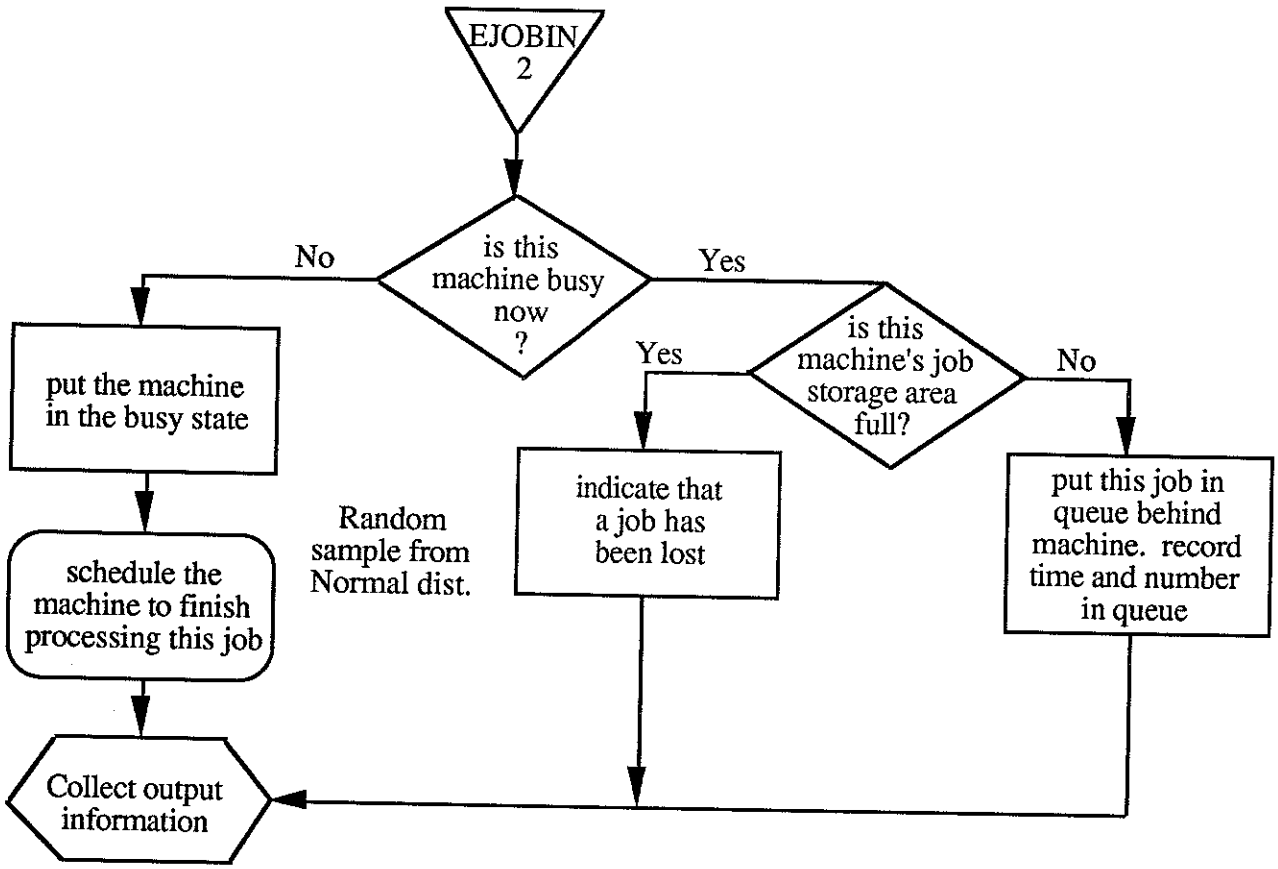


Figure 9. The Flow-Chart Representation of an Event (EJOBIN) in GASP

Language Requirement	Basis for Provision	Examples
Random Number Generation	FORTRAN II resident library, NRANDM(d) with d = 1, 2, 3, or 4 gives an integer $[1,10^d]$	JDGT = NRANDM(I) – 1
Process Transformers	Uniform, Poisson, Normal and Erlang distributions. An empirical distribution also could be defined.	NUM = NPOISN (3) (where 3 specifics the row of the input parameter list which contains the actual parameter value)
List Processing Capability	Subroutines provided the capability for filing (FILEM) and retrieving (FETCHM) elements using FIFO or LIFO disciplines in a FORTRAN array.	CALL FILEM (TMDUE,JDDLR, KMATL,QUEUE,1) CALL FETCHM (FLOATF(JDOLR), XINTF(TMDUE),KMATL,QUEUE,1) (Note: first argument is the basis for insertion and removal)
Statistical	Subroutines COLECT: the sum, sum of squares, extreme values and sample size for up to 20 designated variables HISTOG: forms a histogram for each up to 20 designated variables	CALL COLECT(WAITM,3) (Note: 3 designates the row of output for the collected values of WAITM)
Report Generation	The OUTPUT subroutine must be written by the user but the statistical subroutines above and PRINTQ provide the functionality.	
Timing Executive	Event scheduling with a scheduling subroutine and using an associated list organization developed by Kiviat [1962a, 1962b]	CALL SCHDL(ARVTM,JARVL, KCUST)

Figure 10. The GASP Capabilities for Meeting the Six Requirements

A GASP model was made up of elements, e.g., people, machines, orders, with elements described in terms of *attributes*. A list of attributes describing an element corresponded to the later concept of a record. Changes in the value of attributes occur through *activities*, which could either be instantaneous or require a time duration. Event routines were described by the user, the operating personnel, through flow charts subsequently translated into executable representations by programmers.

Subroutines written in FORTRAN II provided the list processing capabilities (queue insertion, item removal, etc.) relying on the array data structure completely. Reporting was centralized in a single subroutine (OUTPUT) to which transfers were made, dependent on the data being collected.

The FORTRAN II resident library provided the random number generator, and GASP utilized transformations for the generation of variates following the uniform, Poisson, normal, and Erlang distributions. In addition, sampling could produce a value from an empirical distribution (based on sample data).

A rather unique feature of GASP was the provision of a regression equation from which sampling could be used to produce input values primarily. The number of arguments for the regression equation was limited to 10 or less, and the error term was expected to follow a normal distribution.

Timing was the responsibility of the GASP EXECUTIVE, which utilized the data supplied by the modeler's use of scheduling (SCHED) calls to sequence events properly. Debugging was also a major concern, and features were provided in a special subroutine (MONITR) that assisted in this effort. A routine for error determination reported 13 types of errors.

3.5.2.2 Non-technical Influences

A revision to GASP, labeled GASP II, was underway in early 1963. No doubt, further developments were hindered by the departure of Kiviat who took a position at the RAND Corporation. A later version of the GASP User's Manual, which is believed to be in the 1965 time frame, was authored by Jack Belkin and M.R. Rao [Belkin, 1965?].

3.6 OPS-3

OPS-3 is classed as a major SPL, not because of conceptual influences but because it was an innovative and technical effort considerably ahead of its time. Growing out of classroom instruction at MIT during the spring, 1964, OPS-3 was a prototyping effort building on two earlier versions, each the result of classroom experiments. The MIT time sharing system, CTSS, stimulated the experiments in interactive model building and simulation programming.

3.6.1 Background

The major contributors to OPS-3 were Martin Greenberger, Malcolm M. Jones, James H. Morris, Jr., and David N. Ness. However, a number of others are cited in the preface of [Greenberger, 1965]: M. Wantman, G.A. Gorry, S. Whitelaw, and J. Miller. The implication is that all in this second group were students in the Alfred P. Sloan School of Management.

The OPS-3 *system* was intended to be a multi-purpose, open-ended, modular, compatible support for creative researchers. The users were not intended to be computing experts, but persons involved in problem solving and research that necessitated model building. Additions to OPS-3 could be from a variety of programming languages or represent a synthesis of subprograms without modification [Greenberger 1965, pp. 1-2].

3.6.2 Rationale for Language Content

OPS-3 was intended to be a general programming tool with extended capabilities for handling modeling and simulation to support research users. As pointed out above, OPS-3 was viewed as a system and not simply a language.

3.6.2.1 Influencing Factors

The major influence on OPS-3 was the interactive, time-sharing environment provided by CTSS. For the first time simulation modeling was to be done in a quasi-real-time mode, but not necessarily with interactive assistance beyond that provided by CTSS. The AGENDA, provided as a compound operation (KOP), enabled the scheduling, cancellation, and rescheduling activities following the "spirit of simulation languages such as SOL and SIMSCRIPT," [Greenberger 1965, p. 7]. The power of interactive model execution was recognized as something totally new.

3.6.2.2 Language Design and Definition

OPS-3 is structured in a hierarchical fashion with a basic calculation capability furnishing the lowest level of support. The system provides five modes or user states: (1) Execute, (2) Store, (3) Store and Execute, (4) Run, and (5) Guide. Simple operators such as PRINT are available in *execute mode*. Cascading a series of operations into a program is enabled with *store mode*. If single operators are to execute within a program then *store and execute* mode was used. *Run* mode

enabled execution of compound operations (KOPs) without interruption. The assistance or help was accessible in *guide mode*. In the terminology typical of the time, OPS-3 would be categorized as emphasizing operators rather than data structures. An operator took the form of a closed subroutine written in MAD (Michigan Algorithmic Decoder), FORTRAN, FAP (an assembler), or any other source language available under CTSS. Over 90 operators were provided in the language, and 75 subroutines were provided in the library.

In terms of the six language requirements for simulation, Figure 11 shows the capabilities provided by OPS-3. Especially impressive were the statistical capabilities for input data modeling and output analysis through an operator set that provided linear least-squares fit, inter-correlation and partial correlation, tests of homogeneity, contingency table analysis and others. A guided form of each test ushered the novice user through the requirements for performing the test. A short form was available for the experienced user.

The determination of the next activity is based on a three-phase method resembling that used in Tocher's GSP [Tocher, 1960]. The contingent event is placed on the AGENDA in an order dictated by TIME, the system clock, in the sequence of conditional events. Parameters can be passed with the SCHED operator, and the AGENDA can be accessed by the user as a debugging aid.

Language Requirement	Basis for Provision	Examples
Random Number Generation	Defined function, presumably a linear congruential method. DRAW arguments: returned val, distribution, parm1, parm2. Initialization possible with SEED an odd integer.	DRAW PRIORITY RANDOM 1 7 SEED 6193
Process Transformers	The second argument of DRAW specified the distribution: exponential, normal or modeler specified. Default parameters of 0 (mean) and 1 (std, dev.) for normal.	DRAW SERVTM EXPONE THETA DRAW MEMRY DEMAND CLASS
List Processing Capability	Major control through user-accessible symbol table. Major data structure is array but with more flexible manipulation than with FORTRAN. Stack operators permit entry, removal, and examination of values.	RESETQ WLINE Empties the queue WLINE GETOLD REDYQ PROCQ Places oldest item in REDYQ into PROCQ
Statistical	Extensive statistical operators with capability for input data modeling and output analysis.	CNTING X 2-way contingency table with X a floating-point matrix. TTESTS X N1 N2 A t-test between two samples of size N1 and N2 with the values in the matrix X
Report Generation	Nothing beyond the usual interactive input and output.	
Timing Executive	A three-phase method built around the KOP AGENDA. SCHED activity option basis CANCEL option activity variable	SCHED SRVCE AT 100 SCHED ENDSV NOW CANCEL FIRST ARRVL ATIME

Figure 11. The OPS-3 Capabilities for Meeting the Six Requirements

3.6.2.3 Non-technical Influences

This period in MIT computing was certainly one of the most exciting. Time sharing and interactive computation were in their infancies. Innovative thinkers such as J.C.R. Licklider, John McCarthy, and R.M. Fano were pushing the boundaries of computing technology to the utmost. Consider the insightful perspective of Licklider concerning the future role of languages for dynamic interactive modeling [Licklider 1967, p. 288-289].

> 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.

Within less than two years of publication, the DEC PDP-10 would supply even more advanced facilities than afforded to the developers of OPS-3. The efforts of Greenberger, Jones, and others with OPS-3 deserve recognition for the perceptions and demonstration of technology potential rather than the effect on SPL developments.

3.7 DYNAMO

The DYNAMO (DYNAmic MOdels) language served as the executable representation for the industrial dynamics systems models developed by Jay Wright Forrester and others at MIT during the late 1950s and into the 1960s and beyond. DYNAMO is the lone non-discrete event simulation language included in this history. Justification for this departure is that concept, techniques, and approaches utilized by the MIT group had a significant impact on those working in the development of discrete event SPLs [Kiviat, 1991b].

3.7.1 Background

The predecessor of DYNAMO, a program called SIMPLE (Simulation of Industrial Management Problems with Lots of Equations), was developed by Richard K. Bennett for the IBM 704 computer in the spring of 1958. (Note that this is a contemporary with GSP in the discrete-event SPL domain.) SIMPLE possessed most of the basic features of DYNAMO, but the model specifications from which the program was derived were considered to be rather primitive [Pugh 1963, p.2]. DYNAMO is attributed to Dr. Phyllis Fox and Alexander L. Pugh, III with assistance from Grace Duren and David J. Howard. Modifications and improvement of the original SIMPLE graphical plots was provided by Edward B. Roberts [Forrester 1961, p. 369].

Originally written for an IBM 704 computer, the DYNAMO compiler, consisting of about 10,000 instructions, was converted to an IBM 709 and then an IBM 7090 by Pugh. The DYNAMO compiler as described in the User's Manual represented about six staff-years of effort (including that required to develop SIMPLE), and the maximum model size was about 1500 equations. A FORTRAN package (FORDYN), intended for users who did not have access to the large IBM 7090 or 7094, was developed by Robert W. Llewellyn of North Carolina State University in 1965 [Llewellyn, 1965]. (Note that neither DYNAMO nor FORDYN are included in Figure 1.)

3.7.2 Rationale for Language Content

While applicable to the modeling of any information feedback system, DYNAMO was developed as the means of implementing industrial dynamics models, which addressed the application of simulation to large scale economic and social systems. The level of modeling granularity does not address the individual items or events typically associated with the job shop models of that time. A thorough understanding of the basis for systems dynamics modeling, the term later employed, is beyond the scope of this paper (see [Forrester, 1961]).

3.7.2.1 Influencing Factors

The design of DYNAMO was strongly influenced by four desirable properties [Pugh 1963, p.2]:

(1) The language should be easily understood and easy to learn by the user group, who in general might not be professional programmers. Flexibility would be sacrificed for both ease of use and correctness.

(2) A very efficient (at the time) compilation phase, with no object language production, eliminated the usual practice of the time of saving object text.

(3) Output in the form of graphical plots was recognized as more useful than extensive numerical results; however, both were provided to the user.

(4) Error detection and error correction were considered a primary responsibility. Both initial conditions and the order of computation were subject to detection and correction. The claim was that DYNAMO checked for almost every logical inconsistency, and provided comments on errors that were easily understood.

3.7.2.2 Language Design and Definition

DYNAMO is described as a language expressing zero- and first-order difference equations. A system is described in terms of flows and accumulations. Flows in orders, material, personnel, money, capital equipment, and most importantly information, affect the decision making that leads to changes in the rate of flow occurring in subsequent time periods. Over time the changes in flow contribute to the redefinition of levels (of money, materials, information) which have subsequent influence on future rates of flow. Each model must begin with an initial state definition, and the constant increment of time (DT) is selected to be sufficiently small to avoid inherent instability.

Figure 12 portrays the timing sequence for each set of DYNAMO computations. The present time (K), has values determined based on levels of accumulation at J modified by rates of flow over the JK interval. Levels at present time (K) then contribute to the definition of rates over the interval KL, where L = K + DT.

Figure 13 is an example taken from [Pugh 1963, p.17] showing the model of a retail store that illustrates the DYNAMO representation. The equational representation in Figure 13 is understood by noting that the number of each equation is suffixed by the type of equation: L for level; A for auxiliary; R for rate; N for initial value. NOTE is a comment statement, and special statements such as PRINT, PLOT, and SPEC provide instructions on the execution process. The subscripting enables not only a determination of direct statement formulation, but also of statement ordering and typing. While discontinuities were permitted in systems dynamics, and could be represented in DYNAMO, they were discouraged unless absolutely necessary as a modeling technique.

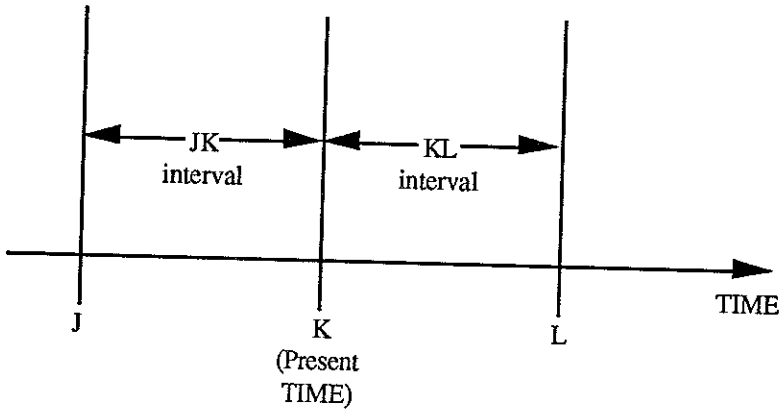


Figure 12. Time Notation (from [Pugh 1963, p.4]

```
*       M478-248,DYN,TEST,1,1,0,0
RUN     2698JP
NOTE    MODEL OF RETAIL STORE
NOTE
1L      IAR.K=IAR.J+(DT) (SRR.JK-SSR.JK)          INVENTORY ACTUAL
1L      UOR.K=UOR.J+(DT) (RRR.JK-SSR.JK)          UNFILLED ORDERS
20A     NIR.K=IAR.K/DT                            NEGATIVE INVENTORY
20A     STR.K=UOR.K/DFR                           SHIPMENTS TRIED
54R     SSR.KL=MIN(STR.K,NIR.K)                   SHIPMENTS SENT
40R     PSR.KL=RRR.JK+(1/DIR) (IDR.K-IAR.K)       PURCHASE ORDERS SENT
12A     IDR.K=(AIR) (RSR.K)                       INVENTORY DESIRED
3L      RSR.K=RSR.J+(DT) (1/DRR) (RRR.JK-RSR.J)   REQUISITIONS SMOOTHED
39R     SSR.KL=DELAY3(PSR.JK,DTR)                 SHIPMENTS RECEIVED
NOTE
NOTE    INITIAL CONDITIONS
NOTE
12N     UOR=(DFR) (RRR)
6N      RSR=RRR
6N      IAR=IDR
NOTE
NOTE    INPUT
NOTE
7R      RRR.KL=RRI+RCR.K                          REQUISITIONS RECEIVED
45A     RCR.K=STEP(STH,5)                         REQUISITION CHANGE
NOTE
NOTE    CONSTANTS
NOTE
C       AIR=8 WKS                                 CONSTANT FOR INVENTORY
C       DFR=1 WK                                  DELAY IN FILLING ORDERS
C       DIR=4 WKS                                 DLY REFILLING INVENTORY
C       DRR=8 WKS                                 REQUISITION SMTHNG  T C
C       DTR=2 WKS                                 DELAY IN TRANSIT
C       RRI=1000 ITEMS/WK                         REQ. RECEIVED INITIALLY
C       STH=100 ITEMS/WK                          STEP HEIGHT
NOTE
PRINT   1)IAR,IDR/2)UOR/3)RRR,SSR/4)PSR,SRR
PLOT    IAR=I,UOR=U/RRR=R,SSR=S,PSR=P,SRR=Q
SPEC    DT=0.1/LENGTH=10/PRTPER=5/PLTPER=0
```

Figure 13. Listing of model.

The graphical output produced on a lineprinter consisted of letters showing the relative values for designated variables. The use of plotters with smoothed curvilinear, coded representations is provided in the book by Forrester, but not available with the early version of DYNAMO. The fixed format field definition required by the translator was to be replaced in later versions.

3.7.2.3 Non-technical Influences

Systems dynamics and DYNAMO, although developed totally at MIT, attracted considerable nationwide interest. Financial support from the Ford Foundation and the Sloan Research Fund are acknowledged in the Preface of [Forrester, 1961]. Additional support from a number of sources is identified there also.

Since DYNAMO was not a discrete event SPL, and because its influence was limited to the developmental period, successive versions of the language are not described. However, the language continued, both in refined versions [Pugh, 1973] and extensions [Pugh, 1976].

3.8 Other Languages

Numerous SPLs emerged during this advent period. Most, experienced some use and disappeared. A few had distinctive characteristics which deserve some note in a paper of this type.

MILITRAN was produced by the Systems Research Group for the Office of Naval Research [Systems Research Group, Inc., 1964]. Krasnow [1967, p. 87] states that little concession to military subject matter was given in MILITRAN. Event scheduling was the world view promoted, and a distinction between permanent (synchronous) and contingent events was made.

The Cornell List Processor (CLP) was a list processing language used extensively for simulation instruction. The developers had the goal of producing a list processing language that required no more than a "FORTRAN level" knowledge of programming [Conway, 1965]. CLP relying on CORC, a general algebraic language used at Cornell also for instructional purposes, enabled students to define entities, use them in set manipulation (INSERT, REMOVE) without the major effort of learning a language such as SIMSCRIPT [Conway 1965, p. 216]. The student still had to write his or her own timing, statistical analysis, and report generation routines.

QUIKSCRIPT was a SIMSCRIPT derivative simulator, a set of subroutines, based on the 20-GATE algebraic language used at Carnegie Institute of Technology [Tonge, 1965]. The GATE subroutines did not provide all the facilities of SIMSCRIPT (e.g., no report generation), and the definitional forms of the latter were omitted.

SIMPAC, produced at the System Development Corporation, was distinguished by its fixed-time-increment timing routine. This would appear to make it the only U.S. language to adhere completely to the activity scan world view [Bennett, 1962]. (OPS-4 departed from OPS-3 by accommodating all three world views.)

SOL (Simulation Oriented Language) was developed by Knuth and McNeley as an extension to ALGOL. The language is structured much like GPSS, even using terms such as SEIZE, RELEASE, and ENTER. Sets are represented as subscripted variables. In contrast with SIMULA, SOL focused on the dynamic interaction of temporary objects, again much like GPSS [Knuth, 1964a] [Knuth, 1964b]. Of note in SOL was the explicit use of a *wait on state condition* that was not present in GPSS or in SIMULA, since the prevailing view was that such an indeterminate expression could lead to gross inefficiencies (see [Nygaard 1981, p. 452] for a discussion specific to SIMULA and [Nance, 1981] for a more general discussion). A second, more anecdotal, item related to SOL was the response of Knuth when asked at the IFIP Conference what were the plans for SOL, he replied that there were no plans for he found SIMULA to have all the capabilities of SOL (except for the *wait on state condition*) [Knuth, 1992] and more.

3.9 Language Comparisons

The end of the advent period marked the beginning of a period of intense interest in simulation programming language comparison and evaluation. Such interest was manifested in the

publication of a number of papers during the 1964-67 timeframe that reviewed and compared existing languages with respect to many criteria [Young, 1963; Krasnow, 1964; Tocher, 1965; Kiviat, 1966; Teichrow, 1966; Krasnow, 1967; Reitman, 1967]. These sources also refer to lesser known languages that are not mentioned here. An in-depth comparison of SIMSCRIPT 1.5 and SIMULA I is given in [Wegner 1968, pp. 332-348].

Interest in comparison and evaluation was likely stimulated by the perceived cost associated with using "another language." The cost of acquiring the translator was minor; the cost of training people, maintaining the language and supporting its migration to subsequent hardware systems could be very high. For that reason, packages in a general purpose language had considerable appeal.

This sharp interest in language comparisons is also marked by the number of workshops, symposia, and conferences addressing SPL issues. Daniel Teichrow and John F. Lubin are cited by Philip Kiviat as the "honest brokers" in that they had no commitment to a particular language but were instrumental in developing forums for the exchange of concepts, ideas, and plans. A workshop organized by them and held at the University of Pennsylvania on March 17-18, 1966 refers to an earlier one at Stanford in 1964 [Chrisman, 1966]. Session chairs at the 1966 workshop included Kiviat, R.L. Sisson (University of Pennsylvania), Julian Reitman (United Aircraft), and J.F. Lubin. Comparison papers were presented by Harold G. Hixson (Air Force Logistics Command), Bernard Backhart (General Services Administration), and George Heidorn (Yale University). Among the 108 attendees were several who were involved in SPL development: W.L. Maxwell of Cornell (CLP), Malcolm M. Jones of MIT (OPS), Howard S. Krasnow and Robert J. Parente of IBM, and Julian Reitman of Norden (GPSS/Norden).

In between the two workshops described above was the IBM Scientific Computing Symposium on Simulation Models and Gaming held at the T.J. Watson Research Center in Yorktown Heights, New York on December 7-9, 1964. A session entitled, "Simulation Techniques" included a paper by Geoffrey Gordon that compares GPSS and SIMSCRIPT [Gordon, 1966] and a paper by Tocher that presents the wheel chart as a conceptual aid to modeling [Tocher, 1966]. In another session, the Programming by Questionnaire (PBQ) technique for model specification is described [Geisler, 1966]. (More about PBQ is to follow.) The Symposium took a broad applications view of simulation, and among the 175 attendees at this conference were Jay W. Forrester (MIT), who described the principles of industrial dynamics, Richard M. Cyert (CIT), Herbert A. Simon (CIT), Philip Morse (MIT), J.C.R. Licklider (MIT), Harold Guetzkow (Northwestern), Richard W. Conway and William L. Maxwell (Cornell), Oscar Morgenstern (Princeton), and Guy H. Orcutt (Wisconsin).

The most notable technical conference on simulation languages was the IFIP Working Conference on Simulation Programming Languages, chaired by Ole-Johan Dahl and held in Oslo, 22-26 May 1967. The *Proceedings* were edited by John N. Buxton and appeared the following year. The list of presenters and attendees reads like a "Who's Who," not only in computer simulation but also in computer science. The "by invitation only" participants, with their role or presentation subject in parentheses, included Martin Greenberger and Malcolm M. Jones (OPS-4), Michal R. Lackner (graphic forms for conversational modeling), Howard S. Krasnow (process view), Donald E. Knuth (Session Chair), Jan V. Garwick (Do we need all these languages?), R.D. Parslow (AS: an Algol language), Ole-Johan Dahl and Kristen Nygaard (class and subclass declaration), L. Patrone (SPL: a simulation language based on PL/I), John G. Laski (interactive process description and modeling languages), G.K. Hutchison (multiprocessor system modeling), Robert J. Parente (simulation-oriented memory allocation), G. Molner (self-optimizing simulation), G.P. Blunden (implicit interaction), John L. McNeley (compound declarations), C.A.R. Hoare (Session Chair), A.L. Pugh III (DYNAMO II), T.B. Steel, Jr. (standardization), and Evzen Kindler (COSMO). The highly interactive group of participants included: Richard W. Conway, Douglas T. Ross, Christopher Strachey, Robin Hills, and John N. Buxton. Issues and problems surfacing in this symposium on occasion resurface, for example, in the annual Winter Simulation Conferences.

In November 1967, Harold Hixson, Arnold Ockene and Julian Reitman collaborated to produce the Conference on Applications of Simulation Using GPSS held in New York. Hixson,

representing SHARE (the IBM User Group), served as General Chair, Ockene of IBM served as Publicity Chair, and Reitman of Norden Systems was the Program Chair. In the following years this conference expanded its programming language scope (beyond GPSS) and became known as the Winter Simulation Conference (WSC), which celebrated its twenty-fifth anniversary in December 1992 in Washington. The three individuals named above, together with other pioneers in the early years of the WSC, were brought together in a panel session entitled "Perspectives of the Founding Fathers" [Wilson, 1992].

4. THE FORMATIVE PERIOD (1966-1970)

Following the bustle of activity surrounding the emergence of new SPLs, the period from 1966-1970 marked a consolidation in conceptual clarification. The concepts, possibly subjugated earlier in the necessities of implementation, were reviewed and refined to promote more consistent representation of a world view and improve clarity in its presentation to users. Nevertheless, rapid hardware advancements and vendor marketing activities forced some languages, notably GPSS, to undergo major revisions.

4.1 GPSS

GPSS II and III are included in the advent period as is the ZIP/ZAP compiled version of the language translator. With the introduction of SYSTEM/360 in the 1960s, GPSS/360 represented an improved and extended version of GPSS III that appeared in two versions (see [Schriber 1974, p. 496]). An RCA version of the language, called *Flow Simulator*, represents a combined version of the two [Greenberg 1972, p. 8]. *Flow Simulator* appeared in 1967 [Flow Simulator, RCA , 1967] [Flow Simulator Information Manual, RCA, 1967]. A Honeywell version, GPS K appeared in 1969 [GPS K, 1969]. Two versions of a User's Manual for GPSS V appeared in 1970-71 [GPSS V, 1970] [GPSS V, 1971]. GPSS/UCC, corresponding closely to GPSS/360, was developed by the University Computing Corporation in the late 1960s.

GPSS/360 extended the prior version (GPSS III) by increasing the number of block types from 36 to 44. Set operations were expanded by the introduction of groups. Extensions to the GENERATE block improved storage use. A HELP block permitting access to routines in other languages was provided. Comparisons of the language versions for GPS K, *Flow Simulator*, and GPSS III with GPSS/360 can be found in the Appendices of [Greenberg, 1972].

GPSS V provided more convenience features but made no major changes in the language. A run timer could be set by the modeler, extensions were made to the HELP block, and free format coding (removal of statement component restrictions to particular fields) was permitted. A detailed comparison of differences in GPSS/360 and GPSS V is provided in Appendix A of [Schriber, 1974].

4.2 SIMULA 67

The realization of shortcomings in SIMULA I and the influences of language and translator developments during the mid-1960s led to an extensive revision of the language. This revision, described in detail in [Nygaard, 1981], is not repeated here. Needless to say, the class concept was a major innovative contribution. Clearly, SIMULA 67 served to crystallize the process concept, the co-routine and quasi-parallel processing capabilities and demonstrate the implementation of abstract data types. The result was a language well beyond the power of most of its contemporaries.

4.3 SIMSCRIPT II

SIMSCRIPT II, although dependent on SIMSCRIPT I.5 for its basic concepts of entity, attribute, and set, was clearly a major advancement over the earlier version. SIMSCRIPT II is

intended to be a general purpose language; while SIMSCRIPT I.5 was intended to be a simulation programming language. SIMSCRIPT II in appearance looks much different from SIMSCRIPT I [Markowitz 1979, p. 28]. An expressed goal of SIMSCRIPT II was to be a self-documenting language. SIMSCRIPT II was written in SIMSCRIPT II, just as its predecessor was written, for the most part, in SIMSCRIPT I.5.

4.3.1 Background

The RAND Corporation provided the organizational support and financial underpinnings for SIMSCRIPT II. Philip J. Kiviat, having come from the United States Steel Company, took over the leading role in the design and development of the language from Harry Markowitz, who had started the project. Markowitz, whose ideas had formed the basis for SIMSCRIPT I and SIMSCRIPT I.5, was in the process of leaving RAND during the major part of the language project. Bernard Hausner, the principal programmer on SIMSCRIPT I.5, had departed, and Richard Villanueva assumed this role.

While the claim was made that SIMSCRIPT II was a general purpose language, RAND's principal interest in developing SIMSCRIPT II was to enhance its capability within discrete event simulation and to offer greater appeal to its clients [Kiviat 1968, p. vi]. Many of RAND's models were military and political applications, necessitating extremely large complex descriptions. The intent was to create a language that, through its free-form and natural language appearance, would encourage more interaction with application users.

Contributions to the language are acknowledged in the preface to the aforementioned book. George Benedict and Bernard Hausner were recognized as contributing much of the basic compiler design and programming. Joel Urman of IBM influenced the language design as well as programmed much of the I/O and operating system interface routines. Suggestions and criticisms were attributed to a number of persons, including Bob Balzer, John Buxton, John Laski, Howard Krasnow, John McNeley, Kristen Nyggard, and Paula Oldfather [Kiviat 1968, p. vii].

In addition to a free-form, English-like mode of communication, the language designers desired a compiler to be "forgiving," and to correct a large percentage of user syntax errors. Furthermore, forced execution of a program was felt to reduce the number of runs to achieve a correct model implementation. Debugging statements and program control features were central to the translator.

4.3.2 Rationale for Language Content

4.3.2.1 Influencing Factors

Certainly, SIMSCRIPT I.5 and the entity, attribute, and set concepts had the major influence on the language design. Nevertheless, the intent to involve application users and to provide a language working within SYSTEM/360 and OS/360, both still somewhat in development, had some impact. At one point, the language designers considered writing SIMSCRIPT II in NPL (the New Programming Language), subsequently PL/I, but the idea was rejected because of the instability of NPL [Kiviat, 1992]. Kiviat [1991b] acknowledges that ideas and information came from interaction with other language developers and designers. The Working Conference on Simulation and Programming Languages, cited above, was a primary source of such ideas [Buxton, 1968].

4.3.2.2 Language Design and Definition

The design of SIMSCRIPT II can be represented by the reverse analogy of "peeling the onion." The language designers refer to "levels of the language," and use that effectively in describing the rather large language represented by SIMSCRIPT II.

Level 1 is a very simple programming language for doing numerical operations. It contains only unsubscripted variables and the simplest of control structures with only rudimentary input (READ) and output (PRINT) statements.

Level 2 adds subscripted variables, subroutines, and extended control structures to offer a language with roughly the capability of FORTRAN 66 but lacking the FORTRAN rigidities.

Level 3 adds more general logical expressions with extended control structures, and provides the capability for storage management, function computations, and statistical operations.

Level 4 introduces the entity, attribute, set concepts needed for list processing. The point of structures, implied subscripting, and text handling go well beyond the capabilities of SIMSCRIPT I.5.

Level 5 contains the dynamic capabilities necessary for simulation: time advance, event processing, generation of statistical variates, and output analysis.

4.3.2.3 Non-technical Influences

Markowitz [1979] attributes the lack of interest by RAND in developing SIMSCRIPT II as more than a simulation language to dictating the limitation of five levels in the implementation. He identifies Level 6 as that which dealt with database entities, and Level 7 as a language writing "language," used in the implementation of SIMSCRIPT II so that a user could define the syntax of statements and the execution of more complex commands [Markowitz 1979, p. 29]. Kiviat recalls discussion of the extension but that no concrete proposal was ever made [Kiviat, 1992].

4.3.3 Variants

SIMSCRIPT I.5 was developed as a commercial product by Consolidated Analysis Centers, Incorporated (CACI). SIMSCRIPT II became a commercial product in SIMSCRIPT II Plus through Simulation Associates, Incorporated, co-founded by P.J. Kiviat and Arnold Ockene, formerly IBM GPSS Administrator. CACI purchased the rights to SIMSCRIPT II Plus and marketed it as SIMSCRIPT II.5. Markowitz [1979, p. 29] also describes a SHARE version of the translator.

4.4 ECSL

ECSL, or E.C.S.L. as Clementson [1966] preferred, was developed for Courtaulds Ltd. by the originators of CSL [Buxton, 1964]. The extensions that distinguish ECSL from its predecessor are likened to those of CSL 2 developed by IBM United Kingdom, but the provision of the features took forms quite different on the Honeywell 400 and 200 series machines. A single man-year of effort was required for each of the Honeywell versions (400 and 200) [Clementson 1966, p. 215].

The most striking difference in CSL and ECSL is the departure from FORTRAN taken with the latter. This decision was based on the desire to facilitate use of ECSL by those having no knowledge of any other programming language [Clementson 1966, p. 215]. The approach taken was to adopt a columnar field formatting akin to that used in GPSS. Abandonment of FORTRAN also led to a richer I/O capability. The FORTRAN I/O was felt to be "the major shortcoming of C.S.L., as originally conceived," [Clementson 1966, p. 218]. ECSL became the target language for the Computer Aided Programming System (CAPS), the first *interactive* program generator, developed by Clementson in 1973 [Mathewson, 1975].

A contrary argument to the separation from FORTRAN was raised almost a decade later in the simulation package EDSIM, which claimed both to emulate ECSL and to be event based

[Parkin, 1978]. In actuality, EDSIM closely followed the ECSL design as an activity scan language.

4.5 GASP II

GASP II appears twice in the genealogical tree in Figure 1. A preliminary description of the revised version appears in manual form available from Pritsker at Arizona State University [Pritsker, 1967]. While the published description appears in the book by Pritsker and Kiviat [1969], a listing of FORTRAN subprograms designated as a GASP II compilation on the IBM 7090 is dated "3/13/63." This early revision, the work of Kiviat alone, predates his departure from U.S. Steel by only a few months.

An examination of the 1963 and 1967 versions of GASP II reveals both the addition and elimination of routines. The most prominent addition is the routine SET. The use of NSET as the primary data structure improved the organization of the package. A basic and extended version of GASP II are described [Pritsker 1967, pp. 22-23].

A version of GASP II, written in the JOSS interactive language, called JASP was developed by Pritsker during the summer of 1969 at RAND. Pritsker notes that memory limitations of 2,000 words forced data packing and overlay techniques developed by Lou Miller of RAND. JASP is the only example of a time-sharing simulation language developed outside of MIT prior to 1970.

4.6 OPS-4

Defined in the Ph.D. thesis of Malcolm M. Jones [1967], the description of OPS-4 is taken here primarily from [Greenberger, 1968]. Based on PL/I, OPS-4 encouraged the incremental construction and test of model components. All three world views (event scheduling, activity scan, and process interaction) were supported in the language. Additional routines for the generation of random variates were provided, and the numerous statistical routines of OPS-3 were continued. Extensive debugging and tracing capabilities were available along with immediate on-line diagnostic explanations for error detection. Model execution could be interrupted for examination and redefinition of values.

OPS-4 was described as a project in the planning stage. Intended for operation under the MULTICS operating system, OPS-4 raised the issue of the degree to which an SPL might use underlying operating systems support for simultaneous execution and asychronous processing [Greenberger 1968, p. 22, p. 24]. Ambitious in scope, OPS-4 did not become a commercial language, and little is known about its eventual use.

4.7 Other Languages

BOSS (Burroughs Operational Systems Simulator) was initiated in 1967 as the generalization of a gun placement simulator developed by the Defense, Space, and Special Systems Group of the Burroughs Corporation. Principals in the development of BOSS were Albert J. Meyerhoff, the informal group leader, Paul F. Roth, the conceptual designer, and Philip E. Shafer, the programmer. Supported at a high level by independent research and development funds, BOSS was viewed as a potential competitor with GPSS. The first prototypes for external use were completed in early 1969. Roth [1992] acknowledges experience with GPSS II as contributing to his design of BOSS, and the transaction flow and digrammatic specification of GPSS are evident in the language description [Meyerhoff, 1968?]. An extensive set of symbols enabled a wide variety of graphical structures to be represented, and the subsequent diagram was transformed into an executable ALGOL program. Roth [1992] attributes an internal competition with SIMULA as the reason that BOSS was never converted into a marketable product, which was the motivation of the developing group. A very interesting application of BOSS during the 1970s was to message/communication processing by CPUs in a network model developed by Scotland Yard. Roth [1992] believes the language continued in use until approximately 1982.

Q-GERT, appearing in the early 1970s, is described by Pritsker as the first network-oriented simulation language [Pritsker 1990, p. 246]. It is based on the GERT network analysis technique developed by Pritsker, and is first reported in Pritsker and Burgess [1970]. The intent with Q-GERT was to provide a user friendly modeling interface with a limited number of node types that could be combined through edge definition to describe routing conditions. Obviously, Pritsker's experience with GASP II had an influence, and the GERT family tree shown in Figure 14 shows the mutual influence between GERT and GASP.

Buxton [1968] contains descriptions of several languages offered in concept or perhaps developed to some extent during this period. Presentations at the 1967 IFIP workshop included: SPL (Petrone), AS (Parslow), SLANG (Kalinichenko).

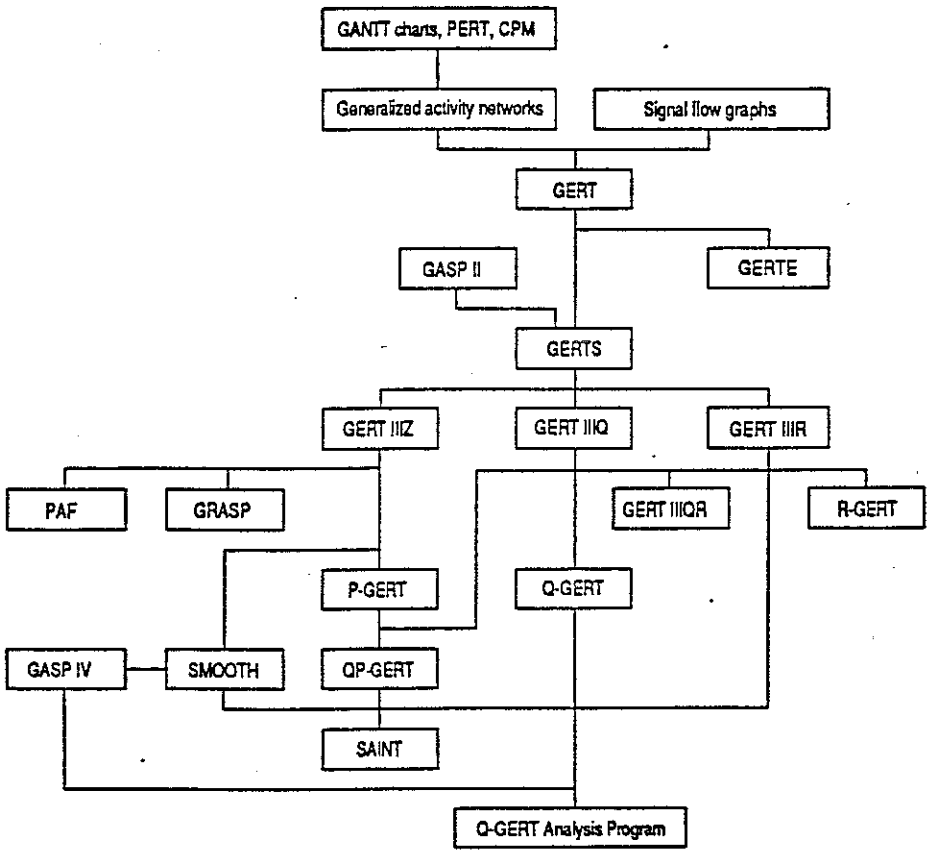


Figure 14. The GERT Family Tree (from [Pristker 1990, p. 243])

5. THE EXPANSION PERIOD (1971-1978)

The title for this period in SPL development emanates from the major expansions and extensions to GPSS, SIMSCRIPT II.5, and GASP. The program generator concept, initiated in the U.S., took hold in the U.K. and became a major model development aid.

5.1 GPSS

The major advances in GPSS came from outside IBM as the success of the language caused others to extend either its capabilities or the host machine environment.

5.1.1 GPSS/NORDEN and NGPSS

Advances in GPSS were underway by the Norden Division of United Aircraft Corporation in the late 1960s. Labeled GPSS/NORDEN, this version of the language is an interactive, visual, on-line environment for executing GPSS models. Programmed in COBOL (the language known best by the programming team) [Reitman, 1977?], GPSS/NORDEN permitted the user to interact through a CRT terminal, examining the behavior of the model (queues forming at facilities, storages filling and emptying, etc.) during execution. The user could interrupt model execution and redefine certain standard numerical attributes and then resume execution. The NORDEN report generator is described as a radically new free format English-like compiler which provides an ideal method for intermixing data, text, and titles [GPSS/NORDEN 1971, pp. 1-3].

GPSS/NORDEN in effect was a redesign around the matrix organization of GPSS. Model redefinition was accomplished through manipulation of matrices, and interactive statistical display and manipulation routines enabled both the inspection and alteration of matrix data, supported by the VP/CSS time sharing system. A film showing the use of this version of the language was distributed by the United Aircraft Corporation during the 1970s.

A version of the language entitled NGPSS (NORDEN GPSS) is described in a report dated 15 December 1971. Characterized as a superset of GPSS/360 and GPSS V for both batch and interactive versions, the User's Guide attributes the translator with the capability of handling very large databases, utilizing swapping capability to IBM 2311 or 2314 disk drives. A limited database capability is provided through the GPSS language to permit a modeler's access to matrix savevalues so that model components can be created and stored in a library [NGPSS, 1971]. The User's Guide is replete with CRT displays of program block structures, data input, and model execution. A light pen is illustrated as used with the display [NGPSS 1971, p. 71].

5.1.2 GPSS V/6000

A version of GPSS V was developed by Northwestern University for Control Data Corporation in a project completed in 1975. Program copyright is shown in [GPSS V/6000, 1975]. GPSS V/6000 was intended to be fully compatible with the IBM product bearing the name GPSS V. In some cases perceived restrictions imposed in the IBM version were removed in order to allow compatibility with an earlier version GPSS III/6000 [GPSS V/6000 1975, p. v-1]. The translator resided under the SCOPE 3.3 operating system (or later versions) and was a one-pass assembler in contrast to the two-pass IBM version. Differences between the two versions are described in detail in Appendix B of [GPSS V/6000, 1975].

5.1.3 GPDS and GPSS 1100

A version of GPSS, based apparently on GPSS/360, called the General Purpose Discrete Simulator, was a program product of Xerox Data Systems for the Sigma Computer line. Little is known about it beyond the identification in [Reifer, 1973]. Similarly, a UNIVAC product labeled GPSS 1100 is referenced with a 1971 date [UNIVAC, 1971a, 1971b]. Little else is known about

this version; particularly, if it was related in any way to GPSS/UCC, which was also implemented on a UNIVAC 1108.

5.1.4 GPSS/H

In 1975, James O. Henriksen published a description of a compiler for GPSS that produced a 3:1 performance enhancement [Henriksen, 1976]. With the announcement of the termination of support for GPSS V by IBM in the mid-1970s, Henriksen was poised to market his version of the language which was already gaining proponents. In 1977 GPSS/H was released by Wolverine Software Corporation, which have supported the language with extensive enhancements since that time [Schriber 1991, p. 17]. Other versions of the language have been developed, both on PCs and mainframes, but GPSS/H (the "H" for Henriksen) remains to date the most popular version.

5.2 SIMULA 67

The historical description in [Wexelblatt, 1981] describes the period of expansion for SIMULA as the development of a pure system description language called DELTA [Holbaek-Hanssen, 1977]. Beginning from the SIMULA platform, the DELTA Project sought to implement system specification for simulation execution through a series of transformations from the high level, user perspective, represented in the DELTA specification language, through an intermediate language called BETA, culminating with an executable language called GAMMA. Probably due to lack of funding, the project is not viewed as having been successful.

Beyond the recognition of the lack of success of the DELTA project, little can be said conceptually about SIMULA during this period. However, the originating organization (Norwegian Computing Center) issued publications which noted comparisons of the language with other SPLs and sought to promote the language in this way [Virjo, 1972] [Røgeberg, 1973]. Houle and Franta [1975, pp. 39-45] published an article showing that the structural concepts of SIMULA encompassed those of other languages, notably GPSS and SIMSCRIPT, and included a comparison with the language ASPOL [MacDougall, 1973]. They specifically claim that GPSS and SIMSCRIPT concepts are a subset of the structural capabilities of SIMULA.

5.3 SIMSCRIPT

As the decade of the 70's opened, simulation languages provided contrasting positions in the representation of world views identified by Kiviat in the classic RAND report [1969]:

event scheduling – SIMSCRIPT II, GASP II,
activity scan – ECSL, and
process interaction – GPSS V, SIMULA 67.

However, those studying model and language representation concepts readily recognized a major distinction between GPSS and SIMULA in the representation of the process interaction world view, i.e., GPSS lacked the flexibility of SIMULA in portraying interactions among permanent entities, which made the former cumbersome for problems where such relationships must be represented.

The expansion period marked the beginnings of the breakdown in conceptual distinctions among languages. Nowhere was this more apparent than with SIMSCRIPT II.

Supported by the U.S. Department of Agriculture, Agricultural Research Service, CACI completed the extension of SIMSCRIPT II.5 to include continuous simulation components in 1976. This effort is documented with a report authored by Claude M. Delfosse [1976] describing C-SIMSCRIPT. This report describes the definitions and program structure for continuous simulation and integration control variables within SIMSCRIPT II.5. Examples of both continuous and combined models are given.

Almost concurrently, the process view with focus on temporary entities was added to the language. Using the distinction of processes to describe the typical GPSS transaction and resources to represent facilities and storages, the modeling approach enabled a definition of resources as passive (permanent) entities and temporary entities represented as processes. Process instances are created similar to the creation and scheduling of events. An ACTIVE or CREATE statement is used. Within the process a WORK or WAIT statement provides for time passage, the latter with a process in a passive, the former an active, state. Processes may SUSPEND, INTERRUPT, or RESUME other processes.

A second form of expansion took place in the form of computer system modeling languages based on SIMSCRIPT. Two such vehicles were ECSS [Kosy, 1975] and CSP II [Iwata, 1975]. Developed at the RAND Corporation, ECSS became a favored language for computer system performance evaluation studies.

5.4 GASP

During this period GASP was the subject of notable changes and considerable experimentation by Pritsker and his graduate students at Purdue. These activities are described extensively in Pritsker [1990].

5.4.1 GASP IV

GASP IV became available on a published basis with a book in 1974 [Pritsker, 1974]. The contributions of Nicholas R. Hurst and C. Elliott Sigal are noted in the preface to the book. GASP IV proceeded philosophically in the differentiation between state and time as modeling characterizations. State events and time events became means of describing the event scheduling concept of an event and the activity scan concept of a condition. However, the concept of state variables was used to extend the discrete event capability to a continuous representation.

Mirroring the SIMSCRIPT II strategy, GASP augmented the transaction flow world view in the 1975-76 time period [Washam, 1976b] [Washam, 1976a]. In [Pritsker 1990, p. 252] GASP is described as an on-going language development that is intended to produce subsequent versions at least up to GASP VI.

The increased capability and flexibility of treating both continuous and discrete models within the same language provided a claimed advantage for GASP over its contemporaries. This advantage was short-lived since SIMSCRIPT followed with the capability in 1976 [Delfosse, 1976].

5.4.2 GASP_PL/I

In the early 1970s, several efforts were made to map a simulation language onto PL/I. Some of these are noted in the final section describing this period. GASP_PL/I, developed by Pritsker and Young [Pritsker, 1975] produced a version of GASP. A doctoral student examining the coded subroutines commented at the time that it appeared as if a FORTRAN programmer had done a statement by statement syntactic translation of FORTRAN into PL/I, and none of the PL/I features were utilized. Kiviat notes that preoccupation with a PL/I implementation was rather widespread at this time [Kiviat, 1992].

5.5 Program Generators

Simulation program generators were intended to accept a model definition in a non-executable form and produce an executable program or one that would admit execution after slight modification. In a sense the early definitional forms of SIMSCRIPT suggested such an approach, replacing the 80-column card specification with custom forms for entity definition and report generator layout. The first program generator was Programming By Questionnaire (PBQ), developed at the RAND Corporation [Ginsberg, 1965] [Oldfather, 1966]. With PBQ model

definition was accomplished through a user's completion of a questionnaire. The questionnaire, with the user's responses, forms a specification that produces a SIMSCRIPT program.

While PBQ was under development at RAND, K.D. Tocher was showing the utility of "wheel charts" for model description in GSP [Tocher, 1966]. The "wheel chart," based on the U.K. "machine based" (permanent entity) approach, pictured the cyclic activity of machines going from idle to busy as they interacted with material requiring the machine resource. In this same paper Tocher comments on the potential for real-time simulation for process control.

Interactive program generators, those permitting an interactive dialogue between modeler and program generator, appeared in the early 1970s in the U.K. CAPS-ECSL (Computer Aided Programming System/Extended Control and Simulation Language), developed by Alan T. Clementson at the University of Birmingham, is generally considered the first [Mathewson, 1975]. DRAFT, developed by Stephen Mathewson at Imperial College, appeared in several versions in which the activity cycle diagrams (also known as entity cycle diagrams), successors to the "wheel charts," are translated into DRAFT/FORTRAN, DRAFT/GASP, DRAFT/SIMON, or DRAFT/SIMULA programs [Mathewson, 1977].

Two efforts, not specifically program generators but related to them, deserve mention in this section. The first is HOCUS, a very simple representational form for simulation model specification, first suggested by Hills and Poole in 1969 but subsequently marketed both in the U.S. and the U.K. in the early to mid-1970s [Hills, 1969]. The second is the natural language interface to GPSS, a project of George E. Heidorn at the Naval Postgraduate School in the early 1970s. Heidorn's work actually began at Yale University in 1967 as a doctoral dissertation [Heidorn, 1976]. The project had as its eventual goal to enable an analyst working at a terminal to carry on a two-way dialogue with the computer about his simulation problem in English. The computer would then develop the model, execute the simulation, and report the results, all in an English dialect [Heidorn 1972, p.1].

5.6 Other Languages

5.6.1 The PL/I Branch

The cessation of support for GPSS V coincided with a decision by IBM to create the language SIMPL/I, a PL/I preprocessor [SIMPL/I, 1972]. SIMPL/I has the distinct flavor of GPSS in its provision of the list processing, random number generation, and statistical routines, to assist in the modeling and simulation studies. Later, SIMPL/I and GPSS V commands were made interchangeable, with the translation of both into PL/I. Such a package, described as SIMPL/I X or PL/I GPSS, provided even greater flexibility to the modeler [Metz, 1981]. Only a few IBM 360/370 Assembly Language routines augmented the PL/I implementation.

SIMPL is a descendent of OPS-4, created by Malcolm W. Jones and Richard C. Thurber [Jones, 1971a] [Jones, 1971b]. Implemented on the MULTICS time sharing system, SIMPL was envisioned as a simulation system consisting of both a programming language and a run time support system, quite similar to its ancestors. Again, PL/I provided the underlying translation capability with SIMPL source code being compiled into PL/I. Use of SIMULATE (model name) set up the special environment for simulation and initiated the model. Like OPS-4, the interactive time sharing environment, now provided by MULTICS, served as the major vehicle.

A third member of the PL/I family, SIML/I, actually appeared in 1979 [MacDougall, 1979]. Intended for computer system modeling, as was one of its predecessors ASPOL [MacDougall, 1973], SIML/I provided representational capability "which extends into the gap between system-level and register-transfer-level simulation languages," [MacDougall 1979, p. 39]. The process interaction world view is apparent, perhaps influenced more by SOL than SIMULA. Process communication occurs via signals that can be simple or synthesized into more complex expressions. The influence of the application domain is readily apparent in the representation of signals.

5.6.2 The Pritsker Family

A. Alan B. Pritsker and Pritsker and Associates, Incorporated have contributed significantly to the development of simulation languages. In addition to GASP II and subsequent versions, GERTS (GERT Simulation program) provided a network modeling capability for simulation solutions to activity network models and queueing network formulations based on GERT, created earlier by Pritsker. Extensions to the network modeling capability resulted in a family of programs generally identifiable as one form of GERT or another. The GERT family tree is shown in Figure 14 which also shows the relationship to GASP [Pritsker 1990, pp. 242-243].

SAINT (Systems Analysis for Integrated Networks of Tasks) was developed by Pritsker and Associates under an Air Force contract that sought a modeling capability which enabled the assessment of contributions of system components especially human operators to overall performance [Wortman, 1977a]. SAINT utilized a graphical approach to modeling, similar to Q-GERT, and depended on an underlying network representation. The continuous component was similar to the provided in GASP IV [Wortman, 1977a, p.532]. Like GASP IV, SAINT is actually a simulator consisting of callable FORTRAN sub-routines and requiring approximately 55,000 decimal words of internal storage [Wortman, 1977b] [Wortman, 1977c]. The contributions of numerous students and employees are generously described in Pritsker [1990].

5.6.3 Interactive Simulation Languages

The rapid acceptance and increased availability of time-sharing operating systems led to the emergence of interactive modeling systems based on user dialogue or conversational style. Two such languages during this period are CML (Conversational Modeling Language) developed by Ronald E. Mills and Robert B. Fetter of Yale University [undated]. CML was intended as a creation language as well as a language for simulation purposes. The reference describes the use of CML for creation of a simulation language used to study hospital resource utilization. The power of CML was found in its ability to provide deferred run-time definition of model parameters and its ability for model modification and redefinition. The creators describe CML as equally useful as a simulation language, a general purpose programming tool, a specialized "package" for certain tasks, and an environment for systems programming.

CONSIM was developed as a doctoral dissertation by Sallie Sheppard Nelson at the University of Pittsburgh. Considered a prototype conversational language, CONSIM is patterned after SIMULA 67 because of the recognized advanced capabilities of the language [Sheppard Nelson 1977, p. 16]. Co-routine sequencing, process description, and dynamic interaction in the CONSIM interpreter follow the design found in SIMULA 67.

6. CONSOLIDATION AND REGENERATION (1979-1986)

The period from 1979 to 1986 might be characterized as one in which predominant simulation languages extended their implementation to many computers and microprocessors while keeping the basic language capabilities relatively static. On the other hand, two major descendants of GASP (in a sense) appeared to play major roles: SLAM II and SIMAN.

6.1 Consolidation

Versions of GPSS on personal computers included GPSS/PC developed by Springer Cox and marketed by MINUTEMAN Software [Cox, 1984] and a Motorola 68000 chip version of GPSS/H [Schriber 1984, p.14]. Joining the mapping of GPSS to PL/I are a FORTRAN version [Schmidt, 1980] and an APL version [APL GPSS, 1977].

CACI extended SIMSCRIPT II.5 even further by providing application dependent interfaces: NETWORK II.5 for distributed computing in September 1985 and SIMFACTORY II.5 for the modeling of manufacturing problems in October 1986 [Annino, 1992]. An added interface in

November 1987, COMNET II.5, addressed the modeling of wide and local area communications networks.

An extension of SIMULA, to capture an explicit transaction world view, called DEMOS, was introduced by Birtwistle [1979]. DEMOS was intended to provide modeling conveniences (built in resource types, tracing, report generation capability) that were lacking in SIMULA [Birtwistle 1981, p. 567].

6.2 SLAM AND SLAM II

The Simulation Language for Alternative Modeling (SLAM), a GASP descendent in the Pritsker and Associates, Inc. software line, sought to provide multiple modeling perspectives: process, event, or state variables, each of which could be utilized exclusively or joined in a combined model [Pritsker, 1979]. SLAM appeared in 1979; SLAM II, in 1983 [O'Reilly, 1983]. (Note that the acronym SLAM is also used for a continuous simulation language developed in the mid-1970s [Wallington, 1976]. The two are not related.)

6.2.1 Background

The background for SLAM introduction included the coding of processes as an addition to GASP IV in a version called GASPPI. This work was done as a master's thesis by Ware Washam [Washam, 1976a]. Jerry Sabuda in a master's thesis at Purdue did the early animation work for Q-GERT networks that led to its incorporation eventually in the SLAM II PC animation program and SLAMSYSTEM® [Pritsker 1990, p. 292]. Pritsker [1990, p.290-293] describes the long, evolutionary process in developing highly usable simulation languages.

The intended purpose of SLAM was to join diverse modeling perspectives in a single language that would permit a large degree of flexibility as well as strong capability to deal with complex systems. Modeling "power" was considered the primary support capability influencing the design of the language and the later evolution of SLAMSYSTEM® [Pritsker, 1991].

6.2.2 Rationale for Language Content

Clearly, GASP, Q-GERT, SAINT, and several variations of each contributed to the creation of SLAM. In fact, many of the identical subroutine and function names in GASP IV are repeated in SLAM.

6.2.3 Language Design and Definition

Unlike its predecessors which were simulators, SLAM is a FORTRAN preprocessor, with the preprocessing requirements limited to the network modeling perspective. The simulator aspect is preserved in the representation of continuous and discrete event models. Improvements were made to the functions and subroutines making up the discrete event and continuous components, but the major structure of the SLAM design followed that of GASP.

6.2.4 Non-Technical Influences

After the completion and delivery of SLAM as a product, differences occurred between Pritsker and Pegden over the rights to the language and a court case ensued. Settlement of the case dictated neither party should say more about the relationship of SLAM to SIMAN, a language developed in 1980-1983 by Pegden. The statements below are taken from Prefaces to the most current sources for each language.

C. Dennis Pegden led in the development of the original version of SLAM and did the initial conception, design, and implementation. Portions of SLAM were based on

Pritsker and Associates' proprietary software called GASP and Q-GERT. Since its original implementation, SLAM has been continually refined and enhanced by Pritsker and Associates [Pritsker 1986, p. viii].

Many of the concepts included in SIMAN are based on the previous work of other simulation language developers. Many of the basic ideas in the process-orientation of SIMAN can be traced back to the early work of Geoffrey Gordon at IBM who developed the original version of GPSS. Many of the basic ideas in the discrete-event portion of SIMAN can be traced back to the early work by Philip Kiviat at U.S. Steel who developed the original version of GASP. SIMAN also contains features which Pegden originally developed for SLAM. Some of the algorithms in SIMAN are based on work done by Pritsker and Associates. The combined discrete-continuous features of SIMAN are in part based on SLAM [Pegden 1990, p. xi].

6.3 SIMAN

The name SIMAN derives from SIMulation ANalysis for modeling combined discrete event and continuous systems. Originally couched in a manufacturing systems application domain, SIMAN possesses the general modeling capability for simulation found in languages such as GASP IV and SLAM II.

6.3.1 Background

Developed by C. Dennis Pegden, while a faculty member at Pennsylvania State University, SIMAN was essentially a one-person project. A Tektronix was used as the test bed for the language, which originated as an idea in 1979 and moved rapidly through initial specifications in 1980, to final specification and a prototype in 1981. A full version of the language was completed in 1982, and the release date was in 1983.

6.3.2 Rationale for Language Content

SIMAN incorporates multiple world views within a single language:

(1) a process orientation utilizing a block diagram similar to that of GPSS,

(2) an event orientation represented by a set of FORTRAN subroutines defining instantaneous state transitions,

(3) a continuous orientation, utilizing dependent variables representing changes in state over time (state variables).

Thus, SIMAN is either a FORTRAN preprocessor or a FORTRAN package depending on the selected world view. The use of FORTRAN as the base for SIMAN was justified by the wide availability of the latter language on mainframes and mini computers. Pegden [1991] acknowledges that in today's technical market place it would be much easier if written in C or C++.

6.3.2.1 Influencing Factors

SIMAN draws concepts from GPSS, SIMSCRIPT, SLAM, GASP, and Q-GERT [Pegden, 1991]. The primary contributions are the combined process, next event, and continuous orientation from SLAM and the block diagram and process interaction concepts from GPSS.

New concepts introduced in SLAM include general purpose features embedded in specialized manufacturing terminology, e.g., stations, conveyors, transporters. SIMAN has claimed to be the

first major simulation language executable on the IBM PC and designed to run under MS-DOS constraints [Pegden, 1991]. Macro submodels provide convenient repetition of a set of objects without replication of entire data structures.

6.3.2.3 Non-technical Influences

Systems Modeling Corporation in marketing SIMAN recognized the major advantage of output animation and created a "companion" product called CINEMA.

6.4 The PASCAL Packages

The emergence of yet another popular general purpose language --PASCAL-- stimulated a repetition of history in the subsequent appearance of simulation packages based on the language. Bryant [1980; 1981] developed SIMPAS as an event scheduling language through extensions of PASCAL that met the six requirements for simulation (described in Section 1.1.2). Implemented as a preprocessor, SIMPAS 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.

INTERACTIVE, described as a network simulation language, used graphical symbols in a four-stage model development process that supported interactive model construction and execution [Lakshmanan, 1983]. The implementation on microcomputers, coupled with the ability of PASCAL to support needed simulation capabilities, motivated the development of the package [Mourant 1983, p. 481].

6.5 Other Languages

INSIGHT (INS) was developed by Stephen D. Roberts to model health care problems and as a general simulation modeling language [Roberts, 1983a]. Utilizing the world view of a network of processes, INSIGHT adopts the transaction flow characterization of object interaction with passive resources. Described as a simulation modeling language rather than a programming language or parameterized model [Roberts 1983b, p. 7], INSIGHT provides a graphical model representation that must be translated manually into INSIGHT statements. INSIGHT provides the usual statistical utilities for random number generation but also provides assistance for output analysis. A default output of model behavior can be supplemented through a TABLE statement, and the utilization of FORTRAN is always available for the user of this preprocessor.

7. CONCLUDING SUMMARY

The history of simulation programming languages is marked by commercial competition, far more intense than that of general purpose languages. Perhaps that single fact explains the identifiable periods of remarkably similar behavior. Such competition might also be the motivator for the numerous concepts and techniques that either originated with an SPL or gained visibility through their usage in the language. Among the most significant are:

- the process concept,
- object definition as a record datatype,
- 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.

Acknowledged as the first object-oriented programming language, SIMULA 67 with its combination of features such as encapsulation, inheritance, the class and co-routine concepts, still remains a mystery to the large majority of the programming language community. Moreover, it remains an unknown to the majority of those hailing the object-oriented paradigm, irrespective of their knowledge in discrete event simulation.

While the ALGOL roots are often cited as the cause of SIMULA's relative obscurity, what dooms SIMSCRIPT to a similar fate? Is it the FORTRAN roots? Addressing the issue more seriously, the entity-attribute-relational view of data was both implicitly and explicitly enunciated in SIMSCRIPT fully ten years before Peter Chen's landmark paper in database theory [Chen, 1976]. Yet, neither the database systems nor the programming languages community took notice.

Given the early prominence attached to simulation programming languages, reflected by the significance of meetings such as the IFIP Working Conference [Buxton, 1968] and the eminence of the attendees, what has led to the appearance of lack of interest by the larger community (be it programming languages or computer science)? Has this subdisciplinary insularity been counter-productive in the development of current general purpose languages? Is this insularity a two-way phenomenon and, if so, is MODSIM (a recently developed object-oriented SPL) likely to suffer for it? While a series of questions might be thought unseemly to close a paper such as this, the answers to them seem crucial to answering a more fundamental question of the programming language community:

Is the past only a prologue?

ACKNOWLEDGEMENTS

I am deeply indebted to Philip J. Kiviat for his guidance throughout the development of this paper. I am grateful also to those who provided personal information on various languages and other topics: Joseph Annino, Donald E. Knuth, C. Dennis Pegden, A. Alan B. Pritsker, and Paul F. Roth. My thanks also to my colleagues, James D. Arthur, for his help in language and translator characteristics, and Osman Balci, for discussions of language and model representational issues.

REFERENCES

[Ackoff, 1961]
Ackoff, Russell L., ed., *Progress in Operations Research, Volume 1*, John Wiley and Sons, Inc., pp. 375-376, 1961.

[Annino, 1992]
Annino, Joseph, Personal Communication, April 1992.

[Belkin, 1965?]
Belkin, J. and M.R. Rao, *GASP User's Manual*, United States Steel Corporation, Applied Research Laboratory, Monroeville, Pennsylvania, undated (believed to be in 1965).

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

[Birtwistle, 1979]
Birtwistle, Garham M., *DEMOS: A System for Discrete Event Modeling on SIMULA*, London: New York: Macmillan, 1979.

[Birtwistle, 1981]
Birtwistle, Graham M., Introduction to Demos, In *Proceedings of the 1981 Winter Simulation Conference*, T.I. Ören, C.M. Delfosse, C.M. Shub, eds., pp. 559-572, 1981.

[Bryant, 1980]
Bryant, R.M., SIMPAS -- a simulation language based on PASCAL, In *Proceedings of the 1980 Winter Simulation Conference*, T.I. Ören, C.M. Shub, and P.F. Roth, eds., pp. 25-40, 1980.

[Bryant, 1981]
Bryant, R.M., A tutorial on simulation programming with SIMPAS, In *Proceedings of the 1981 Winter Simulation Conference*, T.I. Ören, C.M. Delfosse, and C.M. Shub, eds., pp. 363-377, 1981.

[Buxton, 1966]
Buxton, J.N., Writing simulations in CSL, *The Computer Journal*, 9:2, pp. 137-143, 1966.

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

[Buxton, 1962]
Buxton, J.N. and J.G. Laski, Control and simulation language, *The Computer Journal*, 5:3, pp. 194-199, 1962.

[Buxton, 1964]
————, Courtaulds All Purpose Simulator, Programming Manual, Courtaulds Ltd., 1964.

[CACI, 1983]
CACI, *SIMSCRIPT II.5 Programming Language*, CACI, Los Angeles, 1983.

[Chen, 1976]
Chen, P.P., The entity-relationship model – toward a unified view of data, *ACM Transactions on Database Systems*, 1:1, pp. 9-36, 1976.

[Chrisman, 1966]
Chrisman, J.K., *Summary of the Workshop on Simulation Languages Held March 17-18, 1966*, University of Pennsylvania, Sponsored by the IEEE Systems Services and Cybernetics Group and the Management Center, April 28, 1966.

[Clementson, 1966]
Clementson, A.T., Extended control and simulation language, *The Computer Journal*, 9:3, pp. 215-220, 1966.

[Conway, 1965]
Conway, R.W., J.J. Delfausse, W.L. Maxwell, and W.E. Walker, CLP – The Cornell LISP processor, *Communications ACM*, 8:4, pp. 215-216, April 1965.

[Conway, 1967]
Conway, Richard W., William L. Maxwell, and Louis W. Miller, *Theory of Scheduling*, Addison-Wesley Publishing Company, 1967.

[Cox, 1984]
Cox, Springer, *GPSS/PC User's Manual*, MINUTEMAN Software, Stowe, Massachusetts, 1984.

[Crookes, 1982]
Crookes, John G., Simulation in 1981, *European Journal of Operational Research*, 9: 1, pp. 1-7, 1982.

[Delfosse, 1976]
Delfosse, Claude M., Continuous Simulation and Combined Simulation in SIMSCRIPT II.5, Arlington, VA: CACI, March 1976.

[Derrick, 1988]
Derrick, Emory Joseph, Conceptual Frameworks for Discrete Event Simulation, Master's Thesis, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, August 1988.

[Esso, 1963]
C.S.L.: Reference Manual/Control and Simulation Language, Esso Petroleum Company, Ltd. and IBM United Kingdom, Ltd., 1 March 1963.

[Fishman, 1973]
Fishman, George S., *Concepts and Methods in Discrete Event Digital Simulation*, New York: John Wiley and Sons, pp. 22-58, 1973.

[Flow Simulator, RCA , 1967]
Flow Simulator, RCA publication #70-05-008, October 1967.

[Flow Simulator Information Manual, RCA, 1967]
Flow Simulator Information Manual, RCA publication #70-35-503, October 1967.

[Forrester, 1961]
Forrester, Jay W., *Industrial Dynamics*, Cambridge, MA: The MIT Press, 1961.

[Geisler, 1966]
Geisler, Murray A. and Allen S. Ginsberg, Man-machine simulation experience, In *Proceedings IBM Scientific Computing Symposium on Simulation Models and Gaming*, White Plains, New York, pp. 225-242, 1966.

[Ginsberg, 1965]
Ginsberg, Allen S., Harry M. Markowitz, and Paula M. Oldfather, Programming by Questionnaire, Memorandum RM-4460-PR, RAND Corporation, April 1965.

[Gordon, 1966]
Gordon, Geoffrey, Simulation languages for discrete systems, In *Proceedings IBM Scientific Computing Symposium on Simulation Models and Gaming*, White Plains, New York, pp. 101-118, 1966.

[Gordon, 1981]
———, The development of the general purpose simulation system (GPSS), *History of Programming Languages*, Richard L. Wexelblatt (Ed.), Academic Press, pp. 403-437, 1981.

[GPS K, 1969]
General Purpose Simulator K, Honeywell publication, file #123.8405.001 K, April 1969.

[GPSS/NORDEN, 1971]
GPSS/NORDEN Simulation Language, Norden Division of United Aircraft Corporation, form 910-1, March 1971, pp. 1-3.

[GPSS V, 1970]
General Purpose Simulation System V, User's Manual, IBM publication #SH20-0851-0, October 1970.

[GPSS V, 1971]
General Purpose Simulation System V, Introductory User's Manual, IBM publication #SH20-0866-0, October 1971.

[GPSS V/6000, 1975]
GPSS V/6000 General Information Manual, Control Data Corporation, 84003900, 1 December 1975.

[Greenberg, 1972]
Greenberg, Stanley, *GPSS Primer*, John Wiley and Sons, Inc., 1972.

[Greenberger, 1968]
Greenberger, Martin and Malcolm M. Jones, *On-Line Incremental Simulation, in Simulation Programming Languages*, J.N. Buxton, (Ed.) North-Holland, pp. 13-32, 1968.

[Greenberger, 1965]
Greenberger, Martin, Malcolm M. Jones, James H. Morris, and David N. Ness, *On Dash Line Computation and Simulation: The OPS-3 System*, Cambridge, MA: MIT Press, 1965.

[Heidorn, 1972]
Heidorn, George E., Natural Language Inputs to a Simulation Programming System, Naval Postgraduate School, NPS-55HD72101A, October 1972.

[Heidorn, 1976]
———, Automatic programming through natural language dialogue, a survey, *IBM Journal of Research and Development*, July 1976, pp. 302-313.

[Henriksen, 1976]
Henriksen, James O., Building a better GPSS: a 3:1 enhancement, In *Proceedings of the 1975 Winter Simulation Conference*, Montvale, NJ: AFIPS Press, 1976, pp. 465-469.

[Hills, 1969]
Hills, B.R. and T.G. Poole, A Method for Simplifying the Production of Computer Simulation Models, TIMS Tenth American Meeting, Atlanta, Georgia, October 1-3, 1969.

[Holbaek-Hanssen, 1977]
Holbaek-Hanssen, Erik, Peter Händlykken, and Kristen Nygaard, System Description and the DELTA Language, DELTA Project Report No. 4, Second Printing, Norwegian Computing Center, February 1977.

[Houle, 1975]
Houle, P.A. and W.R. Franta, On the structural concepts of SIMULA, *The Australian Computer Journal*, 7:1, March 1975, pp. 39-45.

[IBM, 1965]
International Business Machines, General Purpose Simulator III: Introduction, Application Program, Technical Publications Department, White Plains, New York, 1965.

[IBM, 1977]
International Business Machines, APL GPSS, Form #G320-5745 and SH20-1942, Armonk, NY, 1977.

[IBMUK, 1965]
IBM United Kingdom Ltd., CSL Reference Manual, 1965.

[Iwata, 1975]
Iwata, H.Y. and Melvin M. Cutler, CSP II – A universal computer architecture simulation system for performance evaluation, *Symposium on the Simulation of Computer Systems*, pp. 196-206, 1975.

[Jones, 1967]
Jones, Malcolm M., Incremental Simulation on a Time-Shared Computer, unpublished Ph.D. dissertation, Alfred P. Sloane School of Management, Massachusetts Institute of Technology, January 1967.

[Jones, 1971a]
Jones, Malcolm M. and Richard C. Thurber, The SIMPL Primer, October 4, 1971.

[Jones, 1971b]
———, SIMPL Reference Manual, October 18, 1971.

[Karr, 1965]
Karr, H.W., H. Kleine, and H.M. Markowitz, SIMSCRIPT I.5, CACI 65-INT-1, Los Angeles: CACI, Inc., 1965.

[Kelly, 1962]
Kelly, D.H. and J.N. Buxton, Monte code – an interpretive program for Monte Carlo simulations, *The Computer Journal*, 5, p. 88.

[Kiviat, 1962a]
Kiviat, Philip J., Algorithm 100: add item to chain-linked list, *Communications of the ACM*, 5:6, p. 346, June 1962.

[Kiviat, 1962b]

————, Algorithm 100: remove item from chain-linked list, *Communications of the ACM*, 5:6, p. 346, June 1962.

[Kiviat, 1963a]

————, GASP: General Activities Simulation Program Programmer's Manual, Preliminary Version, Applied Research Laboratory, Monroeville, PA: United States Steel Corporation, January 16, 1963.

[Kiviat, 1963b]

————, GASP – A General Purpose Simulation Program, Applied Research Laboratory 90.17-019(2), United States Steel Corporation, undated (believed to be around March 1963).

[Kiviat, 1963c]

————, Introduction to Digital Simulation, Applied Research Laboratory 90.17-019(1), United States Steel Corporation, April 15, 1963 (stamped date).

[Kiviat, 1966]

————, Development of new digital simulation languages, *Journal of Industrial Engineering*, 17:11, November 1966, pp. 604-609.

[Kiviat, 1967]

————, Digital Computer Simulation: Modeling Concepts, RAND Memo RM-5378-PR, RAND Corporation, Santa Monica, California, August 1967.

[Kiviat., 1968]

Kiviat, Philip J., R. Villanueva, and H.M. Markowitz, *The SIMSCRIPT II Programming Language*, Prentice Hall, Inc., 1968.

[Kiviat, 1969]

————, Digital Computer Simulation: Computer Programming Languages, RAND Memo RM-5883-PR, RAND Corporation, Santa Monica, January 1969.

[Kiviat, 1991a]

————, Personal Communication, May 13, 1991.

[Kiviat, 1991b]

————, Personal Communication, July 5, 1991.

[Kiviat, 1992]

————, Personal Communication, February 3, 1992.

[Knuth, 1992]

Knuth, Donald E., Personal Communication, November 24, 1992.

[Knuth, 1964a]

Knuth, D.E. and McNeley, J.L., SOL – a symbolic language for general purpose system simulation, *IEEE Transactions on Electronic Computers*, EC-13:4, pp. 401-408, August 1964.

[Knuth, 1964b]

———, A formal definition of SOL, *IEEE Transactions on Electronic Computers*, EC-13:4, pp. 409-414, August 1964.

[Kosy, 1975]

Kosy, D.W., The ECSS II Language for Simulating Computer Systems, RAND Report R1895-GSA, December 1975.

[Krasnow, 1967]

Krasnow, H.S., Dynamic presentation in discrete interaction simulation languages, *Digital Simulation in Operational Research*, S.H. Hollingsdale (Ed.), American Elsevier, pp. 77-92, 1967.

[Krasnow, 1964]

Krasnow, H.S. and R.A. Merikallio, The past, present, and future of general simulation languages, *Management Science*, 11:2, pp. 236-267, November 1964.

[Lackner, 1962]

Lackner, Michal R., Toward a general simulation capability, In *Proceedings of the SJCC*, San Francisco, California, 1-3 May 1962, pp. 1-14.

[Lackner, 1964]

———, Digital Simulation and System Theory, System Development Corporation, SDC SP-1612, Santa Monica, California, 6 April 1964.

[Lakshmanan, 1983]

Lakshmanan, Ramon, Design and Implementation of a PASCAL Based Interactive Network Simulation Language for Microcomputers, unpublished Ph.D. dissertation, Oakland University, Rochester, Michigan, 1983.

[Law, 1991]

Law, Averill M. and David Kelton, *Simulation Modeling and Analysis*, 2nd Ed., McGraw-Hill, Inc., 1991.

[Licklider, 1967]

Licklider, J.C.R., Interactive dynamic modeling, *Prospects for Simulation and Simulators of Dynamic Systems*, George Shapiro and Milton Rogers, eds., Spartan Books, 1967.

[Llewellyn, 1965]

Llewellyn, Robert W., *FORDYN: An Industrial Dynamic Simulator*, Typing Service, Raleigh, North Carolina, 1965.

[MacDougall, 1979]

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

[MacDougall, 1973]

MacDougall, M.H. and J. Stuart MacAlpine, Computer simulation with ASPOL, In *Proceedings Symposium on the Simulation of Computer Systems*, pp. 92-103, 1973.

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

[Markowitz, 1963]
Markowitz, Harry M., Bernard Hausner, and Herbert W. Karr, SIMSCRIPT: A Simulation Programming Language, The RAND Corporation, Prentice Hall, Inc., 1963.

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

[Mathewson, 1977]
Mathewson, Stephen C. and John A. Alan, DRAFT/GASP – A program generator for GASP, In *Proceedings of the Tenth Annual Simulation Symposium*, Tampa, Florida, W.G. Key, et. al., eds., pp. 211-228, 1977.

[Metz, 1981]
Metz, Walter C., Discrete event simulation using PL/I based general and special purpose simulation languages, In *Proceedings of the Winter Simulation Conference*, T.I. Ören, C.M. Delfosse, C.M. Shub, eds., pp. 45-52, 1981.

[Meyerhoff, 1968?]
Meyerhoff, Albert J., Paul F. Roth, and Philip E. Shafer, BOSS Mark II Reference Manual, Burroughs Corporation, Defense, Space, and Special Systems Group, Document #66099A, undated.

[Mills, undated]
Mills, Ronald E. and Robert B. Fetter, CML: A Conversational Modeling Language, Technical Report #53, undated.

[Morgenthaler, 1961]
Morgenthaler, George W., The theory and application of simulation and operations research, *Progress in Operations Research I*, John Wiley and Sons, New York, 1961.

[Mourant, 1983]
Mourant, Ronald R. and Ramon Lakshmanan, INTERACTIVE -- a user friendly simulation language, In *Proceedings of the 1983 Winter Simulation Conference*, S. Roberts, J. Banks, and B. Schmeiser, eds., pp. 481-494, 1983.

[Nance, 1981]
Nance, Richard E., 1981, The time and state relationships in simulation modeling, *Communications ACM*, 24:4, pp. 173-179, April 1981.

[NGPSS, 1971]
User's Guide to NGPSS, Norden Report 4339 R 0003, Norden Division of United Aircraft Corporation, 15 December 1971.

[Nygaard, 1981]
Nygaard, Kristen and Ole-Johan Dahl, The development of the SIMULA languages, *History of Programming Languages*, Richard L. Wexelblatt (Ed.), Academic Press, pp. 439-491, 1981.

[Oldfather, 1966]
Oldfather, Paula M., Allen S. Ginsberg, and Harry M. Markowitz, Programming by Questionnaire: How to Construct a Program Generator, Memorandum RM-5129-PR, November 1966.

[O'Reilly, 1983]
O'Reilly, Jean J., SLAM II User's Manual, Pritsker and Associates, Inc., West Lafayette, Indiana, 1983.

[Overstreet, 1986]
Overstreet, C. Michael and Richard E. Nance, World view based discrete event model simplification, *Modeling and Simulation Methodology in the Artificial Intelligence Era*, Maurice S. Elzas, Tuner İ. Ören, and Bernard P. Zeigler, eds., North Holland, 1986, pp. 165-179.

[Parkin, 1978]
Parkin, A. and R.B. Coats, EDSIM – Event based discrete event simulation using general purpose languages such as FORTRAN, *The Computer Journal*, 21:2, pp. 122-127, May 1978.

[Pegden, 1991]
Pegden, C. Dennis, Personal Communication, June 11 1991.

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

[Pritsker, 1967]
Pritsker, A. Alan B., GASP II User's Manual, Arizona State University, 1967.

[Pritsker, 1969]
Pritsker, A. Alan B. and Philip J. Kiviat, *Simulation with GASP II: A FORTRAN Based Simulation Language*, Prentice Hall, 1969.

[Pritsker, 1970]
Pritsker and R.R. Burgess, The GERT Simulation Programs: GERTs III, GERTs III-Q, GERTs, III-R, NASA/ERC contract NAS12-2113, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1970.

[Pritsker, 1974]
Pritsker, A. Alan B., *The GASP IV Simulation Language*, John Wiley and Sons, Inc., 1974.

[Pritsker, 1975]
Pritsker, A. Alan B. and R.E. Young, *Simulation with GASP_PL/I* John Wiley, 1975.

[Pritsker, 1979]
Pritsker, A. Alan B. and Claude Dennis Pegden, *Introduction to Simulation and SLAM*, John Wiley and Sons, 1979.

[Pritsker, 1986]
———, *Introduction to Simulation and SLAM II*, Systems Publishing Corporation, 1986.

[Pritsker, 1990]
———, *Papers, Experiences, Perspectives*, Systems Publishing Company, 1990.

[Pritsker, 1991]
———, Personal Communication, 13 June 1991.

[Pugh, 1963]
Pugh, Alexander L., III, DYNAMO User's Manual, The MIT Press, 2nd Ed., 1963.

[Pugh, 1973]
———, DYNAMO II User's Manual, including DYNAMO II (Subscript F), The MIT Press, 4th edition, 1973.

[Pugh, 1976]
———, DYNAMO User's Manual, including DYNAMO II/370, DYNAMO II/F, DYNAMO III, Gaming DYNAMO, The MIT Press, 5th Ed., 1976.

[Reifer, 1973]
Reifer, Donald J., Simulation Language Survey, Hughes Aircraft Corporation, Interdepartmental Correspondence, Ref. 2726.54/109, 24 July 1973.

[Reitman, 1967]
Reitman, Julian, The user of simulation languages – the forgotten man, In *Proceedings ACM 22nd National Conference*, 1967, pp. 573-579.

[Reitman, 1977?]
Reitman, Julian, Personal Communication, around 1977.

[Reitman, 1992]
Reitman, Julian, How the 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*, James J. Swain, Robert C. Crain, and James R. Wilson, eds., 1992 (to appear).

[Roberts, 1983a]
Roberts, Stephen D., Simulation Modeling and Analysis with INSIGHT, Regenstrief Institute for Health Care, Indianapolis, Indiana, 1983.

[Roberts, 1983b]
Roberts, Stephen D., Simulation modeling with INSIGHT, In *Proceedings of the 1983 Winter Simulation Conference*, Stephen D. Roberts, Jerry Banks, Bruce Schmeiser, eds., 1983, pp. 7-16.

[Røgeberg, 1973]
Røgeberg, Thomas, Simulation and Simulation Languages, Norwegian Computing Center, Publication No. S-48, Oslo Norway, October 1973.

[Rosen, 1967]
Rosen, Saul, Programming systems and languages: a historical survey, *Programming Systems and Languages*, S. Rosen, Ed., McGraw-Hill Book Company, 1967.

[Roth, 1992]
Roth, Paul F. Personal Communication, February 1992.

[Sammet, 1969]
Sammet, Jean E., *Programming Languages: History and Fundamentals*, 1st Ed., Englewood Cliffs, NJ: Prentice-Hall, p. 650, 1969.

[Schmidt, 1980]
Schmidt, Bernd, *GPSS – FORTRAN*, New York: John Wiley and Sons, Inc., 1980.

[Schriber, 1974]
Schriber, Thomas J., *Simulation Using GPSS*, John Wiley and Sons, 1974.

[Schriber, 1984]
———, Introduction to GPSS, In *Proceedings of the 1984 Winter Simulation Conference*, Sallie Sheppard, Udo W. Pooch, C. Dennis Pegden, eds., pp. 12-15, 1984.

[Schriber, 1991]
———, *An Introduction to Simulation*, John Wiley and Sons, 1991.

[Shantikumar, 1983]
Shantikumar, J.G. and R.G. Sargent, A unifying view of hybrid simulation/analytic models and modeling, *Operations Research*, 31:6, pp. 1030-1052, November-December 1983.

[Sheppard Nelson, 1977]
Sheppard Nelson, Sallie, Control Issues and Development of a Conversational Simulation Language, unpublished Ph.D. dissertation, University of Pittsburgh, 1977.

[Shreider, 1966]
Schrieder, Yu A., ed., *The Monte Carlo Method: The Method of Statistical Trials*, Pergamon Press, 1966.

[SIMPL/I, 1972]
SIMPL/I (Simulation Language Based on PL/I), Program Reference Manual, IBM publication number SH19-5060-0, June 1972.

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

[Teichroew, 1966]
Teichroew, Daniel and John Francis Lubin, Computer simulation – discussion of the technique and comparison of languages, *Communications of the ACM*, 9:10, pp. 723-741, October 1966.

[Tocher, 1965]
Tocher, K.D., Review of simulation languages, *Operational Research Quarterly*, 16:2, pp. 189-217, June 1965.

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

[Tocher, 1979]
————, Keynote Address, In *Proceedings of the 1979 Winter Simulation Conference*, Harold J. Highland, Mitchell G. Spiegel, and Robert Shannon, eds., pp. 645-654, 1979.

[Tocher, 1960]
Tocher, K.D. and D.G. Owen, 1960, The automatic programming of simulations, In *Proceedings of the Second International Conference on Operational Research*, pp. 50-68.

[Tonge, 1965]
Tonge, Fred M., Peter Keller, and Allen Newell, Quikscript – a simscript-like language for the G-20, *Communications of the ACM*, 8:6, pp. 350-354, June 1965.

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

[UNIVAC 1100, 1971b]
UNIVAC 1100 General Purpose Systems Simulator II Reference Manual, UP 4129, 1971.

[Uyeno, 1980]
Uyeno, D.H. and W. Vaessen, PASSIM: a discrete-event simulation package for PASCAL, *Simulation* 35:6, pp. 183-190, December 1980.

[Virjo, 1972]
Virjo, Antti, A Comparative Study of Some Discrete Event Simulation Languages, Norwegian Computing Center Publication #S-40 (reprint of a presentation at the Nord Data 1972 Conference, Helsinki).

[Wallington, 1976]
Wallington, Nigel A. and Richard B. Were, SLAM: a new continuous-system simulation language, In *SCS Simulation Councils Proceedings Series: Toward Real-Time Simulation (Languages, Models, and Systems)*, Roy E. Crosbie and John L. Hays, eds., 6:1, pp. 85-89, December 1976.

[Washam, 1976a]
Washam, W.B., GASPPI: GASP IV with Process Interaction Capabilities, unpublished Master's thesis, Purdue University, West Lafayette, Indiana, 1976.

[Washam, 1976b]
Washam, W.B. and A. Alan B. Pritsker, Putting Process Interaction Capability into GASP IV, ORSA/TIMS Joint National Meeting, Philadelphia, 1976.

[Wexelblatt, 1981]
Wexelblatt, Richard L. *History of Programming Languages*, Academic Press, ISBN 012 745040 8. This is the Proceedings of the ACM SIGPLAN History of Programming Languages Conference, 1-3 June 1978.

[Wegner, 1986]
Wegner, Peter, *Programming Languages, Information Structures and Machine Organization*, McGraw-Hill Book Company.

[Wilson, 1992]
Wilson, James R., The winter simulation conference: perspectives of the founding fathers, twenty-fifth anniversary panel discussion, In *Proceedings of the 1992 Winter Simulation Conference*, James J. Swain, Robert C. Crain, and James R. Wilson, eds., 1992 (to appear).

[Wortman, 1977a]
Wortman, David B., Steven D. Duket, and Deborah J. Siefert, Modeling and analysis using SAINT: a combined discrete/continuous network simulation language, In *Proceedings of the 1977 Winter Simulation Conference*, Harold J. Highland, Robert G. Sargent, J. William Schmidt, eds., pp. 528-534, 1977.

[Wortman, 1977b]
Wortman, David B., Steven D. Duket, R.L. Hann, G.P. Chubb, and Deborah J. Siefert, Simulation Using SAINT: A User-Oriented Instruction Manual, AMRL-TR-77-61, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio, 1977.

[Wortman, 1977c]
Wortman, David B., Steven D. Duket, R.L. Hann, G.P. Chubb, and Deborah J. Siefert, The SAINT User's Manual, AMRL-TR-77-62, Aerospace Medical Research Laboratory, Wright-Patterson Air Force Base, Ohio, 1977.

[Young, 1963]
Young, Karen, A User's Experience with Three Simulation Languages (GPSS, SIMSCRIPT, and SIMPAC), System Development Corporation, Report #TM-1755/000/00, 1963.