ESD-TR-71-227

MTR-2040

SURVEY OF SIMULATION LANGUAGES AND PROGRAMS

J. C. DesRoches

JULY 1971

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 5720
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract F19(628)-71-C-0002

ESD-TR-71-227 MTR-2040

SURVEY OF SIMULATION LANGUAGES AND PROGRAMS

J. C. DesRoches

JULY 1971

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 5720
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract F19(628)-71-C-0002

FOREWORD

The survey of simulation languages and programs was prepared for ESD in order to provide the Air Force a report which would in a form of a compendium present information on the techniques and procedures in operation and/or in development of ADP simulation capabilities. This report was prepared by The MITRE Corporation, Bedford, Mass., under contract No. F19(628)-71-C-0002, MITRE Project 5720. The ESD program monitor is Mr. William J. Letendre, Technology Application Division, Directorate of Systems Design and Development. Additional work on the subject matter is continuing. Inquiries regarding the current status of this effort should be directed to the Deputy for Command and Management Systems, Directorate of Systems Design and Development, Attn. Mr. William J. Letendre, L.G. Hanscom Field, Bedford, Mass. 01730.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved.

For [illegible]
EDMUND P. GAINES, JR., Colonel, USAF
Director, Systems Design & Development
Deputy for Command and Management Systems

ABSTRACT

This report documents a survey of available simulation languages and programs of potential applicability to the simulation of ADPE systems. The major features of the subject languages are discussed and a comprehensive bibliography is included.

ACKNOWLEDGMENT

The author acknowledges with gratitude the contribution of Barbara N. Wagner in automating the bibliography; of Peter B. Burleson in reviewing GPSS, SIMSCRIPT, S3, PLS, and COST; of John H. McIntosh in reviewing ECSS and BOSS; and of Roy N. Gamse for his editorial assistance.

TABLE OF CONTENTS

TABLE OF CONTENTS (Concluded)

LIST OF TABLES

SECTION I

INTRODUCTION

PURPOSE AND SCOPE

This volume documents the first phase of a study of the available simulation languages, programs, and techniques which could be of benefit to ESD in the simulation of ADPE systems. Phase I has consisted of an extensive survey of the field of simulation languages and has included the preparation of a comprehensive bibliography on the subject. Each of the identified simulation language/packages is discussed with reference to its historical development and major distinguishing technical characteristics.

A collection of documents considered to be of a general, tutorial, or survey nature has been assembled; and an annotated bibliography has been computerized for ease of reference and rapid retrieval of pertinent information.

In addition, a survey of the literature for papers relative to the simulation of computer systems has been initiated and the documentation is being assembled. This effort represents the second phase of the study and will be reported on in Volume 2 of the MTR to be released at a later date.

ORGANIZATION OF THE REPORT

The report is organized into three major sections:

1. Theory of Simulation
2. Simulation Language/Package Compendium
3. Bibliography

The section on Theory of Simulation is included to introduce the basic concepts and terminology of simulation. The treatment is not definitive; rather it is only of a depth necessary to assure comprehension of the subject matter in the compendium.

The compendium represents the major content of the report. Languages have been classified as general purpose (suitable for the simulation of discrete, recursive systems[1]) and special purpose

[1]Computer systems are so classified.

(designed specifically for the simulation of computer systems). Additionally, special purpose program packages exist for the evaluation of computer systems. All of the identified components of each subject class are discussed. It is the view of the author that this report might serve as a reference guide, and to this end some languages which are only of limited academic interest are included.

The Bibliography constitutes the final section of the report.

SECTION II

THEORY OF SIMULATION

MODELING AND SIMULATION

Simulation is the technique of constructing a model which is a representation of the significant properties of an object, process, or system and then performing experiments on the model. This experimentation serves to identify behavioral characteristics and makes possible the prediction of performance in a real-world environment. Variable levels of abstraction range from the construction of physical models to mathematical representation.

The development of large-scale, digital computers coincidental with advances in the theory of simulation has greatly expanded the scope of studies which are amenable to this methodology. Simulation has become a valuable tool for studying the dynamic responses of systems that consist of a large number of interacting processes of such a complex nature that an analytical solution is practically impossible.

In a computer simulation the object system is represented by a computer program which describes the system components, the process relationships, and the environment.

CONTINUOUS AND DISCRETE MODELS

Simulation models are classified into two major types: continuous change models and discrete change models. Continuous change models are those in which the system entities are considered in the aggregate and in a state of continuous interaction. These systems are generally represented mathematically by a set of n^{th} order differential equations that define the rates at which the system variables change with time, given specified constraints and equilibrium relationships. Continuous change models have historically been simulated on analog computers because of their continuous mode of operation. But in many cases applications are restricted because of hardware limitations and lack of accuracy. Recently, digital-analog simulator programs have been developed which simulate the elements and organization of analog computers. These allow continuous system problems to be programmed for execution on digital computers in a manner approximating analog computer problem formulation. A discussion of this topic is beyond the scope of this paper; papers by Clancy and Fineberg and by Brennan and Linebarger provide a comprehensive treatment of the subject. Languages developed for continuous change simulations are identified and listed for reference in Table I.

Systems which are characterized by changes which are predominantly discontinuous are called discrete systems. In this view the interaction between state variables occurs infrequently and "instantaneously." In reality, of course, all activity is continuous. Continuous activity is approximated by computing state changes only at the instants of interaction and assuming no activity takes place between interaction times. Discrete systems are modeled as network flow systems. A network flow system is comprised of components which perform functions according to specified operating rules. Items flow through the system according to prescribed paths and are processed, giving rise to sequences of events (i.e., changes in system state) which frequently occur in a stochastic manner.

DISCRETE SIMULATION LANGUAGES

Historical Background

Discrete systems simulation traces its early origin to the field of operations research and in particular to the statistical technique of distribution sampling. In the early 1950's Monte Carlo techniques were applied without benefit of a computer to the study of the dynamic behavior of physical systems. With advances in the field of computer technology, these studies became prime candidates for automation. The first programs were special-purpose models, coded in assembly or compiler language, each designed to solve a specific problem. As experience was gained techniques applicable to specific classes of problems (i.e., job-shop and inventory management simulations) were abstracted and implemented to produce special purpose computer simulation packages. To use these the analyst supplies parameter values and control variables which are specific to his application.

General-purpose simulation programming languages applicable to a wide range of discrete-change models originated in the late 1950's and have evolved since then through the incorporation of advanced concepts, as theories of simulation have been formalized.

Common Features

Simulation languages are specialized algorithmic languages which provide features of particular relevance to simulation. While

Table I

Continuous Change Simulation Languages/Packages

ASTRAL

BHSL
BLODI

360/CSMP
COBLOC
CSSL

DAS
DEPI
DEPI-4
DES-1
DIAN
DIDAS
DSL/90
DYANA
DYNAMO
DYNASAR
DYSAC

EASL
FORBLOC
HYBLOC
JANIS

MADBLOC
MIDAL
MIDAS and ENLARGED MIDAS
MIMIC

PACER
PACTOLUS
PARTNER
PLIANT

SADSAC
SALEM
SCADS
SIMTRAN
SLANG
SLASH
SPLASH

TAF
UNITRAC
WIZ

they differ in their conceptualization of a "world view"[2] and in their manner of implementation, all provide the necessary facilities to describe the object system's static and dynamic structure. The basic components of a simulation language include:

1. A data structure which allows for the identification, description, classification and manipulation of the constituent entities of a system.

2. A collection of commands to modify the state of the system.

3. A master timing routine to provide the sequencing control.

4. A capability to generate pseudo-random numbers from specified probability distributions.

5. Facilities for the aggregation and presentation of the statistical results.

WORLD VIEW

A description of a real world system consists of two prime components:

1. A static representation.

2. A dynamic representation.

Kiviat states: "The static structure of a simulation model is a time-independent framework within which system states are defined... Dynamic system processes act and interact within a static data structure, changing data values and thereby changing system states."[3]

Static Representation

The static structure of a system is described in terms of the objects which it contains and the relationships that exist between objects. Objects are characterized by attributes or properties

---

[2]Krasnow and Merikallio define "world view" to be "the conceptual foundation upon which a representation of a system may be built." Howard S. Krasnow and Reino A. Merikallio, "The Past, Present, and Future of General Simulation Languages," Management Science, Vol. 11, No. 2, November 1964, p. 237.

[3]Philip J. Kiviat, "Digital Computer Simulation: Computer Programming Languages," RAND Memorandum RM-5883-PR, January 1969, p. 10.

whose values may be permanent or temporary. The status of an object, at any point in time, is uniquely defined by the values of its attributes.

Because of the large number of objects in a typical system and because of a lack of uniqueness, objects are aggregated into classes. A system is defined in terms of its constituent classes and a qualitative delineation of the set of attributes associated with each. The actual status of each object is defined dynamically as the simulation model is exercised.

The data structures onto which these status descriptions are mapped and the referencing techniques are functions of the language design. Many languages allow for the dynamic creation and destruction of objects and provide the capabilities to form disjoint classes of objects based on differences in status. Ordered sets are used for representing queues and files of objects. The languages provide operators of varying power to manipulate set memberships.

Dynamic Structure

The dynamic structure of a system is concerned with representing the possible state changes within a system and the sequential relationships that exist between them. Every simulation program is comprised of modules which describe specific system behavior patterns and an executive routine which provides the time mechanism and the control to assure the execution of the modules in the proper time, logic and priority sequence.

Simulation languages may be classified according to their principal dynamic modeling orientation:

1. Event-oriented languages.

2. Activity-oriented languages.

3. Process-oriented languages.

An event represents a change in state. It is conceptualized as occurring instantaneously and taking no time. In contrast, an activity represents a time consuming action within the system and is bounded by two events, namely the initiation and the termination of the action. A process is a set of events which are associated with a specific system behavior pattern. It is like an activity in that it exists over a period of simulated time. The characteristics of some physical systems are such that they would dictate the choice of one language class as dominant for the execution of a simulation study.

The terminology event-oriented simulation derives from the fact that the model is segmented into routines called events. These routines consist of procedural statements describing the instantaneous changes in system state which result from the occurrence of the subject events. An event is scheduled (i.e., assigned a time of occurrence) by special language statements either when the logic of the model indicates that all necessary conditions have been satisfied, or alternatively, when a random drawing from a specified statistical distribution indicates that it should be scheduled. Schedule occurrence times are maintained on an event list. It is the function of the executive program to maintain both the currency and the chronology of the list and to execute the proper subprogram as each event "occurs" in simulated time.

An activity-oriented simulation language differs from an event-oriented one in that it contains no explicit scheduling statements within its repertoire. Under this conceptual approach, the model is segmented into modules called activities. A moduie consists of both (1) a description of the time consuming action which produces change in system status, and (2) the set of conditions upon which the initiation of the action depends. Within activity modules element clocks are maintained which indicate the time at which entities associated with the activity are due to change state. The action section is comprised of a set of state-changing and time-setting instructions.

In an activity-oriented model the scheduling is achieved via the mechanism of an activity scan. Before each simulated time advance, the control program scans all activities, testing the conditions and determining the potential of each for execution. If all test conditions are satisfied, the action section is executed, otherwise control is transferred to the next activity. This procedure is repeated in cyclic fashion to account for the interdependency between the activities. To improve operational efficiency, some activity-oriented languages also incorporate the feature of an event list to schedule events which are non-interactive.

The third orientation for the modeling of system dynamics is the process concept. A process, as previously defined, is a set of events which are descriptive of a specific system behavior pattern. It represents a combination of both the activity and event formulations. In a process-oriented simulation the model is segmented into units called processes each of which consists of multiple event subprograms. The execution of a process extends over system time and consists of alternate periods of activity and inactivity. Only one process is in the active state at any point in time but processes interact and are joined together by time-dependent or status-dependent conditions. Thus the outcome of a simulation study is described by the sequence of active phases of the component processes. This mode of operation has been defined as quasi-parallel.

In addition to arithmetic and logic statements, sequencing statements are provided which cause the transfer of control from process to process. The execution of a sequencing statement causes control to leave the process, and the location following such a statement is defined as a reactivation point. Control is transferred back to this point at the initiation of the next active phase of the process. Proper sequencing is maintained by an executive program. Event-scheduling techniques are employed when a time-oriented scheduling statement is encountered, and an activity-scan is initiated when a condition-oriented scheduling statement is encountered.

This method has some obvious advantages. It allows for the concise notation of activity-oriented languages which is useful in describing highly interactive processes and takes advantage of the increased run time efficiency of event scheduling. The result is a highly sophisticated executive program.

Some authors further classify simulation languages as machine or material based. If the major emphasis of the system study is the observation of actions executed <u>by</u> entities, the system is considered machine oriented; conversely, if the emphasis is on the observation of actions performed <u>upon</u> entities, the system is material oriented.

SIMULATION LANGUAGE TECHNIQUES

<u>Timing Concepts</u>

In a computer simulation the independent variable is time. A master timing routine must provide the point of reference within the model to control the advancement of time. It must also provide the scheduling logic to control this advancement in a manner consistent with the time and sequence relationships between the components of the simulated system. The problem is complicated by the fact that computers are sequential in nature while real world physical systems are sometimes characterized by the occurrence of many parallel and interdependent activities.

Basic to the representation of simulation time is the concept of a master simulation clock. It is a variable which is successively updated to represent the current point on the simulation time scale.

The first simulation clocks were counters that were advanced in fixed increments representing the basic time unit of the simulation (i.e., seconds, minutes, hours, etc.). The system was examined every unit of clock time to determine whether any events were due to occur. Fixed time increment methods are computationally efficient in the simulation of systems in which events occur with predictable regularity. When this is not the case, the method leads to long periods during which the computer is "idle."

An alternative concept for controlling time has been called next-event or critical event simulation. This method involves scanning the event list and advancing the master simulation clock by the amount necessary to cause the occurrence of the next most imminent event. In this manner the program is stepped from event to event, maximizing operational efficiency by compressing those increments of time in which no state changes occur. Since most physical systems are categorized by the occurrence of non-periodic random changes, the variable time increment method of controlling time has been implemented in the vast majority of discrete simulation languages.

List Processing

The state of a system at any point in time is completely specified by the list of entities, their associated attributes and set memberships. The process of simulation consists of changing the relationships among these data as a function of time. Hence, data structures and list processing techniques are of importance for accessing and updating this information.

List processing techniques are utilized in the representation and management of queues and in the scheduling of events. Lists are series of words in non-contiguous storage whose relationships are maintained by the use of pointers. Chaining techniques associate entities with their attributes and set memberships. This capability facilitates the ability to change set memberships, to scan for entities possessing specific attributes, and to maintain queue discipline.

Computer storage is usually dynamically allocated. This is appropriate since the dynamic nature of simulation is such that the lengths of queues and set memberships are variable and change as a function of time.

Random Number Generation

Simulation studies are characterized by the random and probabalistic occurrence of events. Basic to simulation languages is the incorporation of techniques to allow for the generation of random variables, the transformation of these into variates from specified statistical distributions, and the maintenance of independent and reproducible streams of random numbers. The following are representative of the distributions commonly available in discrete simulation languages:

1. Normal
2. Poisson
3. Geometric
4. Exponential
5. Rectangular

SECTION III

SIMULATION LANGUAGE/PACKAGE COMPENDIUM

This section presents the results of a survey of simulation languages/packages which might have applicability in the simulation of ADPE systems. While some languages are not applicable or are obsolete, they have been included for historical reasons.

The languages/packages have been classified by major orientation as:

1. General Purpose Simulation Languages

2. Computer Systems Simulation Languages

3. Computer Systems Simulation Packages

Each language is described within a common format:

- Background Information
  - Author
  - Originating Organization
  - Date of Release
  - Status
  - Machine Implementations
- Major Features
- Applications
- References

The information is as described in the literature. No attempt has been made at verification; and especially in the matter of machine implementation, the data may be incomplete.

As stated previously, Volume 2 of this report will cover in depth the use of simulation languages in the modeling of computer systems. Therefore, information relevant to applications is not consistently included and is of a general and cursory nature.

GENERAL PURPOSE SIMULATION LANGUAGES

AS - An ALGOL Simulation Language

1. Background Information

   Author: R. D. Parslow

   Originating Organization: Brunel University, London

   Date of Release: 1967

   Status: Under continuous revision

   Machine Implementation: Written for KDF9 at National Physics Laboratory, adaptable to any machine with ALGOL compiler.

2. Major Features

AS, an activity oriented language, consists of a collection of ALGOL procedures designed to provide the basic capabilities necessary for simulation. It enables the user to write a simulation program in ALGOL incorporating the relevant modules from the package. Its method of operation is derived from GSP (General Simulation Program) by K. D. Tocher.

Matrix notation is used to record the system elements, both static and dynamic, together with their associated data.

The system elements are defined as follows:

a. ents -- time dependent entities

b. depots -- serve as records or represent stores

c. activities

   b act -- bound activity, set to operate on a particular entity at a predetermined time

   c act -- conditional activity, requires availability of several entities

d. pools -- sets of entities possessing a common attribute; pools may themselves consist of sets of pools

The matrix notation facilitates the dynamic operation of the system and provides easy access to such information as:

a. Entities

   * status - busy or idle

   * time of completion if busy

   * next scheduled b act

   * committal parameters (i.e., distribution type and parameters, randon stream, etc.)

   * final location in pool at completion

b. Pools

   * number of members

   * earliest time of availability

   * reference number of earliest available member

c. Activities

   b act

   * committal parameters for entities engaged in multiple activities

   c act

   * records ents or pools required for c acts

d. Utilization of entities and pools

   * total number of activities engaged in

   * total committal time

The phase structure is controlled by the procedure "next" which is called at the completion of each activity block in the object program. It controls the cycles of four phases of the simulation:

a. T-Phase - advances time to the next event. This is facilitated by grouping time-dependent entities into pools. The minimum time for each pool is recorded in the matrix.

b. B-Phase - scans all entities to enter all bound activities due and to release working entities whose committal time is over. A tolerance value may be input which allows the grouping of all events due to occur within the specified ΔT.

c. C-Phase - causes transfer to the first conditional activity. Conditions are tested by a Boolean procedure "c act." If satisfied: activity is committed. If not satisfied: enter A-Phase which tests if unavailable entities will be free after present task and if so reserves them for the specific "c act."

After the execution of the C-Phase, control is transferred back to the T-Phase.

The major emphasis in the development of the language was to provide a framework within which simulation studies could be easily formulated and speed of operation was considered to be of secondary importance.

Reference

R. D. Parslow, "AS: An ALGOL Simulation Language," in J. N. Buxton, ed., Simulation Programming Languages, North Holland, Amsterdam, 1968, pp. 86-100.

CLP - The Cornell List Processor

1. Background Information

Authors: J. W. Conway
J. J. Delfausse
W. L. Maxwell
W. E. Walker

Originating Organization: Cornell University, Department of Engineering

Date of Release: 1964

Status: Current

Machine Implementation: Control Data 1604
Burroughs 220

2. Major Features

CLP was developed as a teaching vehicle to introduce the concepts of simulation and other list processing techniques in a language that did not require a high level of programmer expertise. This was accomplished by including CORC, a general algebraic language, as a subset of CLP.

The major feature of the language is a special purpose data element called an entity which has associated properties called attributes. Entities are dynamic in that the number varies as a function of time. Special language statements allow for the creation and destruction of entities and for the creation of sets ordered on the value of a particular attribute of its component members. The dynamic assignment of core storage and overlay techniques make possible the ephemeral nature of the data structure.

CLP derives some of its concepts from SIMSCRIPT, in particular it incorporates a Report Generator to specify variable output formats. It does not include a timing routine and this function remains a programmer responsibility.

Reference

R. W. Conway, J. J. Delfausse, W. L. Maxwell, W. E. Walker, "CLP - The Cornell List Processor," Communications of the ACM, Vol. 8, No. 4, April 1965, pp. 215-216.

CSL - Control and Simulation Language

1. Background Information

Authors:	J. N. Buxton
	J. G. Laski

Originating Organization:	CSL	- ESSO Petroleum Co., Ltd. and IBM United Kingdom, Ltd.
	CSL2	- IBM United Kingdom, Ltd.
	ECSL	- Courtaulds, Ltd. and Honeywell Controls, Ltd.

Date of Release: 1963

Status:	CSL Current
	CSL2 in Preparation
	ECSL Current

Machine Implementation:	CSL	- IBM 7090
	CSL2	- IBM 7090/7094
	ECSL	- Honeywell 400 and 200 Series

2. Major Features

CSL is a programming language designed to simulate industrial scheduling and waiting line systems involving logical decisions of great complexity. It is a descendant of GSP; and although developed independently, it has many conceptual features similar to SIMSCRIPT.

The language was designed as an experiment to test the premise that a combination of the techniques of queueing theory with those of predicate calculus would provide a capability for the formulation and solution of complex logical problems. As a result the principal feature of CSL is the concept of sets whose members are entities which are in a specific state or possess a common attribute. Since the order in which entity names are placed on a set is preserved at all times, a set can be used to represent a queue of its members. The language offers extensive facilities for set creation and manipulation. The state of a system at any point in time is defined by lists of entities belonging to each set in the system and by their attribute values stored in arrays.

A CSL program is divided into sectors called "activities" which completely describe all possible operations within the system. Each activity includes a set of conditions which must be satisfied before a specific operation of the system can be initiated and a specification of state changes which are enacted if the conditions are satisfied. This type of sequencing is defined as interrogative sequencing and is necessitated by the inability to predict in advance the system time at which a given event should take place.

Time in a CSL simulation is controlled via the mechanism of T-cells which are associated with entities and indicate the time at which the entity is next due to change its state. All time values are relative to the present simulated instant. Time advancement is controlled via a repeated two-phase process. During Phase 1 all time cells are scanned to locate the smallest, positive, non-zero value. Simulated time is advanced by this amount, and all T-cells are correspondingly reduced. Phase 2 consists of an attempt to execute each activity in sequence. During the activity scan the conditions for each activity are tested, and unless explicitly stated otherwise the standard path is: after success go to next statement, after failure go to next activity. Each instance of an activity constitutes an event. To allow for interdependency between events, the RECYCLE statement causes the activity list to be rescanned at the same simulated time.

The form of arithmetic and assignment statements is similar to FORTRAN. In the initial version of the language, CSL statements were first compiled into FORTRAN statements before compilation into IBM 7090 machine code. Newer versions of the language, CSL2 and ECSL (Extended Control and Simulation Language), have abandoned this approach; and compilers are available to translate directly into assembly language for IBM and Honeywell computers respectively. This is a more efficient mode of operation and allows for extensions to the language not previously possible because of FORTRAN restrictions.

3. Application

In an extract from a forthcoming work by Buxton and Laski, the authors express the opinion that the language is not well suited to describing problems which are concerned with flow and in which the logical decisions made are not of great complexity. Examples cited are the flow of information through a computing system or the flow of work through a single production unit.

References

J. N. Buxton and J. G. Laski, "Control and Simulation Language," Computer Journal, October 1962, pp. 194-200.

J. N. Buxton, "Writing Simulations in CSL," Computer Journal, August 1966, pp. 137-143.

A. T. Clementson, "Extended Control and Simulation Language," Computer Journal, November 1966, pp. 215-220.

ESP - The Elliott Simulator Package

1. Background Information

Author: J. W. J. Williams

Originating Organization: Elliot Scientific Computing Division

Date of Release: 1964

Status: Current

Machine Implementation: Elliot 503 and 803

2. Major Features

ESP, an activity oriented language based on GSP, is comprised of a set of ALGOL procedures.

The system concept is that of a set of actions which manipulate a set of objects. Objects are represented numerically, and the actions are represented by program segments. Actions have been categorized into:

a. delayed -- scheduled to occur at a specific time

b. conditional -- dependent upon the satisfaction of a set of logical conditions

Each delayed action is represented by a program segment, while all conditional actions are grouped into a single section.

The process by which the occurrence of a delayed action is scheduled is referred to as "calling" that action. Any action may call any other including itself. The statement "call (i,t)" schedules the i^{th} delayed action to be executed after t time units have elapsed. Parameters are transferred via the "send and get" arrays and are made available at the time of scheduling. This produces sets of local variables defined in time. The use of sophisticated sorting routines and dynamic storage allocation provides for multiple occurrence of each event with independent sets of parameters.

The ESP program controls the maintenance of simulation time and the correct sequencing of actions. This is accomplished in the following manner. A list of times of pre-scheduled (i.e., delayed) actions is scanned by a procedure called "next," and the action(s) associated with the minimum time is executed. After this the conditional actions are tested sequentially, and any whose criteria for operation are satisfied are also executed. The two stage cycle is then repeated.

Additional features include facilities for generation of independent streams of random numbers, construction of histograms, and manipulation of queues.

Reference

J. W. J. Williams, "ESP - The Elliott Simulator Package," Computer Journal, January 1964, pp. 328-331.

FORSIM IV

1. Background Information

Author: E. Famolari

Originating Organization: The MITRE Corporation

Date of Release: 1964

Status: Current

Machine Implementations:	IBM 7030	GE 635
	IBM 7090	CDC 3400
	IBM 7094	CDC 3600
	IBM 360/50	

2. Major Features

FORSIM IV is a subroutine-structured general purpose simulation language written (with very minor exceptions[1]) in FORTRAN IV. With minimal effort it can be adapted to any computer with a FORTRAN IV compiler and may therefore be considered "machine-independent."

The conceptual framework of the language is based on that of the Control and Simulation Language (CSL). However, it provides additional capabilities, and the modular structure allows for easy expansion of the command set.

FORSIM IV employs the concepts of entity, class, and set with the standard definitions. Sets serve the same function as in GPSS; that is, as stores, facilities, and queues. The set of subroutines which perform the actions and tests associated with the relationships between sets, entities, and attributes is classified as Set-Entity. It provides the unifying concept for the 50 subroutine components of the package, allowing the entities to be accessed by the use of index values. A complete classification of the command routines is:

a. Set-Entity

b. Time

[1]The debug routine and random number generators were written in STRAP, the 7030 machine code.

c. Histogram

d. Statistical

e. Random number and debug

Set-Entity routines are further classified as Action routines, which perform unconditionally, and Dual routines, which perform only if certain logical conditions are met.

The I/O, arithmetic, decision logic, and normal processing functions are provided by FORTRAN IV. The program consists of the appropriate FORTRAN IV statements and FORSIM IV subroutines organized in a manner which reflects the logic of the simulated system. In general terms the program will consist of initialization, working, and termination sections. The working section is comprised of a sequence of routines each describing a unique type of system action involving time-dependent or dynamic elements of the system. Each routine is termed an Activity, and the sequences is called a List.

Time is advanced in variable time increments to the next most imminent action time. The dynamics of the language allow time to be associated with both entities and activities. A recycle feature allows an additional cycle through the activities list with the simulation time held constant.

Reference

E. Famolari, "FORSIM IV User's Guide SR-99," The MITRE Corporation, February 1964.

GASP - General Activity Simulation Program

1. Background Information

Author: Original Version: P. J. Kiviat
Revised Version: J. Belkin & M. R. Rao

Originating Organization: United States Steel Corporation
Applied Research Laboratory

Date of Release: Original Version: 1963
Revised Version: 1965

Status: Current

Machine Implementation: IBM 1130, 1620, 1830, 7040/7044, 7090/7094
GE 225, 415
XDS 930
NCR 315
XDS Sigma 7
Burroughs 3500, 5500
CDC 3400, G-20

2. Major Features

GASP consists of a collection of 23 FORTRAN subroutines designed to be used in FORTRAN-written simulation programs. These routines are representative of operations common to computer simulations. A simulation utilizing GASP consists of a set of FORTRAN subroutines, called GASP events, that are organized via the GASP EXECUTIVE, which maintains the temporal order of the model. This approach has the dual advantages of modularity and machine independence.

GASP provides the facilities to:

a. control the sequence of activities

b. maintain list structures

c. sample from statistical distributions

d. aggregate results and compute statistics

e. automatically monitor program variables and flow

Special symbols and GASP-oriented conventions are used to describe system behavior in the form of flow charts which delineate operations, decisions, transfers and control. These flow charts are then translated into FORTRAN statements.

The basic component in GASP is termed an element. It may be permanent or temporary and is described by a set of attributes. The logic that describes the interaction of elements and the attendant change of system status is called an event. It occurs instantaneously in time and represents the beginning and/or end of an activity. Each event is written as a separate FORTRAN subroutine which is executed when the event "occurs" in simulated time.

It is the function of the GASP executive program to maintain the proper chronology between the events. This is achieved via an "event list" ordered on the time of occurrence of each scheduled event. GASP is a variable-increment time model in which the scheduling of events is the only mechanism of advancing time. This is accomplished within GASP events; each event is capable of scheduling itself or any other event.

Reference

P. J. Kiviat, "GASP - A General Activity Simulation Project," available from Applied Research Laboratory, United States Steel, Monroeville, Pa., 1963.

GPSS - General Purpose System Simulator

1. Background Information

Authors: R. Efron
G. Gordon

Originating Organization:

GPSS (B5500)	- Burroughs, GPSS/360 capability
GPSS III	- Control Data Corp.
GESIM	- General Electric, GPSS/360 capability
GPS-K	- Honeywell
GPSS/360	- IBM
Flow Simulator	- RCA, GPSS/360 capability
GPSS II	- Univac, FORTRAN implementation
GPDS	- Xerox Data Systems, GPSS/360 capability due February 1971

Date of Release: Several versions, the first of which appeared in the early 1960's

Status: Maintained and supported by IBM

Machine Implementation:

GPSS	- B5500
GPSS III	- CDC 3600
GESIM	- GE 600 series
GPS-K	- Honeywell 200 series
GPSS/360	- IBM 360
Flow Simulator	- RCA Spectra 70
GPSS II	- Univac 1107-1108
GPDS	- XDS Sigma 5, 7

2. Major Features

GPSS has been extended and improved since its origination, and present implementations (e.g., GPSS/360) provide a versatile and fairly powerful language which can be learned and applied by non-programmers in a rather minimal amount of time. These desirable characteristics derive primarily from the block diagram nature of the language. The system being simulated is modeled using a fixed set of predefined block types. These block types represent the various actions and functions occurring in real systems. The interconnections between these block types in a particular system model reveal the structure of the system; i.e., the ordering of the succession of events. Since there exists a one-to-one correspondence between blocks and GPSS coding, a model can be encoded directly from its diagram. GPSS is, therefore, a highly structured language relying on predefinition for power and simplicity.

GPSS has other advantages, particularly to the novice. Statistical analyses of system events are provided automatically, along with a rather extensive set of error diagnostics. For all the above reasons, GPSS has achieved the wide acceptance indicated by the machine implementation schedule shown above. It is certainly beneficial to the GPSS user that he can find GPSS/360 capability with machines made by Burroughs, GE, RCA, and XDS, as well as IBM.

On the negative side, GPSS does suffer some drawbacks relative to other alternatives. The operations specified by GPSS coding are executed interpretively; i.e., GPSS programs are not compiled or assembled, but are examined by the IBM GPSS program. The actions specified by the user coding are carried out under control of this IBM Program. Execution of GPSS simulations, therefore, always requires submission of a source deck (the only object program is that supplied by IBM). The interpretive execution of this source deck usually requires more computation time than would be required by simulation languages which do not operate interpretively. Also, this contributes to GPSS core storage requirements, which are generally in excess of the alternatives. The data structures offered by GPSS are not as powerful as those available in many other simulation languages. Since simulation is performed by the time-phased alteration of set relationships, this disadvantage to GPSS can become important when the modeling requirements become complex.

GPSS, therefore, permits the rapid development of a simulation capability from negligible beginnings; it facilitates the generation of workable programs (and hence answers) in minimum elapsed time, and it provides a common standard language which can be used by a large number of machines and personnel. It is, however, somewhat inefficient, cumbersome, and rigid.

References

Arnold Ockene, "Losing Business? Simulation Makes It Easier to See Why," Computer Decisions, March 1970, pp. 36-40.

R. Efron and G. Gordon, "A General Purpose Digital Simulation and Examples of Its Applications," IBM Systems Journal, Vol. 3, No. 1, 1964, reprinted in Computer Simulation Techniques, by Thomas H. Naylor, Joseph L. Balintfy, Donald S. Burdick, and Kong Chu, John Wiley & Sons, 1966, pp. 248-270.

"General Purpose Systems Simulator II, Reference Manual," IBM.

"General Purpose Simulation System/360, Application Description," H20-0186-2, IBM.

"General Purpose Simulation System/360, Introductory User's Manual," H20-0304-1, IBM.

"General Purpose Simulation System/360, User's Manual," H-20-0326-2, IBM.

GSP - General Simulation Program

1. Background Information

   Author: K. D. Tocher

   Originating Organization: United Steel, Ltd.

   Date of Release: 1960

   Current Status: GSP MK II

   Machine Implementation: Ferranti Pegasus
   Elliott 503

2. Major Features

GSP represents one of the pioneering works on simulation languages. It is tersely written and intended primarily for mathematically-oriented users.

Particularly designed for the simulation of systems found in manufacturing plants, the structure consists of representing the system as a collection of machines in various states. The definition of a machine may encompass any system entity. To this extent, GSP is considered a "machine-based language"; i.e., emphasis is placed on activities executed by entities. Formulation of the model consists of defining the set of machines, the initial conditions, rules governing their operation, and a description of the actions which they perform.

The descriptive unit of GSP is called an activity; each execution of an activity constitutes an event; i.e., a change in the state of a machine. The user defines the necessary states of machines for activities to commence, and the conclusion is determined probabilistically.

The scheduling of a machine to its next activity is referred to as engaging the machine. Machines are considered to be either committed or available, and their status may be interrogated.

Activities are categorized as:

a. B-activity - A bound activity consists of a set of statements which describe the changes to be made in the state of the plant when a machine becomes available.

b. C-activity - In addition to a delineation of the state changes, a conditional activity consists of a set of statements which specify the circumstances under which it may occur.

GSP is principally an activity-oriented language, but it employs both activity scan and event-selection in its control algorithm. It operates in a three phase manner:

a. Phase A - Scans the list of engaged machines and advances the clock time to the earliest time of machine availability.

b. Phase B - Bound activities associated with available machines are executed.

c. Phase C - Scans list of conditional activities and performs all possible state changes.

Both references present the language specifications in a level of detail beyond the scope of this paper. Reference 2 deals with MK II, the revised version of GSP. Advanced language features are discussed with reference to MK I, MONTECODE, SIMSCRIPT, CSL and RSP.[1] In summary, MK II eliminates the restrictions of the MK I version by providing more sophisticated techniques for activity selection, set manipulation, monitoring facilities, sampling procedures and report generation.

References

K. D. Tocher and D. G. Owen, "The Automatic Programming of Simulations," Proceedings of Second International Conference on Operations Research, 1960, pp. 50-68.

K. D. Tocher and D. A. Hopkins, "New Developments in Simulation," Proceedings of Third International Conference on Operations Research, 1963, pp. 832-848.

[1]Recursive Simulation Program is an interim program with limited facilities which reduces a simulation to a recursive calculation in order to obtain a rapid appraisal of system behavior. A more comprehensive program is in preparation.

MILITRAN

1. Background Information

   Author: Shapiro

   Originating Organization: Systems Research Group, Inc.

   Date of Release: 1964

   Current Status: Obsolete

   Machine Implementation: IBM 7090/7094

2. Major Features

Although MILITRAN is considered obsolete, it is included in this compendium for historical reasons. It is a general purpose, problem-oriented language which was designed to facilitate the simulation of military systems. The development of the language was jointly sponsored by the Office of Naval Research and the Air Force Systems Command (ESD).

Its special features include object modes, list processing statements, event processing procedures, special retrieval arrays, etc.

The basic elements in the language consist of individual objects, object types and object classes. The essence of simulation consists in maintaining the status and interaction of the participating objects by the processing of events. An event is categorized by time of occurrence, participating objects, necessary conditions, and associated processing. MILITRAN classifies events as permanent and contingent. They differ in the manner of occurrence; permanent events perform activities which cannot be readily scheduled but occur at periodic intervals, while contingent events occur at a critical juncture in time and are scheduled when circumstances dictate.

The CONTINGENT EVENT statement fulfills the following functions:

1. Defines an event type.
2. Associates with the event a list of potential event occurrences, each of which has the following parameters:

P_1 - scheduled-event time

$$\left.\begin{array}{l} P_2 \text{ - attacking object} \\ P_3 \text{ - target object} \\ P_4 \text{ - } \\ \cdot \\ \cdot \\ \cdot \\ P_i \end{array}\right\} \text{Optional}$$

3. Delineates the program steps representative of the occurrence of the event.

The potential events are generated dynamically as the simulation progresses through time. Thus, at any point in the simulation the set of future potential events is defined by the set of all entries on CONTINGENT EVENT lists. A PERMANENT EVENT does not necessarily require a list.

The NEXT EVENT statement is used to start the event processing and to maintain the proper chronological sequencing. Permanent events, if they exist, are executed in sequence and always precede every contingent event. After the completion of this phase the contingent event list is scanned (either in its entirety or a specified subset thereof), and the event with the minimum time component on its list is selected for processing.

Reference

"MILITRAN Programming Manual," Systems Research Group, Inc., ESD Report ESD-TDR-64-320.

MONTECODE

1. Background

   Authors: D. H. Kelley
   J. N. Buxton

   Originating Organization: British Iron and Steel Research Association (BISRA), United Kingdom

   Date of Release: 1959

   Status: Unknown

   Machine Implementation: Ferranti Pegasus I

2. Major Features

Montecode, one of the first simulation languages, was developed in England in 1959. Many of the later languages represent an extension of the basic concepts of Montecode. It is an interpretive language and was developed as a means of expediting the Monte Carlo simulation of industrial scheduling problems. The basic language features of Montecode are similar in many respects to those of Autocode.

The distinguishing feature of Monte Carlo simulations is that probability distributions are used to determine the inter-arrival and service times of each independent variable. Montecode provides for the random sampling from distributions, the maintenance of queues, sequencing of events based on the critical event principle, and the presentation of results in histogram form.

The controlling elements in the simulation are defined as the "action times." Each action time is associated with an event and indicates the time, relative to current simulation time, when the event will next occur. When the action time becomes zero, the event "occurs"; i.e., a programmed subroutine is executed.

Reference

D. H. Kelley and J. N. Buxton, "Montecode - An Interpretive Program for Monte Carlo Simulation," Computer Journal, July 1962, pp. 88-93.

NSS - New Simulation System

1. Background Information

Authors: K. R. Blake
G. P. Blunden
K. S. Krasnow
B. M. Leavenworth
L. J. Parente
S. C. Pierce

Originating Organization: IBM Corp., Advanced Systems Development Division

Date of Release: Not released

Status: IBM Proprietary

Machine Implementation: A prototype processor currently being implemented

2. Major Features

NSS is an experimental language which was designed for use in the representation and simulation of systems with a high degree of interaction and parallelism; the authors consider multiprocessing to be a representative example. Additionally, the language incorporates features which easily accommodate the experimental nature of simulation studies.

The language is process-oriented; and although patterned after PL/I, it is based on its own data and program structures.

In the context of this language, an entity is any system component, static or dynamic, that can be modified, categorized or observed. By this definition, sets and processes are classified as entities. The characteristics of an entity are described by attributes whose values, taken in the aggregate, completely define the current state of the system.

As previously stated the unifying concept of the language classifies a process as an entity. A process is defined as an entity that exists over time and possesses dynamic characteristics that render it capable of altering the system state (such as creating and destroying entities, altering set membership, modifying the state of an entity and affecting interactions in the system) on the basis of complex logical decisions. As such, a process is not completely specified by its associated attributes and must be defined in terms of its behavior pattern. Each type of system process is defined in terms of its behavior pattern, and the system dynamics are completely defined by the set of all behavior patterns. The "component description" section of the model identifies and specifies the attributes of each system entity; the "behavior description" section specifies the behavior patterns for all process types. A process is similar to a static entity in that it is a system component, possessing attributes and capable of being modified and observed; it differs in that it produces change. A static entity may be conceptualized as a data carrier that does nothing.

Entities and processes may be defined whose behavior is related solely to the experimental aspects of the simulation study; no distinction is made between these and model entities. Representative of these are facilities for specifying initialization and environmental conditions, defining processes which monitor and measure system entities, analyze the data and provide output reports.

The statements of a process may be organized into blocks that are executed in line or into FUNCTIONS which are accessed in the manner of a subroutine. Functions are recursive, and the arguments may take the form of an expression or a reference to any attribute.

The value of an attribute may be the name of another entity. This dynamic referencing capability allows for the definition of complex relationships between entities and facilitates the construction of generalized program modules.

Interaction within and between processes may occur in either a time-dependent or state-dependent manner; the sequencing algorithm provides the control necessary to allow for the simulation of parallel processes on a sequential machine.

A macro facility provides for the definition of new processes in terms of those already defined, thereby extending the language and allowing it to be organized for the simulation of specialized classes of problems.

In conclusion, the language attempts to provide a simpler conceptual framework and advanced experimental capability.

References

G. P. Blunden and H. S. Krasnow, "The Process Concept as a Basis for Simulation Modeling," Simulation, August 1967, pp. 89-93.

H. S. Krasnow, "Highlights of a Dynamic System Description Language," IBM Advanced Systems Development Division, TR-195, 1966.

R. J. Parente, "A Language for Dynamic System Description," IBM Advanced Systems Development Division, TR-180, 1965.

R. J. Parente and H. S. Krasnow "A Language for Modeling and Simulating Dynamic Systems," Communications of the ACM, Vol. 10, No. 9, September 1967, pp. 559-567.

OPS-3/OPS-4

1. Background Information

Authors: Martin Greenberger
Malcolm M. Jones
James H. Morris, Jr.
David N. Ness

Originating Organization: Massachusetts Institute of Technology

Date of Release: 1965

Status: OPS-3 current
OPS-4 specified but not programmed

Machine Implementation: OPS-3 Modified IBM 7094 (CTSS)
OPS-4 GE 645

2. Major Features

OPS-3 is a simulation language which was designed to extend the facilities of the Compatable Time-Sharing System (CTSS) to allow for on-line programming and model building.

On-line facilities allow the user to interact with the computer and to build, modify, test, and run simulations in an incremental and repetitious manner and also to incorporate into the simulation the actual phenomenon being simulated. This mode of simulation dictates a specially designed simulation language and represents the rationale for the development of OPS-3.

OPS-3 is characterized by the following features:

a. Easy to modify model structure without recompilation or reloading.

b. Data structures are specified and initialized dynamically.

c. Complete or partial reinitialization of the model is possible.

d. Scheduling mechanism, called the AGENDA, is accessible to the user.

e. Comprehensive tracing and debugging facilities are provided.

f. General algebraic language is available.

g. Linkage with subroutines written in any language is provided for.

The installation of a new time-sharing system called MULTICS provided the impetus to redesign OPS-3, to correct its deficiencies, and to make optimum utilization of the more powerful features of the new system. OPS-3 is an interpretive language and, therefore, execution is slow; it has limited data entities and statistics gathering routines and an awkward syntactical structure.

OPS-4 exists only as a set of specifications and has not been programmed. Since on-line simulation appears to be a future trend in computer simulations, the more important features of the language will be briefly discussed.

MULTICS, implemented on the GE 645, employs paging techniques and allows multiple users to access the same program simultaneously while maintaining unique data segments.

The language OPS-4 is a subset of PL/1 from which it derives its basic algebraic and data handling capability. Special simulation features as well as specialized data types (sets, queues, and tables) have been incorporated. It encompasses the features of both event and activity oriented languages; events may be scheduled directly as in SIMSCRIPT or implicitly as in SOL. The structure of OPS-4 is organized such that an activity is described by a program which may encompass multiple events. The execution of an activity or an event may be conditional or time-dependent. Activities are independently compiled and are written as external procedures which have local data bases and may selectively share data bases with other activities. Since the computer has a multiprocessor capability, each activity declaration must include a specification for sequential or simultaneous operation. The multiprocessor capabilities are limited and in no way negate the need for sequencing algorithms to model simultaneous events. The multiprocessor capabilities are of value in performing functions which do not affect the course of the simulation, such as user interaction, debugging monitors, statistical processing, etc.

The design of the language has been predicated on the philosophy that a simulation is repetitive and experimental in nature. The language, therefore, has features which allow for the incremental design and testing of a model. The user may interrupt the model during execution, redirect the sequence and execution of events and, if desired later, restore the model to its initial state and continue the simulation from the point of interruption.

References

Martin Greenberger, Malcolm M. Jones, James H. Morris, Jr. and David N. Ness, _On-Line Computation and Simulation: The OPS-3 System_, M.I.T. Press, 1965.

Malcolm M. Jones, "On-Line Simulation," _Proceedings - ACM National Meeting,_ 1967.

QUIKSIM

1. Background Information

   Author: David G. Weamer

   Originating Organization: National Cash Register Company

   Date of Release: NCR proprietary

   Status: To be reprogrammed for NCR Century series

   Machine Implementation: NCR 315 RMC

2. Major Features

QUIKSIM, a block structured language written in SIMSCRIPT, represents an attempt to incorporate the best features of block type and algebraic languages into a general purpose simulation language. SIMSCRIPT, itself a high level simulation language, has been used to produce an interpreter for QUIKSIM, an even higher level language. QUIKSIM is, therefore, a block-structured, interpretive language.

In QUIKSIM the simulated system is described by a sequence of block types, representative of the system logic, called an application. Transactions called jobs flow through the application executing the blocks. The system is also comprised of entities which are used to analyze the system traffic; representative of such are:

a. processing entities - facilities, storages, and switches

b. data collection entities - tables

c. computational entities - functions

Each block and entity are represented internally by a temporary data record, linked together by filing them in lists. Memory is dynamically allocated, and it is possible to insert new blocks during program execution.

The QUIKSIM interpreter consists of three major routines:

a. Input and set up routine which reads in the input, sets up temporary entities, and files them in lists.

b. A driver routine which controls the flow of a job through an application.

c. A SIMSCRIPT driver which is a simulated time clock and drives the system from event to event.

The interpreter allows for modularity in the block structure, and although QUIKSIM contains 27 block types to perform the major functions of simulation, block routines written by the user in either SIMSCRIPT or FORTRAN may be incorporated without restriction.

While no attempt will be made here to explain the functions of all the standard block types, a few of the salient features should be noted:

a. Processing entities may be grouped into classes to facilitate the searching of lists.

b. Sequences of blocks may be executed like subroutines.

c. The capability of generating jobs facilitates the simulation of parallel processing.

d. QUIKSIM assumes each processing entity has its own queue and provides for automatic queue management.

e. The need to simulate the transmission of signals in computer simulation models was responsible for the definition of two block types.

3. Application

Included in the reference as an illustration of the features of the language is an example of a remote-terminal communications system operating under a polling discipline.

Reference

David G. Weaner, "QUIKSIM - A Block Structured Simulation Language Written in SIMSCRIPT," Proceedings of Third Conference on Application of Simulation, Los Angeles, 1969, pp. 1-11.

SIMON II

1. Background Information

Author: P. R. Hills

Originating Organization: Bristol College of Science and Technology, England

Date of Release: 1965

Status: Current

Machine Implementation: Elliott 503 and 803
Any Machine With ALGOL Compiler

2. Major Features

SIMON, which is in the form of a set of ALGOL procedures, is an activity-oriented language designed for writing simulation programs based on the Tocher model. This model consists of a series of machines or entities, the state of which at any given point in time will completely define the state of the system.

Provision is made for grouping entities with common attributes into sets. Since position is maintained within the set, it has the properties of an ordered queue.

At any point in time the model machines are either in a time-dependent active state or a time-independent idle state. Time is advanced in discrete stages to the next most imminent event. To facilitate the sequencing control, the names of all machines in a time-dependent state are preserved in a list called "timeset."

Initialization of the program consists of input specifying the entities, their associated attributes and processing type, the sets, distributional data, and the initial starting conditions.

The program control consists of three phases:

A Phase - Advances simulation time to the minimum non-zero time as recorded in timeset and transfers the identity of the associated machine to Phase B.

B Phase - Contains the disengaging instructions to change the status of the machine from time-dependent to time-independent.

C Phase - Contains the engaging instructions. An iterative procedure is executed in Phase C which tests conditions and sets up all possible new time-dependent states. Control is then transferred back to Phase A.

Reference

P. R. Hills, "SIMON - A Computer Simulation Language in ALGOL," in S. H. Hollingdale, ed., Digital Simulation in Operational Research, American Elsevier Publishing Company, Inc., New York, 1967, pp. 105-115.

SIMPAC - Simulation Package

1. Background Information

   Authors: M. R. Lachner
   J. Kagdis

   Originating Organization: System Development Corporation

   Date of Release: 1961

   Current Status: Obsolete

   Machine Implementation: IBM 7090

2. Major Features

SIMPAC is a simulation package developed to simulate waiting line and scheduling problems, but it has had limited use outside of SDC where it was developed. It is considered to be a package in that it provides:

a. Modeling concepts for describing the model.

b. A language to generate computer code.

c. A program to move the simulation through time and record performance.

d. An output presentation capability.

The basic modeling concepts are predicated on the world view that a system is a complex of activities, activity performers, transient information records, and queues. Each activity is described by an algorithm which delineates the necessary conditions for the set up phase and the program steps descriptive of the execution phase. Performance of an activity is contingent upon the availability of both the activity performers and the required information. Each component of the model has an associated control block which maintains data such as addresses, current status, and time and utilization statistics.

The simulation model is comprised of symbolic expressions in combination with SIMPAC macro expressions. The basic SIMPAC program consists of sixty subroutines whose functions include:

a. Initialization and generation of exogenous events

b. Clock time keeping

c. Queue manipulation

d. Associative storage control

e. Data recording

f. Data analysis and output presentation

g. Distribution sampling

h. Mathematical functions

SIMPAC is atypical in that it utilizes a fixed increment time mechanism. The time increment may be varied or made a function of the state of the model. A cycling routine continually executes the set of algorithms and during each cycle moves each component forward in time by the specified amount. The control block associated with each activity records the performance time to date and the performance time necessary for completion.

Reference

Michael R. Lachner, "Toward A General Simulation Capability," Proceedings of Western Joint Computer Conference, 1962, pp. 1-14.

SIMPLE - Simulation Program Language

1. Background Information

Authors: J. J. Donovan
M. M. Jones
J. W. Alsop

Originating Organization: Massachusetts Institute of Technology, Project MAC

Date of Release: Not released

Status: Experimental

Machine Implementation: MULTICS Time Sharing System (M.I.T.)
IBM 1130

2. Major Features

SIMPLE is an interactive on-line simulation system that incorporates facilities for graphical display. The language is imbedded in PL/I; and while some of the simulation features are integrated into the language, others are implemented as external routines. This organization allows for the inclusion in the language of both the transaction-oriented approach of GPSS and the activity-oriented approach of SIMULA.

The language provides extensive facilities for the user to interact with the system, to describe and display the topology of the system in hierarchical levels of detail, to display time-series plots and frequency distributions, and to validate the system in real time.

These features are representative of new trends in the design of simulation languages.

3. Applications

The applicability of the real time feature of the system to the simulation of computer systems on the micro logic level is dependent upon the ability of the host computer to keep pace with real time.

Reference

J. J. Donovan, J. W. Alsop and M. M. Jones, "A Graphical Facility for an Interactive Simulation System," Proceedings of IFIPS Congress, 1968, pp. 593-596.

SIMSCRIPT

1. Background Information

Authors/Originating Organization/Date of Release:

SIMSCRIPT I

H. M. Markowitz, B. Hausner and H. W. Karr/
The RAND Corporation/1963

SIMSCRIPT 1.5

H. W. Karr, H. Kleine and H. M. Markowitz/
California Analysis Center, Inc./1966

SIMSCRIPT II

H. M. Markowitz, B. Hausner, P. J. Kiviat and R. Villanueva/
The RAND Corporation/1968

SIMSCRIPT II Plus

P. J. Kiviat/Simulation Associates, Inc./1970

Machine Implementation:

SIMSCRIPT I

IBM 7040/44, 7090/94

SIMSCRIPT 1.5

IBM 7040/44, 7090/94
CDC 3600/3800/6400/6600/6800
Philco 210/211/212
UNIVAC 490/1107/1108
GE 625/635 (implemented by Digitek, Inc.)
CDC 6400/6500/6600 (labeled SIMSCRIPT 2.0)

SIMSCRIPT II

IBM 360 (implemented by RAND and CAC)
CDC 6400/6500/6600 (implemented by CAC)

SIMSCRIPT II PLUS

IBM 360
RCA Spectra 70, DOS

2. Major Features

SIMSCRIPT is a versatile, powerful, and efficient simulation language which, through several improvements since its inception in 1963, has kept pace with (if not paced) the development of special languages for use in simulation. Its static structure is based upon entities which may be permanent (lasting) or temporary (created and/ or destroyed during the simulation). The properties of entities are denoted by associated attributes. Entities may be grouped together into sets. The definition of an entity results in a class of objects, each possessing the same list of attributes and the same list of potential set memberships. SIMSCRIPT is basically event-oriented with discrete time intervals as in GPSS, although the programmer can perform scheduling directly in SIMSCRIPT but not in GPSS. The changes in state represented by events are accomplished through the direct coding of event routines which alter data interrelationships, cause and cancel events, and transfer control either to the scheduler or to another event routine. Events occur in zero simulated time so that activities must be represented as a sequence of events.

SIMSCRIPT I programs are translated into FORTRAN and then compiled, but all succeeding versions are compiled directly into machine code. Hence, SIMSCRIPT I accepts coding in FORTRAN, and other versions permit machine language coding; but use of FORTRAN or machine language limits the transferability of SIMSCRIPT programs between compilers of different versions.

SIMSCRIPT 1.5 provides a capability almost identical to its predecessor with rather minor syntax changes and direct compilation as noted above. SIMSCRIPT II represents a major departure from previous versions since it is designed and produced as a general programming language. It can be used for such widely varying purposes as business data processing, scientific programming, list processing, and through special purpose features, simulation. The language structure and syntax employed differ significantly from its predecessors, and a translator planned to provide forward compatibility (SIMSIFT) was never produced. Programs written in SIMSCRIPT I or 1.5, therefore, cannot be run with SIMSCRIPT II. The newest version, SIMSCRIPT II Plus, has a more flexible and compact data structure and better diagnostics, including some error correction (described as a "large percentage of user syntax errors"[1]) which is used to force execution of every complete program. It is also faster at program assembly than SIMSCRIPT II. Execution time is claimed to be 20% faster than earlier versions of SIMSCRIPT.[2] The comments which follow apply to SIMSCRIPT II, though many apply to all versions.

The language is described in terms of five levels which are identified as follows:[3]

Level 1: A simple teaching language designed to introduce programming concepts to nonprogrammers.

Level 2: A language roughly comparable in power with FORTRAN but departing greatly from it in specific features.

Level 3: A language roughly comparable in power to ALGOL or PL/I, but again with many specific differences.

Level 4: That part of SIMSCRIPT II that contains the entity - attribute - set features of SIMSCRIPT. These features have been updated and augmented to provide a more powerful list - processing capability. This level also contains a number of new data types and programming features.

Level 5: The simulation-oriented part of SIMSCRIPT II containing statements for time advance, event-processing, generation of statistical variates, and accumulation and analysis of simulation-generated data.

SIMSCRIPT II has an extended timing routine which permits activities as well as events to be represented. Output reports are produced by a report generator which utilizes a user-coded sample report for input.

---

[1]P. J. Kiviat, promotional pamphlet from Simulation Associates, Inc.

[2]Ibid.

[3]P. J. Kiviat, et. al., <u>The SIMSCRIPT II Programming Language</u>, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1968, p. v.

3. Applications

SIMSCRIPT applications are broad and extensive, including analysis of the operations of railroads, canals, warehouses, elevator banks, production lines, the construction of computer systems, and procedures for aircraft maintenance and supply. The language has been utilized in producing a model called Extendable Computer System Simulator (ECSS) referenced elsewhere in this document.

References

Control Data 6400/6500/660 Computer Systems SIMSCRIPT Reference Manual, Control Data Corporation, Pub. No. 60178300, Palo Alto, California, 1968.

M. A. Geisler and A. M. Markowitz, "A Brief Review of SIMSCRIPT as a Simulating Technique," The RAND Corporation, RM-3778-PR, 1963.

G. Gordon, System Simulation, Prentice Hall, Inc., Englewood Cliffs, New Jersey, pp. 239-275.

H. W. Karr, H. Kleine and H. M. Markowitz, "SIMSCRIPT 1.5," California Analysis Center, Inc., Santa Monica, California, 1966.

P. J. Kiviat, "Introduction to the SIMSCRIPT II Programming Language," in the Digest of the Second Conference on Applications of Simulation, December 2-4, 1968, New York City.

P. J. Kiviat, H. J. Shukiar, J. B. Urman and R. Villanueva, "The SIMSCRIPT II Programming Language: IBM 360 Implementation," The RAND Corporation, RM-5777-PR, 1969.

P. J. Kiviat, R. Villanueva and H. M. Markowitz, The SIMSCRIPT II Programming Language, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1968.

H. M. Markowitz, B. Hausner and H. W. Karr, SIMSCRIPT-A Simulation Programming Language, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1963.

H. M. Markowitz, "Simulating with SIMSCRIPT," Management Science, Vol. XII, No. 10, June 1966, pp. 396-409.

SIMTRAN

1. Background Information

   Author: C. R. Dowling

   Originating Organization: International Business Machines Corporation

   Date of Release: 1965

   Status: Obsolete

   Machine Implementation: IBM 7030

2. Major Features

   A modified version of SIMSCRIPT written especially for the IBM 7030.

Reference

D. M. Braddock, C. R. Dowling, K. Rochelson, "SIMTRAN - A Simulation Programming System for the IBM 7030," IBM SDD, Poughkeepsie, N. Y., July 1965.

SIMULA - Simulation Language

1. Background Information

   Authors: Ole-Johan Dahl
   Kristen Nygaard

   Originating Organization: Norwegian Computing Center, Oslo, Norway, under contract with The Sperry Rand Corporation, Univac Division

   Date of Release: 1965

   Status: Current

   Machine Implementation: Univac 1107

2. Major Features

SIMULA, an ALGOL based, process oriented simulation language, was designed to provide facilities for the formal description and simulation of discrete event systems. It includes ALGOL 60 as a subset and provides the mechanisms for extensive list processing, quasi-parallel processing, sampling from distribution functions, and accumulating system time integrals and histograms.

A process-oriented language describes a discrete event system as a collection of modules called processes which specify the behavior of the system components and may be conceptualized as operating in parallel. The actions and interactions of the processes completely define the behavior of the system as the simulation progresses through time.

For the sake of clarity, some SIMULA terminology is defined:

a. Activity declaration -- the description of a process in terms of data declarations and operation rules (i.e., sequences of state changes).

b. Activity -- a class of processes described by the same declaration.

c. Process -- one dynamic instance of an activity declaration.

d. Event -- active state of a process. The associated system time remains constant during the execution of an event.

e. Interaction point -- a point within an activity declaration at which control may pass to another process.

f. Reactivation point -- a point within a process where processing is to resume during the next active state.

Each process is, therefore, one instance of an activity (i.e., it has a unique set of data values and maintains its own identity as it executes the behavior pattern). A process may span one or more interaction points at which time it passes from an active to a non-active state and control is surrendered to another process. The sequence of operations in the system may be defined by the active states of its component processes. This mode of operation in conjunction with the appropriate time control mechanism has been termed quasi-parallel.

The dynamic occurrence of events in a SIMULA model is controlled via scheduling and sequencing statements and by a scheduling entity called an event notice. An event notice records the scheduled time of the next active phase (i.e., event) of a referenced process. The set of event notices is termed the sequencing set and is arranged in order by the value of the time references. The first member is representative of the current system time and the currently active process. At the completion of its active phase, the event notice is deleted and control is transferred to the reactivation point of the process referenced by the next sequential event notice.

At any point in time a process may exist in one of four possible states:

a. Active -- Process is currently active and via sequencing statements may alter the state of any process including its own.

b. Suspended -- A suspended process has an associated event notice and a reactivation point.

c. Passive -- A passive process has a reactivation point, but no associated event notice.

d. Terminated -- No further events may occur in association with this process; it consequently has neither an event notice nor a reactivation point.

References

Ole-Johan Dahl and Kristen Nygaard, "SIMULA - An ALGOL-Based Simulation Language," Communications of the ACM, Volume 9, No. 9, September 1966.

John McNeley, "Simulation Languages," Simulation, August 1967, pp. 95-98.

SIMULA 67

1. Background Information

Authors: O. J. Dahl
B. Myhrhaug
K. Nygaard

Originating Organization: Norwegian Computing Center, Oslo, Norway

Date of Release: Unknown

Status: Unknown

Machine Implementation: Unknown

2. Major Features

SIMULA 67 is a general purpose programming language with a built-in simulation capability. Conceived to improve the facilities of SIMULA I as a simulation language, it evolved into a general purpose programming language and finally into a language for generating special application languages. It contains ALGOL 60, with minor revisions, as a subset.

The "object" is the central concept in SIMULA 67. A process class declaration (called an activity declaration in SIMULA I) defines a system behavior pattern in terms of the associated data and actions. An object is a self-contained program having its own local data and executing actions as defined by its process class declaration. Objects conforming to the same behavior pattern are considered members of the same class. Objects are manipulated and related to each other by list processing facilities. As in SIMULA I the simulation may be described as a sequence of the active phases of the constituent objects.

SIMULA 67 may be used as a language for generating problem-oriented languages by defining a class containing the concepts necessary for the special application area and adding this class as a prefix to the program.

SIMULA 67 contains a class called "SIMULATION" as part of the language. SIMULATION, by defining a time axis and a list structure and by organizing the active phases of an object through the time axis, transforms SIMULA 67 from a general purpose programming language into a simulation language. This same technique may be utilized to generate other problem-oriented languages.

Reference

O. J. Dahl, B. Myhrhaug, and K. Nygaard, "Some Features of the SIMULA 67 Language," Second Conference on Applications of Simulation, New York, 1968, pp. 29-31.

SOL - Simulation Oriented Language

1. Background Information

Authors: D. E. Knuth
J. L. McNeley

Originating Organization: Burroughs Corporation
Case Institute of Technology

Date of Release: 1964

Status: Not available for distribution. Currently being used for research within the Burroughs Corporation.

Machine Implementation: Burroughs B5000/5500
Univac 1107

2. Major Features

SOL is a general purpose algorithmic language designed for the simulation of complex systems. It incorporates the essential characteristics of GPSS expressed in an ALGOL-like structure. Additionally, SOL provides the capability of:

a. parallel computation

b. arithmetic expressions incorporating random elements

c. automatic generation of statistics

The descriptive unit in SOL is called a "process." A model is conceptualized as a program consisting of a number of individual processes which may be executed in parallel. Provision is made for both intra- and inter-process parallelism. Objects within a process have been defined as transactions, and each has an associated set of local variables unique to the process. Interaction between processes is achieved via global entities and control statements. Global entities are of three major types: variables, facilities, and stores. Global variables may be accessed for reference or change by any transaction in the system. A facility is a global element which is controlled by a single transaction at any point in time. Facilities are seized on a priority basis through the concept of the "control strength" of the requesting transaction. Interrupts occur and may be nested to any depth. Stores are space-shared global elements which are assigned on a first-come, first-served basis with no provision for preemption.

Control is transferred from process to process as delays are encountered by transactions waiting for time to pass, for logical conditions to be satisfied, for a facility, or for space within a store. Time is advanced in variable time increments, meaning that after all transactions have been processed at the current time, time is advanced to the earliest completion time for a wait statement.

Operationally, a SOL program is translated by the compiler into an interpretive pseudo-code which is executed by the interpreter to produce the statistical results of the simulation.

3. Applications

Included in the first reference is a detailed model for a multiple online console system. Included is the complete SOL program, an explanation of each statement, and some representative output reports.

It is the opinion of the authors of that article that SOL is especially advantageous for simulating computer systems since the language lends itself readily to the description of computer workloads.

References

D. E. Knuth and J. L. McNeley, "SOL - A Symbolic Language for General Purpose Systems Simulation," IEEE Transactions on Electronic Computers - 13, August 1964, pp. 401-408.

D. E. Knuth and J. L. McNeley, "A Formal Definition of SOL," IEEE Transactions on Electronic Computers - 13, August 1964, pp. 409-414.

SOLPASS - Simulation Oriented Language Programming and Simulation System

1. Background Information

Authors: Donald J. Miller
Harry C. Page

Originating Organizations: International Computer Sciences, Inc.
USAECOM, Fort Monmouth

Date of Release: 1968

Status: Current

Machine Implementation: Burroughs B5500 Remote Access System

2. Major Features

SOLPASS is an extension of SOL developed under government contract to simulate large-scale communications networks. It is, however, a general purpose simulation language of wide-range applicability for both discrete and continuous simulations.

The language was implemented in ALGOL and a complete ALGOL capability is provided as a subset of the language. Since SOLPASS operates with virtual memory, the number of transactions simultaneously in process is limited by disk memory and not core storage. The use of core swapping and overlay techniques provide the capability for a virtually unlimited number of queueing variables and simultaneous queues. A special language construct called a "trunk" was added to the basic language to facilitate simulation of large communications trunk lines. A trunk, like a store, is a space-shared variable with a discrete capacity but possesses the characteristic of a facility in that it is pre-emptable on a priority basis.

The SOLPASS system consists of the following basic components:

a. A compiler which generates an ALGOL program from the SOL source program.

b. An ALGOL compiler which generates object code.

c. A simulation control module which includes the scheduling algorithms.

d. A statistical analysis program.

During the execution phase an external event log file is generated which records the occurrence of each event and the pertinent data relative to each queueing variable. These data are subject to post-simulation statistical analysis. A variety of options for graphical output aid in the presentation of the simulation results.

The efficient utilization of the system is enhanced by the incorporation of extensive debugging and error analysis features. In addition, a unique capability exists to run a model until steady state is reached, perform a breakout, and then run a series of restarts imposing various experimental conditions on the steady state system.

3. Applications

Most of the experience gained with the system is in the area of traffic and systems simulations. The authors state that SOLPASS seems to be ideally suited to traffic studies but also performs very well in systems simulations applications. The following systems are among those types of systems simulated:

a. Control Systems

b. Computer Systems

c. Communications Systems

d. Transmission Systems

Reference

J. Armstrong, H. Ulfers, D. J. Miller and H. C. Page, "SOLPASS - A Simulation Oriented Language Programming and Simulation System," Proceedings, Third Conference on Applications of Simulations, Los Angeles, 1969, pp. 24-37.

SPL - Simulation Programming Language

1. Background Information

Author: Luigi Petrone

Originating Organization: Olivetti General Electric, Milan, Italy

Date of Release: Not Released

Status: Experimental

Machine Implementation: Potentially available for any machine with PL/I compiler

2. Major Features

SPL was designed to be translated into PL/I by a preprocessor written in PL/I and is, therefore, said to be completely defined on PL/I. Two defining methods are presented. Implementation A, discussed in depth, is based primarily on the tasking concept of PL/I. Implementation B, discussed in cursory fashion, is based on the use of programmed allocation and, in particular, on the based variables concept.

The SPL language is patterned principally on the concepts of SIMULA and to a lesser degree on those of SOL. One purpose of the language was an attempt to construct the synchronizing mechanism of SIMULA within the parallel programming capability of PL/I, and thus to make quasi-parallel processing a particular case of parallel processing.

An evaluation of SPL language features must be predicated on an in-depth knowledge of both PL/I and SIMULA and will not be attempted here.

Reference

Luigi Petroni, "On A Simulation Language Completely Defined Onto The Programming Language PL/I," in J. N. Buxton ed., Simulation Programming Languages, North Holland, Amsterdam, 1968, pp.305-318.

CSS - Computer System Simulator

1. Background Information

Author: Unknown

Originating Organization: International Business Machines Corp.

Date of Release: Not released

Status: IBM proprietary

Machine Implementations:

IBM 7040/7044

IBM 7090/7094

IBM 360

2. Major Features

The Computer System Simulator, CSS, provides a language and a structure specifically designed for modeling computer systems to evaluate their performance. The language adapts many of the techniques of GPSS and traces its evolution to the inability of GPSS to accomplish this specialized application efficiently.

CSS provides the capability to model a large variety of computer systems in varying levels of detail. The basic input consists of the following components:

a. statement of system configuration

b. description of operating programs

c. description of the system environment

The system configuration identifies the system components and specifies such associated data as the size of core storage, data transfer rates for I/O devices, and I/O device/channel connections. The operational characteristics of some IBM hardware are included in the model.

The operating programs include the users' applications programs and the system control programs which perform the functions of task scheduling, interrupt handling, etc. Programs of each type must be defined in terms of logic, timing information, and statistical requirements. This implies that the user must have a complete and comprehensive understanding of the monitor routines of the object operating system.

The system environment is implicit in both of the previous specifications. Input message rates, job streams, and task generation are explicitly defined in the environmental section.

The simulator automatically provides features to update the clock, maintain lists of future events, generate random numbers and perform statistical functions.

From the input specifications the CSS program assembles a model, generates the tasks, and operates the model in an interpretive fashion until the occurrence of a user-specified stopping condition.

The output report provides a summary of the run statistics and includes:

a. listing of the input specifications

b. utilization of all units of equipment

c. statistics on all system queues

d. statistics on usage of system resources

e. activity statistics (number of messages generated/terminal etc.)

The CSS is based on the concept that computer system operation may be represented by the interaction of three basic simulator elements: equipment, transient data units, and programs. To this end the CSS entities consist of:

a. equipment entities - processors, channels, control units, tapes, disks, terminals, etc.

b. transient entities - messages, tasks, events, etc.

c. programs - applications, control, and environment

d. modeling entities - clock, queues, polling lists, etc.

Associated with each entity is a set of attributes describing its characteristics. The state of the system at any point in time is defined by the attribute values of the system entities.

The CSS instruction set consists of 40 instructions which resemble a high-level macro language and may be classified into eight categories of related usage:

a. processing control

b. attribute access

c. entity control

d. branching control

e. line control

f. I/O control

g. run control

h. library linkage

The user may build his own set of macroinstructions from the basic set provided, thereby increasing the scope of the language.

3. Applications

The CSS does not optimize the system design, but it does provide a means to determine the extent to which system performance meets desired criteria. The CSS was designed principally to model computer installations having a real time capability and is most advantageous when analyzing large systems where the interaction of processing and I/O is to be studied.

Examples of configurations that may be modeled are:

a. single-processor card or tape systems

b. disk systems

c. multiprocessor systems

d. teleprocessing systems

Representative types of operation that may be modeled are:

a. sequential

b. multiprogram

c. real-time

d. time-sharing

The study by P. H. Seaman and R. S. Soucy was undertaken in an attempt to provide a general submodel of the SYSTEM/360 operating system. The provision of such a submodel assures the user of the correct logic and timings of the supervisor and data management routines. Additionally, it provides OS/360 developers with a vehicle to test proposed changes:

Reference

P. H. Seaman and R. C. Soucy, "Simulating Operating Systems," IBM System Journal, No. 4, 1969, pp. 264-279.

ECSS - An Extendable Computer System Simulator

1. Background Information

Author: N. R. Nielsen

Originating Organization: The research for ECSS was sponsored by NASA under a RAND Corporation contract. Present development is being sponsored by Project RAND.

Date of Release: Not yet released

Status: Under development

Machine Implementation: ECSS is presently on an IBM 360. A SIMSCRIPT II compiler is required in order to implement ECSS. Core requirements vary depending on the specific simulation being conducted, but about 100K bytes seem to be the norm.

2. Major Features

ECSS is an extension of SIMSCRIPT II and combines the full SIMSCRIPT II capability with more specific capabilities for modeling computer systems. ECSS language statements are written in a fairly free form, translated into SIMSCRIPT II commands, and subsequently compiled by SIMSCRIPT II. Statements can be formed by combining ECSS and SIMSCRIPT II type statements, and the user can augment commands. Flow-oriented problems, not suitably handled by SIMSCRIPT itself, can be handled by ECSS due to special commands built into ECSS.

A binary deck is produced as an output when an ECSS simulation is run. This deck can be used in subsequent runs to avoid the translation process.

ECSS appears to be a good combination between a proven general purpose simulation language (SIMSCRIPT II) and an easy to use flexible command structure for modeling computer systems. Its present drawbacks include a weak report generation (depends on SIMSCRIPT II) and weak software definition capability (again requiring SIMSCRIPT II coding). There is presently no user documentation available and experience with the prototype system is limited.

3. Application

Since ECSS is still in development, there is little data available on application experience. Several small models have been written and tested at RAND, but full field testing (due early in 1971) must be completed before ECSS potential in the application area can be appraised.

References

D. W. Kosy, "Experience with the Extendable Computer System Simulator," Proceedings of the Fourth Conference on Applications of Simulation, New York City, December 1970.

N. R. Nielsen, "ECSS: An Extendable Computer System Simulator," The RAND Corporation, RM-6132-NASA, February 1970.

PDL - Program Description Language

1. Background Information

   Author: G. K. Hutchinson

   Originating Organization: Texas Technological College, Lubbock, Texas

   Date of Release: 1968

   Status: Unknown

   Machine Implementation: Unknown

2. Major Features

The PDL language was designed to provide a capability for describing the programs to be processed in the simulation of multi-processor computer systems. A program is defined in terms of the logical relationships between its constituent routines and in terms of each routine's resource requirements. The routines are defined to be uniform in resource requirements and without parallelism.

Routines may be constrained until the necessary resources are available and assigned by the operating system and/or until required predecessor routines have been processed. Routine dependency may be expressed as a logical AND or OR.

Parallelism between programs is simulated by the simultaneous execution of routines and is unlimited, constrained only by the availability of resources; it is thus possible to simulate reentrant code.

The language set consists of 38 commands which describe the interdependency of the program routines and which control the logical flow of the program and the progression of time.

Hutchinson expresses the opinion that as multiprocessor operating systems become more comprehensive, users may be required to detail the logical relationships and resource requirements of the various routines in their programs and that command languages with features similar to PDL will be developed for this purpose.

Reference

G. K. Hutchinson, "Some Problems in the Simulation of Multiprocessor Computer Systems," in J. N. Buxton, ed., Simulation Programming Languages, North Holland, Amsterdam, 1968, pp. 305-324.

PLS - Product Line Simulator

1. Background Information

   Author: Computer Programming Laboratory

   Originating Organization: Hughes Aircraft Company

   Date of Release: Proprietary

   Status: Currently available through service contract

   Machine Implementation: IBM 360/50 with 256 K

2. Major Features

PLS is a computer systems simulation language and processor, somewhat like ECSS and CSS, that was developed by Hughes primarily for use in evaluating computer system designs. Since designs cannot be evaluated independent of their environment, PLS also provides for the description of operating systems and application programs.

The descriptive material on PLS available to date provides only a rather cursory look at the system elements modeled and the purposes toward which utilization might be directed. These elements and purposes appear roughly comparable to CSS, ECSS, and S3. No information concerning the structure and format of the language was given in the document reviewed, other than the statement that library routines are incorporated in the language.

Reference

Hughes Aircraft Company, Ground Systems Group, Simulation Models - A Tool for the Development of Real-Time Computer Systems, August 1970.

SEAL - Simulation Evaluation and Analysis Language

1. Background Information

Author: D. M. Braddock

Originating Organization: International Business Machines Corp.

Date of Release: Unknown

Status: Unknown

Machine Implementation: Unknown

2. Major Features

SEAL, an event-oriented language, is an extension of SIMSCRIPT 1.5 and includes PL/I-like data structures and improved list processing features. It does, however, lack the capability for automatic sampling from probability distribution functions.

References

D. M. Braddock and C. B. Dowling, "Simulation Evaluation and Analysis Language," IBM 360 D15, 1.005.

"Simulation Evaluation and Analysis Language (SEAL), System Reference Manual," IBM Corp., 17 January 1968.

SIMCOM - The Simulation Compiler

1. Background Information

Author: Thomas G. Sanborn

Originating Organization: Space Technology Laboratory, under purchase order from Thompson Ramo Wooldridge, Inc., in support of their contract to supply technical direction to the Automatic Data Processing Facility, U. S. Army Electronic Proving Ground.

Date of Release: 1959 (approximate)

Status: Unknown

Machine Implementation: IBM 709

2. Major Features

In the context of SIMCOM a computer simulation program is one which will allow one computer to assume the operational characteristics of another computer which is unavailable or may exist only in the design stage. SIMCOM was developed to assist in the preparation and modification of computer simulation programs. It is not itself a simulation program, but a compiler which accepts input statements written in a specialized simulation-oriented source language and generates instructions in the machine language of the host computer (i.e., SCAT for the IBM 709). SCAT is included as a subset of the language.

The statement is the fundamental unit of the SIMCOM language; two classes exist:

a. definition statements which define the components of the simulated computer

b. procedural statements which define the data manipulation or control functions which are associated with the execution of instructions within the simulated computer

A simulation program written in SIMCOM consists of the following components:

a. Machine definition - describes the static computer in terms of registers, memory, input, output, keys, and indicators.

b. Instruction interpretation - describes the computer in operation. This section is written in terms of procedural statements and describes the accessing of instructions from the storage of the simulated computer plus a description of the effects of the execution of each instruction.

c. Panel operation - describes the effect of operator intervention. This section includes an interrogation of the status of each console key and a description of computer response to activation of the key.

The output from SIMCOM consists of a translation of the source program into SCAT. This includes, in addition to a direct expansion of the procedural statements, the generation of an input/output supervisor and a storage management routine which allocates the storage of the simulated computer to the various 709 storage media.

There are language statements for keeping track of time and by specifying the execution times of instructions, it is possible to assess total elapsed time in the computer.

3. Application

Sanborn has expressed the opinion that SIMCOM provides the vehicle for describing various computers; but that beyond the timing of benchmark programs, it has no mechanism for computer evaluation.

Reference

Thomas G. Sanborn, "SIMCOM - The Simulation Compiler," Proceedings of the Eastern Joint Computer Conference, 1959, pp. 139-142.

SLANG - Simulation Language

1. Background Information

Author: L. A. Kalinichenko

Originating Organization: Ukranian Academy of Sciences, Institute of Cybernetics, Kiev, U.S.S.R.

Date of Release: Unknown

Status: Unknown

Machine Implementation: Unknown, possibly the BESM 6

2. Major Features

SLANG, a simulation-oriented experimental programming language, was designed to be used for the simulation of computer systems.

Computer systems are characterized by a great number of devices for storage, processing, and transfer of data; operation of which is overlapped. At the initial stage of system design, the following basic system characteristics must be defined:

a. system components and associated parameters

b. configuration of communications paths between system components

c. number of control levels of different processes

d. priority system for various data flows

e. specification of operational algorithms of the system control elements

Kalinichenko feels that problem-oriented languages are not suitable for the simulation of computer systems and that special simulation languages are necessary and should provide the following capabilities:

a. ability to use the language for a formalized system description

b. ability to model the generation of flows of non-uniform requests

c. ability to construct and process list structures; this requirement derives from the modeling of scheduling algorithms

d. ability to generate random variables from specified distributions and to provide convenient means for gathering statistics

After a comparative analysis of current simulation languages (GPSS II, GPSS III, SIMSCRIPT, SOL, SIMULA) and experimental programming of processes representative of computer systems, the author concluded that SOL provided the most capability. It has natural and brief notation; its special objects are similar to computer system components; and implementation is facilitated by the simplicity of its procedural part. It was, therefore, chosen as the basis for the experimental language, with the intention of correcting some of its inherent deficiencies.

In contrast to SOL, SLANG allows for interaction among processes by means of exchange transactions and further allows one transaction to reference the local variables of another. List processing techniques, similar to those of SIMULA, have been incorporated to improve the flexibility of control of transaction movement in the system. The SOL techniques for synchronization of concurrent processes were expanded. With a list processing capability, SLANG now has the capability to control event sequencing by two alternative methods and to compare their relative effectiveness.

3. Application

An example of simulation of a multi-terminal, time-sharing computer system is included in the reference to illustrate the language features.

Reference

L. A. Kalinichenko, "SLANG - Computer Description and Simulation-Oriented Experimental Programming Language," in J. N. Buxton, ed., Simulation Programming Languages, North Holland, Amsterdam, 1968, pp. 101-116.

COMPUTER SYSTEMS SIMULATION PACKAGES

BOSS - Burroughs Operational Systems Simulator

1. Background Information

Authors: A. J. Meyerhoff
P. F. Roth
J. P. Troy

Originating Organization: BOSS was originated in the Burroughs Defense Space and Special Systems Group of the Advanced Development Organization although it is presently being supported by the Modeling and Simulation Group of the DSSSG Marketing Support organization at Paoli, Pennsylvania.

Date of Release: 1968

Status: Current

Machine Implementation: BOSS is implemented on the Burroughs B5500 system. BOSS (implemented in ALGOL) is a Burroughs proprietary package and is not available for implementation on other than Burroughs computers at the present time. Specific information on peripheral requirements, core requirements, etc. for the B5500 implementation or implementation on other Burroughs computers is not available at this time.

2. Major Features

BOSS is similar in many respects to GPSS. It is a discrete-event, block-diagram oriented system. BOSS maintains a "next-event" file which is triggered by the changes in state of the system being modeled as opposed to triggering on fixed increments of time.

BOSS considers the world as composed of processes, tasks, and units. In this context a process defines the priority and sequence of tasks to be performed and controls the generation of process starts. A task is any discrete operation which consumes time and resources. The unit is considered as a resource element and is used in a pool concept by the various tasks to carry out a process.

BOSS generates several reports including data on queue statistics, unit utilization, processor job mix statistics, memory utilization, etc..

The block symbology used for defining inputs is structured in a fairly simple way and appears to facilitate the definition job itself, but it may lack flexibility if used in widely dispersed application areas.

3. Applications

BOSS is designed for the simulation of computer systems, although it can be used for systems with similar characteristics in the sense that BOSS is discrete-event oriented.

References

A. J. Meyerhoff, P. F. Roth, and J. P. Troy, "BOSS, Applications Manual," July 19, 1968, Burroughs Corporation, Defense Space and Special Systems Group.

P. F. Roth, "The BOSS Simulator - An Introduction," Proceedings of the Fourth Conference on Applications of Simulation, New York City, December 1970.

CASE - Computer Aided Systems Evaluation

1. Background Information

   Author: Unknown

   Originating Organization: Computer Learning and Systems Corp.

   Date of Release: 1969

   Status: Current version CASE III

   Machine Implementations:

      IBM 360/50

      GE 635

      CDC 6000

      1108 Univac

2. Major Features

CASE is a statistical simulator capable of analyzing batch processing, multiprogramming, multiprocessing, time-sharing, and real-time computer systems. It is written in ASA FORTRAN with some event features similar to SIMSCRIPT. Like SCERT, CASE consists of a collection of timing algorithms used in conjunction with a library of hardware and software factors to simulate a computer system and evaluate its performance in processing a given workload.

Input definitions of the workload and system configuration are provided via coding forms and considerable flexibility in level of detail is allowed. The workload specification is in machine independent form and includes such items as general system type, file definitions, run sequences, and for each constituent run, its identification, language, frequency, I/O files, internal processing activity, and output reports. The processing activity is charged against files and may be specified in terms of higher level macros and subroutines or by explicitly delineating the operations. Frequency of occurrence may be expressed exactly or as a probability.

The system configuration to be analyzed is specified in terms of the hardware and operating system software. Information is provided relative to the operating system, CPU, channels, tapes, immediate access storage devices, and peripherals. Each unit is described by its model number, quantity, and physical connection.

File allocations, file blocking factors, and device/channel assignments either may be specified or can be optimized by the simulator. The CASE library is an input to a CASE simulation and provides the source of information relative to the technical characteristics and cost of the hardware and software system packages marketed by the major computer manufacturers. These data are used as parameters in a generalized model of a computer hardware/software configuration.

The CASE control program is comprised of four major subsystems. Each is described in terms of its major function:

a. Independent Processing Analyzer (IPA) simulates the system in batch mode operation. For each constituent run in the workload, a Run Description Report is prepared which summarizes data relative to the file requirements, processing activity, and output reports. This report is of value in the preparation of specifications. Additionally, summary statistics for system component utilization are generated over all the runs in the workload.

b. Concurrent Processing Analyzer (CPA) operates on inputs from the other analyzers in conjunction with any configuration changes, determines an optimum run schedule based on considerations of run dependencies and priorities, and simulates the workload in a multi-programming mode of operation. This simulation provides reports to document the run schedule, to provide a time history and activity analysis of the component jobs, and to document overall system performance and requirements.

c. The Real Time Analyzer (RTA) provides the algorithms and logic necessary to construct a model of a computer system operating in a real-time environment. The RTA simulates generation and flow of information through the system and provides reports which record response time, maximum queue lengths, and configuration utilization.

d. Time Sharing Analyzer (TSA) is a specialized package which analyzes complex time sharing systems. The capability exists to perform detailed analysis on systems employing such advanced techniques as virtual memory, partial execution, and page swapping.

The execution of CASE is an iterative process in which the analyst modifies the input and makes the configuration and design adjustments necessary to achieve an optimal system design for processing the specified workload. Design adjustments take the form of consolidating, simplifying, or eliminating certain files and runs. Configuration adjustments are of greater magnitude and involve the addition or deletion of channels, controllers, or memory or the specification of a different set of mass storage units, etc.

These decisions are made on the basis of detailed output reports which represent a distillation of the statistics gathered during the execution of the simulation. Reports are provided at the completion of each simulation iteration. In addition to the traditional outputs of an automatic timing routine, CASE supplies reports which aid the design process by providing the theoretical limits which can be achieved by sub-optimizing system components. After an optimal system configuration is achieved, reports of a higher level of abstraction are prepared for management which document the final solution.

3. Applications

Reference 2 delineates the basic features of CASE, SCERT, and SAM. Bairstow reports the user impression that CASE, being of more recent origin, is better designed than SCERT to model third generation features such as multiprogramming, real-time, and time-sharing systems. There was no user experience with multiprocessing systems. However, the author makes the point that CASE is able to measure hardware contentions in a single system and may be able to do so for a multiprocessor configuration.

References

"A Technical Description of the CASE System," Software Products Corporation.

Jeffrey N. Bairstow, "A Review of Systems Evaluation Packages," Computer Decisions, June 1970, p. 20.

William C. Thompson, "The Application of Simulation in Computer System Design and Optimization," Second Conference on the Applications of Simulation, New York, 1968, pp. 286-290.

COST - Computer Optimized System Tradeoffs

1. Background Information

Author: George Pan

Originating Organization: Interactive Sciences Corporation, now operated by System Architects

Date of Release: Not released

Status: Proprietary

Machine Implementation: GE-635
PDP-10

2. Major Features

COST is neither a language nor a model, but an overall methodology combining simulation with theoretical analysis for the purpose of analyzing computer systems. The simulations employed utilize one or more of a group of programs along with a library of system entities and their attributes. Its originators claim result accuracies of ± 10% without past history from users and without a benchmark profile, through utilizing the "simplest deterministic saturation analysis model."[1] If the subject for analysis is a presently functioning system and actual and consistent user profiles are available, then "one of the generic Monte Carlo models ... used for a particular COST run ... can (produce) accuracies ... on the order of ± 0.5%."[2] The major strengths of this approach, and its differences from other simulators such as SCERT, lie in the optimization of file design, data base design, and network design. Many of COST's applications have been in this area.

3. Applications

The capability summarized by the acronym COST has been applied in a large number of studies done for, among others, the Library of

[1]Interactive Sciences Corporation, Memorandum titled "Computer Systems - Analysis and Simulation," p. 9.

[2]*Ibid.*, p. 9.

Congress, the University of California, Harvard University, Dartmouth College, NASA, Office of Education of H.E.W., ONR, a number of private industrial firms including Honeywell EDP, GE Information Systems, IBM, and Control Data, and for the Decision Sciences Laboratory of ESD as well as MITRE Washington.

References

"A Brief Introduction to Interactive Sciences Corporation," Interactive Sciences Corporation, Braintree, Mass.

"Computer Systems - Analysis and Simulation," Interactive Sciences Corporation, Braintree, Mass.

Conversation with George Pan, now of System Architects, Randolph, Mass.

DPSS - Data Processing Systems Simulator

1. Background Information

Authors: Michael I. Youchah
Donald D. Rudie
Edward J. Johnson

Originating Organization: Systems Development Corporation

Date of Release: 1964

Status: Expanded DPSS - Model C

Machine Implementation: AN/FSQ - 32 - V
IBM 7090/7094

2. Major Features

The Data Processing System Simulator (DPSS) is a general purpose computer system simulation package designed to aid in the evaluation of the performance and response characteristics of a proposed computer system. The design being tested can be subjected to various workloads and data processing disciplines. Constructed by the Systems Development Corporation for use in analyzing the operating performance requirements for the Strategic Air Command Control System (Project 465L), the DPSS has been primarily applied in the area of real-time management information and command/control type systems.

The DPSS is an event-oriented simulator, written in JOVIAL, which employs higher order macros combined in a single logical sequence designed so as to permit the representation of a wide range of system configurations and processing algorithms.

The DPSS is used to model computer systems which may be conceptualized as consisting of input, processing, and output phases. The inputs consist of message or information units which are entered into the system either locally or via remote terminals. These units are queued and processing is initiated under control of the batching, priority, and interrupt rules in effect. The processing phase consists of retrieving from auxiliary storage the appropriate programs and environment and executing the specified sequences of instructions. During the terminal phase, the necessary data is aggregated, formatted, and displayed as required.

Input to a representative DPSS simulation would include the specification of message types, traffic rates, batching criteria, task processing times to load and operate (expressed as probability distributions with associated parameters), message-task and message-display relationships, task sequences, etc. Output consists of statistics related to the transmission history of each message in the system and to the state of the system during each replication of the cycle.

An expanded DPSS (Model C) has been developed which allows for the simulation of more sophisticated computer systems. Future developments are expected to be in the area of multi- and parallel processing.

3. Applications

The DPSS was used successfully in the design and development of 465L SACCS. Details of this application are included in the reference. Additionally, it proved useful in evaluating original concepts relating to the Space Surveillance Project and the New York State Identification and Intelligence System.

The authors of DPSS state that "it is a powerful tool in performing system feasibility studies, simulating the operation and performance of computer-based data processing systems, and in evaluating equipment and data processing discipline combinations as a function of system operational requirements."

Reference

M. I. Youchah, D. D. Rudie, and E. J. Johnson, "The Data Processing System Simulator (DPSS)," Proceedings AFIPS 1964 Fall Joint Computer Conference, Vol. 26, Part 1, pp. 251-276.

S3 - The System and Software Simulator

1. Background Information

Author: Leo J. Cohen

Originating Organization: CEIR, Inc., Under Contract from the United States Army Computer System Evaluation Command

Date of Release: 1968

Status: Only basic documentation available.

Machine Implementation: Written in FORTRAN IV; successfully executed on IBM 360/50/65/75, and Univac 1108

2. Major Features

S3 is a discrete event-oriented simulator designed specifically for modelling computer systems. Its operation is dynamic (i.e., a user defined system description results in a representative simulation such that the actual time to run the simulation depends upon the length of the time period being simulated). It utilizes five types of information: a hardware characteristics data base, a description of the operating system, configuration data for hardware and software, user program descriptions, and file organization information for application programs. S3 produces sets of statistics concerning utilization of hardware resources, effectiveness of the operating system, and performance measures for application programs.

The five types of input data are described in a defined specification language, and each type is substantially independent of the others. For example, the descriptions of particular units of hardware are independent of the way in which the hardware units are interconnected. The level of detail involved can be appreciated by viewing the table which follows.

Processing by S3 is done in four phases. In the first of these, the hardware/software specifications in the defined programming and job control "languages" are "assembled", i.e., they are reduced to compact description through a translation process. Next, this assembled code is used to generate a model of the system to be simulated. In the third phase, this model is used to produce a record of the simulated performance of the system over a period of time.

Table II

Sample S3 Specifications

ELEMENT	DESCRIPTORS
1. Hardware Performance	
Printer	Characters/Line, Lines/Second, Start/Stop Time, Skip Time
Tape Drive	Data Storage Unit (bit, byte, word), Inter-Record Gap, Start/Stop Time, Rewind Time, Successive I/O Reference Delays
Data Controllers	Data Unit Transmitted (bit, byte, word), Selector/Multiplexor, Simplex/Half/Full Duplex
CPU	Add/Multiply/Divide Time for Decimal/Fixed/Floating Cycle Rate, Average Instruction Time
Core Memory	Access Rate, Data Storage Units (bit, byte, word), Size
2. Hardware Configuration	Controllers/Channels to which each unit is connected; allowable CPU-memory connections; number and type of processors and networks of computers
3. Operating System	"Programming Language" to express O/S flowchart logic; core allocation and reassignment techniques; queue manipulation doctrines; control methodology; interrupts internal and external to particular CPUs; overlay and/or multiprogram structure
4. Application Programs	"Program Language" for representation of program flow, including program/subprogram structure, I/O commands, parallel processing

Table II (Concluded)

ELEMENT	DESCRIPTORS
5. Application File Structure	"Job Control Language" describes application program file structure and distribution among peripherals, amount of storage required for instructions/constants/data, frequency of program operation using fixed/stochastic interval, variability in storage requirements during processing, calls to external programs

Finally, this performance data is reduced through statistical analysis to performance reports.

These performance reports are composed of statistical data outlining the operation of each defined facility and each program in the system. Included are data concerning utilization and overhead time for devices, controllers, channels, CPUs and memory, a detailed summary of interrupts, memory repacking (if utilized), queue history, and a breakout of area memory utilization. Minimum, maximum and average turn-around and execution times are reported for application programs along with breakouts of time allocated for CPU, I/O, and queue delay. The minimum, maximum and average number of file references is reported for each program.

3. Applications

S3 has been utilized in designing multiprogramming control of a computer used for process control with off-line program background, for design of real-time airborne hardware/operating systems for multiprocessor configurations, and for performance evaluation of systems utilizing the Litton 3050M, the IBM 4Pi, and the Burroughs D84T computers.

References

Leo J. Cohen, System and Software Simulator, Volume I, CEIR, Inc., Washington, D.C., December 1968, Available from Clearinghouse for Federal Scientific and Technical Information.

Leo J. Cohen, "S3, the System and Software Simulator," Digest of the Second Conference on Applications of Simulation, December 2-4, 1968, New York, pp. 282-285.

SAM - Systems Analysis Machine

1. Background Information

   Author: Leo J. Cohen

   Originating Organization: Applied Data Research Inc.

   Date of Release: 1970

   Status: Current

   Machine Implementation: IBM 360/50

2. Major Features

SAM, System Analysis Machine, is a computer program written in FORTRAN designed for the discrete-step simulation of the operations of digital computer systems. The facilities allow for the modeling of the hardware, operating system, utility programs, and applications programs which comprise a computer system.

Unlike SCERT and CASE in which system data is specified via definition forms, SAM provides a dynamic programming language as the means for describing computer hardware/software characteristics.

A complete SAM model is comprised of the following basic elements:

a. The Hardware Module consists of SAM declarators which describe the operational characteristics and interconnections of the hardware components (example: I/O devices, channels, CPU's, controllers, etc.).

b. The Program Module describes the component software of the system. The declarators provide information relative to size, language, storage requirements, and I/O files. Additionally, the logic of each program as described by flow charts is encoded in SAM statements.

c. The File Module describes the characteristics of the files in terms of direction (input or output), organization, and format.

d. The Media Module describes the peripheral storage devices and delineates organization, physical properties, and file assignments.

e. The System Module correlates the Hardware, Program, File, and Media Modules and incorporates any user defined FORTRAN subroutines.

f. The Creation Module provides data relative to the characteristics of the jobs and the rate at which they will be generated and entered into the simulation.

The SAM System consists of five components; the functions of each will be briefly described.

a. The Model Library consists of pre-defined models of the hardware, operating systems, and utility packages of the major computer manufacturers. User defined models may be coded in the SAM language and entered into the library. During a simulation, macro calls will extract the appropriate models from the library.

b. The SAM Translator is comparable to a compiler in that it converts "source models" written in the SAM language into "object models" in the lower level code required by the Model Generator.

c. The Model Generator performs the functions of a linkage editor. Operating on the output from the Translator, the Generator produces a complete model from the component modules (i.e., hardware, program, file, media system, and creation).

d. The model produced by the Generator is executed by the Interpreter. Data relative to system performance, device utilization, and program efficiency are collected and formatted for input to the Data Analyzer and Reporter.

e. The Data Analyzer and Reporter generates the statistical results of the simulation. At the option of the analyst, a wide range of output reports describing system performance is available.

3. Applications

SAM addresses itself to a wide spectrum of problems. The referenced manual states that the Systems Analysis Machine is capable of modeling real-time, time-sharing, multiprocessor, and multi-computer systems.

To illustrate the features of the language, an example of a tele-processing system is included.

References

Jeffrey N. Bairstow, "A Review of System Evaluation Packages," Computer Design, July 1970.

"Systems Analysis Machine - Introductory Guide," Applied Data Research, Inc.

SCERT - Systems and Computers Evaluation and Review Technique
COMET - Computer Operated Machine Evaluation Technique

1. Background Information

Author: Fred C. Ihrer

Originating Organization: COMRESS, Inc., Rockville, Maryland

Date of Release: 1962

Status: Current version SCERT 50

Machine Implementations:	IBM 360 series
	UNIVAC 1108
	Spectra 70 series

2. Major Features

SCERT was the first of the commercially available systems evaluation packages. The Air Force presently owns a version of the simulator under the acronym COMET. Since COMET represents a basic simulation capability within the Air Force, it is treated in somewhat greater depth than the other packages described in this section.

It provides the capability to simulate the performance of a user's processing requirements against cost/performance models of selected computer hardware/software systems. Written in assembly language, it is essentially an algorithmic simulator incorporating discrete event simulation techniques only in the case of randomly occurring processing.

SCERT is an evolutionary tool and has been modified as necessary to keep pace with and accurately reflect advances in computer technology. Originally designed to simulate batch systems, it has been updated to the current version, SCERT 50, which is capable of simulating multiprogramming, multiprocessing and real-time systems.

SCERT may be run in either of two modes, review or evaluation. The evaluation mode includes optimizing features such as blocking of records, assignment of peripheral units to I/O channels, allocation of memory, and scheduling of a multiprogramming workload. In the review mode these parameters are specified by the user, and the simulator merely determines the execution timings and hardware utilization. The review mode of operation is useful in validating a base case configuration by comparing the simulation results with known throughput data.

The user may then change values of parameters and run in the evaluation mode, and SCERT will apply the appropriate optimizing algorithms. These, however, are limited to those previously noted, and it remains the responsibility of the systems analyst to study the output reports and initiate changes in equipment configuration and program design which have the potential of enhancing system performance.

The SCERT package consists of four major components:

1. Definition languages
2. Factor library
3. Simulation programs
4. Output reports

The definition languages define the applications systems and the hardware/software configurations to be analyzed. Processing requirements are delineated for each computer program or random event which comprise the projected workload. Each application program is defined in terms of priority, pre-requisites, frequency, I/O files, and internal processing requirements. These processing requirements may be specified in terms of basic computer activity (add, compare, move, etc.) or at a higher functional level (extract, validate, sort, etc.) depending upon the stage of program design. Random processes are modeled as a series of sequential sub-processes where each sub-process is defined as one computer run. Additionally, each file involved in the system processing is defined in terms of number of records, characters per record, number of alpha and numeric fields, media, etc.

The hardware/software configurations are specified by component model number and quantity. The hardware must be specified in exact detail including all adaptors, channels, and control units. The user may specify the allocation of peripheral devices to channels and the terminal interfaces in communications networks. The software packages which are to be used in conjunction with the hardware are also specified on input definition forms. Included are a specification of the operating system, compilers, sort and IOCS packages, etc.

Finally, the system environment definition includes such items as programmer experience profile, salaries, corporate cost of money, and system life expectancy.

The SCERT factor library is a technical data base which contains factors concerning the performance of commercially available computer hardware and software. The hardware factors include cost, environment, specification, and performance data relative to central processing units, peripheral components, control units, and special features. The software factors include capability, capacity, and overhead data relative to generalized software packages such as compilers, operating systems, IOCS, and sort routines.

Conceptually, SCERT simulation programs may be viewed as having five functional phases. Phase 1 accepts as input the systems, file and environment definitions, and builds a hardware-independent, mathematical model of the processing requirements of each computer run and real-time event in the system. These models are verified and output diagnostics generated as necessary.

Phase 2 accepts as input the definitions of the hardware components and software packages which comprise the configuration to be analyzed. Appropriate factors are retrieved from the library and mathematical models are built which represent the hardware configuration, the software packages, and the effect of the integration of hardware and software. The Phase 1 processing models are validated for compatibility with the Phase 2 models.

Phase 3 of SCERT consists of the execution of the pre-simulation algorithms. The processing requirements developed in Phase 1 are adjusted to conform to the capability of the hardware/software complex specified in Phase 2. For each program or random event in the applications workload the following data are computed:

1. Internal processing time
2. Memory requirements
3. Assignment of files to peripheral devices and channels
4. Structuring of the files to the hardware
5. Throughput timing and memory requirements for all I/O functions
6. Timing of generalized software packages
7. Pre- and post-run timing

The actual simulation is executed in Phase 4 via three distinct stages. Stage 1 is considered the throughput simulation. SCERT explodes each program run and random event into its maximum number of iterations, simulates the flow of each through the specific computer configuration, and derives net throughput timing. These data are based on the composite representation of I/O and internal timing generated by the pre-simulation algorithms, and all available simultaneity is accounted for. The resultant timing reflects system operation in a sequential batch mode (i.e., a non-multiprogrammed environment).

Stage 2 consists of the event-oriented simulation and is executed only if the defined processing includes real-time events. Probabalistic models of events occurring during the total time frame are developed; and the flow of these models through the computer system is analyzed to determine the load imposed on potential queue points (CPU, mass storage, peripherals), the average and maximum delay times for messages, and the effect of real-time interrupts on background batch jobs.

Stage 3 consists of the time-oriented simulation; it is entered whenever the system specifications include a multiprogramming or multiprocessing capability. Employing critical path techniques, SCERT determines the degree of concurrency available and derives a schedule of tasks based on considerations of priorities, processing requirements, and pre-requisites.

In summary, Stages 1 and 2 determine the net elapsed time (vertical utilization) and capacity requirements (horizontal utilization) for each scheduled run and real-time event. Stage 3 generates time stream records of processing requests which are sorted into sequence by priority within a time zone. These time stream records are passed against a model representing the total capacity of the computer system in terms of memory available, total peripherals, internal processing capacity, channels available, etc. Each record is examined, requirements are matched to available capacity, and a schedule of tasks that should be run in parallel to achieve maximum throughput is determined.

The last functional component of SCERT consists of the output report generator. A full complement of fourteen reports of varying levels of detail is available for management analysis. Each of these may be selectively requested as dictated by the objectives of the simulation. In summary, the output reports are as follows:

- System Identification - for each program run and real-time event this report outlines the input file data, internal functional activities, and output file data.

- Configuration Identification and Cost

    Computer Complement Report - identifies the configuration being simulated and includes certain basic cost and environment data.
    Cost Summary Report - the lease, purchase and maintenance costs of the hardware are presented, based on expected utilization.

- Sequential Processing Projections

    Central Processor Utilization - projects for each run the processing time, set-up time, and memory requirements. Additionally, a summary report is generated which prorates these projections into daily, weekly and monthly average utilization.
    Computer Capabilities Report - tabulates for each run the hardware devices utilization.

Detail Systems Analysis Report - a three part report produced for each scheduled run or real-time event simulated.

Part 1 - analyzes the I/O requirements of the run in terms of device assignment, file blocking, buffer space, throughput timing, etc.

Part 2 - derives the internal processing time, program steps, and memory attributable to arithmetic operations, decision and control, data handling, and I/O control.

Part 3 - analyzes pre- and post-run overhead functions.

- Random Real-Time Projections

Event Processing Time - analyzes each real-time event in terms of its unique input, output, and processor time independent of any possible queueing delays.

Hardware Utilization - analyzes all potential queue points in terms of utilization and derives expected and worst case queue lengths.

System Response Data - the expected response of each event through the central computer complex and through the communications network is presented for the 99th, 95th and 50th percentile probability levels.

Memory Requirements - tabulates for each random event the memory requirements for program areas and data areas. Also presented are the expected and worst case background memory requirements.

- Multiprogramming/Multiprocessing Projections

Detailed Multiprogramming Schedule - provides a snapshot of the computer system at every point in simulated time when a program starts, terminates or changes state.

Multiprogramming Throughput Summary - tabulates the complete time history for each scheduled run and provides data for throughput analysis.

Multiprogramming Utilization Summary - presents for each time zone a measure of system utilization in terms of memory and processor capacity.

- Implementation Projections

Programming Requirements Report - breaks down each program run into the number of instructions to be programmed by the user and supplied by utility routines. Additionally a projection is made of the programming effort in man-months.

Application Summary Report - presents a summary of the central processor utilization and programming effort for each unique application area in the workload.

References

Jeffrey N. Bairstow, "A Review of Systems Evaluation Packages", Computer Decisions, June 1970, p. 20.

"Data Processing Planning Via Simulation", EDP Analyzer, Vol. 6, No. 4, April 1968.

Donald J. Herman, "SCERT: A Computer Evaluation Tool", Datamation, February 1967, p. 26-28.

Donald J. Herman and Fred C. Ihrer, "The Use of a Computer to Evaluate Computers" Proceedings - Spring Joint Computer Conference, 1964, p. 383- 395.

Allan J. Pomerantz, "Predict Your System's Fortune: Use Simulations Crystal Ball", Computer Decisions, June 1970, p. 16-19.

"A Technical Description of SCERT", COMRESS, Inc.

SECTION IV

BIBLIOGRAPHY

The bibliography has been automated via the KWIC INDEXING[1] system which reads reference data and permutes the titles on a complete or selective word basis. It is possible to annotate the titles by keyword descriptors, and permutation on the annotators is an additional option. Where appropriate the references have been so annoted to indicate the subject language(s). Preliminary documentation has been collected relevant to computer systems simulation; these references have been annoted by language/computer system. The entire bibliography, organized by author, is presented as Appendix A; the bibliography indexed on the languages delineated in Section III is presented as Appendix B.

Several of the references are of an expository nature and deal with the basic similarities and differences that exist between simulation languages. Table II identifies these reference sources and delineates the languages so compared.

---

[1]OS/360 QUIC (KWIC INDEXING), IBM Corporation, Contributed Program Library, 360D-06.7.022.

Table III

Comparison of Simulation Languages

LANGUAGE	BIBLIOGRAPHY REFERENCE				
	D0010	K0130	T0010	T0030	T0033
CLP			X		
CSL	X	X	X	X	X
DYNAMO		X			
ESP				X	
GASP			X		
GPSS	X	X	X	X	
GSP				X	X
HSL			X		
MONTECODE				X	X
RSP					X
SIMON				X	
SIMPAC		X		X	
SIMSCRIPT	X	X	X	X	X
SIMULA	X			X	
SOL	X		X		

D0010 - O. J. Dahl, "Discrete Event Simulation Languages"
K0130 - H. S. Krasnow, R. A. Merikallio, "The Past, Present and Future of General Simulation Languages"
T0010 - D. Tiechroew, J. F. Lubin, "Computer Simulation - Discussion of the Techniques and Comparison of Languages"
T0030 - K. D. Tocher, "Review of Simulation Languages"
T0033 - K. D. Tocher, A. P. Amiry, "New Development in Simulation"

Complete references may be found in the computerized bibliography listing.

****

APPENDIX A

BIBLIOGRAPHY ON SIMULATION

```
A0005 ACM/AIIE/IEEE/SHARE/SCI/TIMS (SPONSOR)
      FOURTH CONFERENCE ON APPLICATIONS OF SIMULATION
      NEW YORK, NEW YORK (DEC.1970)
      APPLICATIONS


A0010 ANDERSON, H A
      SIMULATION OF THE TIME-VARYING LOAD ON FUTURE REMOTE-ACCESS
      IMMEDIATE-RESPONSE COMPUTER SYSTEMS


A0020 APPLIED DATA RESEARCH, INC.
      SYSTEM ANALYSIS MACHINE
      APPLIED DATA RESEARCH, INC. (APRIL 1970)
      SAM


A0030 ARMSTRONG, J        ULFERS, H
      MILLER, D J         PAGE, H C
      SOLPASS, A SIMULATION ORIENTED LANGUAGE PROGRAMMING AND
      SIMULATION SYSTEM
      PROC. THIRD CONF. ON APPLICATIONS OF SIMULATION, LOS ANGELES
      , 1969, P.24-27 (ACM/AIIE/IEEE/SHARE/SCI/TIMS)
      SOLPASS


B0003 BAIRSTOW, J N
      A REVIEW OF SYSTEMS EVALUATION PACKAGES
      COMPUTER DECISIONS (JUNE 1970), P.20
      CASE   SAM   SCERT


B0010 BLUNDEN, G P        KRASNOW, H S
      THE PROCESS CONCEPT AS A BASIS FOR SIMULATION MODELING
      SIMULATION (AUG.1967), P.89-93


B0030 BRENNAN, R D        LINEBARGER, R N
      A SURVEY OF DIGITAL SIMULATION
      SIMULATION (DEC.1964), P.22-36
      CONTINUOUS


B0040 BUXTON, J N                                                    **
      SIMULATION PROGRAMMING LANGUAGES
      PROC. OF IFIP WORKING CONF. ON SIMULATION PROGRAMMING
      LANGUAGES, NORTH-HOLLAND PUBLISHING CO.-AMSTERDAM (1968)


B0050 BUXTON, J N
      WRITING SIMULATIONS IN CSL
      COMPUTER JOURNAL (AUG.1966), P.137-143
      CSL
```

B0060 BUXTON, J N   LASKI, J G
CONTROL AND SIMULATION LANGUAGE
COMPUTER JOURNAL 5,3 (OCT.1962), P.194-200
CSL

C0010 CALINGAERT, P
SYSTEM PERFORMANCE EVALUATION: SURVEY AND APPRAISAL
COMMUNICATIONS OF THE ACM (JAN.1967), VOL.10, NO.1, P.12-18
APPLICATIONS

C0020 CANNING, R G
DATA PROCESSING PLANNING VIA SIMULATION
EDP ANALYZER (APRIL,1968) VOL.6, NO.4

C0025 CHENG, P S
TRACE-DRIVEN SYSTEM MODELING
IBM SYST J 8,4 (1969), P.280-289

C0030 CLANCY, J J   FINEBERG, M S
DIGITAL SIMULATION LANGUAGES: A CRITIQUE AND A GUIDE
PROC. FJCC (1965), P.23-36
CONTINUOUS

C0032 CLARK, S R   ROUKE, T A
A SIMULATION STUDY OF COST OF DELAYS IN COMPUTER SYSTEMS
PROC. FOURTH CONF. ON APPLICATIONS OF SIMULATION ACM/AIIE
IEEE/SHARE/SCI/TIMS NEW YORK,NEW YORK (DEC.1970), P.195-200
APPLICATIONS

C0040 CLEMENTSON, A T
EXTENDED CONTROL AND SIMULATION LANGUAGE
COMPUTER JOURNAL 9.3 (NOV.1966), P.215-220
ECSL

C0050 COHEN, L J
SYSTEM AND SOFTWARE SIMULATOR
C-E-I-R, INC. WASHINGTON, DC (DEC.1968), AD 679 269, VOL.1

C0060 COHEN, L J
S3, THE SYSTEM AND SOFTWARE SIMULATOR
DIGEST OF THE SECOND CONFERENCE ON APPLICATIONS OF
SIMULATION (DEC.1968), P.282-285
S3

C0070 COMRESS - ROCKVILLE, MARYLAND
A TECHNICAL DESCRIPTION OF SCERT

```
C0080 CONWAY, R W          DELFAUSSE, J J
      MAXWELL, W L         WALKER, W E
      CLP - THE CORNELL LIST PROCESSOR
      COMM OF THE ACM (APRIL 1965), VOL.8, NO.4, P.215-216
      CLP


D0010 DAHL, O J
      DISCRETE EVENT SIMULATION LANGUAGES
      LECTURES DELIVERED AT THE NATO SUMMER SCHOOL,
      VILLARD-DE-LANS (SEPT.1966)
      CSL   GPSS   SIMSCRIPT   SIMULA   SOL


D0020 DAHL, O J            NYGAARD, K
      SIMULA - AN ALGOL-BASED SIMULATION LANGUAGE
      COMMUNICATIONS OF THE ACM (SEPT.1966), VOL.9, NO.9,
      P.671-678
      SIMULA


D0030 DAHL, O J            MYHRHAUG, B
      NYGAARD, K
      SOME FEATURES OF THE SIMULA 67 LANGUAGE
      DIGEST OF THE SECOND CONFERENCE ON APPLICATIONS OF
      SIMULATION (DEC.1968), P.29-31
      SIMULA


D0040 DONOVAN, J J         ALSOP, J W
      JONES, M M
      A GRAPHICAL FACILITY FOR AN INTERACTIVE SIMULATION SYSTEM
      PROC. OF IFIPS CONGRESS, (1968), P.593-596
      SIMPLE


D0050 DOWNS, H R           NIELSEN, N R
      WATANABE, E T
      SIMULATION OF THE ILLIAC IV - B6500 REAL-TIME COMPUTING
      SYSTEM
      PROC. FOURTH CONF. ON APPLICATIONS OF SIMULATION ACM/AIIE/
      IEEE/SHARE/SCI/TIMS NEW YORK,NEW YORK (DEC.1970), P.207-212
      FORTRAN   BURROUGHS/6500   APPLICATIONS


E0010 EVANS, G W           WALLACE, G F                           **
      SUTHERLAND, G L
      SIMULATION USING DIGITAL COMPUTERS
      PRENTICE-HALL, INC. (1967)


F0003 FAMOLARI, E
      FORSIM IV - FORTRAN IV SIMULATION LANGUAGE USER'S GUIDE
      SR-99 MITRE CORP. (JAN.1964)
      FORSIM
```

```
F0004 FINE, G H          MCISAAC, P V
      SIMULATION OF A TIME SHARING SYSTEM
      MANAG. SCI. 12,6 (FEB.1966), P.180-194
      APPLICATIONS


F0007 FOX, D             KESSLER, J L
      EXPERIMENTS IN SOFTWARE MODELING
      PROC. AFIPS FJCC (1967), VOL.31, P.429-436
      APPLICATIONS


F0010 FREEMAN, C E
      DISCRETE SYSTEMS SIMULATION
      SIMULATION 7,3 (SEPT.1966), P.142-148
      GPSS


F0020 FREEMAN, C E
      PROGRAMMING LANGUAGES EASE DIGITAL SIMULATION
      CONTR. ENG. (NOV.1964), P.103-106A
      CSL  GPSS  SIMSCRIPT


G0010 GAINEN, L
      COMPLEX BUSINESS PROBLEMS? TRY SIMSCRIPT, A POWERFUL
      SIMULATION LANGUAGE
      COMPUTER DECISIONS (APRIL 1970), P.52-56
      SIMSCRIPT


G0020 GEISLER, M A       MARKOWITZ, H M
      A BRIEF REVIEW OF SIMSCRIPT AS A SIMULATING TECHNIQUE
      RAND CORP. RM-3778-PR (AUG.1963)
      SIMSCRIPT


G0025 GLINKA, L R        BRUSH, R M
      UNGAR, A J
      DESIGN, THRU SIMULATION, OF A MULTIPLE-ACCESS INFORMATION
      SYSTEM
      PROC. AFIPS FJCC (1967), VOL.31, P.437-447
      APPLICATIONS


G0030 GORDON, G
      SIMULATION LANGUAGES FOR DISCRETE SYSTEMS
      PROC. IBM SCIENTIFIC COMPUTING SYMPOSIUM SIMULATION MODELS
      AND GAMING (1966), P.101-118


G0040 GORDON, G                                                     **
      SYSTEM SIMULATION
      PRENTICE-HALL, INC. (1969)
```

G0050 GREENBERGER, M   JONES, M M
MORRIS, J H   NESS, D N
ON-LINE COMPUTATION AND SIMULATION: THE OPS-3 SYSTEM
THE MIT PRESS (AUG.1965)
OPS3

H0010 HAWTHORNE, G B
DIGITAL SIMULATION AND MODELING
THE MITRE CORP. SR-111 (MARCH 1964)

H0020 HERMAN, D J
SCERT: A COMPUTER EVALUATION TOOL
DATAMATION (FEB.1967), P.26-28
SCERT

H0030 HERMAN, D J   IHRER, F C
THE USE OF A COMPUTER TO EVALUATE COMPUTERS
PROC. SJCC (1964) SPARTAN BOOKS, WASHINGTON,DC P.383-395
SCERT

H0033 HILLS, P R
SIMON: A SIMULATION LANGUAGE IN ALGOL
DIGITAL SIMULATION IN OPERATIONAL RESEARCH,
ED. S.H.HOLLINGDALE, (1967), LECTURES PRES. AT NATO
SCIENTIFIC AFFAIRS CONF., HAMBURG, GERMANY, (SEPT.1965),
P.105-115
SIMON

H0035 HOARE, C A R
RECORD HANDLING
PROGRAMMING LANGUAGES, NATO ADVANCED STUDY INSTITUTE,
LECTURES DELIVERED AT NATO SUMMER SCHOOL, VILLARD-DE-LANS,
ACADEMIC PRESS, (1968), P.324-336

H0040 HOLLAND, R C   MERIKALLIO, R A
SIMULATION OF A MULTIPROCESSING SYSTEM USING GPSS
IEEE TRANSACTIONS ON SYSTEMS SCIENCE AND CYBERNETICS
(NOV.1968) VOL.SSC-4, NO.4, P.395-400
GPSS

H0045 HOLLINGDALE, S H **
DIGITAL SIMULATION IN OPERATIONAL RESEARCH
LECTURES DELIV. AT NATO SCIENTIFIC AFFAIRS CONF., HAMBURG,
GERMANY, AMER. ELSEVIER PUB. CO., (1967)

H0050 HUESMAN, L R   GOLDBERG, R P
EVALUATING COMPUTER SYSTEMS THROUGH SIMULATION
COMPUTER J. 10,2 (AUG.1967), P.150-156
SCERT CTSS SIMSCRIPT SIMTRAN CSS MDL APPLICATIONS

H0053 HUGHES AIRCRAFT GROUND SYSTEM GROUP
A TOOL FOR THE DEVELOPMENT OF REAL-TIME COMPUTER SYSTEMS
(PRODUCT LINE SIMULATOR)
HUGHES AIRCRAFT CO, NO.05256-2, (AUG.1970)
PLS

H0057 HUTCHINSON, G K
SOME PROBLEMS IN THE SIMULATION OF MULTIPROCESSOR COMPUTER
SYSTEMS
SIMULATION PROGRAMMING LANGUAGES (1968) PROC. OF IFIP
WORKING CONFERENCE ON SIMULATION LANGUAGES, OSLO, NORWAY
(1967), P.305-324
APPLICATIONS

H0060 HUTCHINSON, G K MAGUIRE, J N
COMPUTER SYSTEMS DESIGN AND ANALYSIS THROUGH SIMULATION
PROC. FJCC (1965), P.161-167
SIMSCRIPT UNIVAC/1107 APPLICATIONS

I0003 IBM
BIBLIOGRAPHY ON SIMULATION
320-0924-C IBM (1966)

I0010 IBM APPLICATION PROGRAM
GENERAL PURPOSE SIMULATION SYSTEM/360 APPLICATION
DESCRIPTION
IBM (1966,1967), H20-0186-2
GPSS

I0020 IBM APPLICATION PROGRAM
GENERAL PURPOSE SIMULATION SYSTEM/360 INTRODUCTORY USER'S
MANUAL
IBM (1967), H20-0304-1
GPSS

I0030 IBM APPLICATION PROGRAM
GENERAL PURPOSE SIMULATION SYSTEM/360 USER'S MANUAL
IBM (1967,1968), H20-0326-2
GPSS

I0035 IBM APPLICATION PROGRAM
GENERAL PURPOSE SYSTEMS SIMULATOR III INTRODUCTION
GB20-0001-0 IBM (FEB.1970)
GPSS

I0040 IBM DATA PROCESSING APPLICATION
GENERAL PURPOSE SYSTEMS SIMULATOR II
IBM (1963), B20-6346-1
GPSS

J0020 JONES, M M
ON-LINE SIMULATION
PROC. 22ND NAT. CONF. ACM PUB. P-67, (1967), P.591-599
OPS3 OPS4

K0005 KALINCHENKO, L A
SLANG - COMPUTER DESCRIPTION AND SIMULATION-ORIENTED
EXPERIMENTAL PROGRAMMING LANGUAGE
SIMULATION PROGRAMMING LANGUAGES (1968) PROC. OF IFIP
WORKING CONFERENCE ON SIMULATION LANGUAGES, OSLO, NORWAY
(1967), P.101-116
SLANG

K0010 KATZ, J H
AN EXPERIMENTAL MODEL OF SYSTEM/360
COMMUNICATIONS OF THE ACM (NOV.1967), VOL.10, NO.11,
P.694-702
SIMSCRIPT IBM/360 APPLICATIONS

K0020 KATZ, J H
SIMULATION OF A MULTIPROCESSOR COMPUTER SYSTEM
PROC. SJCC (1966), VOL.28, P.127-139
APPLICATIONS

K0025 KELLEY, D H BUXTON, J N
MONTECODE - AN INTERPRETIVE PROGRAM FOR MONTE CARLO
COMPUTER J. (JULY 1962), VOL.5, P.88-93
MONTECODE

K0030 KIVIAT, P J
DEVELOPMENT OF DISCRETE DIGITAL SIMULATION LANGUAGES
SIMULATION 8,2 (FEB.1967), P.65-70

K0040 KIVIAT, P J
DEVELOPMENT OF NEW DIGITAL SIMULATION LANGUAGES
P-3348, RAND CORP., SANTA MONICA, CALIF., (APRIL 1966)

K0050 KIVIAT, P J
DIGITAL COMPUTER SIMULATION: COMPUTER PROGRAMMING LANGUAGES
RM-5883-PR RAND CORP., SANTA MONICA, CALIF., (JAN.1969)
CSL GPSS SIMSCRIPT SIMULA

K0060 KIVIAT, P J
DIGITAL COMPUTER SIMULATION: MODELING CONCEPTS
RM-5378-PR RAND CORP., SANTA MONICA, CALIF., (AUG.1967)

K0070 KIVIAT, P J
INTRODUCTION TO THE SIMSCRIPT II PROGRAMMING LANGUAGE
DIGEST OF THE SECOND CONFERENCE ON APPLICATIONS OF
SIMULATION (DEC.1968), P.32-36
SIMSCRIPT

K0074 KIVIAT, P J VILLANUEVA, R **
MARKOWITZ, H M
THE SIMSCRIPT II PROGRAMMING LANGUAGE
PRENTICE-HALL, INC. (1968)
SIMSCRIPT

K0077 KIVIAT, P J SHUKIAR, H J
URMAN, J B VILLANUEVA, R
THE SIMSCRIPT II PROGRAMMING LANGUAGE: IBM 360
IMPLEMENTATION
RM-5777-PR RAND CORP. (JULY 1969)
SIMSCRIPT

K0080 KLEINE, H
A SURVEY OF USERS' VIEWS OF DISCRETE SIMULATION LANGUAGES
SIMULATION (MAY 1970), P.225-229

K0090 KNUTH, D E MCNELEY, J L
A FORMAL DEFINITION OF SOL
IEEE TRANS. EC-13 (AUG.1964), P.409-414
SOL

K0100 KNUTH, D E MCNELEY, J L
SOL - A SYMBOLIC LANGUAGE FOR GENERAL-PURPOSE SYSTEM
SIMULATION
IEEE TRANS. EC-13 (AUG.1964), P.401-408
SOL

K0103 KOSY, D W
EXPERIENCE WITH THE EXTENDABLE COMPUTER SYSTEM SIMULATOR
FOURTH CONF. ON APPLICATIONS OF SIMULATION, ACM/AIIE/IEEE/
SHARE/SCI/TIMS NEW YORK,NEW YORK (DEC.1970), P.235-242
ECSS
APPLICATIONS

K0108 KRASNOW, H S
DYNAMIC REPRESENTATION IN DISCRETE INTERACTION SIMULATION LANGUAGES
DIGITAL SIMULATION IN OPERATIONAL RESEARCH,
ED. S.H.HOLLINGDALE, (1967), LECTURES PRES. AT NATO SCIENTIFIC AFFAIRS CONF., HAMBURG, GERMANY, (SEPT.1965), P.77-92
CSL GPSS GSP SIMSCRIPT SIMULA SOL MILITRAN

K0110 KRASNOW, H S
HIGHLIGHTS OF A DYNAMIC SYSTEM DESCRIPTION LANGUAGE
IBM ADV. SYSTEMS DEVELOPMENT DIV. TECHNICAL REPORT 17-195
NSS

K0120 KRASNOW, H S
SUMMARY REPORT: INTERNATIONAL FEDERATION OF INFORMATION PROCESSING WORKING CONFERENCE ON SIMULATION LANGUAGES
SIMULATION (FEB.1968), P.79-80

K0130 KRASNOW, H S   MERIKALLIO, R A
THE PAST, PRESENT, AND FUTURE OF SIMULATION LANGUAGES
MAN. SCI. 11, (NOV.1964), P.236-267
CSL DYNAMO GPSS SIMPAC SIMSCRIPT

K0140 KRIBS, P
SHARE DIGITAL SIMULATION GLOSSARY
SDC SP-1562 (FEB.20,1964)

L0003 LACKNER, M R
TOWARD A GENERAL SIMULATION CAPACITY (SIMPAC)
WESTERN JOINT COMPUT. CONF. (1962), P.1-14
SIMPAC

L0010 LEHMAN, M M   ROSENFELD, J L
PERFORMANCE OF A SIMULATED MULTIPROGRAMMING SYSTEM
PROC. FJCC (1968), VOL.33, PART 2, P.1431-1442
APPLICATIONS

M0003 MACDOUGALL, M H
COMPUTER SYSTEM SIMULATION: AN INTRODUCTION
COMPUTING SURVEYS (SEPT.1970), VOL.2, NO.3, P.191-209

M0005 MACDOUGALL, M H
SIMULATION OF AN ECS-BASED OPERATING SYSTEM
PROC. AFIPS SJCC (1967), VOL.30, P.735-741
APPLICATIONS

M0020 MARTIN, F F   **
COMPUTER MODELING AND SIMULATION
JOHN WILEY AND SONS, INC. (1968)

M0026 MCAULAY, S E
JOBSTREAM SIMULATION USING A CHANNEL MULTIPROGRAMING FEATURE
PROC. FOURTH CONF. ON APPLICATIONS OF SIMULATION ACM/AIIE/
IEEE/SHARE/SCI/TIMS NEW YORK,NEW YORK (DEC.1970), P.190-194
CSS IBM/360 APPLICATIONS

M0028 MCCREDIE, J W SCHLESINGER, S J
A MODULAR SIMULATION OF TSS/360
PROC. FOURTH CONF. ON APPLICATIONS OF SIMULATION ACM/AIIE/
IEEE/SHARE/SCI/TIMS NEW YORK,NEW YORK (DEC.1970), P.201-206
SIMULA TSS/360 APPLICATIONS

M0030 MCNELEY, J L
SIMULATION LANGUAGES
SIMULATION 9,2 (AUG.1967), P.95-98
SIMULA

M0040 MERIKALLIO, R A HOLLAND, F C
SIMULATION DESIGN OF A MULTIPROCESSING SYSTEM
PROC. FJCC (1968), VOL.33, PART 2, P.1399-1410
GPSS APPLICATIONS

M0050 MEYERHOFF, A J
BOSS SIMULATION OF A TIME SHARING MESSAGE PROCESSING SYSTEM
FOR BANK APPLICATIONS
BURROUGHS CORP. DEFENSE, SPACE AND SPECIAL SYSTEMS GROUP,
(SEPT.5,1969), NO.61088
BOSS
BOSS APPICATIONS

M0060 MEYERHOFF, A J ROTH, P F
TROY, J P
BOSS: BURROUGHS OPERATIONAL SYSTEMS SIMULATOR - APPLICATIONS
MANUAL
BURROUGHS CORP. DEFENSE, SPACE AND SPECIAL SYSTEMS GROUP
(JULY 19,1968)
BOSS

M0070 MIZE, J H COX, J G **
ESSENTIALS OF SIMULATION
PRENTICE-HALL, INC. (1968)

N0010 NAYLOR, T H
BIBLIOGRAPHY 19. SIMULATION AND GAMING
COMPUTING REVIEWS (JAN.1969), P.61-69

N0020 NIELSEN, N R
COMPUTER SIMULATION OF COMPUTER SYSTEM PERFORMANCE
PROC. ACM NATIONAL MEETING (1967), P.581-589
APPLICATIONS

N0030 NIELSEN, N R
ECSS: AN EXTENDABLE COMPUTER SYSTEM SIMULATOR
RM-6132-NASA RAND CORP., (FEB.1970)
ECSS

N0040 NIELSON, N R
THE SIMULATION OF TIME-SHARING SYSTEMS
COMM. ACM 10,7 (JULY 1967), P.397-412
IBM/360/67 APPLICATIONS

N0050 NYGAARD, K DAHL, O J
BASICS CONCEPTS OF SIMULA, AN ALGOL BASED SIMULATION LANGUAGE
DIGITAL SIMULATION IN OPERATIONAL RESEARCH,
ED. S.H.HOLLINGDALE, (1967), LECTURES PRES. AT NATO
SCIENTIFIC AFFAIRS CONF., HAMBURG, GERMANY, (SEPT.1965),
P.116-124
SIMULA

O0010 OCKENE, A
LOSING BUSINESS? SIMULATION MAKES IT EASIER TO SEE WHY
COMPUTER DECISIONS (MARCH 1970), P.36-40

P0010 PARENTE, R J
A LANGUAGE FOR DYNAMIC SYSTEM DESCRIPTION
IBM TECHNICAL REPORT 17-180 (NOV.19,1965)
NSS

P0020 PARENTE, R J KRASNOW, H S
A LANGUAGE FOR MODELING AND SIMULATING DYNAMIC SYSTEMS
COMMUNICATIONS OF THE ACM (SEPT.1967), VOL.10, NO.9,
P.559-567
NSS

P0030 PARSLOW, R D
AS: AN ALGOL SIMULATION LANGUAGE
SIMULATION PROGRAMMING LANGUAGES (1968) PROC. OF IFIP
WORKING CONFERENCE ON SIMUALTION LANGUAGES, OSLO, NORWAY
(1967), P.86-100
AS

```
P0040 PETRONE, L
      ON A SIMULATION LANGUAGE COMPLETELY DEFINED ONTO THE
      PROGRAMMING LANGUAGE PL/1
      SIMULATION PROGRAMMING LANGUAGES (1968) PROC. OF IFIP
      WORKING CONFERENCE ON SIMULATION LANGUAGES, OSLO, NORWAY
      (1967), P.61-85
      SPL


P0050 POMERANTZ, A G
      PREDICT YOUR SYSTEM'S FORTUNE: USE SIMULATION'S CRYSTAL BALL
      COMPUTER DECISIONS (JUNE 1970), P.16-19
      SCEPT


R0010 REHMANN, S L       GANGWERE, S G
      A SIMULATION STUDY OF RESOURCE MANAGEMENT IN A TIME-SHARING
      SYSTEM
      PROC. FJCC (1968), VOL.33, PART 2, P.1411-1430
      APPLICATIONS


R0012 REITMAN, J         INGERMAN, D
      KATZKE, J          SHAPIRO, J
      SIMON, K           SMITH, B
      A COMPLETE INTERACTIVE SIMULATION ENVIRONMENT GPSS/360
      PROC. FOURTH CONF. ON APPLICATIONS OF SIMULATION ACM/AIIE/
      IEEE/SHARE/SCI/TIMS NEW YORK,NEW YORK (DEC.1970), P.260-270
      GPSS  APPLICATIONS


R0020 ROTH, P F
      THE BOSS SIMULATOR - AN INTRODUCTION
      FOURTH CONF. ON APPLICATIONS OF SIMULATION, ACM/AIIE/IEEE/
      SHARE/SCI/TIMS NEW YORK,NEW YORK (DEC.1970), P.244-250
      BOSS
      APPLICATIONS


S0005 SANBORN, T G
      SIMCOM - THE SIMULATOR COMPILER
      EASTERN JOINT COMPUT. CONF. (1959), P.139-142
      SIMCOM


S0010 SEAMAN, P H
      ON TELEPROCESSING SYSTEM DESIGN (PART VI) THE ROLE OF
      DIGITAL SIMULATION
      IBM SYSTEMS JOURNAL (1966), VOL.5, NO.3, P.175-189
      APPLICATIONS


S0020 SEAMAN, P H        SOUCY, R C
      SIMULATING OPERATING SYSTEMS
      IBM SYSTEM JOURNAL (1969), NO.4, P.264-279
      CSS
      CSS   IBM/360/OS  APPLICATIONS
```

S0030 SHUBIK, M
BIBLIOGRAPHY ON SIMULATION, GAMING, ARTIFICIAL INTELLIGENCE AND ALLIED TOPICS
JOURNAL AMERICAN STATISTICAL ASSOCIATION (DEC.1960), VOL.55, P.736-751

S0040 SOFTWARE PRODUCTS CORP.
A TECHNICAL DESCRIPTION OF THE CASE SYSTEM
SOFTWARE PRODUCTS CORP. (1968)
CASE

S0050 STATLAND, N
METHODS OF EVALUATING COMPUTER SYSTEMS PERFORMANCE
COMPUTERS AND AUTOMATION (FEB.1964), P.18-23
IBM/1410 APPLICATIONS

S0070 SYSTEMS RESEARCH GROUP INC.
MILITRAN PROGRAMMING MANUAL
REPORT ESD-TDR-64-320, (JUNE 1964)
MILITRAN

T0010 TEICHROEN, D LUBIN, J F
COMPUTER SIMULATION - DISCUSSION OF THE TECHNIQUE AND COMPARISON OF LANGUAGES
SIMULATION 9,4 (OCT.1967) P.181-190
CLP CSL GASP GPSS HSL SIMSCRIPT SOL

T0020 THOMPSON, W C
THE APPLICATION OF SIMULATION IN COMPUTER SYSTEM DESIGN AND OPTIMIZATION
DIGEST OF THE SECOND CONFERENCE ON APPLICATIONS OF SIMULATION (DEC.1968), P.286-290

T0030 TOCHER, K D
REVIEW OF SIMULATION LANGUAGES
OPERATIONS RESEARCH QUARTERLY 16,2 P.189-218
CSL ESP GPSS GSP MONTECODE SIMON SIMPAC SIMULA SIMSCRIPT

T0031 TOCHER, K D
THE ROLE OF MODELS IN OPERATIONAL RESEARCH
J. ROYAL STATISTICAL SOCIETY, A, 124, (1961), P.121-142

T0033 TOCHER, K P AMIRY, A P
NEW DEVELOPMENTS IN SIMULATION
PROC. THIRD INTERN. CONF. ON OPERATIONS RESEARCH, (1963), P.832-848
GSP MKII

T0035 TOCHER, K D OWEN, D G
THE AUTOMATIC PROGRAMMING OF SIMULATORS
PROC. SECOND INTERN. CONF. OPERAT. RES. ENGLISH UNIV. PRESS
(1960), P.50-68
GSP

T0040 TOGNETTI, K P
DISCRETE SIMULATION LANGUAGES WITH REFERENCE TO A
BIO-SIMULATION - A USER'S IMPRESSIONS
PROC. OF FOURTH AUSTRALIAN COMP. CONF., ADELAIDE, SOUTH
AUSTRALIA, (1969)

T0050 TURK, C W
INCREMENTAL MODELING IN THE FORWARD DIRECTION
PROC. FOURTH CONF. ON APPLICATIONS OF SIMULATION ACM/AIIE/
IEEE/SHARE/SCI/TIMS NEW YORK,NEW YORK (DEC.1970), P.251-259
APPLICATIONS

U0010 USS ENGINEERS AND CONSULTANTS, INC.
GENERAL ACTIVITY SIMULATION PROGRAM - GASP
USS APPLIED RESEARCH LAB., MATHEMATICAL SERVICES NO.46,
(AUG.1969)
GASP

W0010 WEAMER, D G
QUIKSIM - A BLOCK STRUCTURED SIMULATION LANGUAGE WRITTEN IN
SIMSCRIPT
PROC. THIRD CONF. ON APPLICATIONS OF SIMULATION, LOS ANGELES
, (1969), P.1-11
QUIKSIM

W0015 WILLIAMS, J W J
THE ELLIOTT SIMULATOR PACKAGE (ESP)
COMPUTER J. (JAN.1964), VOL.6, NO.4, P.328-331
ESP

W0020 WORKSHOP ON SIMULATION LANGUAGES
UNIVERSITY OF PENNSYLVANIA MANAGEMENT SCIENCE CENTER
MARCH 17,18 1966

Y0010 YOUCHAH, M I RUDIE, D D
JOHNSON, F J
THE DATA PROCESSING SYSTEM SIMULATOR (DPSS)
PROC. AFIPS 1964 FALL JOINT COMPUT. CONF., VOL.26, PT.1,
P.251-276
DPSS

APPENDIX B

BIBLIOGRAPHY ON SIMULATION
(LISTED BY LANGUAGE)

```
P0030                                AS#
M0050                                BOSS#
M0060                                BOSS#
R0020                                BOSS#
B0003                                CASE SAM SCERT#
S0040                                CASE#
T0010 T SOL#                         CLP CSL GASP GPSS HSL SIMSCRIP
C0080                                CLP#
B0030                                CONTINUOUS#
C0030                                CONTINUOUS#
K0130 PT#                            CSL DYNAMO GPSS SIMPAC SIMSCRI
T0030 ON SIMPAC SIMULA SIMSCRIPT#    CSL ESP GPSS GSP MONTECODE SIM
T0010 L#                         CLP CSL GASP GPSS HSL SIMSCRIPT SO
K0108 SOL MILITRAN#                  CSL GPSS GSP SIMSCRIPT SIMULA
D0010                                CSL GPSS SIMSCRIPT SIMULA SOL#
K0050                                CSL GPSS SIMSCRIPT SIMULA#
F0020                                CSL GPSS SIMSCRIPT#
B0050                                CSL#
B0060                                CSL#
S0020                                CSS#
Y0010                                DPSS#
K0130                            CSL DYNAMO GPSS SIMPAC SIMSCRIPT#
C0040                                ECSL#
K0103                                ECSS#
N0030                                ECSS#
T0030 IMPAC SIMULA SIMSCRIPT#    CSL ESP GPSS GSP MONTECODE SIMON S
W0015                                ESP#
F0003                                FORSIM#
T0010                        CLP CSL GASP GPSS HSL SIMSCRIPT SOL#
U0010                                GASP#
T0030 C SIMULA SIMSCRIPT#    CSL ESP GPSS GSP MONTECODE SIMON SIMPA
K0108 MILITRAN#                  CSL GPSS GSP SIMSCRIPT SIMULA SOL
T0010                   CLP CSL GASP GPSS HSL SIMSCRIPT SOL#
K0130                     CSL DYNAMO GPSS SIMPAC SIMSCRIPT#
D0010                            CSL GPSS SIMSCRIPT SIMULA SOL#
K0050                            CSL GPSS SIMSCRIPT SIMULA#
F0020                            CSL GPSS SIMSCRIPT#
F0010                                GPSS#
H0040                                GPSS#
I0010                                GPSS#
I0020                                GPSS#
I0030                                GPSS#
I0040                                GPSS#
I0035                                GPSS#
T0033                                GSP MKII#
T0030 ULA SIMSCRIPT#    CSL ESP GPSS GSP MONTECODE SIMON SIMPAC SIM
K0108 RAN#                   CSL GPSS GSP SIMSCRIPT SIMULA SOL MILIT
T0035                                GSP#
T0010              CLP CSL GASP GPSS HSL SIMSCRIPT SOL#
S0070                                MILITRAN#
K0108 GPSS GSP SIMSCRIPT SIMULA SOL MILITRAN#                  CSL
T0033                            GSP MKII#
T0030 SIMSCRIPT#    CSL ESP GPSS GSP MONTECODE SIMON SIMPAC SIMULA
K0025                                MONTECODE#
P0020                                NSS#
P0010                                NSS#
K0110                                NSS#
```

```
J0020                                OPS3 OPS4#
G0050                                OPS3#
J0020                           OPS3 OPS4#
H0053                                PLS#
W0010                                QUIKSIM#
B0003                           CASE SAM SCERT#
A0020                                SAM#
H0030                                SCERT#
B0003                       CASE SAM SCERT#
H0020                                SCERT#
P0050                                SCERT#
S0005                                SIMCOM#
T0030    CSL ESP GPSS GSP MONTECODE SIMON SIMPAC SIMULA SIMSCRIPT#
H0033                                SIMON#
K0130              CSL DYNAMO GPSS SIMPAC SIMSCRIPT#
T0030  ESP GPSS GSP MONTECODE SIMON SIMPAC SIMULA SIMSCRIPT#   CSL
L0003                                SIMPAC#
D0040                                SIMPLE#
D0010                       CSL GPSS SIMSCRIPT SIMULA SOL#
K0108                   CSL GPSS GSP SIMSCRIPT SIMULA SOL MILITRAN#
K0050                       CSL GPSS SIMSCRIPT SIMULA#
T0010          CLP CSL GASP GPSS HSL SIMSCRIPT SOL#
K0070                                SIMSCRIPT#
K0130         CSL DYNAMO GPSS SIMPAC SIMSCRIPT#
K0077                                SIMSCRIPT#
K0074                                SIMSCRIPT#                          **
F0020                       CSL GPSS SIMSCRIPT#
G0010                                SIMSCRIPT#
G0020                                SIMSCRIPT#
T0030 MONTECODE SIMON SIMPAC SIMULA SIMSCRIPT#   CSL ESP GPSS GSP
T0030 SS GSP MONTECODE SIMON SIMPAC SIMULA SIMSCRIPT#   CSL ESP GP
K0108         CSL GPSS GSP SIMSCRIPT SIMULA SOL MILITRAN#
D0010             CSL GPSS SIMSCRIPT SIMULA SOL#
K0050             CSL GPSS SIMSCRIPT SIMULA#
D0030                                SIMULA#
D0020                                SIMULA#
M0030                                SIMULA#
N0050                                SIMULA#
K0005                                SLANG#
K0108 CSL GPSS GSP SIMSCRIPT SIMULA SOL MILITRAN#
K0100                                SOL#
K0090                                SOL#
D0010     CSL GPSS SIMSCRIPT SIMULA SOL#
T0010 P CSL GASP GPSS HSL SIMSCRIPT SOL#                          CL
A0030                                SOLPASS#
P0040                                SPL#
C0060                                S3#
```

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* The MITRE Corporation Bedford, Massachusetts 01730	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

SURVEY OF SIMULATION LANGUAGES AND PROGRAMS

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

5. AUTHOR(S) *(First name, middle initial, last name)*

Joan C. DesRoches

6. REPORT DATE JULY 1971	7a. TOTAL NO. OF PAGES 126	7b. NO. OF REFS 74

8a. CONTRACT OR GRANT NO. F19(628)-71-C-0002	9a. ORIGINATOR'S REPORT NUMBER(S) ESD-TR-71-227
b. PROJECT NO. 5720	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* MTR-2040
d.	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Electronic Systems Division, Air Force Systems Command, L. G. Hanscom Field, Bedford, Massachusetts 01730

13. ABSTRACT

This report documents a survey of available simulation languages and programs of potential applicability to the simulation of ADPE systems. The major features of the subject languages are discussed and a comprehensive bibliography is included.

DD FORM 1 NOV 65 1473

Security Classification

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
ANALOG SIMULATION						
COMPUTERIZED SIMULATION						
COMPUTER PROGRAMMING						
COMPUTERS						
DATA PROCESSING						
DIGITAL SIMULATION						
HYBRID SIMULATION						
MATHEMATICAL MODELS						
SIMULATION						
PROGRAMMING LANGUAGES						
PROGRAMMING MANUALS						

Security Classification