SOLPASS
A Simulation Oriented Language Programming and Simulation System

James Armstrong
Horst Ulfers
USAECOM, Fort Monmouth, New Jersey

Donald J. Miller
Harry C. Page
International Computer Sciences, Inc.
Neptune, New Jersey

Abstract

SOL, A Simulation Oriented Language, was described by Messrs. Knuth and McNeley of Burroughs Corp. in 1964. In 1968 a complete 2-pass compiler system was developed under Government contract by Messrs. Page and Miller of Patterson-Smith, Inc. Since then a number of features have been added to the SOL language, making it a very powerful tool for discrete simulations. SOLPASS has found a broad field of applications within the USAECOM Laboratories served by the Burroughs B5500 installation at Fort Monmouth. The Simulation System has been implemented to make most efficient use of the remote access system to the B5500 computer for compilation, debugging, test runs and short production runs. This arrangement has increased the programming efficiency to the extent that discrete simulation models can now be implemented within a fraction of the time previously needed. Discrete simulations of large scale communication systems and control systems have been performed successfully using SOLPASS.

An important goal of the Army is the development and production of a lightweight and mobile battlefield communication network to eventually link all the field armies and incorporate many types of messages, including teletype, voice, and high speed data transmission. Communication links are to include hardwire, RF, microwave, and satellite relays. Switching modes will include conventional relay equipment, switching computers, and store and forward computers. The Army has devoted considerable effort to gather the traffic statistics and to simulate the traffic flow through the various system configurations to determine component requirements of data links and modes. The Advanced Systems Design Branch of the U.S. Army Electronics Command Communication and Automatic Data Processing Laboratories at Fort Monmouth New Jersey selected SOL to be the simulation language used in the communication traffic simulation.

The SOLPASS simulation system was developed to enable efficient simulations of large networks on the Burroughs B5500 computer. The SOL language, as defined by Mr. McNeley and Professor Knuth was used as basis for this implementation. The SOLPASS system constitutes a general purpose simulation package of wide range applicability and the only one available for the B5500 computer. Special complementary language constructs were added to allow efficient simulations of large communication systems, with a variety of features for preemption and similar control actions.

SOL is an algorithmetic language used to construct models of general systems for simulation. The model builder describes his model in terms of processes whose number and detail are completely arbitrary and definable within the constraints of the language elements. A

SOL model consists of a number of statements and declarations which have a character similar to that found in programming languages such as ALGOL. The statements and declarations in each process define the rules for each transaction using that process. Each process may have local variables and each transaction for that process will have its own set of local variables. Thus, a process to describe a message in a communication system may be written and each message would be a discrete transaction in the process.

The SOL language is characterized by several predefined queuing variables which are global to all processes in the simulation model. Each queuing variable type has discrete disciplines which relieve the modeler from performing his own queue management. There are basically five distinct queuing constructs in the SOL language as follows:

(1) WAIT (N time units)
This construct permits the transaction to wait for a passage of time.

(2) WAIT UNTIL (Boolean expression)
This construct permits the modeler to define one or more queues based on relationship of model variables.

(3) FACILITY
A facility is a time-shared variable which is preemptable on a priority basis. A facility is either in a state of busy or not busy. It is controlled by a transaction seizing it and is made not busy by a transaction releasing it. The degree to which a transaction may control a facility is determined by a control strength associated with the seize. Transactions attempting to seize the same facility with a higher control strength will interrupt the transaction controlling the facility. Transactions attempting to seize the facility with less than or equal to the control strength will be queued on control strength, transaction priority, and first-in/first-out basis. The interrupted transaction may be cancelled, returned to the facility queue or may branch to a prespecified statement at the option of the modeler. The interrupt action is a function of each transaction and not a function of the overall model.

(4) STORE
A store is a space-shared variable which has a discrete capacity. Transactions control all or part of a store by entering the store and release control by leaving the store. They may enter or leave with N units of space. If a transaction attempts to enter a store with more units than are currently available, the transaction will be queued, based on transaction priority, first-in/first-out. Multiple transactions may hold space in the same store.

(5) TRUNK
The trunk is a space-shared variable which may be interrupted in a fashion similar to that of the facility. The trunk was implemented to facilitate simulation of large communication trunk lines which may handle a variety of message types simultaneously. The interrupt capability permits high priority messages to pre-empt parts of a data link for message transmission. The transaction or transactions which are interrupted from a trunk are predicated on a priority basis.

The first four queuing constructs are a part of the basic language. The trunk is an extension which was defined for this implementation. There are several other extensions to the SOL language which will be defined later in this paper.

The implementation of the SOL compiler system was performed under Government contract by Patterson-Smith, Inc., a subsidiary of International Computer Sciences, Inc. The project was started in the Fall of 1967 and was completed in early 1968. Additional refinement of the SOLPASS system was performed after the completion of the initial implementation.

Because of the anticipated simultaneous processing of up to 10,000 messages through the network, SOLPASS was designed to operate with virtual memory, making full utilization of the on-line disk storage of the Burroughs B5500. The number of transactions which may be simultaneously in process is a function of the disk memory availability and not the core storage capability. Further criteria was the capability to define and run models from re-

mote on-line terminals without significant sacrifice of large scale simulation speed. It was further required that the number of queuing variables and number of simultaneous queues be virtually unlimited to facilitate implementation of large scale networks.

Since SOL is an ALGOL-type language, it was decided to implement SOL in ALGOL and provide a complete ALGOL capability as a sub-set of the SOL language. Toward this end, the complete Burroughs extended ALGOL has been included in the SOLPASS system. The SOLPASS system has three basic components:

(1) a SOL translator which syntaxes SOL source statements and generates an ALGOL program,

(2) a simulation control module which is appended during the ALGOL compilation,

(3) and a post-simulation statistical analysis system which performs calculations necessary for resource usage documentation.

This structuring of the SOLPASS system into three discrete modules permits considerable flexibility and provides some unique capabilities which are desirable in a simulation system. The system flow involves writing the model in SOL with ALGOL language inserts as required, compiling the model in SOL to generate an ALGOL program, compiling the ALGOL program with the automatic insertion by the system of the control module queuing algorithms and intrinsics; running the model; and, finally, performing statistical analyses after the simulation has been run. The separation of the simulation model from the statistical routine is accomplished in a rather unique fashion. Statistics on the queuing variables are not retained in memory during the simulation run, but rather the events upon which the statistics are based are placed on an external file as the events occur. As a normal byproduct of any simulation, the occurence of each event and the time of each event for all queuing variables are placed on the external file. Additional information on the external file may be specified by the modeler by defining tables for gathering statistics on non-queuing variables and arithmetic expressions. The capturing of the events rather than the statistics during the simulation run provides the modeler with considerable flexibility for analyzing the statistics. The event file may be retained by the modeler for as long as desired and multiple analysis runs of the statistics may be made. With an event log file it is possible to do an analysis of discrete variables as a function of time and to select discrete time spans for analyses. Available in the SOLPASS implementation are tabular printouts of queuing variables and tables, graphic displays via bar charts, and plot functions with the ability to plot multiple variables on the same graph with different scaling factors. These capabilities simplify preparation of reports and presentation of the simulation results as well as analysis of the simulation run. The separation of statistics from the simulation run simplifies the problem of defining the model because upon successful completion of a given model one may then determine what statistics are significant rather than attempting to predefine statistical output prior to the running of the model.

A prime consideration in the implementation of SOLPASS was the requirement for providing convenient debugging tools and the ability to analyze and identify errors in the physical aspects of the model as well as the logical aspects of the model. Several levels of debugging and error analysis have been incorporated into the SOLPASS system. The first such level is during the compile phase. All SOL statements are thoroughly checked for syntactical correctness and error messages generated at compile time describe any errors which might occur. The ALGOL compilation provides a secondary level of source language checking on the ALGOL statements which may be included in the SOL program. A third level of error detection and analysis occurs while the simulation is in process to ensure proper management of arithmetic and queuing variables. Error termination conditions include invalid array indexing, division by zero, mismanagement of facilities, stores, or trunk queuing variables, and excessive transactions simultaneously in a process where this criteria is controlled by the modeler. If errors are detected during the simulation run, an error termination routine for the simulation is evoked which annotates the error and dumps to an external file all global queuing and arithmetic variables with the status of all queues. Additionally, disk files containing all local variables and transactions are made available for diagnosis. These external files may then be examined after the simulation is terminated to provide a complete analysis of the state of the simulation when the error occurred. It is additionally possible to specify dynamic monitoring and dumping of discrete variables

during the simulation run to permit tracing of the logic flow.

To further facilitate the utilization of the system a complete breakout-restart capability has been implemented in the simulation run. This gives the modeler the ability to stop a program at discrete points and restart it with or without alterations. This facilitates computer time scheduling for large simulations and minimizes the possibility of reruns due to computer failure. It also provides the unique capability to run a model until it has reached a steady state condition, perform a breakout, and then run a series of restarts of the steady state model impressing varying loading factors upon the steady state condition. The maintenance of a separate event log facilitates analysis of the subsequent restarts and permits statistical analysis of the entire simulation including the restart or only the simulation time involving the imposition of varying loading factors. This particular capability has proved to be very valuable in the analyses of communication networks.

The random number generator implemented for SOLPASS has been tested extensively at the USAECOM R & D Laboratories. It is fast, non-recurring over long sequences, satisfied fundamental statistical criteria and is repeatable.

Starting with three arbitrary real numbers between zero and one, a new number between zero and one is calculated as the sum of the three numbers. The oldest of the original numbers is replaced by the new number and the sequence is repeated for the next random number. Within the SOL language, a random number is obtained by use of the syntactical primary RANDOM.

There are several intrinsic distributions to the SOL language which also utilize the random number generator. These are the normal, exponential, poisson, and geometric distributions. There is, additionally, a probability function in the SOL language which has been defined as PR = RANDOM LESS THAN OR EQUAL TO. Thus, if one desires an event to occur with the probability of .25, it would be specified as IF PR .25; An extension to the SOL language which calls upon the random number generator is the distribution function. This function permits sampling from a discrete probability distribution. Values of the random variables are given in an expression with the associated frequency of occurence. Frequencies need not sum to any particular number; they are summed and normalized before the function is evaluated. The random number generator is called by the distribution function for the selection of the random variables.

There are several characteristics of this implementation pursuant to routine efficiency which are of interest to the modeler. The SOLPASS system has been designed so that memory is not a restriction on the size of the model. Most models will normally run in 8,000 words of resident memory on the B5500. This is brought about by several design features in the control modules of the SOL system. In other simulation systems, considerable core memory is required to retain statistics. The generation of an event log tape as opposed to maintaining in-core statistics minimizes the actual simulation model core requirement. Secondly, the system is responsive to the number of transactions in any one process and, if the number of transactions exceeds a predetermined quantity which is a function of the number of local variables in the transaction, a disk management module is evoked which provides efficient core swapping of transaction groups for each process. A third factor in the effectiveness of the simulation system is the queue management capability. The queues, where all queuing variables are maintained in a single segmented dynamic array, may be overlayed by the operating system of the B5500. All entries into the queue are via link lists and several levels of algorithms are utilized for queuing and un-queuing based upon the current size of a particular queue.
The use of multiple algorithms for transaction management and queue management permit efficient running of both large and small simulations. Small simulations will run completely in core, not requiring disk access. Large simulations, on the other hand, will tend to run the most active segments in core with overflow to disk only as required to handle large volumes of transactions.

The SOLPASS simulation package is in use on a Burroughs B5500 computer with a remote access user system. To make most efficient use of the remote access facilities all SOLPASS programs have been developed for use on the remote access system as well as for batch processing. Figure 1 illustrates the flow through the SOLPASS system. The generation of the Object Program code is performed in two steps via an intermediate ALGOL language state. Both compiler programs, the SOL compiler as well as the ALGOL compiler, can easily be executed from the remote access

units. The error listings on disk are available for debugging immediately, thus eliminating the long turnaround times of batch processing.

The simulation model can be tested by short test runs of the Object Program. In case of test runs executed from the remote access units disk files are used instead of a Log Tape and Error Tape. A subsequent Log Analysis Program Run also executable from the remotes will produce the statistical data either on a hard copy or on disk memory. A second Analysis Program is used to generate load plots of specific model components, a very useful item in verifying the model.

Simulation production runs in general are performed by batch processing over night or weekends.

Generally a Simulation Model consists of three basic segments: the transaction generator, the basic simulator, and the analyzer. As experience has shown it is better for operational reasons to run segments of long simulations individually. This is in particular true for simulations whereby extensive numbers of transactions with local variables are to be generated from statistical data. For instance, in traffic simulations detailed statistics are usually available to generate messages with all parameters such as originator, destination, mode, length, precedence levels, route through the network, etc. For large communication networks the computer time to generate these messages approaches the basic simulation time. All segments of the program should be run separately and intermediate results of the simulation saved by the breakout feature provided in the extended SOL language. The separation of the traffic generation program usually will cut the running time 50% for each segment. The separation of the traffic generator avoids redundancy whenever several simulation runs are planned with identical traffic statistics.

The SOLPASS system separates statistical analysis program from the basic simulation program. This approach proves very efficient since the log tape of the simulation usually has to be analyzed more than just once and a variety of parameters can be specified for this analysis, for statistical listing as well as graphical illustrations.

Typical run times for simulations of varying sizes are contained in Table 1.

Applications for SOLPASS

Discrete Simulations
- Traffic Simulations
  - Road Traffic Studies
  - Air Traffic Studies
  - Communications Traffic Studies

Systems Simulations
- Control Systems
- Computer Systems
- Communication Systems
- Transmission Systems

Hardware Simulations
- Error Detection/Correction
- Encoding/Decoding Devices
- Logical Circuits

Continuous Simulations
- Flood Control Systems
- Analog Processes
- Electronics Circuits
- Transient Analysis

The expanded SOL language has been developed for general purpose applications. Consequently SOLPASS has found a broad spectrum of applications within the USAECOM laboratories. Using the SOLPASS system most of the listed applications have been modelled and programmed in SOL and executed on the Burroughs B5500 computer. Within the USAECOM Communication and Automatic Data Processing Laboratory SOLPASS had been developed for the simulation of traffic flow across large military communication networks. Consequently, most of the experience has been gained in Traffic Simulations and Systems Simulations.

Traffic Simulations are concerned primarily with the development of traffic load patterns for existing planned networks, to derive proper dimensions for the network components. For all types of traffic studies SOLPASS seems to be ideally suited.

Systems Simulations are usually performed to verify a certain system design without building specific hardware. Traffic considerations are here of secondary importance only. Also in this class of application SOLPASS performed very well.

Hardware Simulations are the third class of applications. Not enough experience has been gathered in this particular class for a proper

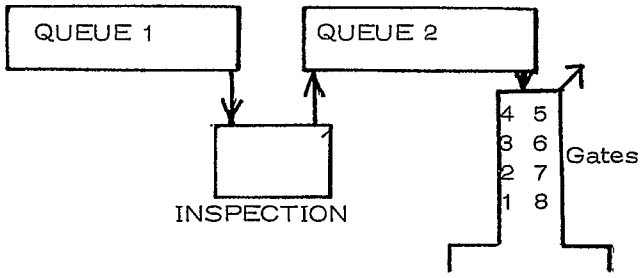
evaluation. However, indications are that SOL-PASS could be applied quite successfully in this field.

The fourth field is the off-beat application to continuous processes. Although only recently conceived and tested on a number of basic problems, this application does not seem to be as far off as one might expect. As a matter of fact, programming seemed to be less involved than on analog computers. Using the remote access system, results and plots were available immediately after relatively short runs.

SIMULATION EXAMPLES

Airport Simulator

The following illustrates a rather simple traffic simulation problem, which has been used as a classroom problem in a SOL programming class. The SOL program listing for this problem is shown as a language sample because of its small size.



An arrival building has 8 gates. Before planes can approach a gate they have to pass a health inspection station. Only one plane can be inspected at a time. It takes 4 seconds per passenger to complete the inspection. The plane then enters a queue until a gate is assigned. It will occupy a gate immediately if vacant. A plane will stay at a gate for an average time of 20 minutes sampled from an exponential distribution. Planes arrive at a rate of 24 per hour, poisson distributed. 60% of the planes carry 50 passengers, 30% carry 100 passengers, and 10% carry 150 passengers.

Determine: the maximum queues waiting for inspection.

Note: simulation time units in seconds.

SOL program listing

```
BEGIN
   FACILITY INSPECTION;
   STORE 8 GATES,
      100 QUEUE1, 100 QUEUE2;
ALGOL
BEGIN
   DEFINE MINUTES = X 60#,
          HOUR    = X 3600#;
END;
 PROCESS CONTROL:
   BEGIN
      WAIT 1 HOUR;
      STOP;
   END;
 PROCESS SIMULATE;
   BEGIN
      INTEGER PASSENGERS;
      LABEL START, ARRIVAL;
      START:
         WAIT EXPONENTIAL (1 HOUR/24);
         NEW TRANSACTION TO ARRIVAL;
         GO TO START;
      ARRIVAL:
         PASSENGERS:=DISTRIBUTION
          (50#60, 100#30, 150#10);
         ENTER QUEUE1;
         SEIZE INSPECTION;
         WAIT (4/PASSENGERS);
         RELEASE INSPECTION;
         LEAVE QUEUE1;
         IF GATES FULL THEN
          BEGIN
            ENTER QUEUE2;
            ENTER GATES;
            LEAVE QUEUE2;
          END ELSE
         ENTER GATES;
         WAIT EXPONENTIAL (20 MINUTES);
         CANCEL;
 END;
END.
```

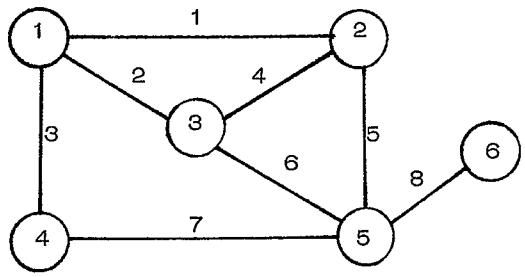
Communications Network Simulator

In general, simulations of large scale communication systems are performed because of the following two reasons:

(1) To optimize an existing network with respect to optimum systems utilization,

(2) To size links and switches in a communication network to be developed.

The following figure shows a simple network layout for illustration purposes only. Real networks tend to be much more complex in their layouts.



This sample network shows 6 nodes and 8 links. Node 6 as an Access Node has no impact on alternate routing. Therefore, Access Nodes in a network will have to be treated differently than the regular Trunk Nodes.

Traffic statistics between pairs of users (needlines) are usually available or can be derived. A further preprocessing of those user requirements will lead to traffic statistics between nodes only (nodal needlines). These are recorded in matrix form based on busy hour traffic. This matrix will serve as input to a traffic generator.

During the simulation, individual messages will then be routed through the network according to a special routing algorithm and statistics are collected for the loading of links and nodes. Important statistical factors are the maximum load occuring during the simulation and the average utilization.

In SOLPASS, the most convenient way to model the links between nodes is to utilize the STORE utility as provided by the SOL language. The store is a utility, which can be used by a number of transactions simultaneously until the capacity of the store is exhausted. Then further arriving transactions can either be queued until enough capacity of the store becomes available or they can be sent to a special procedure to take any other action desired. Each transaction represents a message as specified in the Traffic File and conserves its specific Traffic File information until it is terminated (cancelled).

If preemption on links is required the TRUNK facility as specified in SOL should be used. A simple substitution of stores by trunks will accomplish this in an existing model. Because the simulation using trunks requires many more bookkeeping processes than for stores, the model designer should use the TRUNK facility only when preemption has to be implemented although trunks will perform all functions of the regular STORE facility.

The nodes pictured in the shown network are, of course, very simplified and the system designer is interested in statistics about the following nodal components as illustrated in Figure 3 at the end of the text.

Figure 3 shows the block diagram of the Node Sub-Model. The Node is represented by 9 distinct modules, which do not necessarily represent hardware modules but which have been selected because each of them has a very distinct meaning in the simulation. They are all represented by STORE utilities.

LOCALSWITCH - This store registers calls which are both locally originated and also received.

TRUNKSWITCH - This store registers calls which are relayed at this node between trunk groups.

LOCALTOTRUNK and TRUNKTOLOCAL - These stores register traffic which is locally originated and inserted into the trunking network or which is remotely originated and received locally.

SANDF - This store is entered by Store/Forward (S/F) messages in addition to the regular stores. The SANDF store is only entered at the node to which the originating and receiving users are connected.

RTNQUEUE, PRTQUEUE, ROWQUEUE, FLHQUEUE - These four stores are only entered by a store/forward message, if the message could not be delivered because of systems blockage. The stores represent message queues, separated by precedence levels: RTNQUEUE for routine, PRTQUEUE for priority, ROWQUEUE for immediate, and FLHQUEUE for flash. To ensure that the higher priority messages in queue will always be considered first, attempts to deliver queued up flash messages are made every second, for immediate messages every two seconds, for priority messages every four seconds, and for routine messages every 8 seconds.

All stores are decreased by the capacity requirement of a message, whenever this message was terminated (cancelled).

Call Examples

The following actions will take place as can be best explained by these examples:

(1) Local Direct Call
The transaction representing the local call enters the LOCAL SWITCH store at origination time. A record is made on the Log Tape concerning the change in load on this store. An origination message is printed out on the event listing, specifying node, message number, time, originating node, receiving node and links along primary route. The transaction remains active until the holding time expires. Then the LOCAL SWITCH store load is reduced by the capacity of the transaction, a record is made on the Log Tape indicating the reduction in load on this store. A completion message is printed out on the event listing specifying message number and the time of completion. The call completion count is increased by 1 on this Nodal Needline. Then the message (transaction) is cancelled.

(2) Direct Call from Trunk Node to Trunk Node via Intermediate Relay Trunk Nodes
The transaction representing this call prints the origination message on the event file listing. Then the availability of the preselected primary route contained in the traffic file is tested. If not available, an alternate route of the same length or one link longer is searched for. If no alternate route is found, a "blocked" message is printed out in the event file listing and the lost call count is incremented by one. Otherwise, the loads on the following stores are increased by the message capacity.

At the originating Node:
LOCALTOTRUNK

At the receiving Node:
TRUNKTOLOCAL

At all intermediate Trunk Nodes:
TRUNKSWITCH

All links along the route:
TRUNK

As each store is loaded, a corresponding entry is made on the log tape. After the holding time has expired, the loads of all stores entered by this transaction are reduced by the capacity of the transaction and corresponding entries are made on the log tape. The message count of completed messages is increased by one. Then the transaction is cancelled.

(3) S/F Message from Trunk Node to Trunk Node via Intermediate Trunk Nodes
The same action takes place as previously described for the direct message except for the following: At the originating and receiving node the S and F store is also utilized. In case there is no route available the message is not cancelled, but depending upon its precedence level, will be entered into one of the following stores: FLHQUEUE, ROWQUEUE, PRTQUEUE, or RTNQUEUE.

A corresponding entry is made on the log tape. In certain time intervals depending upon the precedence level, an attempt is made to deliver the message. A record of S/F delays between pairs of nodes is maintained. If an S/F delay exceeds the time of delivery requirement, then the transaction is cancelled. An entry is made in a special file that registers S/F messages not being delivered on a node to node basis. Then the message is cancelled.

The following communications network model for large communication networks was programmed in SOL and has been run extensively for network sizes up to 127 nodes and for various degrees of traffic loads.

The Basic Model

The basic simulation model consists of three programs, which have to be run sequentially: The Traffic Generator Program, the Network Simulator Program and the Statistical Analysis Program.

(1) The Traffic Generator is a program that generates a complete message file for the busy hour. Two sets of input files have to be prepared for this program: frequency files and routing files. There are five frequency files, one for each mode of traffic (TTY, voice, etc.). These contain in matrix form the number of messages occurring between pairs of nodes during the busy hour. The routing files contain all possible alternate shortest and next longer routes. These files have also been generated by a pre-simulation run using the basic connectivity file of the network to be simu-

lated. There are a number of parameters, which must be set for each run, specifying precedence distributions, average inter-arrival time, and average holding times. The following data are recorded: Interarrival time, precedence, message number, mode, holding time, originating node, receiving node, a shortest route. All times are in terms of milliseconds. The route does not include the two terminating nodes. Routes from or to access nodes are only recorded to their respective trunk node.

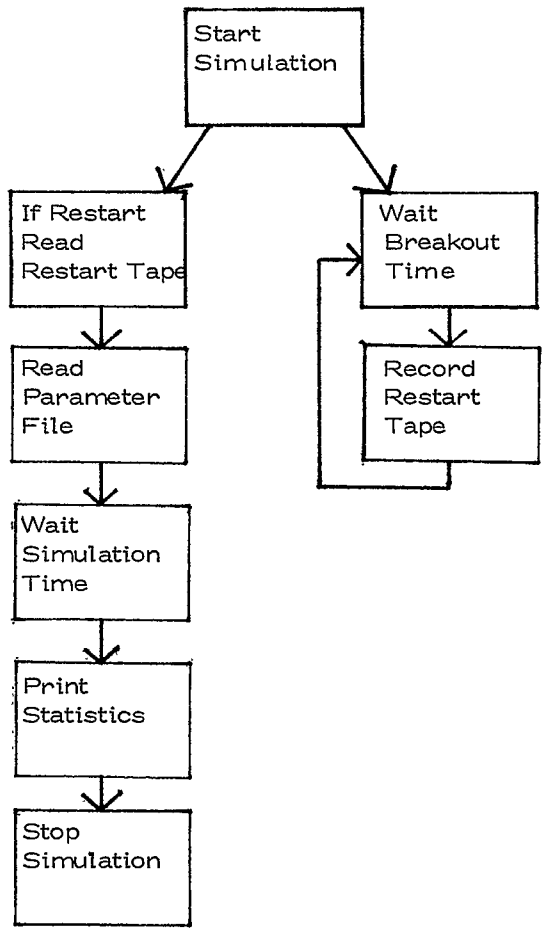
(2) The Network Simulator Features

The Network Simulator is the most important of the 3 simulation programs. It reads the Traffic File and performs the bookkeeping tasks for all network components to be monitored while messages (transactions) are passing through the model logic. The information about all active transactions, queues and occupied utilities (stores, facilities, trunks) is recorded and maintained on disk. Whenever a utility or a table is entered a corresponding entry is made on a log tape. Each entry contains time of entry, numeric entry value, type of utility or table, identification code, and index (if used). The program also provides for a breakout/restart feature which allows saving the contents of all disk files at certain time intervals. This information is stored in a Restart File on magnetic tape. In case of machine failures, the program then can be restarted at the last breakout point. This feature is also useful if certain network parameters are to be changed during the simulation run, since at restart these parameters can be changed, if desired. The Network Parameters are contained in files on disk and are read once at the beginning of each simulation run and each restart. They contain information about Access Nodes to Trunk Nodes connections, simulation time, breakout intervals, link capacities and the time increment for printing the number of active transactions. During the simulation two printer listings are produced: the Event Listing and the Transaction Listing. The Event File records all important simulation events in order of time. The Transaction Listing contains a record of all active transactions, the total number of transactions during the simulation and at the end, a record of the computer time used since begin of simulation or restart.

At the end of simulation and at each breakout a special printout of Grade-of-Service for the total system and in matrix format for each Nodal Needline is produced. Furthermore, the number of messages delivered between nodal pairs is also printed in matrix format. For S/F messages, delays exceeding the specified time of delivery are printed separately for each precedence level and each Nodal Needline.

Network Simulator Structure

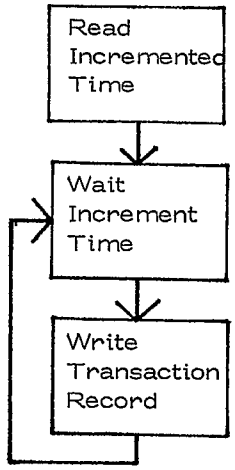
The model consists of three parallel processes:



This process initiates the simulation, reads the parameter file and sets all the simulation parameters. In certain time intervals, this

process records snapshots of the simulation on the RESTART TAPE. After expiration of the specified simulation period this process terminates the simulation and causes computer running time information to be printed out.

Process 2.
This process records in specified time intervals the number of active transactions as well as the total number of transactions entered since the start of the simulation (i.e. it records the transaction (or message) statistics). This is an important feature for observance when the network loading saturates.



Process 3. (Figure 4 at end of the text.)
This process is the heart of the simulation. It contains all the logic corresponding to the features of the model. It reads the traffic file, waits for the exponentially distributed interarrival time and then initiates a new transaction at the begin of the process. After a record of message origination has been printed either an unblocked route through the network is loaded or, in case of local calls only, the local node elements are loaded. The transaction stays in queue for the duration of the holding time, then the completion message is printed, the statistics recorded and, finally, the transaction is cancelled.

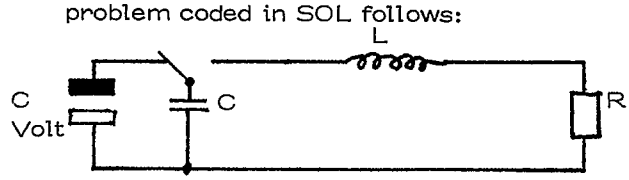
(3) The Statistical Analysis Program
This is a standard program of the SOL Simulation System, as described before. No basic options exist to change the format of the listings produced by this program. Since the simulation made use only of stores and tables, all other utility listings were suppressed. Statistical printing of stores not used during the simulation was also suppressed.

For stores the following items are listed: the name of the store with index in brackets for subscripted stores, the time of printout, the initial capacity of the store, the maximum capacity used (maximum load imposed during the simulation), the total occupancy (the sum of all message capacities multiplied by their respective holding times), and the average utilization (ratio of total occupancy to store capacity multiplied by simulation time). See Table 2.

Continuous Simulations

A NOVEL TYPE OF APPLICATION of the discrete type SOLPASS simulation system has been found in the simulation of transients in continuous processes as in electronic circuits. This application is still in its initial trial phase, but promises to be very powerful in simulation of circuits with nonlinear parameters, which are hard or impossible to simulate on analog computers. The basic method to implement these processes is to describe the interaction of the individual circuit components in finite but small increments in discrete time steps. Usually currents or voltages can be used to represent the interactive relations. However, there is no need to formulate differential equations. The initial conditions can easily be set by initiating the process properly. There is no scaling problem that would lead to invalid results. Results can be made as accurate as desired by choosing the proper time increments. The regular SOLPASS utilities as facilities and stores can be used to model network components as capacities, etc. Running times are usually very short and can be executed from remote access units in the matter of seconds. The resulting Log File now on disk can then be analyzed by running the PLOT program that will produce any desired plot of the process by direct printout on the remote access unit. Parameters then can be reset and the simulation repeated until optimum results are achieved.

A simple example of a continuous simulation problem coded in SOL follows:



```
BEGIN
INTEGER C, L. CVOLT, DI, DT, BIAS, CURRENT,
  SIMTIME, I, R, LVOLT;
STORE 2000 CIRCUIT;
ALGOL BEGIN
FILE IN INFILE DISK SERIAL "SWING"
  "IN" (1,10,30);
END;
PROCESS CONTROL, 1;
BEGIN
ALGOL BEGIN
READ (INFILE,/, C, L, CVOLT, DT, SIMTIME,
  BIAS, R);
END;
WAIT SIMTIME;
STOP;
END;
PROCESS SWINGER, 1;
BEGIN;
LABEL RETURN;
RETURN:
LVOLT:=CVOLT IXR;
DI:=(LVOLTXDT)/L;
I:=I+DI;
CVOLT:=CVOLT-(IXDT)/C;
CURRENT:=I+BIAS;
ENTER CIRCUIT, CURRENT;
WAIT DT;
LEAVE CIRCUIT, CURRENT;
GO TO RETURN;
END;
END.
```

This model represents resonant circuit consisting of the inductance L, the capacity C, and a Resistance R in series. The initial conditions can be set as parameters and the current can be studied as a function of time. Presently the system has been set up to plot the current as a function of time by a subsequent run of the SOL/PLOT program. However, it is planned to implement a plot function, which will allow to plot these functions at remote access consoles, while the program is running.

The SOLPASS implementation has proved to be effective and efficient, as well as an easily learned language for simulation programming. It is believed to be the first unrestricted implementation of the defined SOL language. Modifications and extensions have been added to increase the capability for model debugging and statistical analysis. The method of system implementation paves the way for further logical extensions currently being planned which will make the language an even more valuable simulation tool.

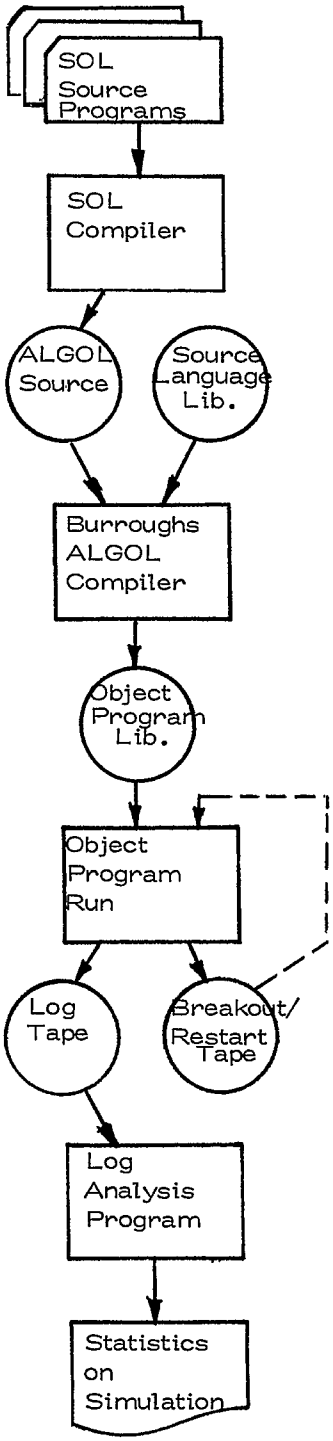


Figure 1

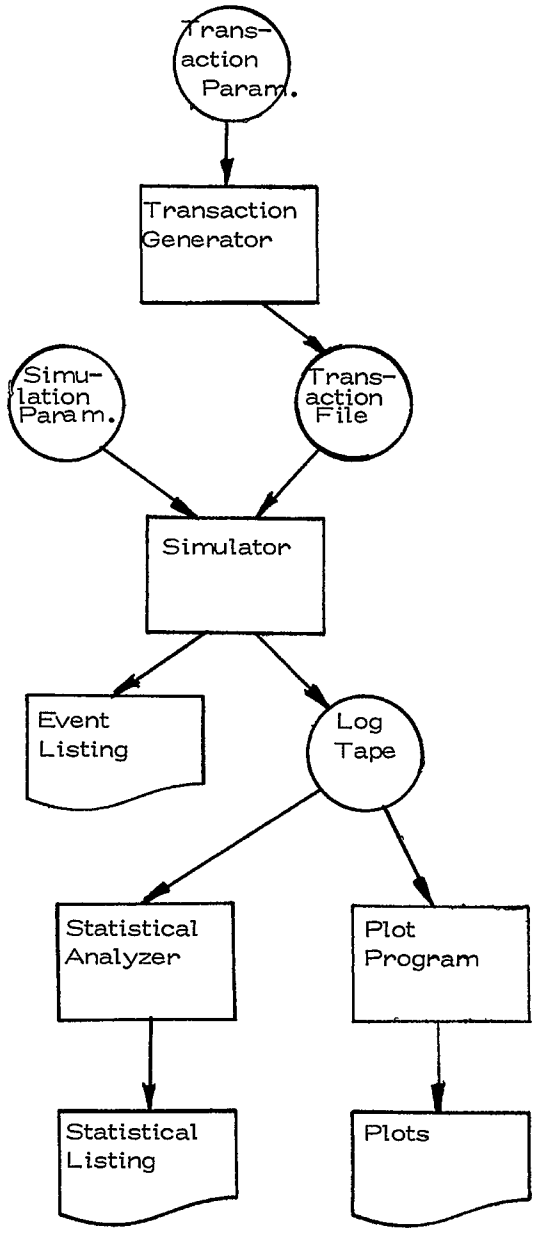


Figure 2

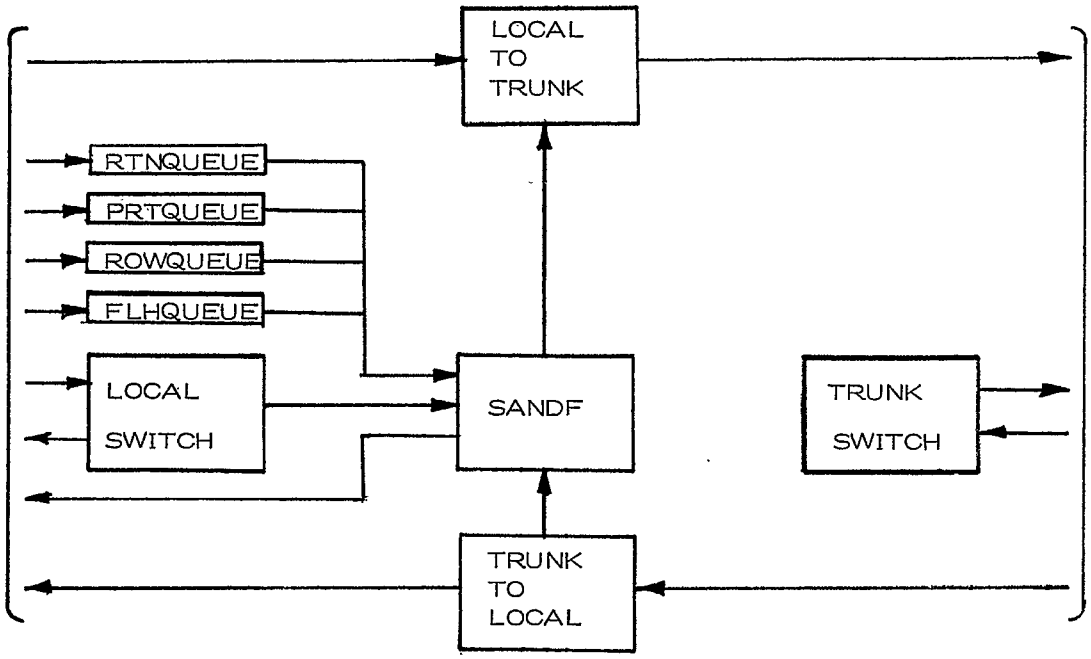


Figure 3

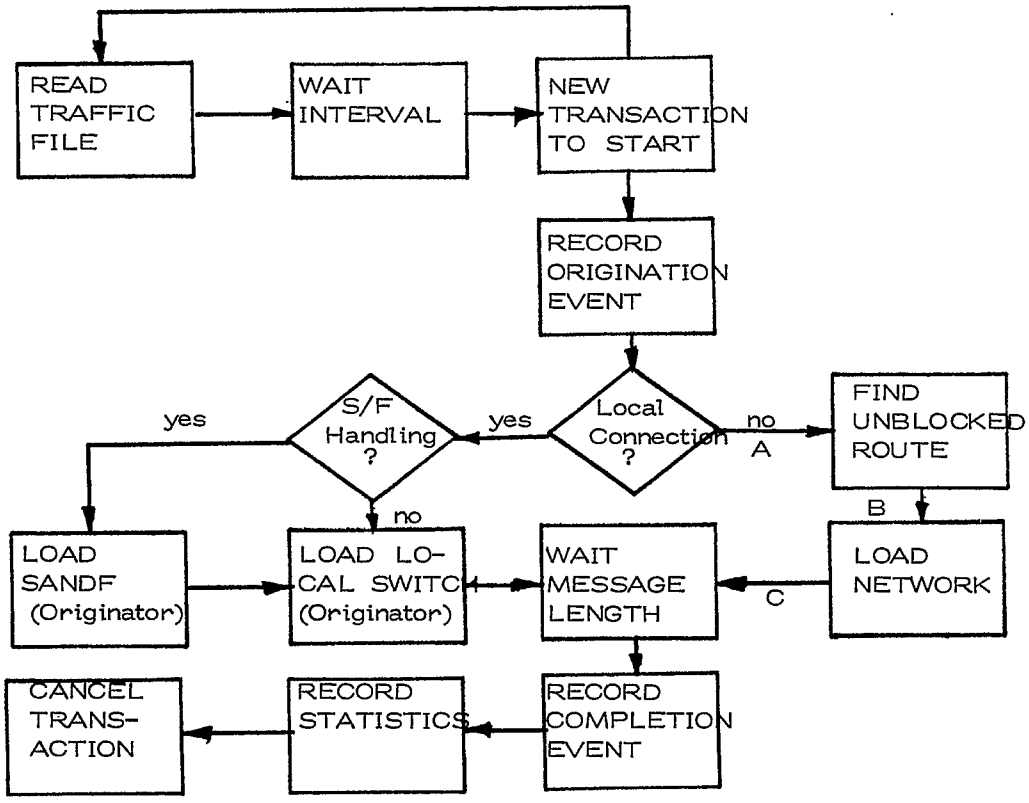


Figure 4

Table 1

Typical Run Times

Active Transactions*	Time/Transaction Processor Sec.	I/O Sec.
6	.06	.21
22	.16	.20
34	.23	.27
1150	.44	1.38
1230	.45	1.54

Each transaction queued two times/minute.

*Average active transactions

Table 2

NAME OF STORE		TIME	CAPCTY	MAX USD	TOTAL OCCP	AVG UTL
TRNK(	1)	1900135	512	379	529774858	0.5445
TRNK(	2)	1900135	512	387	570644927	0.5866
TRNK(	3)	1900135	512	388	506864532	0.5210
TRNK(	4)	1900135	512	160	179039663	0.1840
TRNK(	5)	1900135	512	26	13135275	0.0135
TRNK(	6)	1900135	512	475	614295039	0.6314
TRNK(	7)	1900135	512	392	477109550	0.9404
TRNK(	8)	1900135	512	512	828680327	0.8518
TRNK(	9)	1900135	512	511	782505257	0.8043
TRNK(	10)	1900135	512	273	312436429	0.3211
TRNK(	11)	1900135	512	506	669208291	0.6879
TRNK(	12)	1900135	512	113	118039645	0.1213
TRNK(	13)	1900135	512	74	78371754	0.0806
TRNK(	14)	1900135	512	88	75557580	0.0777
TRNK(	15)	1900135	512	512	807316577	0.8298