This paper describes the SIMSCRIPT *system simulation language and the philosophy of system structure on which it is based.*

Application of the language to programming both discrete and continuous models is indicated and illustrated with examples.

SIMSCRIPT *processing is described and statistics regarding operating characteristics are given.*

The SIMSCRIPT *system was developed at The RAND Corporation*[1,4] *by a group including the second author.*

A description of the SIMSCRIPT language

by B. Dimsdale and H. M. Markowitz

The production of a digital simulator program, or of any program for that matter, involves two steps: creating the model, then writing the program. Fundamentally, the writing of the program is a technical detail which must of necessity wait upon the creation of the model. Nevertheless, the nature of the machinery available for producing simulation programs is bound to exercise an influence on the nature of the model. This is true because effective modeling requires abstraction of the essence of the system under investigation, the direction taken in the abstraction being determined by the goals of the investigation. For complex systems it is very often not clear which of many possible abstractions is most valid for the purposes at hand.

The choice in this case is naturally made of the one which is easiest to handle even, perhaps, if it appears slightly less desirable than another one which will clearly lead to great difficulties in programming. At this point, it is worth remarking that the normal course of a simulation is not described by: modeling, programming, end of process; but rather by: modeling, testing, modeling, testing, etc., until an adequate model is developed. It seems a natural conclusion that the less restraint placed upon the modeler by the nature of his tools and their ease of use, the more rapidly this process will converge on the average, this phenomenon being the more pronounced for the more complex systems. From

this it follows that the nature of the programming language should facilitate program debugging, modification, and repetitive testing. Storage capacity should not be wasted, reports of simulation results should be easy to arrange and use, and not too much time should be necessary to test models.

SIMSCRIPT universe

The SIMSCRIPT language[1,2] is based upon a description of systems involving concepts denoted by entity, attribute, set, state, and event. In this language, these terms have been assigned the following meanings. Briefly, an *entity* is a class of objects described by a fixed collection of parameters called *attributes*. Individual members of an entity class have specific numerical values assigned to their parameters. *Sets* are collections of individual entities having certain common properties. The *state* of the model at any given instant is completely described by the current list of individual entities, their attributes, and set memberships. The dynamics of the system are represented by changes of state; that is, addition or deletion of individuals, change of attribute values, set memberships, or some combination of these. These changes take place instantaneously at discrete points in simulated time and are called *events*. The time at which an event is to occur is most frequently prescribed by SIMSCRIPT programming as current time plus some increment. The occurrence of the event is caused automatically by the SIMSCRIPT system at the prescribed time. That is to say, changes of state take place automatically and instantaneously (with reference to simulated time) at successive discrete points in time. At the conclusion of any event, simulation time is automatically increased to the time of the next event.

In order to indicate the nature of SIMSCRIPT programming, it is necessary to discuss some of the sub-categories of the concepts mentioned above. As a matter of convenience, both with regard to conserving computer storage space and ease of programming, entities have been separated into two categories, permanent and temporary. *Permanent* entities are known in advance to be present during the entire simulation. *Temporary* entities are of known form, but individuals, in general, appear and disappear during the simulation. There is a special kind of temporary entity called an *event notice* which is used to schedule future events, that is, changes of state. The distinction is made because event notices, as distinguished from other temporary entities, affect the automatic timing operations.

Sets may have 0, 1, or 2 subscripts specifying the number of entities with which the set is associated, that is, owner entities. That is to say, an unsubscripted set belongs to the system, a singly subscripted set belongs to one entity class, a doubly subscripted set belongs to a pair of entity classes. The members of the set may be ordered as FIFO, LIFO, or ranked on some attribute, high or low.

Events are of two kinds, endogenous and exogenous. *Endogenous* events are those internally generated by the system itself, *exogenous* are those imposed upon the system from the external

world. Simulations consisting only of endogenous events (aside from initialization and requests for reports of a kind which do not influence the simulation) are called *closed* simulations elsewhere, all others being called *open*.

It is worth noting at this point that continuous processes are apparently, but only apparently, excluded. The difficulty with continuous processes is not contained in the SIMSCRIPT view of the universe but rather in the fact that arithmetic processes performed by digital computers are finite. The resolution of the difficulty lies in the use of well-known numerical methods for solving differential equations. This point will be dealt with at some length later.

SIMSCRIPT programming

The examples which follow will serve to indicate some of the features of SIMSCRIPT programming. Consider a small neighborhood market assumed to have one checkout stand and a fixed number of carts, customers arriving according to a predetermined schedule, each with a shopping list. Also, the customer is assumed to have certain personal characteristics: the length of time he is willing to wait for a cart, and the length of time he is willing to wait in the checkout line. At the time the customer enters the simulation, his first act is to look for a cart. If he finds one, he immediately proceeds to shop, for a length of time determined by his shopping list plus a random increment. Once he finishes shopping, he moves on to the checkout stand. If the stand is not in use, he immediately checks out, this process taking an amount of time determined by the number of items plus a random number. When he is checked out, his cart is returned for further use, and he leaves. If, on arrival at the store, no cart is available, he waits in a FIFO queue until one does become available, in which case he proceeds to shop as before, unless his maximum waiting time is exhausted, in which case he leaves without shopping. If, on arrival at the checkout, the stand is in use, a similar process takes place. It is assumed, in the event he leaves without checking out, that his cart is returned for further use.

The information desired is of three kinds:

- Number of customers completely processed and the associated quantities of merchandise.
- Number of customers lost by cart shortage and their merchandise statistics.
- Number of customers lost at checkout and their merchandise statistics.

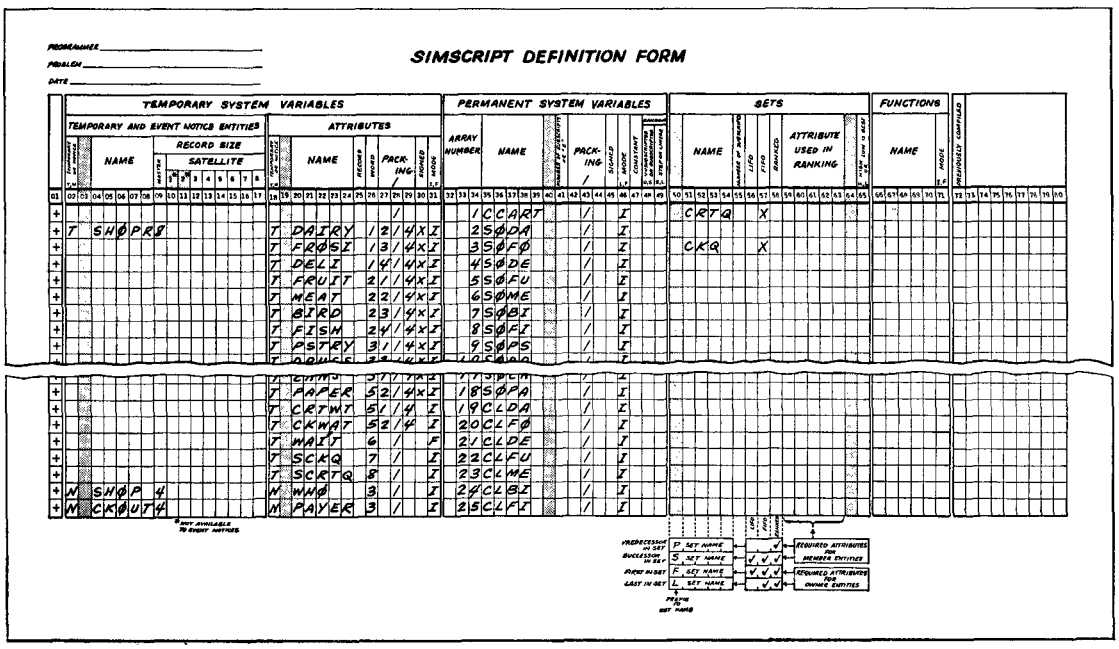
In this model there is only one temporary entity, aside from event notices, customers whose name is SHOPR, names being limited to five letters. A SHOPR has ten attributes: CANS, DAIRY, FROST, DELI, FRUIT, MEAT, and several other items as shown in Figure 1. Each of these attributes is used to contain the number of items of the various categories of merchandise on the shopping list. Provision is made to allow random generation of any of these numbers to account for impulse buying. Three

more attributes are CRTWT, CKWAT, and WAIT. The first two contain maximum waiting times for carts and for checkout. The last is used to contain the time at which waiting begins in either case. In addition to these thirteen attributes, there are SCRTQ and SCKQ. These are defined in order to be used by the SIMSCRIPT system in handling queues. They contain the names of the successors to this shopper in the cart and checkout queues respectively and are used only by the SIMSCRIPT system. For any sensible limitation on number of items and length of waiting time, SIMSCRIPT memory organization allows this information to be packed into an eight-word IBM 7090/94 record. (The SIMSCRIPT system was programmed for this equipment.)

There are two event notices. One is called SHOP and comes into being when a SHOPR begins shopping. It schedules the end of shopping procedure. It has one attribute, WHO, which is used to cross reference the record of a SHOPR. The other event notice is called CKOUT and comes into being when a SHOPR begins checking out. It schedules the end of checkout procedure. It has one attribute, PAYER, which is used to cross-reference the record of a SHOPR.

The definition form reproduced in Figure 1 shows how the above information is formally assembled. In a large measure, the form explains itself. It may be worth noting, however, that entity records may have from 1 to 72 words, consisting of a master record and as many as 8 satellites. Each may have 1, 2, 4, or 8 words, each word being capable of storing 1, 2, 3, or 4 items of information. These can be signed or unsigned, floating or integer,

Figure 1



with floating items being restricted to either a half or whole word. In this example, permanent system variables are used to provide storage for the number of carts currently available (CCART), or for various statistical quantities and other incidental information.

The endogenous event SHOP is described in detail to demonstrate SIMSCRIPT statements and events sequencing. The career of the customer begins with an exogenous event CUST which creates a SHOPR and loads his record appropriately. The same event creates an event notice SHOP and schedules it. The event exhibited below takes place on schedule.

```
    STORE WHO (SHOP) IN PRSN
    DESTROY SHOP
    IF(CKWT)EQ(O), GO TO 660
    LET CCK = CCK + 1
    FILE PRSN IN CKQ
    STORE TIME IN WAIT (PRSN)
    GO TO 760
660 STORE 1 IN CKWT
    CREATE CKOUT
    CALL CHEKT
    STORE PRSN IN PAYER (CKOUT)
    CAUSE CKOUT AT TIME + CT
760 RETURN
```

The first instruction retrieves the pointer to the SHOPR and stores it in temporary storage called PRSN. The second instruction is used here to return to dynamic storage the space used for the SHOP event notice record, just as a CREATE statement removes from dynamic storage the space for the record being created. The third statement investigates availability of the check stand. If not available, the next statement files the pointer in its appropriate place in the queue and adjusts the queue count. FILE (and REMOVE) involve dynamic storage allocation again. Present time is filed for later computation of the time spent by the SHOPR in the queue, and this part of the process ends. If the check stand is available, it is made busy. An event notice CKOUT is created in order to schedule the end of checkout event. A subroutine CHEKT is used to compute the amount of checkout time. The pointer for the SHOPR is stored, and the CAUSE statement schedules the end of checkout event at current time plus a time computed in the subroutine. To underline a point about sequencing, note that this event creates and schedules a next event. Obviously it could have created and scheduled any number of events.

The above routine illustrates all the SIMSCRIPT "entity operations" except CANCEL, REMOVE, and REMOVE FIRST. CANCEL deletes an event from the schedule, and may be followed by destruction of the event notice or by a new CAUSE if the event is to be rescheduled. REMOVE and REMOVE FIRST are set operations of obvious intent. Among the most noteworthy of the remaining SIMSCRIPT statements are the decision commands which may be modified by control phases. For example, the statement

```
FIND FIRST, FOR EACH PRSON OF CKQ,
WITH (CKWAT(PRSN))LE(12), AND (CANS(PRSN))
EQ(0), AND (FRUIT(PRSN))GE(6), OR (BIRD
(PRSN)) LS(2) AND (DRUG(PRSN))GR(3), IF NONE,
GO TO 50
```

has the following effect, if CKQ is a ranked queue: it goes through the members of the queue one by one, seeking the first member who meets at least one of the two conditions:

- The maximum he will wait in the check stand queue is less than or equal to twelve minutes; he is buying no cans; he is buying six or more fruit items.
- He is buying less than two poultry items; he is buying more than three drug items.

If such a member is found, the pointer to his record is stored in PRSN, and the program continues with the next statement.

Another pair of commands of great power are ACCUMULATE, COMPUTE. For example, COMPUTE A1, A2, A3, A4, A5, A6 = MEAN, SUM, SUM-SQUARES, MEAN-SQUARE, VARIANCE, STD-DEV OF CANS (PRSN), FOR EACH PRSN OF CKQ, which is self-explanatory.

Up to this point, no mention has been made of the report generator. The first step required in producing a report is to assign a name to it, whence it may be called as a subroutine. The format is specified on a report generator layout form, as in Figure 2. The name appears in the first line. The form indicates structure of the line, asterisks (*) representing variables to be inserted. The content line specifies which variables are in fact to be inserted.

Figure 2

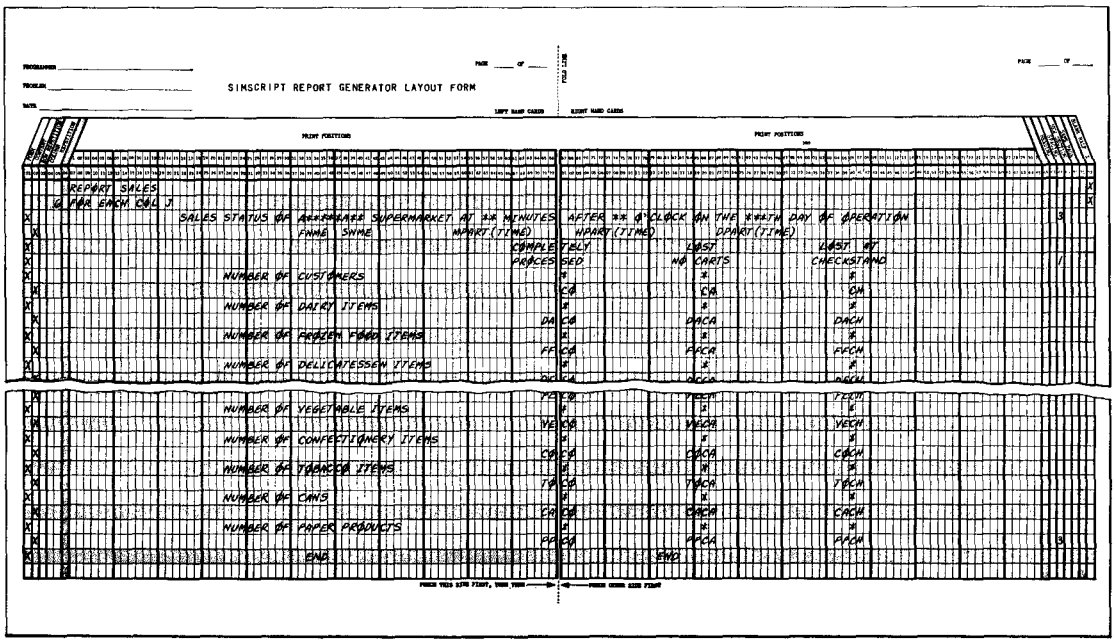


Figure 3 Report generated

SALES STATUS OF XYZ SUPERMARKET
AT 18 MINUTES AFTER 11 O'CLOCK ON THE 7TH DAY OF OPERATION

	COMPLETELY PROCESSED	LOST NO CARTS	LOST AT CHECKSTAND
NUMBER OF CUSTOMERS	387	52	93
NUMBER OF DAIRY ITEMS	729	204	312
NUMBER OF FROZEN FOOD ITEMS	587	62	51
NUMBER OF DELICATESSEN ITEMS	79	102	93
NUMBER OF FRUIT ITEMS	487	62	153
NUMBER OF MEAT ITEMS	78	109	36
NUMBER OF POULTRY ITEMS	983	326	487
NUMBER OF FISH ITEMS	158	62	39
NUMBER OF PASTRY ITEMS	79	0	3
NUMBER OF DRUG ITEMS	426	39	158
NUMBER OF HARDWARE ITEMS	627	128	79
NUMBER OF LIQUOR ITEMS	36	72	58
NUMBER OF PERIODICALS	196	212	108
NUMBER OF VEGETABLE ITEMS	47	63	22
NUMBER OF CONFECTIONERY ITEMS	8	12	15
NUMBER OF TOBACCO ITEMS	386	72	204
NUMBER OF CANS	119	33	47
NUMBER OF PAPER PRODUCTS	59	27	93

The repetition specifications have to do with printing arrays. Thus in the example, the command CALL PROFIT, provides for a report as indicated, including an array whose contents may be specified by the user (at execute time). Figure 3 shows a report produced by this model. Obviously, it is extremely simple to insert special reports for tracing and snapshots. This means, of course, that debugging both the program and the model in the source language is no problem.

The model discussed above is useful only as an illustrative device to introduce SIMSCRIPT concepts. Another supermarket model includes the entrance of the customer as an endogenous event, and the number of check stands as automatically controlled by the volume of business on hand, within limits. A larger variety of statistics is gathered concerning queues at the check stands, and cart usage. In particular, a report is made showing the number of carts in use as a function of time. An entry appears in this table for each change in the number, opposite the time at which the change occurs. The programming of the report is quite simple. An array of proper size is provided, and entries are made in successive positions in that array. With the assistance of a counter, this report is CALLed when the array is full, or when the simulation ends.

continuous models

Returning to the question of continuous models, suppose now that a model involves a set of differential equations as well as a discrete structure. For the moment also suppose that the simplest possible integration technique is adequate. That is to say, given the values of the variables at a point t in time, the values of those variables at a time $t + \Delta t$ is obtained by increasing the values of the variables in proportion to their slopes at time t, the factor of proportionality being Δt. (This is the kind of model to which the DYNAMO system[3] addresses itself, and the integration technique used there.) The SIMSCRIPT technique for dealing with this situation is as follows:

1. The exogenous event which begins the simulation also creates an event notice (perhaps called STEP) and CAUSEs it (that is, schedules it) at TIME + DT, DT being fed in at execute time. It also computes and stores initial slopes.
2. The event routine (STEP) goes to a subroutine to compute updated values of the variables and their slopes, and again CAUSEs STEP at TIME + DT.

The effect of this is to cause a single step to take place every DT in time. If values of x are required for times other than those arising in the integration, a simple interpolation suffices. Output of tables of functions against time can be obtained by the device indicated above for arriving at a chart of cart usage against time.

There are problems inherent in this approach to continuous models which imply the necessity of a more flexible arrangement. The most obvious of these problems is that sufficient accuracy with this kind of integration formula may require an extremely small time interval, hence an inordinate amount of computer time; it is quite possible, in fact, that no time interval exists which provides a specified accuracy of solution. Resolution of such a difficulty may require introduction of a higher order, that is, more precise, integration method. This is, in fact, the reason for subroutinizing the integration process in (2) above. It is a simple matter, once arranged this way, to replace the integration subroutine.

Another case involves models in which a continuous process may be characterized by a set of parameters that eliminates the need for integration. For example, the motion of a satellite may be described by differential equations of motion, and the flight path computed by integration. For certain purposes it may also be described by certain sets of equations, the equation of the ellipse among others, so that position for any specified time, or vice versa, can be computed directly. In a case such as project Mercury, where the requirement may be to find time of arrival at some dozen positions around the earth, this would have the effect of replacing thousands of events involving integration by a dozen events involving evaluation of some formulas.

All of these problems, in addition to problems which arise when there are discontinuities in the solutions or their derivatives, such as those introduced in the various phases of a missile launch, are quite easily handled in the SIMSCRIPT language. This is not to say that the associated mathematical problems are easy. The difficulties associated with these problems are simply not resolvable by programming techniques. However, once they have been resolved by other means, SIMSCRIPT makes it easy to program the result.

processing and performance characteristics

Once the definition and report generator forms are completed, event routines are written, and an events list is made up, cards are punched from all this preparatory to translation. SIMSCRIPT translation then takes place on a 7090/94 under the control of the FORTRAN II monitor. This translation produces a FORTRAN input tape.

Wherever it finds SIMSCRIPT type errors, it incorporates devices into the FORTRAN program to force diagnostics—it also prints SIMSCRIPT diagnostics on line. Examples of such errors are: improper punctuation, missing parentheses, certain misspellings.

The next process is compilation of the FORTRAN program into machine language, under control of the FORTRAN II monitor. This, of course, leads to standard FORTRAN output. It may well be noted that all options available in FORTRAN are also available in SIMSCRIPT, that is, the input deck to the SIMSCRIPT translator may be entirely in SIMSCRIPT or a mixed SIMSCRIPT-binary deck. The user also has the option of requesting, through control cards, a translation and compile only or a translation-compile-execute run. In the latter case, an input data deck must also be provided. The first few cards of the data deck must be of a specified form expected by the housekeeping section of the program as provided by the SIMSCRIPT translator.

The function of these cards is to allow initialization of all permanent system variables; in particular, this allows the user to specify array size at execute time rather than compile time.

Corrections can conveniently be made either at the SIMSCRIPT or machine language level. SIMSCRIPT compilation is arranged in conjunction with segmentation of the SIMSCRIPT deck in such a manner that only those segments which have been changed need to be recompiled. For example, each event routine and each subroutine is a segment, as is the deck of definition cards produced from the definition forms; so that if one event routine is changed only that routine needs to be recompiled, and so forth. Since binary corrections are made in a binary deck produced by the FORTRAN compiler, they are made in the customary manner.

In some cases, it may be desirable to let the computer analyze the results of previous runs (to determine the value of policy parameters for a next run) and then make this next run immediately without getting off the machine. To accomplish this, the calendar may be cleared of coming events, as described elsewhere,[4] then time reset with the statement

```
LET TIME = 0.0
```

In large programs which require more than one core-load, and in "games" in which human participants interact with computer runs, the RECORD and RESTORE statements are useful. The statement

```
RECORD MEMORY ON TAPE J,
RESTORE TO 30
```

puts a complete snapshot of core and registers on the specified tape. When a RESTORE FROM TAPE K statement is executed with tape K positioned at the start of the previously RECORDed file, then the machine will be set up as it was when the RECORD statement was executed, except that control is first given to statement 30. The FORTRAN chain feature is also available.

Some 25 SIMSCRIPT programs in the areas of logistics, manufacturing, medicine, and computer systems were examined. It turns

out that for most of these simulations, execution time is in the neighborhood of 10,000 events per minute. This assumes that event routines are of "reasonable size" (which is usually the case), that the logic is not unusually complex with respect to loop structure, and that there are not too many long set searches. It is to be noted in this connection that complex simulations do not normally owe their complexity to the existence of such conditions, but rather to the existence of a larger variety of events of normal size.

The statistics for compilation time show that SIMSCRIPT compilation normally takes about one third as long as the subsequent FORTRAN compilation. It should be remarked here that if the simulation program were written directly in FORTRAN, the time for compilation of the program would probably be larger than the time for the double compilation required for SIMSCRIPT. The reason for this is that the timing and initialization routines, among others, are provided in binary form. The SIMSCRIPT compiler only retouches them to fit the specific problem at hand, a process which takes very little time, particularly as compared with the FORTRAN compilation of complete routines of this complexity.

As for utilization of storage space, dynamic storage allocation obviously provides substantial advantages with regard to both utilization of available space and speed of execution, since it obviates any necessity for reshuffling storage and wastes no storage capacity on non-existent entities or set members.

Space is also conserved by the flexibility provided for structuring entity records and by the fact that dimensions are not specified until execute time.

The storage requirements of the SIMSCRIPT control routines during execution are as follows: Two thousand cells are required during initialization and are then returned to the pool for temporary entities. Another two thousand cells are required for the other SIMSCRIPT chores, such as control of the dynamic storage allocation process, the exogenous events buffer, random number generator, etc.

Enough teaching experience has been accumulated to state that most persons in this field with no other programming experience can be taught SIMSCRIPT in from 30 to 40 hours of instruction, which includes writing one or two small sample programs.

Those who have learned SIMSCRIPT say that it has helped to structure their view of the system to be simulated. That is, the manner in which SIMSCRIPT looks at a system to be simulated is in itself an aid to model building.

In a trivial sense, SIMSCRIPT can accomplish anything the computer can accomplish since it can do anything FORTRAN and FAP can do. In practice, FORTRAN and FAP code is rarely used. It is significant to note that, to date, no programmable simulation problem has been proposed to the authors which has not been amenable, in rather straightforward fashion, to SIMSCRIPT treatment.

CITED REFERENCES

1. H. M. Markowitz, B. Hausner, and H. W. Karr, SIMSCRIPT, *A Simulation Programming Language*, The RAND Corporation, 1963, Prentice-Hall, Inc., Englewood Cliffs, N. J.
2. Compiler program: SHARE Distribution Number, 3031, RSSIMS, Program Distribution Center, P. O. Box 790, White Plains, N. Y.
3. J. W. Forrester, *Industrial Dynamics*, The M. I. T. Press, John Wiley and Sons (1961) New York.
4. B. Hausner and H. M. Markowitz, "Technical Appendix on the SIMSCRIPT Simulation Programming Language," Memorandum RM-3813-PR, August 1963, The RAND Corporation.