DESIGN CONSIDERATIONS FOR A USER ORIENTED DISCRETE-EVENT SIMULATION LANGUAGE

Mil Basom, Rafael A. Lizarazu, Robert Scott Pallack, Dimas K. Sanchez, Paul Schluter, M. L. Schneider, Computer Systems Program, Florida Atlantic University, Boca Raton, Florida 33431.

ABSTRACT

Discrete event simulation languages can be described in terms of simulation perspective and language structure. A useful language should allow an emphasis to be placed on transactions, queues, and item flow. The language structure should be self-documenting, readable, and descriptive rather than algorithmic in philosophy. SIMFO, a newly designed language, provides the necessary definition of a system using a COBOL-like structure, and complying to the above requirements. A system can be defined in terms of the items, facilities, servers, queues, and item flow. Branching involves only an IF - - - THEN construction. Aside from summary information, all analyses are performed off-line.

INTRODUCTION

With the increasing use of discrete event simulation techniques by many users, some unfamiliar with programming techniques, a descriptive, easy to use language is needed. In examining other languages, two factors should be examined: the structure language and the simulation perspective.

Shannon [1] considers four categories of discrete event simulation perspectives: activity (CSL, ESP, FORSIM IV, GSP, and MILITRAN), event (SIMSCRIPT, GASP, SIMCOM and SIMPAC), process (SIMULA, OPS, SOL), and transaction flow (GPSS and BOSS). Most of these languages emphasize events or the transaction. In some languages, however, queue control plays an important function.

The other consideration is language structure. In most cases, the simulation language is pseudo-algorithmic. For example, SIMULA employs ALGOL structuring, SIMSCRIPT belongs to the FORTRAN set of languages, and GPSS, to the non-user, appears to be an assembler. These language structures, without additional documentation, convey little information about the model to the non-programmer.

LANGUAGE PHILOSOPHY

In order to produce a more usable simulation language, a new approach should be developed, employing a descriptive, rather than an algorithmic approach. This philosophy should furnish the non-sophisticated user with a simple-to-use, yet comprehensive discrete event simulation language. At the same time it should provide the experienced user with a powerful tool for solving complex problems. The two basic requirements should allow flexible simulation perspectives, and should be self-documenting and readable.

Since different applications may require the emphasis to be placed upon the facilities, and others, on the queues, the language should allow the user the ability to implement either approach. In addition, multi-facility, parallel processing and sequential processing should be available with a minimum of effort.

A visable indentation structure, such as in a properly written ALGOL program, is an important aid in program documentation. Although the philosophy of a user oriented simulation language should be descriptive, rather than algorithmic, it is desirable to implement such a structure. This can be accomplished by requiring lower level statements to be indented in relation to the statements which they modify. To reduce coding errors, free format constructions for statements and tables should be permitted whenever they do not conflict with indentation requirements.

Readability of the program is improved when the user can assign meaningful identifiers to items, facilities, attributes, etc. These identifiers should consist of up to thirty characters (chosen from the set A-Z, 0-9, -). This length provides the best compromise between length and information content.

In accordance with the philosophy: "what the user doesn't know, won't hurt him," there should be no reserved word which would preclude its use as an identifier. The translator, or compiler, should, however, issue a warning whenever a user employs a keyword in another context. This facility of the language structure can be extended, permitting the multiple use of identifiers in different contexts (item, attributes, facilities, etc.) without causing any internal ambiguity. However, an indication of the multiple use of identifiers should be given to the user.

Meaninful keywords should be chosen in order to maintain readability. Certain optional noise words, such as those in COBOL, are required in order to improve documentation while retaining the ability to produce a terse program.

Even in self-documenting languages, comments are required. This feature can be provided by two delimiters: the first denoting the end of line, and the second the end of the statement. The syntactical scan for any line can be terminated by the first delimiter, thus allowing comments to be placed on each card. This implies that if the delimiter is placed in the first column, the entire card contains comments information.

To require the user to be concerned with continuation cards can lead to numerous errors. Thus, each statement should end with the second type of delimiter. This is in agreement with the indentation philosophy, which may require many cards when the indentation level is high.

External input is required whenever systematic modifications of the system are made. Thus, the ability to supply input parameters and their values in an important requirement. Data directed input provides the easiest, most fool-proof method for the proposed user.

LANGUAGE STRUCTURE

In order to implement the above philosophy, a new simulation language, SIMFO, has been designed. The simulation method has been described elsewhere [2]. The language structure is COBOL-like, containing paragraphs and sentences.

Statements must appear in columns 10-72, with indentation levels every five spaces. Within each statement, a fixed starting position, aside from the required indentation, is not important. Identifiers conform to the above requirements.

A simulation can be described in eight sections:

DEFINE SYSTEM - All specifications regarding the general system characteristics are defined. For example starting time, checkpoints, and end of simulation criteria.

DEFINE ITEM - The characteristics of each item are specified in this section, including arrival time distribution, and attributes. There is one subsection for each item type in the system.

DEFINE FACILITY - Detailed information pertaining to the facilities used in the model must be specified. These would include server requirements, down time, and number of facilities.

DEFINE SERVER - A facility may require manpower, outside equipment, or additional,reusable items for its operation. These pools are defined with their attributes in this optional section.

DEFINE QUEUE - Although the system will define queues based upon the facility's attributes and entry requirements, the ability to specify queue structures is available.

DEFINE TABLE - Each of the above sections may require information in a tabular form. These tables are placed in a separate section in order not to distract from the internal structure and readability of the other definition sections.

DEFINE FUNCTIONS - This optional section allows the user to define any special functions employed; the rational is the same as for the tables.

SYSTEM FLOW - Because a different use of the above defined items, facilities, servers, and queues are described in this section, a break in the above DEFINE . . . pattern is employed to emphasize this fact.

This section provides the user with the ability to direct the item flow pattern, and changing the pattern if desired.

The DEFINE SYSTEM section has two parts, the start time, and simulation controls; and the checkpoint, restart, file and summary features. The system time span is controlled by the first two specifications even though a particular SIMFO program may have, sufficient information for a longer period. There are two ways a simulation can be completed: first, by the simulation time reaching the end time, and secondly, after a specified number of items have been generated. The second set of features are self explanatory; the checkpoint control can be described similarly to the end simulation.

In order to describe a particular item, and its characteristics, the DEFINE ITEM section is used. This section contains a name (optional), arrival description and attributes of an item. The arrival description can be based upon one of the following criteria: 1) a constant arrival rate, 2) by distributions that are defined a table or 3) values defined by a function. Common distributional functions are provided, as well as the ability to define specific functions.

A class is a type of activity, for example, a check-out-counter, employing one or more facilities; each of which has specific attributes. There may be more than one facility with the same attributes but different identifiers. The DEFINE FACILITY will have: a name, the number of facilities, the time it is "down", with values defined in the same way as for an item's arrival and the time an item remains in the facility as a function of its attributes.

The number, and attributes of servers are defined in the DEFINE SERVER section. Server information includes the number of servers, as well as general information controling their availability and attributes.

If the user wishes to specify a special queue, or wants additional control over the queues, then the DEFINE QUEUES section is employed. Control parameters include 1) the name of facilities for each queue, 2) one queue for each facility, 3) all facilities use one queue, 4) normal default queues, 5) do not allow a facility to use this queue, 6) define queue entry conditions without requiring a facility. Priority of items in each queue can be specified in this section.

The second set of optional sections are the DEFINE TABLE and DEFINE FUNCTION. Tables are entered by defining the information order, and entering the values as sets of n-tuples. Functions can be defined using a number of available system keywords, as well as tables. Their structure is FORTRAN-like; external functions are also allowed.

The flow of items through the system is facilitated by a series of statements that control their movement through time, during the simulation process. To accomplish this objective the user moves items from class to class and within each class, specifies a specific facilities.

The SYSTEM FLOW selection depends upon three factors: 1) the entry conditions of the item, represented by the constraints specified as characteristics of that class; 2) the item atributes specifications, which determine the queue and facility to be used; 3) the exit conditions which specify deviations from the normal flow.

Parallel system flow is controlled by a block structure, bracketed with a begin and end. This provides a readable structure for such processing.

Priorities of queues and facilities can also be defined. Thus, full priority specifications can be defined, both within the queue and between queues.

In the automatic mode, facilities with the same attributes use a common queue. However, if the DEFINE QUEUE is employed, it would override this feature. Thus, the emphasis can be changed from events to queues.

```
* A MODEL OF A BANK WITH SIX TELLERS
* ONE TELLER IS FOR DEPOSIT ONLY
          DEFINE SYSTEM.
               START TIME=1.
               END SIMULATION WHEN CUSTOMER=INPUT(NCUSTOMER).
*                   INPUT SIMULATION TERMINATION INFORMATION
               CHECKPOINT= 10 TIME UNITS.
          DEFINE ITEM.
               ITEM CUSTOMER.
                    ARRIVE EVERY 2 TIME UNITS, RANGED POISSON.
                    ATTRIBUTE TRANS-TYPE, VALUES
                         DEPOSIT 25 *PERCENT
                         CHECK   75.*PERCENT THESE THREE LINES ARE
*                                    ONE STATEMENT, BUT IS MORE
*                                    READABLE
          DEFINE FACILITIES.
               FACILITIES FULL-SERVICE(5), DEPOSIT-ONLY.
                    TRANSACTION TIME FOR CUSTOMER.
                         FOR CHECK 10 NORMAL 2.
                         FOR DEPOSIT 5 NORMAL 1.
*         SERVER, TABLE QUEUE, AND FUNCTION DEFINE SECTIONS ARE
*         NOT NEEDED - THEY ARE SET BY THE SYSTEM
*
          SYSTEM FLOW.
               CLASS TELLER.
                    ENTER FACILITY FULL-SERVICE.* NO CONDITIONS
                    ENTER FACILITY DEPOSIT-ONLY FOR DEPOSIT.
          END SIMFO.
```

CONCLUSION

SIMFO is a newly designed discrete event simulation language which has a high degree of flexibility. Since the language is intended for the non-sophisticated user, it is highly descriptive. The structure is COBOL-like with an indentation format. With full freedom in the choice of keywords and identifiers, the resulting program self-documenting and easily understood. A wide range of simulation viewpoints makes this language applicable for a large number of systems.

REFERENCES

1. R. E. Shannon, Systems Simulation, the Art and Science, pp 120-121, Prentice-Hall Inc., 1975.

2. M. L. Schneider, An Information System Based, Structured, Discrete Event Simulation Language, ACM Computer Science Conference, Anaheim, Feb. 1976.