Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

C Based discrete event simulation support system

SATHYAKUMAR SELVARAJ ERIC L. BLAIR MILTON L. SMITH WILLIAM M. MARCY

DEPARTMENT OF INDUSTRIAL ENGINEERING
TEXAS TECH UNIVERSITY
LUBBOCK, TEXAS

1. ABSTRACT

The **C** programming language is gaining in popularity with professional programmers because of its powerful constructs, versatility and portability. The state of development of **C** compilers has advanced to the degree that applications generally require less memory and execute more quickly when coded in **C** compared to other languages such as **BASIC**, **FORTRAN**, or **Pascal**. **DISC** (**DI**screte event **S**imulation in **C**) provides an environment for creating and executing discrete event simulation models with event routines written in **C**. It provides a library of simulation support functions similar to those provided in **FORTRAN** based simulation languages such as **GASP**, **SLAM** and **SIMAN**. The effective implementation of **C** language capabilities in **DISC**, which are pertinent to simulation, results in the following advantages: dynamic memory management, interactive execution environment, interactive trace, powerful debugging facility, flexible input-output and faster execution.

1. INTRODUCTION

The ease with which large complex systems can be modeled and simulated is highly dependent upon the features and capabilities of the simulation software employed. The selected programming language for the model development should be conceptually simple yet possess powerful features [4]. **C** is a simple, flexible and powerful language which has become the most important language in industrial applications and software development [5]. **C** is particularly popular for programming applications for microcomputers but **C** compilers are also available for mini and mainframe computers. The design of **C** makes it easier to incorporate top-down planning, structured programming and modular design in simulation models. The capabilities of **C** which are most pertinent to simulation applications are:

1. Dynamic memory management,
2. Powerful data handling support using physical pointers,
3. Modular software construction support,
4. Execution time interaction support,
5. Flexible and standard Input/Output support,
6. Portability, and
7. Maintainability and extensibility support.

The simulation support system described here is called **DISC** (**DI**screte-event **S**imulation in **C**). **DISC** provides an environment for creating and executing discrete-event simulation models in which the event routines are coded in **C**. It is designed to be similar to **FORTRAN** based simulation languages such as **GASP**, **SLAM** and **SIMAN** [1,2,3]. In **DISC**, the **C** functions required to support discrete event simulation are organized into library modules. The eight task modules of simulation in **DISC** are:

1. Executive controller tasks,
2. Information storage and retrieval tasks,
3. Dynamic memory management tasks,
4. Data collection and statistical reporting tasks,
5. Model tracing and monitoring tasks,
6. Interactive-execution tasks,
7. Model Input-Output tasks, and
8. Random deviate generation tasks.

By using the various functions included in these modules, a real system can be modeled by specifying the entities, attributes of the entities and the events. The event logic which describes the interaction between the entities is coded in **C**. Entities are created and stored as **C** data structures in which the attributes are stored. Associated with each event is an event entity. The event type and the time of occurrence of the event are the primary attributes of the event entity.

In **DISC**, different types of entities, with differing numbers of attributes, can be represented by a single data type called **structure record** (**RECORD**). The **RECORD** has been defined in such a way that the number of attributes in a **RECORD** can be varied according to the needs of the model entities. A grouping of entities (**RECORD**s) based on some common characteristic is called a file or list. The data structure which has been used to represent the file is a dynamic doubly linked list. The **RECORD**s which are

physically located in different parts of the computer memory, are connected through the physical pointers which have the values of the memory addresses. The pointer connections make the **RECORD**s appear as a continuous chain.

2. EXECUTIVE CONTROLLER MODULE

To implement the overall control logic of a simulation, a hierarchical control structure is required. In this hierarchical scheme, the executive controller has the highest level of control. The **C** function **executive** contains the logic necessary to run the simulation. It is contained in the executive controller module. When control passes to the **executive**, from the user-written **main** program, this function takes charge and controls the overall simulation. The **executive** updates the simulation clock, places the event entities on the simulation calendar (the event file), calls the event functions in the proper chronological order, and passes control to routines in the various modules to perform the required tasks.

3. INFORMATION STORAGE AND RETRIEVAL MODULE

The information storage and retrieval module provides various functions for file manipulation. **RECORD**s in a file can be scanned and their attributes accessed and changed if desired. The file manipulation functions can also be used to add a **RECORD**, remove a **RECORD** or re-arrange the order of **RECORD**s in a file. Fast access to information has been achieved by utilizing the powerful data handling support provided by **C**'s pointer syntax.

4. DYNAMIC FILE RANKING

When adding a **RECORD** to a file, the "file ranking rule" is used to prescribe the placement of the entering **RECORD** into the file. In **DISC**, the file ranking is dynamic in that the ranking rule used to order the file can be changed during simulation by changing the input parameter values passed to the **RECORD** adding function. Therefore, file ranking is not fixed and need not be declared before starting the simulation. The six options given to order a **RECORD** added to a file are:

1. Position according to FIFO order,
2. Position according to LIFO order,
3. Position according to a specific attribute value (ascending or descending value),
4. Position according to the ascending order of time of occurrence attribute (event file),
5. Position according to a specified rank (e.g., place **RECORD** "third" or last in the list), and
6. Position according to FIFO within a priority group.

The function which adds a **RECORD** to a file can be called in six syntactically different ways with a lesser number of parameters to accomplish each of the six ranking rules. Functions are also provided to re-arrange the current **RECORD**s in a file according to a different file ranking rule and to access the memory address of a particular **RECORD** (either by specifying the rank or a specific condition), for the purpose of modifying the attribute values of the **RECORD**.

5. DYNAMIC Memory Management Module

The number of attributes needed to describe a model entity may vary with each different entity type. A simulation language should be capable of accommodating a variable number of attributes depending on the **RECORD** type being created. This language feature is easily implemented in **DISC**. When an entity is created in a **DISC** simulation, memory is allocated dynamically to a **RECORD** using the standard **C** library function calloc(). Thus, the size of a **RECORD** need not be fixed to accommodate the maximum number of attributes over all the **RECORD** types. (as with **FORTRAN** based languages such as **SLAM** and **SIMAN**). Similarly, the maximum number of entities that will be created and used in the model also need not be specified prior to simulation. When an entity leaves the model, the associated memory of the **RECORD** will be freed.

The required number of files needed for the simulation model will also be dynamically created during the start up of the simulation. The data collection facility, which collects data to provide statistical reports, will also be dynamically created to accommodate the number of model variables for which the statistical information is required.

6. INTERACTIVE EXECUTION MODULE

A key feature of **DISC** is the ability to interact with the simulation model as the run develops. During execution, the simulation can be temporarily suspended and the current system state can be viewed and analyzed by the user. This provides the user with the flexibility to check the values of the system variables and take appropriate action during execution, using the various options provided.

The interactive execution module provides the option of advancing the simulation, event by event, to check the values of the system variables after the occurrence of each event. This option can be used as an interactive trace for model debugging. It is also possible to use the interactive feature to examine the progress of important model statistics as the run evolves. In this way, the simulator can determine the point at which transient effects have dampened out and

can assess the sufficiency of sample size for producing confidence interval estimates of the desired precision.

Interaction is accomplished by scheduling interruptions to the simulation run. When an interruption occurs, the following set of seventeen options are provided to the user to check, change, print and/or dump the current system status:

1. Show all the **RECORD**s and associated attributes of a file,
2. Show all the file statistics,
3. Show all the COLLECT variable (based on observation) statistics,
4. Show all the TMST (time persistent) variable statistics,
5. Show all the user defined variables,
6. Schedule a future interrupt for the next interactive execution module,
7. Insert a new simulation stop time,
8. Print the current model results into an ASCII file,
9. Print/dump the current system status into an ASCII file,
10. Modify an attribute value of a **RECORD**,
11. Create and add a new **RECORD** to a file,
12. Delete a **RECORD** from a file,
13. Rearrange all the **RECORD**s in a file according to the new specified file ranking rule,
14. Find a confidence interval for a COLLECT variable using the batch means method, and
15. Change the status of the system variable.
16. Stop the simulation now,
17. Continue the simulation.

Using the first five options, the current system status can be reviewed and checked. Using option 6, a next call to the interactive execution module can be scheduled. After viewing the simulation status, the analyst can change the stopping time of the simulation using option 7. Using option 9, the current system state can be noted and recorded into an ASCII file. Subsequent runs can be initiated from a previously saved system state. This can reduce the transient phase of replicated simulation runs. Using option 11, new entities can be created and loaded into the model and their effect on the system can be studied. Using option 13, the order of **RECORD**s in a file can be changed according to a different order ranking rule. This option could be useful, for example, when comparing different scheduling strategies in a simulation model. Option 14 will produce a confidence interval for a COLLECT variable using the batch means method. If more observations are needed to obtain a sufficiently narrow confidence interval, the simulation stop time can be extended. Option 15 is used to toggle the status of the global system variable **event_mode** from "ON" to "OFF" and vice versa. When **event_mode** is "ON", the simulation will run in "event by event" mode with interruptions interleaving the sequential events. Thus, a flexible trace and debugging facility is provided to check the corresponding changes in the system variables, after the occurrence of each event.

7. METHODS TO PRODUCE INTERRUPTS

Interactive execution of a simulation model can be implemented by producing interrupts obtained through four methods. In the first method, an interrupt can be produced from the key board by pressing the "ESCAPE" (Esc) key, at any time during the execution of the simulation model. The second method uses a **C** function **"pause()"** which can be called from any user-written event routines. The third method is to schedule an interrupt to occur at a time specified during the initial model input phase. In this method, an interrupt is scheduled to occur at a specific future time from the user-written event routines.

8. INPUT/OUTPUT MODULE

When starting the simulation, three options are provided to enter data to the simulation model. Using the first option, the data can be entered interactively through the key-board, in response to a sequence of menus and data entry windows. With the second option data is read directly from an input file consisting of **DISC** input statements. The third method is to load a file created at the termination of a previous simulation run through the interactive facility offered by **DISC**. Using this option, the simulation model state can be initialized to begin where the previous run left off.

9. ADVANTAGES OF DISC

C has already become a dominant language in software development for microcomputers. Developing simulation models coded in **C** allows the simulation analyst to take full advantage of the power and flexibility of the **C** language. The **DISC** simulation software has capitalized on these advanced language capabilities in two ways. First, **DISC** allows the programmer to write event routines in **C**. Secondly, **DISC** is written entirely in **C** and uses the properties of the language to achieve the following features:

Dynamic Memory Management: Memory can be dynamically allocated according to the needs

of the program during program execution. Memory requirements need not be specified prior to simulation. There is no limiting statement to specify the maximum number of concurrent entities in a file or in the model. Thus, computer memory can be more effectively utilized. This will become important under the multitasking/multiuser operating systems such as UNIX and OS/2.

Variable Size Record: In a typical simulation model, there are a number of different entity types each with a different number of attributes. In **DISC**, each entity **RECORD** will occupy only the memory necessary to store the attributes for its entity type.

Dynamic File Ranking: In **DISC**, file ranking is dynamic. The file ranking rule used to place entities into files need not be specified prior to simulation. The rules used for each file can be changed during the simulation and entities currently in a file can be rearranged at will.

Input-Output Flexibility: In DISC, data input and model initialization can be accomplished either from interactive keyboard input, read directly from a user created disk file or read from a file created to save the model state and system parameters at the end of a previous simulation run. Simulation results can also be written to a user specified ASCII file in either a standard default format or in any appropriate form specified by the user.

Interactive-Execution: During execution, simulation can be temporarily suspended by producing interrupts. Using the several options provided by **DISC** the current system status can be viewed and changed. By selecting the **event_mode** option, the simulation can be traced, event by event, and the corresponding changes in the state variables can be checked. This is a very useful tool for debugging and validating the simulation model.

General Advantages: Since **DISC** is written entirely in **C** the user can make use of sophisticated data structures and define his/her own data types using structures; these constructs of structured programming are fully supported by **DISC**. Since **DISC** uses the standard C library functions it is portable across hardware and operating systems. Preliminary experience indicates significant reductions in both execution time and memory requirements when compared with **SLAM** and **SIMAN** versions of the same models.

REFERENCES

1. Pritsker, A. A. B., The GASP IV Simulation Language, John Wiely (1974).

2. Pritsker, A. A. B., Introduction to Simulation and SLAM, John Wiely & Sons, (1986).

3. Pegdon, C. Dennis, "Introduction to SIMAN", Systems Modeling Corporation, Pennsylvania.

4. Hopper, James W., "Strategy-related characteristics of discrete event languages and models," SIMULATION, pp.153-159, April (1986).

5. Kernighan, Brian W. and Dennis M. Ritchie, "The C Programming Language," Prentice-Hall (1978).

Sathyakumar Selvaraj is a Ph.D. student in the Industrial Engineering Department of Texas Tech University. His primary research interests are in the development of simulation languages and the application of simulation modeling to manufacturing systems. Sathyakumar is a big LAKERS fan and particularly enjoys watching Magic Johnson shoot the three point shot.

Eric L. Blair is an Associate Professor of Industrial Engineering at Texas Tech. His research includes the computer simulation modeling of manufacturing and service systems. Eric is the only serious scholar in the group and, coincidentally, wrote all four bio sketches.

Milton L. Smith is a Professor of Industrial Engineering at Texas Tech. His primary research interests lie in the modeling and design of High Tech manufacturing systems with particular attention to scheduling and resource planning. Milt is known to his friends as "the Prince of Pun" for his outstanding, but strange, sense of humor.

William M. Marcy is a professor and director of the Division of Computer Science of the School of Engineering at Texas Tech. His major research interests are in the area of automation and control of manufacturing systems. Bill is an avid sportsman and believes that "gun control" is using both hands to hold the pistol.