*Proceedings of the 1991 Winter Simulation Conference*
*Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)*

# SIMSCRIPT II.5 AND MODSIM II: A BRIEF INTRODUCTION

Dr. Edward C. Russell

Russell Software Technology
1735 Stewart Street
Santa Monica, California 90404

## ABSTRACT

The SIMSCRIPT II.5 and MODSIM II programming languages are described. SIMSCRIPT II.5 with its integrated graphical interface, SIMGRAPHICS, substantially reduces time and effort in simulation model development. Its English-like syntax improves readability of the code and. substantially reduces the need for documentation. MODSIM II is a modern object-oriented language with built-in support for simulation that also has an integrated graphical interface. It is a compiled language which is available for most systems including mainframes, work-stations and PC's.The built-in object-oriented constructs of MODSIM II include single and multiple inheritance, dynamic binding of objects, polymorphism, data abstraction and information hiding.

## 1 INTRODUCTION TO SIMSCRIPT II.5

SIMSCRIPT II.5 is a well established, standardized, and widely used language with proven software support. It assists the analyst greatly in the formulation and design of simulation models and gives the programmer and analyst a common language for describing the model. The benefits of using SIMSCRIPT II.5 can be felt at all stages in the development of a model, including:

The powerful "world-view" consisting of Entities, Attributes, and Sets provides a natural conceptual framework in which to relate real objects to the model.

The modern, free-form language contains structured programming constructs and all the built-in facilities needed for model development. Model components can be programmed so as to clearly reflect the organization and logic of the modeled system.

A well-designed package of program testing facilities is provided. Tools are available to detect errors in a complex computer program without resorting to memory dumps and other archaic means.

The SIMSCRIPT program structure allows the model to evolve easily and naturally from simple to detailed formulation as more information becomes available. Many modifications, such as choices of set disciplines and performance measurements are simply specified in the program preamble in a non-procedural manner. Animation and presentation graphics can even be changed without program modification.

The powerful English-like language allows for modular implementation. Because each model component is readable and self-contained, the model listing can be understood by the end-user who may not be at all familiar with programming. Because the detailed model documentation is the program listing, it is never obsolete or inaccurate.

### 1.1 Overview

The purpose of a simulation must be clearly articulated before embarking on model development. Many modelling efforts have been doomed to failure, because a clear goal was never determined. The natural tendency is to model in great detail that part of the system which is well understood and "sweep under the rug" or over-simplify those parts which are not understood. The detailed model of the well understood parts yields many lines of model code and gives the illusion of great progress, when in fact, a much smaller model of the entire system may actually be of much greater value. In general, a model is an abstraction of the real system under study. It is not necessary or even desirable to include all of the details of the actual system. Deciding which details are essential and which may be omitted for the purposes of the study is perhaps the most difficult task which the modeler faces.

Without its world-view, SIMSCRIPT II.5 would be just another programming language, albeit a very powerful one. But with its world-view, the modeler

is guided in the formulation of a complete specification of the problem. The objects in the real world map very naturally into the SIMSCRIPT II.5 objects, which break down into classes termed TEMPORARY ENTITIES, PROCESSES, and RESOURCES. (All capitalized words are part of the SIMSCRIPT II.5 vocabulary.)

Any entity may have ATTRIBUTES which give it individual characteristic values. While all instances of a particular entity class have the same named attributes, each instance has its own values for the attributes. In addition, entities may be organized into SETS in order to represent any type of ordered list with various ordering disciplines (FIFO, LIFO, or RANKED by any combination of attribute values).

After the static structure of the model has been described, the dynamic aspects are described in terms of process routines. Each process routine corresponds to a declared process entity. Very natural commands are employed for manipulating objects in the process routines. Processes may WORK or WAIT for a period of simulated time. They may be FILEd in sets or REMOVEd from them They may ACTIVATE, INTERRUPT or RESUME one another. Processes may REQUEST or RELINQUISH resources, automatically waiting for those which are unavailable when requested and automatically starting other processes when relinquishing unneeded resources.

Animation in SIMSCRIPT II.5 is a very natural extension of the established world-view. Entities may be declared to be GRAPHIC in order to participate in animated displays. The actual form of the display (the so-called "icon") is described through the use of an editor and may be changed independently of the model.

### 1.2 SIMSCRIPT II.5 Language Features

SIMSCRIPT II.5 is a complete programming language. In addition to its simulation modelling capabilities, it has a full range of input/output capabilities including the ability to specify either formatted or freeform input, screen-oriented output (including cursor placement). generalized reports which may expand to multi-page width as well as length. The TEXT mode of variable declaration permits very general text manipulation of character strings of arbitrary length, including operations such as concatenation, substring search and replace, case change, etc.

The entity/attribute/set structure mentioned above is an extension of a very powerful underlying data structure. Arrays in SIMSCRIPT II.5 may be of any dimension whatever, without limit. The allocation of storage for the arrays occurs during execution and arrays may be deallocated and reallocated with different dimensions.

The support of the representation of statistical phenomena is extensive. Generators exist for random numbers distributed according to uniform, integer uniform, normal, lognormal, exponential, beta, gamma, Erlang, Poisson, binomial, triangular, and Weibull distributions. If these are not sufficient, an arbitrary numerical distribution is available to describe any distribution as a table of values versus probability (individual or cumulative).

The collection of data in the form of statistical performance measures is supported by three very powerful statements: ACCUMULATE, TALLY, and COMPUTE. ACCUMULATE and TALLY update statistical counters as the variable of observation changes values. Then only when the results are needed are the final statistical calculations performed. The measures available include number of samples, sum, average, maximum, minimum, standard deviation, variance, sum of squares and mean square. ACCUMULATE performs these calculations on a time-dependent basis, while TALLY performs them on a sample-basis.

Part of the ongoing development effort of SIMSCRIPT II.5 is to make the interface between user and model easier to understand. Models can be developed in which the parameters can be easily represented as presentation graphics such as pie charts, strip charts, dials level meters, bar graphs, etc. These so-called smart icons are updated on the screen as the simulation proceeds. In addition, animation capabilities have been developed to display moving objects against a static background in order to give further insight into the complex interactions which take place within a system.

The preparation of the presentation graphics as well as the icons for animation is accomplished through the use of editors. The icons are stored with the program but may be modified without having to modify the program or clutter it with non-system related code.

## 2 INTRODUCTION TO MODSIM II

MODSIM II was specifically designed to support large programming projects. It is a compiled, modular, object-oriented language with multiple inheritance. To protect the user's investment in applications, MODSIM can be moved to new computer systems as they become available. Its syntax is based on that of Modula-2. Modularity in MODSIM II improves reliability and code reusability. Objects and routines performing related functions can be grouped into modules. These can be put into libraries for reuse

by other programs. The simulation constructs are based on those used in SIMSCRIPT II.5. The portability of MODSIM II derives from the fact that its compiler emits C code which is compiled, in turn, by each computer's C compiler.

Finally, the integrated dynamic graphics of MODSIM II substantially reduces the time and effort needed to display results with animation and presentation graphics. It only takes a few statements to make dynamic icons, histograms, clocks and meters appear and change as the simulation runs. MODSIM II is a complete, general purpose programming language which is ideal for large software engineering projects.

### 2.1 Objects

An object is essentially an encapsulation of data and code. The data describes the object's current status. The code describes what the object does. As an example of an object in MODSIM II, consider things that move around, such as trucks and airplanes. This is the type definition of a moving object:

```
TYPE
  MovingObj =
  OBJECT
    position : LocTyp;
    course   : [ 0 .. 359 ];
    speed    : INTEGER;
    TELL METHOD GoTo(IN dest : LocTyp,
                     IN spd  : INTEGER);
    ASK  METHOD Stop;
  END OBJECT;
```

`MovingObj` is an object type. It has three data fields which hold information about its location, course and speed. In addition it has two methods. Methods are an object's procedures or routines which define its behavior. GoTo makes the object go to the specified destination from its current position. Stop is used to set the object's speed to zero. Note that the above object type declaration simply describes the state and methods of `MovingObj` and serves as an interface to the object. The actual code for the methods is supplied separately in the object declaration block. For example:

```
OBJECT MovingObj;
 TELL METHOD GoTo(IN dest : LocTyp,
                  IN spd  : INTEGER);
 VAR
   travelTime : REAL;
 BEGIN
   travelTime := ... { compute time }
   course := ...{ some trig calculation }
   speed  := spd;
   WAIT DURATION travelTime
     { simulation time elapses here }
   END WAIT;
   speed := 0;
   position.x := dest.x;  { update }
   position.y := dest.y;  { position }
 END METHOD;
 ASK  METHOD Stop;
 BEGIN
   speed := 0;
 END METHOD;
END OBJECT;
```

`ASK` methods are instantaneous with respect to simulation time. When an `ASK` method is invoked, the caller pauses and control passes to the invoked `ASK` method. When the invoked method completes, the caller resumes. `ASK` methods behave like a procedure call but have direct access to all fields and methods of that object. No simulation time can pass in an `ASK` method.

`TELL` methods are asynchronous. When the `TELL` method is invoked, it is simply scheduled for execution, and the caller immediately continues execution without waiting for the `TELL` method to start. Simulation time can elapse in a `TELL` method.

A `TELL` method starts execution under control of the built-in simulation timing routine. The data fields of an object instance are visible to all other parts of a program and may be read using an `ASK` statement. However an object's fields may be changed only by the object itself. To use an object, we create an instance of that object type and send it messages using `ASK` or `TELL` when we want it to do something.

### 2.2 Information Hiding

As we've seen, the fields of an object can be changed only by the object itself. This is one level of information hiding. However it is still normally possible for any program code to "read" the value of an object's fields using an ASK statement. We can achieve a higher level of information hiding by declaring some of the fields to be private. Private fields cannot be seen except by the object itself. Methods can be `PRIVATE`, too. Methods which are private can be invoked only by other methods of the object.

### 2.3 Inheritance

MODSIM II supports inheritance. With inheritance, new object types can be defined in terms of existing object definitions. While most languages only allow

inheritance from one existing object type, MODSIM II supports multiple inheritance.

Here is a `VehicleObj` type definition created from a `MovingObj`:

```
VehicleObj = OBJECT(MovingObj)
  payload : REAL;
  TELL METHOD Load(IN amount   : REAL);
  TELL METHOD Unload(IN amount : REAL);
END OBJECT;
```

`VehicleObj` inherits all of the fields and methods of a `MovingObj`. In addition it adds a payload field and methods for loading and unloading the vehicle.

If an inherited method is no longer appropriate for the newly defined object, it can be overridden and replaced by a new one of the same name. The old method can be invoked by the replacement method as part of its behavior if desired.

Different object types can adapt methods to fit their own particular behavior. This important and versatile object- oriented capability is known as polymorphism—multiple behaviors invoked with the same method name.

### 2.4 Discrete Event Simulation and Processes

Simulation is supported directly, as in SIMSCRIPT II.5, by built-in language constructs. The `WAIT` statement is used to make simulated time pass. Here is an example using the Load method of `VehicleObj`.

```
 TELL METHOD Load (IN amount : REAL);
CONST
  rate = 0.25; { seconds per passenger }
VAR
  loadingTime: REAL;
BEGIN
  loadingTime := amount / rate;
  WAIT DURATION loadingTime
    OUTPUT("Loading completed");
  ON INTERRUPT
    OUTPUT("Loading NOT completed");
  END WAIT;
END METHOD { Load };
```

The `WAIT DURATION` statement causes the method to suspend execution for the indicated amount of simulation time. Once the wait is started, control returns to the scheduler which then starts execution of the next most imminent process. When the `WAIT` is complete, control returns to this method at the statement after the `WAIT`. Any of the methods of an object which are waiting for completion can be interrupted. If the method receives an interrupt command, it executes the part of the `WAIT` statement after `ON INTERRUPT`.

Two other forms of the `WAIT` statement let methods synchronize themselves.

```
WAIT FOR Flight217 TO Load(324.0);
  ...
END WAIT;
```

This statement schedules the Load method of Flight217 but does not allow the invoking code to proceed with execution until the Load method has completed. Note that this is different from a normal `TELL` invocation which proceeds without waiting.

The other form of the `WAIT` statement uses the built-in trigger object to synchronize methods.

```
WAIT FOR ControlTowerLight TO Fire;
  ...
END WAIT;
```

This statement makes `Flight217` wait for a signal from the `ControlTowerLight` before it moves. `ControlTowerLight` is a trigger object which has a `TELL` method called `Trigger`.

```
TELL ControlTowerLight TO Trigger;
```

The `Trigger` method releases all waiting methods when it is executed.

### 2.5 Development Environment

Transporting programs from one computer system to another has often been a problem. Frequently programs have to be extensively rewritten to eliminate machine dependencies. MODSIM II avoids this problem. It was designed for portability. MODSIM II compiles its source code to C. The MODSIM II compilation manager then compiles and links the C code to a standalone executable.

MODSIM II's compilation manager was designed to facilitate project management of large computer programs consisting of many separate modules and libraries. It manages separate compilation of MODSIM II programs consisting of multiple modules by determining which modules have been edited since the last compilation and then recompiling only those edited modules and any modules which depend on them. This process is accomplished automatically without need for "make" or project files to describe the process.

### 2.6 Dynamic Graphics

Graphically displaying results has typically been a tedious programming task. Graphics programming is made simpler through MODSIM's interface to the SIMGRAPHICS II graphics editor and environment. SIMGRAPHICS II has three major capabilities:

- Animated graphics tied to objects in a program
- Dynamic or static graphs tied to variables and statistics in a program
- Interactive input menus in a contemporary windowed style

Animated icons, graphs and input menus are all interactively edited using the SIMGRAPHICS II editor. These are then tied to existing objects and variables in the user's program. This greatly simplifies the task of creating a graphical user interface. The amount of coding for graphics is drastically reduced. An important side benefit of the editor is that the appearance of objects can be edited without recompiling or changing code. This facilitates both design and subsequent maintenance as well.

Figure 1 shows a screen from a communications satellite model. The satellites are icons which move around the earth. The line between two satellites indicates that a message is being passed. At the top left is a trace plot of message rate versus time. At top right is a level meter showing the current mean message rate. Finally, the clock at the bottom shows that we are 31 seconds into the simulation.

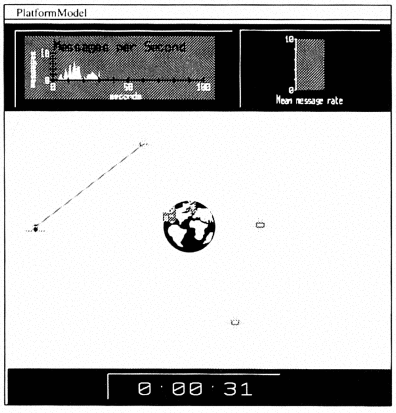


Figure 1: Satellite Communication Simulation

## 2.7 Benefits of MODSIM II

Any high order language is designed to reduce the effort needed to program a set of problems. The object-oriented and modular features of MODSIM II substantially reduce the time and effort needed to write programs.

- Objects improve reliability because they encapsulate data fields and provide a disciplined interface to these fields.
- Development time is reduced because code can be put in libraries and reused.
- Modules permit step-wise development, particularly by separating the definition module from the implementation module.
- Inheritance allows programmers to build on top of previous effort instead of starting from scratch each time.
- Integrated dynamic graphics substantially reduce the time and effort needed to build menus and display results.

## 2.8 Conclusions

MODSIM II is a robust, general purpose programming language with built-in graphics. Its features substantially reduce the time and effort required to write and validate computer programs.

## 2.9 Availability

SIMSCRIPT II.5 and MODSIM II are proprietary products of CACI Products Company. They are sold on a free-trial basis. A special university program of CACI supplies SIMSCRIPT II.5 and MODSIM II to educational institutions for the cost of distribution.

## REFERENCES

CACI. 1988. *SIMSCRIPT II.5 Programming Language*, CACI Products Company, La Jolla, CA.

Law, A.M. and C.S. Larmey. 1984. *An Introduction to Simulation Using SIMSCRIPT II.5*, CACI Products Company, La Jolla, CA.

Russell, E.C. 1989. *Building Simulation Models with SIMSCRIPT II.5*, CACI Products Company, La Jolla, CA.

Belanger, R., B. Donovan, K. Morse and D. Rockower. 1990. *MODSIM II Reference Manual*, CACI Products Company, La Jolla, CA.

Belanger, R., and A. Mullarney. 1990. *MODSIM II Tutorial*, CACI Products Company, La Jolla, CA.

## AUTHOR BIOGRAPHY

Dr. Edward Russell has over twenty years experience applying simulation to commercial and government problems.