Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

SIMSCRIPT II.5 AND SIMGRAPHICS TUTORIAL

Edward C. Russell

Russell Software Technology
1735 Stewart Street
Santa Monica, California 90404, U.S.A.

ABSTRACT

The SIMSCRIPT II.5 programming language is 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.

1. INTRODUCTION

SIMSCRIPT II.5 is a well established, standardized, and widely used language with proven software support. Experience has shown that SIMSCRIPT II.5 reduces simulation programming time and cost several fold when compared to FORTRAN. 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:

1.1 Design

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

1.2 Programming

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 modelled system.

1.3 Testing

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.

1.5 Evolution

The SIMSCRIPT II.5 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.

1.6 Documentation

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.

2. OVERVIEW

2.1 Purpose of simulation

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.

2.2 Concept of a World-View

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 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 **FILE**d in sets or **REMOVE**d 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.

2.3 Self-Documenting Code

Over the years, we have observed numerous unsuccessful simulation projects that had no documentation except for a FORTRAN listing. Many of these listings contained few explanatory comments. Even a thoroughly commented FORTRAN listing is difficult to decipher for anyone other than the person who wrote it. Often, even the original author has difficulty understanding it after a short time.

We have also seen great amounts of money wasted on manuals and flowcharts intended to make it easier to develop, maintain, modify and enhance the model. This waste is a consequence of the realities of model development. Most models evolve over a long period of time because of new and increased understanding of the system, changing goals, and availability of new data. Because of the evolutionary changes, flowcharts, prose documentation, detailed descriptions of routines and variables, and program comments often become obsolete, incomplete, or incorrect shortly after they are written. The longer the model is around—and many models in use today were developed five or more years ago—the more this type of documentation deteriorates.

For the purposes of computer program development, modification, and enhancements, the only dependable documentation in a changing environment is the source program listing. The quality and usefulness of this documentation is determined by the model design and the choice of simulation language. SIMSCRIPT II.5 has been shown to reduce the amount of code required when compared to FORTRAN by at least 75%, a four to one reduction!

2.4 Large model development

SIMSCRIPT II.5 has traditionally been the language of choice for very large models. There are no inherent limits to this size of either SIMSCRIPT II.5 programs or their data structures. The dynamic storage allocation of SIMSCRIPT II.5 frees the modeler from concerns about the size of data elements and all the error-checking code necessary to enforce array limits. The modularity of the language structure permits large teams of developers to work on independent segments of the model without needing to know all of the details of other elements of the model.

2.5 Portability

SIMSCRIPT II.5 is a truly portable language. It was originally developed for large mainframe computers, but it has evolved with the industry to implementations on mini- and now microcomputers. The IBM Personal Computer implementation of SIMSCRIPT II.5 is one of the most advanced software packages available on that computer. The modelling language has been designed and maintained to be compatible across all the implementations. These include PC (DOS, Windows, OS/2), SPARC Station, DEC Station, HP 9000/700, IBM RS6000, VAX, and CRAY.

3. 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 free-form 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. (If there were a need for much reallocation, the temporary entity concept would more likely be used.)

SIMSCRIPT II.5 contains all of the constructs of modern structured programming. Search commands relate to the data structures to be scanned. Program segments may be modularized along functional lines as routines, functions, monitoring routines (to be called implicitly when the monitored variable is either accessed or modified or both), as well as the process and event routines of simulation. Routines may also be executed recursively.

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.

SIMSCRIPT II.5 has recently been enhanced to enable the user to develop models which include processes which change continuously with simulation time. This enables models to be built for those systems which are described in terms of differential equations with superimposed discrete events. The combined capabilities enable the user to define models where dependent variables may change discretely, continuously, or continuously with discrete jumps superimposed.

Part of the ongoing development effort of SIMSCRIPT II.5 is to make the interface between user and model easier to understand. Traditionally, the output of a simulation run was collated tables of data which required extensive analytical capability on the part of the user in order to understand the underlying interactions between various parts of the system under investigation. Much progress has been made in providing facilities within the language whereby these interactions may be represented graphically. This enables models to 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.

4. THE PC SIMLAB ENVIRONMENT

Many of the capabilities of PC SIMSCRIPT II.5 are made possible because of SIMLAB. SIMLAB is an operating environment for the SIMSCRIPT II.5 compiler, editors, and run-time. SIMLAB supports a multitasking environment in which it is possible to perform several tasks simultaneously. During program development, it is possible to edit several portions of the program simultaneously (in different windows). During execution, it is possible to open debug windows, set break points, and track the execution of the program through its source code. It is also possible to have multiple, concurrent output streams to different windows. SIMLAB also makes it possible to write, maintain, and execute programs which are much larger than the actual memory of a PC can contain. Through its virtual addressing mechanism, programs which would normally require main frame capacity are being developed on PCs.

5. EXAMPLE CODE

Simulation involves the passage of time in the life of some object. For example, passengers arrive at an airport, wait for a passenger agent, get a boarding pass and leave.

General purpose programming languages, such as Pascal, C and FORTRAN, lack the constructs to make time pass in a simulation. The programmer has to write these constructs before he or she can get on to writing the simulation.

5.1 Processes

In general, SIMSCRIPT II.5's simulation constructs are built around the concept of a process. This is a routine that describes what happens to the object as it moves through time. For example, here is the SIMSCRIPT.II.5 code that processes a passenger in a terminal.

```
Process PASSENGER
  Define ARRIVAL.TIME as a real variable
  Let ARRIVAL.TIME = time.v
  Request 1 PASSENGER.AGENT
  Let WAITING.TIME = time.v - ARRIVAL.TIME
  Wait exponential.f(MEAN.SERVICE.TIME, 1)
      minutes
  Relinquish 1 PASSENGER.AGENT
End "PASSENGER
```

Before getting into the details, a comment about style is in order. SIMSCRIPT II.5, unlike many other languages, is not case sensitive: `ARRIVAL.TIME` and `arrival.time` are the same variable.

While SIMSCRIPT II.5 will not punish you for failing to capitalize a word, it does require that variables be spelled correctly. If you misspell a word, you will get a warning message: `Local variable used only once`. This is part of SIMSCRIPT II.5's substantial error checking capability.

As a matter of style all SIMSCRIPT II.5 words are shown as capitals and lower case; user defined words are all capitals. This way the substance of the model is apparent as you scan the code. The two apostrophes at the end of the process routine are the beginning of a comment. The first statement declares `ARRIVAL.TIME` to be a local, real variable.

The `PASSENGER.AGENT` is a resource. The `PASSENGER` has to have a `PASSENGER.AGENT` before he can get a boarding pass. So he requests one, if one is available, the next line of code is executed immediately.

If a `PASSENGER.AGENT` is not available, control passes to the timing routine. The timing routine files this `PASSENGER` in queue waiting for a `PASSENGER.AGENT` and starts execution of the next process.

When a `PASSENGER.AGENT` becomes available and this `PASSENGER` is first in the queue, the timing routine removes the `PASSENGER` from the queue and schedules it to continue execution. When this `PASSENGER` is the next process to be executed, control returns to the process routine at the line after the request statement.

The next statement calculates the amount of time the `PASSENGER` had to wait for a `PASSENGER.AGENT`. `Time.v` is the current simulated time.

The `Wait` statement represents the passage of time while the `PASSENGER` gets his boarding pass. The amount of time is drawn from an exponential distribution with a mean of `MEAN.SERVICE.TIME` using a random number stream 1.

As with the resource, control passes back to the timing routine. The timing routine files the `PASSENGER` in a queue of pending events, called an event set. This queue is ranked by the time of reactivation. Reactivation time for this `PASSENGER` is the current simulated time plus the amount of time drawn from the exponential distribution.

When this `PASSENGER` is next up for execution, control returns to the statement after the wait statement. The `PASSENGER` relinquishes the `PASSENGER.AGENT` so that someone else can use it and disappears from the scene.

The great advantage of this construct is the modeler can write the steps of the process in structured English. Having done this, he or she only has to create passengers when they are needed, and SIMSCRIPT II.5 will handle all details of scheduling and execution.

Notice that the process is clear about what is supposed to happen. This reduces the chances of logical errors in coding. More importantly, an airline employee can read the code and know whether it represents what really happens.

5.2 Creating Instances of Processes

The process routine show what happens to one passenger. We need to create many passengers to run a simulation. The general technique involves creating a second process that functions as a passenger generator.

```
Process PASSENGER.GENERATOR
  Define I as an integer variable
  For I = 1 to 120,
  Do
    Activate a PASSENGER now
    Wait exponential.f (INTERARRIVAL.TIME, 2)
       minutes
  Loop
End "PASSENGER.GENERATOR
```

PASSENGERs are created one at a time with some time passing between creation of each PASSENGER. In actuality, this is the wait between arrivals that normally occur at an airport: generally passengers do not all show up at once.

The PASSENGER.GENERATOR will create 120 passengers. An alternative to the For statement is:

```
Until time.v >= RUN.TIME
```

In this case PASSENGERs would be created until some simulated time had passed, say eight hours.

The Activate statement creates the process notice for this instance of the passenger, sets the reactivation time, time.a, to time.v (i.e. now) and files it in the event set. The PASSENGER.GENERATOR then waits some amount of simulated time before looping and creating the next PASSENGER. It passes control back to the timing routine. The timing routine sets the PASSENGER.GENERATOR's reactivation time to the current time plus the time drawn from this exponential distribution.

It then files the PASSENGER.GENERATOR in the event set, gets the next process notice and starts executing it. When the PASSENGER.GENERATOR is first in the event set, the timing routine returns to the statement after the Wait statement, loops, creates the next passenger and goes back to the event set.

5.3 Running the Simulation

All that remains is to start the simulation and build some I/O structure to support the model. As in any full-featured language, SIMSCRIPT II.5 supports formatted input and output of text.

More significantly, SIMSCRIPT II.5's built-in graphics package, SIMGRAPHICS, supports a wide range of presentation graphics and animation. Well-designed animation greatly enhances the effectiveness of the simulation, allowing you to better understand what is happening in the model.

6. SIMSCRIPT II.5 AVAILABILITY

SIMSCRIPT II.5 is a proprietary product of CACI Products Company. It is sold on a free-trial basis. A special university program is supported by CACI in which SIMSCRIPT II.5 is supplied to educational institutions for the cost of distribution.

7. TRAINING

Week-long training courses are given by CACI on a regular basis. These courses are held in their training facilities in La Jolla and Washington, D.C., as well as at other locations throughout the world. The same course is available for on-site training from Russell Software Technology.

REFERENCES

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

CACI (1992), *SIMGRAPHICS User's Guide and Casebook*, CACI Products Company, La Jolla, CA.

CACI (1991), *SIMSCRIPT II.5 Reference Handbook*, CACI Products Company, La Jolla, CA.

CACI (1990), *Major Applications of SIMSCRIPT II.5 with SIMGRAPHICS*, CACI Products Company, La Jolla, CA.

AUTHOR BIOGRAPHY

Dr. Edward C. Russell, President of Russell Software Technology, has over twenty five years experience in the simulation profession. While at CACI, he helped to design and implement the SIMSCRIPT II.5 programming language.

He developed and regularly teaches simulation and modelling courses. He consults with business and government clients on the use of simulation in such wide-ranging applications as manufacturing systems design, missile defense systems, munition distribution systems, oil tanker schedule planning, and computer network design.

Dr. Russell is the author of the popular textbook on simulation entitled ***Building Simulation Models with SIMSCRIPT II.5***.

He regularly teaches course on modelling and simulation for the Department of Engineering and the Anderson Graduate School of Management at UCLA.

His academic credentials include both an M.S. and Ph.D. in Computer Science (Engineering) from UCLA and a B.S. in Electrical Engineering from Wayne State University.