Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

AN INTRODUCTION TO COMPUTER SIMULATION

Michael Pidd
The Management School
Lancaster University
Bailrigg
Lancaster LA1 4YX
UK

ABSTRACT

This paper provides a gentle introduction to the fundamentals of computer simulation. It discusses the difference between the various approaches in common use and highlights the importance of a carefully considered approach to modelling. Its main stress is on discrete event methods within which it describes conceptual modelling, statistical aspects and the types of computer software which are available.

1 THE WHYS AND WHEREFORES OF COMPUTER-BASED MODELLING

Computer simulation methods have been in use since the 1950s and are based on the idea that an experimental or gaming approach can be a useful support to decision making. The idea is to try out a policy before it is implemented. clearly, there are several ways in which this could be done.

- The policy could be tried in the real world, but in a controlled way so that its effects can be understood and analysed. There are obvious problems with this approach, especially in systems which are dangerous or expensive to operate for which experimenting with the real system could turn out to be experimenting with disaster. Nevertheless, this type of direct experimentation does have its place, especially when training people.
- a second option would be to develop a mathematical model of the system being studied, and this is the speciality of operations research. This approach works well for some types of application (for example in simple queuing systems) but not so well in others The basic problem is that the maths needed to represent a complex dynamic system may be impossible to solve or virtually impossible to formulate without excessive approximation.
- Hence, the third option is to simulate the system of interest in a computer-based model and then carry out experiments on that model to see what might be the best policy to adopt in practice.

In addition, a simulation approach is sometimes used in order to understand how an existing system operates or how a proposed system might operate. What are believed to be the rules governing the behaviour of the system are captured in a computer-based model and the behaviour of this model is used to infer how the system being modelled might itself operate.

1.1 The essence of computer-based modelling

A computer-based model is at the heart of any computer simulation and the question which has to be faced is, how can the important details of the system to be simulated be captured within such a model? A later section of this paper will discuss the main features of simulation models, but before considering these features it is a good idea to give some brief consideration to the process of modelling.

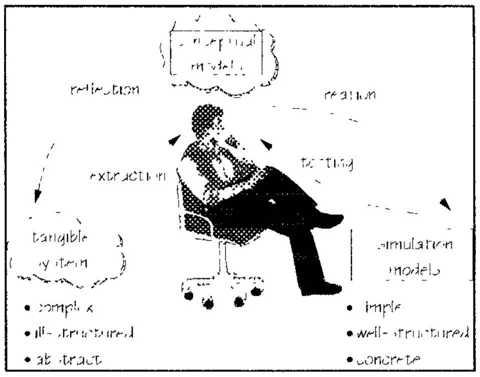

Figure 1: Computer-based modelling

Figure 1 shows that at the core of this process are one or more human beings who are concerned to ensure that their model is appropriate for the purpose for which it is intended. To do this it is useful to consider the main features of Figure 1.

The tangible system which they are attempting to model is separate from them and, to use the ideas of Zeigler (1976, 1984) provides a source of data from which a model will be constructed. Different people may well hold different notions about the operation of this tangible system and these can be described in *conceptual models* which are themselves the result of reflection about the tangible system. The *simulation model*, which eventually becomes a computer program, stems from the conceptual model and is expected to be much simpler than the tangible system. Were it not simpler, then it would be as difficult to use for experimentation as the tangible system itself.

Hence, computer-based modelling is a process of simplification and abstraction in which the modeller attempts to isolate those factors believed to be crucial in the operation of the system being modelled. This process of abstraction depends on data and information about the tangible system and also on the intended purpose of the simulation.

1.2 Modelling complicated dynamic systems

Given enough time, money, expertise and computer power almost any system can be simulated on a computer, but this may not be sensible. Hence the next question to face is, to what type of systems are modern computer simulation methods best suited? The following features tend to characterise the systems best suited to computer simulation.

- They are *dynamic*, that is they display distinctive behaviour which is known to vary through time. This variation might be due to factors which are not well understood and may therefore be amenable to statistical analysis - for example, the apparently random failures of equipment. Or they might be due to well understood relationships which can be captured in equations - for example, the flight of a missile through a non-turbulent atmosphere.
- They are *interactive*, that is, the system is made up of a number of components which interact with one another and this interaction produces the distinctive behaviour of the system. For example, the observed behaviour of aircraft under air traffic control will be due to the performance characteristics of the individual aircraft, the intervention of the air traffic controllers, the weather and any routing problems due to political action on the ground. This mix of factors will be varying all the time and their interaction will produce the observed behaviour of the air traffic.
- They are *complicated*, that is, there are many objects which interact in the system of interest, and their individual dynamics need careful consideration and analysis.

1.3 Continuous, discrete and mixed approaches

It is normal to classify approaches to computer simulation into three groups and this will be done here, but it should be noted that these distinctions are ones made by the modeller and are not ones which occur in the real world being simulated.

1.3.1 Discrete event simulation

The Winter Simulation Conference is traditionally concerned with *Discrete Event Simulation*. This approach is based on a number of building blocks as follows, each of which is discussed in more detail in #2 of this paper.

- *Individual entities*: the behaviour of the model is composed of the behaviour of individual objects of interest which are usually called entities. The simulation program tracks the behaviour of each of these entities through simulated time and will be minutely concerned with their individual logics. The entities could be truly individual objects (e.g. machines, people, vehicles) or could be a group of such objects (e.g. a crowd, a machine shop, a convoy of vehicles).
- *Discrete events*: each entity's behaviour is modelled as a sequence of events, where an event is a point of time at which the entity changes state. For example, a customer in a shop may arrive (an event), may wait for a while, their service may begin (an event), their service will end (an event) and so on. The task of the modeller is thus to capture the distinctive logic of each of these events (e.g. what conditions must hold if a *begin service event* is to occur?). The flow of simulation time in a discrete event simulation is not smooth, as it moves from the event time to event time and these intervals may be irregular.
- *Stochastic behaviour*: the intervals between events is not always predictable, for example the time taken to serve a number of customers in a shop will be observed to vary. There may sometimes be obvious and entirely predictable reasons for this (the server may speed up as the queue of waiting customers increases) or there may be no obvious reason to explain things. In the latter case, the varying intervals between events has to be modelled stochastically by using sampling methods based on probability theory.

1.3.2 Continuous simulation

A quite different approach to simulation is taken in *Continuous Simulation*, which is an approach that is popular amongst engineers and economists. The main building blocks of this approach are as follows.

- *Aggregated variables*: instead of a concern with individual entities, the main concern is with the aggregated behaviour of populations. For example, the changing sales of a product through time.
- *Smooth changes in continuous time*: rather than focusing on individual events, the stress is on the gradual changes which happen as time progresses. Thus, just as the graph of a variable might be smooth, the aim is to model the smooth changes of the variable by developing the suitable continuous equations .

- *Differential or difference equations*: the model consists mainly of a set of equations which define how behaviour varies through time, thus these tend to be differential equations or, in simpler cases such as system dynamics (Forrester, 1961 and Wolstenholme, 1990), difference equations.

Nature does not present itself labelled neatly as discrete or continuous, both elements occur in reality. Modelling, however, as mentioned in #1.1 above, involves approximation, and the modeller must decide which of these approaches is most useful in achieving the desired aim of the simulation?

1.3.3 Mixed discrete/continuous simulation

In some cases, both approaches are needed and the result is a *mixed discrete-continuous simulation*. An example of this might be a factory in which there is a cooking process controlled by known physics which is modelled by continuous equations. Also in the factory is a packing line from which discrete pallets of products emerge. To model the factory might well require a mixed approach.

1.4 The role of software

Computer software plays an essential role in the development and use of computer simulations and is available to support the following aspects.

- *Statistical analysis of input data*: in a discrete simulation it will probably be necessary to model certain aspects by taking samples from probability distributions within the model. Thus the modeller needs to consider which distributions are appropriate for the system being considered. This requires the modeller to collect data (e.g. the times between failure) and to try to fit an appropriate distribution. There are a number of products available to support this task, examples include UniFitII (Vincent & Law, 1993) and SIMSTAT 2.0 (Blaisdale & Haddock, 1993).
- *Rapid modelling*: the last 10 years have seen the development of *Visual Interactive Modelling Systems* (VIMS) such as Witness (AT & T Istel, latest version), XCELL+ (Conway et al, 1990) and ProModel (Harrell & Leavy, 1993) and Stella/I Think (Richmond & Peterson, 1988). These allow the modeller to develop the logic of a model on-screen using a graphical user interface and also control the running of the model.
- *Simulation model programming*: as #4 makes clear, it is sometimes necessary to write a 'proper' computer program and this can be done using a purpose designed simulation language such as SIMSCRIPT II.5 (CACI, 1987), a general purpose language such as C or even on a spreadsheet or database.
- *Statistical output analysis*: It is not always easy to interpret the results of a simulation, especially one which includes a large number of stochastic elements and the resulting output many need careful statistical analysis. There are tools to support this task, some are specifically for simulation modelling (e.g. SIMSTAT, Blaisdale & Haddock, 1993) and others are more generally available packages such as SPSS and SAS.

2 MODELLING IN DISCRETE SIMULATION

As this is the winter simulation conference, the rest of this paper will concentrate on discrete event simulation.

2.1 Events and their logic

A computer program which represents a discrete simulation model will have a number of components as follows.

- *The event logic*: a precise definition of the conditions which govern the state changes of the entities to be included in the model.
- *An executive or control program*: which ensures that the entities' events occur at the right time and in the correct sequence and thus ensures that their aggregate behaviour is a model of the system being simulated.
- *Other components*: such as sampling routines, integration algorithms, graphics and other features needed for a particular model.

If the modeller is using a VIMS or a simulation programming language, then he or she need only be concerned with the event logic, everything else will be provided by the system vendor. If a bespoke program is being written in a general purpose language then all of the features will have to be provided.

2.2 Capturing system logic

2.2.1 The principle of parsimony

Perhaps the best way of modelling complicated event logic is to bear in mind the *principle of parsimony*, which is to keep things as simple as possible for as long as possible. This requires an evolutionary approach to modelling, starting with a deliberately over-simplified model which is gradually elaborated until it is agreed to be valid for the intended purpose. The initially over-simplified model should represent the skeletal logic of the system and should not be elaborated until the modeller is happy with the validity of the skeleton.

2.2.2 Using diagrams

One way of ensuring a parsimonious approach to modelling is to try to capture the essential system logic within some type of network diagram. In some cases, such diagrams can be drawn on-screen or described textually to a computer program which will itself generate the computer-based model from the diagram (see #4.1). In this paper, only a simple type of diagram - the Activity Cycle Diagram (ACD) will be presented, though other forms (for example, Petri nets and GPSS flowcharts) have been used in discrete simulation.

An ACD is an attempt to show how the processes of different entity classes interact, at least in a skeletal form. An ACD has just two symbols as shown in Figure 2.

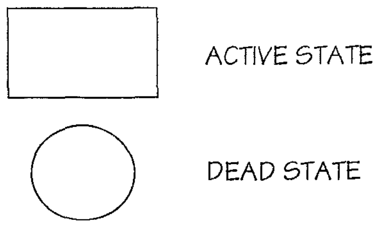

Figure 2: Activity cycle diagram symbols

- An *active* state is one whose time duration can be directly determined at the event which marks its start. This might be because the time duration is deterministic (the bus will definitely leave in 5 minutes) or because its duration can be sampled from some probability distribution (see #3).
- A *dead state* is one whose duration cannot be so determined but can only be inferred by knowing how long the active states may last. In most cases, a dead state is one in which an entity is waiting for something to happen and thus some writers refer the dead states as queues.

These two symbols are used to represent the logic of a system as in the following simple example.

Consider a theatre booking office staffed by one or more clerks who have two tasks - answering the phone and attending to personal callers at the theatre. As this is a skeletal model, suppose that the theatre has a call queuing system with infinite capacity, that there is no limit on the number of waiting personal customers and that all waiting callers have customers are infinitely patient. Hence the diagram of figure 3 can be drawn.

The skeletal logic of the system can be clearly understood from the activity cycle diagram. For example, a personal service can only begin if two conditions hold - there must be an idle clerk and a waiting personal enquirer. Similarly it shows that any clerk may engage in two tasks - attending to personal enquirers or answering the phone. It also shows some of the ambiguities. For example, what should a clerk do if faced, at the same time, with waiting enquirers and a ringing phone?

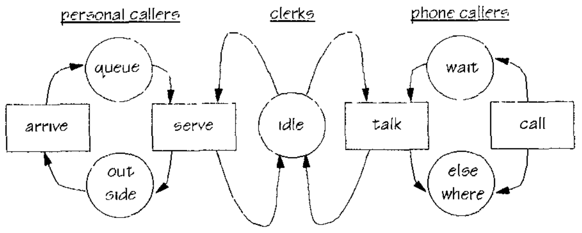

Figure 3: Harassed booking clerks ACD

This type of conceptual representation must eventually become part of a computer program which might involve some programming or could involve a description of the logic being fed as data to a simulator which may be a VIMS (see #4).

3 HANDLING RANDOM AND UNPREDICTABLE BEHAVIOUR

As was made clear earlier, one aspect of systems which are well suited to discrete event simulation is that they may have behaviour which can only be modelled statistically - for example, the time interval between arrivals may be observed to vary and the variation may be modelled by fitting a probability distribution to that variation. To cope with this variation, discrete simulation models employ sampling procedures.

3.1 Basics of random sampling

The idea of random sampling is to ensure that a set of samples is produced that is representative of the distribution from which they were taken and within which set no pattern is evident. This is usually achieved by a two stage sampling process which uses pseudo-random numbers.

A truly random number stream is a sequence of numbers produced by a device which is believed to be random - for example a roulette wheel, which some people find curiously interesting. Truly random numbers streams are not used in discrete simulations because most such devices are slow (millions of random numbers may be needed) and also they cannot be repeated - an important consideration, as will become clear shortly.

A pseudo-random number stream is a sequence of numbers which behave exactly as a stream of random numbers would be expected to behave but which is produced by a well-understood mathematical process.. Thus, when the sequence is examined, there is no pattern in the sequence and all values covered by the range of the random numbers occur equally often. In statistical terms, the sequence must be independent and uniformly distributed with a dense coverage of the range of values.

The two stage process is follows.

- Generate 1 or more pseudo-random numbers.
- Convert these into the samples needed by some suitable algorithm.

3.1.1 Top-hat sampling

To illustrate the basic idea, consider Top-hat sampling, which is an common approach to taking samples from histograms. Figure 4 shows the probability of a clerk selling a certain number of tickets during the service of a customer. Figure 5 is the cumulative histogram.

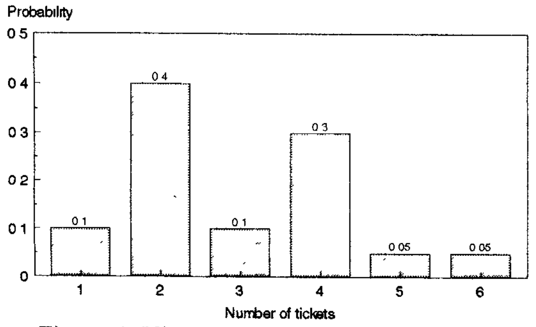

Figure 4: Histogram of ticket sales

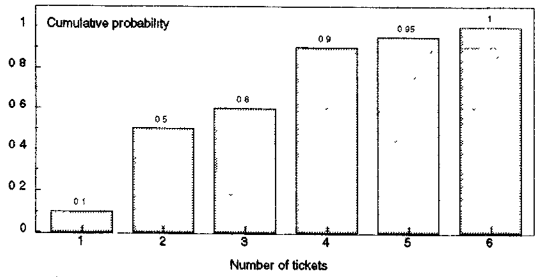

Figure 5: Cumulative histogram of ticket sales

In figure 5, the vertical axis represents the cumulative probability of ticket sales. It runs over the (0, 1) interval and can be replaced by a range of pseudo-random numbers which also runs over (0, 1). Thus, if the pseudo-random sequence includes a set of numbers (0.38, 0.75, 0.53) then reading from the graph of figure 5, these correspond to a set of ticket sales (2, 4, 3). The simple two stage process for Top-hat sampling involves.

- Generate a pseudo-random number.
- Look up the corresponding value from the graph or a look-up table..

As well as top-hat sampling, there are many algorithms in use for different types of probability distribution, Law & Kelton (1991) have more details.

3.2 The effects of statistical variation

Due to its sampling procedures, a discrete simulation may display complicated behaviour which needs careful analysis. For example, there may be separate samples taken for the personal enquirer arrival time, the phone call arrival time, the personal service duration and the phone conversation duration even in a simple model such as the harassed booking clerk.

Typical two stage sampling procedures use the pseudo-random numbers for two purposes. The first is to ensure that the sequence of samples is pattern-free, the second is to select the set of values which are contained in the sequence. These two sources of sampling variation, which will be combined as different samples are combined, means that any discrete simulation which stochastic elements needs to be regarded as a sampling experiment. In any such experiment, it must be recognised that there is a risk of coming to the wrong conclusions. For example, consider figures 6 and 7 which show the (imagined) results from two sets of simulations in which policy A is being compared with policy B - the idea being to decide which one generates the highest profit.

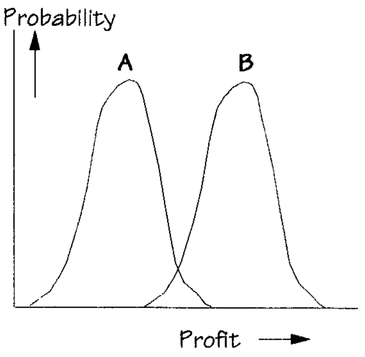

Figure 6: First set of experiments

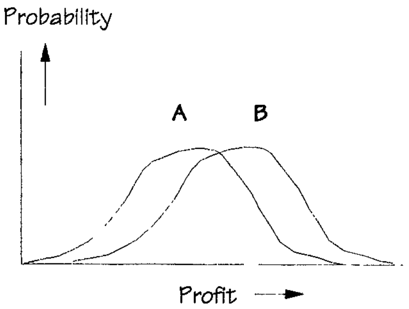

Figure 7: Second set of experiments

The variation due to the pattern-free sequence is wholly desirable (this is random behaviour), that due to a badly selected set is wholly undesirable and is due to the fact that the set is of finite (and possibly rather small) size. This set effect means that the distribution of the samples may not properly represent the distributions from which they came.

In both cases the mean value of the experiments with policy B exceeds those with policy A, but it would be possible to be much more confident that this is a true inference if the experiments turned out like the first set shown in figure 6. The difference between the two sets of experiments is that the output variances are much lower in figure 6 than in figure 7 and thus the risk of a wrong inference is lower. The output variance is a function of the sampling variation which must be controlled.

3.3 Some cautionary advice

In any experimental comparison, whether using real systems or a simulation, it is important to ensure that the comparison is a fair one. That is, the comparisons should be made with the system (real or simulated) operating under similar and typical conditions for all policy options.

3.3.1 Run-in periods

Suppose that someone wished to simulate the effect of adding an extra runway to a civil airport and in particular they wished to discover what extra traffic, if any, this would permit. Part of the experimental control would be to ensure that simulations of the existing runway configuration and the extra runways were conducted in such a way that both options were compared under the same starting conditions. There is probably never a time when a large civil airport has no activity and thus starting the simulations with no activity would not be representative of real conditions. Indeed, there is a risk that this may bias the comparison one way or the other.

Two ways of coping with this are to use 'typical starting conditions' or to use a run-in period. Of the two, the latter is preferable, but why?

The risks with using 'typical starting conditions' are twofold. First, we may know what these are for the existing system configuration but we do not know what they are for the novel alternative. Indeed, if we knew this then there would be no need for the simulation. The second reason is that use of 'typical starting conditions' may bias the results. For example, in the airport example we might reasonably believe that an extra runway will allow extra flights and we may thus ensure that the starting conditions for this policy have more activity than those for the current system configuration. If we are interested in assessing whether the extra runway will permit extra traffic then there is a great danger of a self-fulfilling prophecy.

Hence it is better to employ a run-in period. The idea of this is that, if a simulation starts with no activity, then it should be allowed to run to some time until it is believed to have settled down into some form of steady state. During this run-in period the output from the simulation is ignored and only output generated after that point will be used in the analysis.

Of course, there are simulations in which no form of steady state is possible (e.g. a missile chasing a jinking target) in which case the transient effects are the main focus of interest. In such cases, ruin-in periods should not be used.

3.3.2 Variance reduction

As was pointed out in #3.2, the accuracy of simulation results is related to the observed variation in the sampling processes of the model. Thus it is important to control these if at all possible. There are many techniques available to help in this (for a thorough discussion see Law & Kelton (1992) and Kleijnen & van Groenendal (1992). The simplest approach when comparing different policies is to use *common random numbers*, a technique which works by synchronising sampling processes across policy comparisons.

To use common random numbers, the analyst must ensure that each sampling process has its own controllable pseudo-random number stream. Hence, in the harassed booking clerk example introduced earlier, this means that 4 streams will be needed for each simulation run (one each for personal arrivals, phone calls, personal service and phone conversations) if there is just one clerk and $2+2n$ if there are n clerks.

The technique works by controlling the set of random numbers which are used to generate the required samples. If each policy option is compared using the same random numbers then the same samples will, as far as is possible, be used for each policy comparison. If each policy option is being replicated m times, then the modeller will need to ensure that each of these m replication uses common random numbers and thus will need access to $m(2+2n)$ streams in the above example.

This need for control of the random numbers streams is the main reason why pseudo-random numbers are preferred over truly random streams.

4 SOFTWARE SUPPORT FOR DISCRETE SIMULATION

4.1 Types of software

A thorough review of this is given in Pidd (1992) but a summary here will help to place things in context.

4.1.1 Coding in a general purpose language

Early simulations were written in whatever primitive programming languages were available on the simple computers of the day. This approach, like the rest, persists to this day and it seems likely that a reasonable proportion of simulations are written in languages such as C (Crookes, 1989), C++ (Joines et al, 1992, Pidd 1993), Pascal (Pidd, 1989)and even in FORTRAN and BASIC (Pidd 1988). Using such an approach means that very flexible and bespoke software can be created, but the cost is that such program development is very slow and requires considerable skills and highly specific knowledge in detailed computer programming.

4.1.2 Using a library

Rather than starting each program from scratch it has also long been possible to construct some parts (occasionally, most parts) of a discrete simulation application out of program building blocks taken from a library. This allows quicker program development but still requires detailed programming skills - and faith in the library which being used.

4.1.3 Simulation programming languages

Many simulations are written in special purpose simulation languages such as SIMSCRIPT II.5 (CACI, 1987) because these ease the task of program development by providing language constructs which are designed for discrete simulation. Thus these languages usually provide the event scheduling mechanisms which underpin discrete simulations and also have a syntax which eases the expression of the logical interaction of the simulation entities. However, as with the other two approaches, such software still requires detailed programming skills if it is to be used effectively.

4.1.4 Flow diagram systems

A different approach to easing model implementation is taken in flow diagram systems such as HOCUS (Szymankiewicz et al, 1988)). In these systems, the user develops a flow diagram such as an activity cycle diagram, and then uses a defined command set to describe the features of the flow diagram to the flow diagram system. This description is, in essence, data to a generic simulation model which is suited to a particular domain (e.g. queuing systems). As originally developed, these flow diagram systems did not permit the user to develop or generate a simulation program in any meaningful sense.

The main advantage of this approach is that it supports rapid program development. The main snag is that, without considerable effort, it is very difficult to model complex systems. It is interesting to note that so-called block-structured languages such as GPSS are, at root, flow diagram systems.

4.1.5 Interactive program generators

These, for example CAPS/ECSL (Clementson, 1991) and SIGMA (Schruben, 1991), attempt to combine the benefits rapid development from flow diagram systems and the flexibility of direct programming. They represent the start of attempts to provide layered software development tools for discrete simulation. Their initial use resembles that of a flow diagram but, instead of treating the diagram description just as program data, it is used to generate a working program in some target language by linked together edited pre-written fragments of program code. This code may then be edited so as to allow the modelling of complex systems. Because the user can develop the simulation model at different levels, these interactive program generators represent the start of layered program development systems.

4.1.6 Visual interactive modelling systems

These are flow diagram systems brought up to date, in the sense that they make good use of recent developments in general computing. There are many examples available on the current market such as Witness (AT&T Istel, latest version), Pro-Model (Harrell & Leavy, 1993), SIMFACTORY (CACI, latest version). To use them, the modeller must conceptualise the system of interest as a network around which elements flow, changing their state at the nodes of the network. Icons are placed on-screen and linked together so as to represent the network logic. In some systems, there is considerable scope for expressing the event logic at nodes by the use of special designed macro-type languages. However, it is not normal for these VIMSs to generate proper program code which the user may modify, though some (for example Witness) have a simplified coding language, and they are thus best suited to relatively straightforward network-type applications.

4.2 Choosing software

4.2.1 Type of application

Some tools are better suited to certain applications than to others. To use an analogy of house-painting. If the walls and windows of a house need to be painted, then, the best way to paint the walls is probably to use a large roller or paint-spray. But if these are used on the windows the results tend to obscure the view!

The discrete simulation equivalents of the paint-sprayers are the VIMSs which provide a rapid way of developing models with attractive graphics. For relatively straightforward applications, many of which are found in factories and back-office processing, these tools are hard to beat.

However, there are times when very detailed systems need to modelled and this will require the use of tools which make it easier to express complex system logic and this usually involves programming. Thus for a smaller proportion of discrete simulations (probably less than 20%) there is no escape from the skilled process of developing a computer program.

4.2.2 Required features

The next issue to face is the detailed demands which will be placed on the software. Examples of these issues are the following.

- Is graphical display important?
- Does the software need to interface with corporate systems (e.g. databases)?
- Do you need strong statistical support for input and output analysis?
- Is security important?
- Are debugging tools required?
- What hardware/software platform will it be run on (e.g. UNIX or DOS/Windows?)?

The software vendors need to be asked to specify a system to meet your requirements.

4.3.3 Vendor support and prices

Finally, no user can ignore two very practical issues. How much will the software cost? - and remember that software prices are soft, so negotiation is often possible. Also ask what support you can reasonably expect from the vendor given the vendor's size and given your physical location.

REFERENCES

Blaisdale, W.E. and Haddock, J (1993) *Simulation analysis using SIMSTAT 2.0.* Proc 1993 Winter Sim Conf, Los Angeles Cal.

AT & T Istel (latest version) *Witness system manual.* AT & T Istel, Redditch Worcs, UK

CACI (latest version) *SIMFACTORY introduction and user's manual.* La Jolla, Cal.

CACI (1987) *PC SIMSCRIPT II.5. Introduction and user's manual.* La Jolla, Cal.

Clementson, A.T. (1991) *The ECSL plus system manual.* Available from A.T. Clementson, The Chestnuts, Princes Road, Windermere, Cumbria UK

Conway, R., Maxwell, W.L., McClain, W.L. and Worona, S.L. (1990) *User's guide to XCell+ factory modelling system, release 4.0. (3rd edition).* The Scientific Press, San Francisco, Cal.

Crookes, J.G. (1989) *Simulation in C.* In Pidd, M. (ed) *Computer modelling for discrete simulation*, John Wiley & Sons Ltd, Chichester.

Forrester, J.S. (1961) *Industrial dynamics.* MIT Press, Cambridge, Mass.

Harrell, C.R. and Leavy, J.J. (1993) *ProModel tutorial.* Proc 1993 Winter Sim Conf, Los Angeles Cal.

Joines, J.A., Powell, K.A. and Roberts, S.D. (1992) *Object-oriented modelling and simulation in C++.* Proc 1992 Winter Simulation Conference. Arlington VA.

Kleijnen, J.P.C. & van Groenendal, W. (1992) *Simulation: a statistical perspective.* John Wiley & Sons Ltd. Chichester

Law, A.M. & Kelton, W.D. (1991) *Simulation modeling and analysis, (2nd edition).* McGraw-Hill, New York

Pidd, M (1992) *Object-orientation and three phase simulation.* Proc 1992 Winter Simulation Conference, Arlington VA.

Pidd, M (1989) *Simulation in Pascal.* In Pidd, M. (ed) *Computer modelling for discrete simulation*, John Wiley & Sons Ltd, Chichester.

Pidd, M (1988) *Computer simulation in management science. (2nd edition)*, John Wiley & Sons Ltd, Chichester.

Pidd, M (1992) *Computer simulation in management science. (3rd edition)*, John Wiley & Sons Ltd, Chichester.

Richmond, B.M. and Peterson, S (1988) *A user's guide to STELLA.* High performance systems Inc, Lyme, NH.

Schruben, L (1991) *SIGMA: a graphical simulation system.* The Scientific Press, San Francisco, Cal

Szymankiewicz, J., McDonald, J. and Turner, K. (1988) *Solving business problems by simulation.* McGraw-Hill, Maidenhead.

Vincent, S.G. and Law, A.M. (1993) *UnitIII: total support for simulation input modeling.* Proc 1993 Winter Sim Conf, Los Angeles Cal.

Wolstenholme, E.S. (1990) *System enquiry: a system dynamics approach.* John Wiley & Sons Ltd, Chichester

Zeigler, B.P. (1976) *Theory of modelling and simulation.* Wiley Interscience, New York.

Zeigler, B.P. (1984) *Multifacetted modelling and discrete event simulation.* Academic Press, New York.

AUTHOR BIOGRAPHY

MIKE PIDD is a Professor of Management Studies in the Management School of Lancaster University in the UK. He is author of *Computer simulation in management science* and of *Computer modelling for discrete simulation* (both published by John Wiley). He teaches, researches and consults in discrete simulation and management science. His current interests include research on object orientation and an upcoming book on Modelling.