Proceedings of the 2004 Winter Simulation Conference
R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

SIMULATION WORLDVIEWS - SO WHAT?

Michael Pidd

Department of Management Science
The Management School
Lancaster University
Lancaster, LA1 4YX, UNITED KINGDOM

ABSTRACT

The simulation pioneers had no choice but to write code if they wished to conduct a computer simulation. Hence the early interest in simulation worldviews, which allowed an application model to be separated from a simulation engine. Nowadays, few simulations are developed this way and few students are taught the various simulation worldviews, though they figure in many textbooks. Does this matter, or is an interest in simulation worldviews just a historical curiosity?

1 INTRODUCTION

1.1 Simulation for the People?

I suspect that the vast majority of discrete simulation models are constructed using what I call VIMS (Pidd 2004) and Law and Kelton (2000) term Simulation Packages. These are shrink-wrapped software systems such as Witness, Simul8, MicroSaint and Automod in which models are built by point and click using predefined objects for which the user must supply properties. They require little in the way of programming and their popularity is evidenced by their presence in the WSC Exhibition Hall for the last 10 years. Though there is no proper survey evidence to support this assertion, the proportion of discrete simulation applications that use a VIMS may be as high as 90%, with many of the rest being large-scale models developed in the defence sector. The latter, by contrast, usually involve significant programming and may require explicit management of the events that comprise the dynamic behaviour of the simulation. The pros and cons of the two approaches are discussed at length in Pidd (2004).

Why are VIMS so popular? Because they offer the prospect of rapid application development by people who are not computer professionals. Indeed, the vendors of Simul8 have, from time to time, implied that it could be to simulation what Excel has been to business modelling. There is no doubt that a VIMS can be easy to learn, at least for straightforward applications, as most simulation instructors will testify. However, it is also true that a VIMS can run out of power when faced with large and complex applications. This, of course, may not matter for routine business applications in which the 80:20 rule may apply: that is, a simple model may be good enough. After all, few senior managers would invest very large sums in a one-off capital development based on a complicated stochastic analysis with wide and overlapping confidence intervals. Approximate models are often good enough, and VIMS are good enough for approximate models.

1.2 Inside a VIMS

1.2.1 Machine and Task Networks

Figure 1 shows the logical composition of a typical VIMS. It presents a user interface that exploits the API of whatever operating system is in place (usually a version of Windows™). Within this, the user will be allowed to develop a model, to edit an existing model, to run a model and to conduct controlled experiments. The latter usually allows at least some statistical analysis, and allows the export of results files in some suitable, external format. Models are constructed by selecting icons that represent features of the system being simulated and these are linked together onscreen, and parameterised using property sheets. This is fine if the objects provided are a good fit for the application. However, the default logic provided by the simulator may be inadequate to model the particular interactions of specific business processes. To allow customisation, most VIMS provide a coding language in which interactions can be programmed. Some offer links with general-purpose programming languages (e.g., Visual Basic). Others incorporate simulation quasi-languages that permit little beyond the assignment of attributes, the definition of if statements, loops and limited access to component properties. Some of the limitations of these languages and their part in VIMs are discussed in Melao and Pidd (2004).

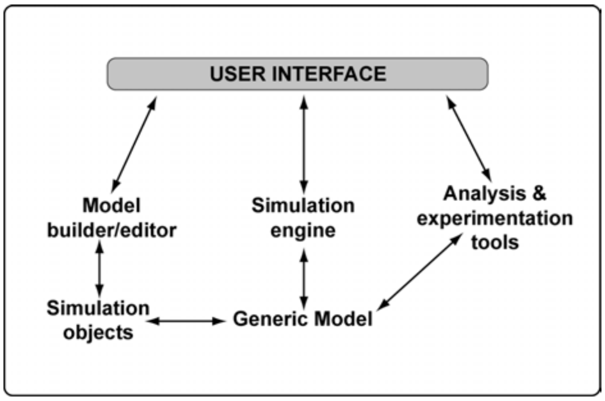

Figure 1: Inside a Typical VIMS

Underneath the user interface is a generic simulation model that is presented to the user in one of two ways, sometimes both. The most common way, as seen in Witness and similar packages, is that of a machine network in which parts flow from work station to work station. For example, a part may go to a lathe, then through a washer, then to an inspection point and then into a shrink wrapper ..etc. A work station may well be able to carry out more than a single task and may be able to cope with more than a single type of part. As parts flow through the network, they sit at a work station for a time, possibly stochastic, often known as the cycle time. As a result of their interaction with the work station during the cycle time, the parts change state. Parts are routed through the machine network and each machine must be parameterised by the completion of a sheet that specifies its predefined properties.

By contrast, systems such as MicroSaint offer a task network that represents the sequence of tasks through which the main entities of the simulation will flow. Thus, passengers may disembark from an aircraft, may walk to immigration, may be processed in immigration, walk to the baggage hall ..etc. Each task requires resources for its completion and tasks may compete for resources. The resources required, and the conditions governing the start of the task are specified in a property sheet, along with the consequences of the task.

Of course, these two types of VIMS network are equivalent; much as the dual formulation of a mathematical programming problem is equivalent to its primal. That is, with suitable imagination, an application that appears to be a sequence of tasks may be modelled as a machine network and vice versa – however, choosing a horse for the right course can make life much easier.

In addition, whatever the vendors may claim, there are very, very few (if any) real applications that can be fully developed just by the completion of property sheets. Instead, each VIMS provides its own simulation language, which is used to capture those aspects of the situation that are not easy to represent in a property sheet (e.g. after 4:00 pm, only jobs judged as urgent will be processed, otherwise the machine is cleaned).

1.2.2 A Generic Model

Inside every VIMS is a generic simulation model that is not usually available to the user. In essence, the generic model takes the network diagram as data, much as GPSS (Gordon 1969) was designed to take a series of punched cards that were then interpreted by GPSS. Thus, if it were thought desirable, a VIMS on-screen network could be replaced by a series of verbal commands each of which carries attribute data to represent the property sheets. The generic model assumes that simulation entities change state and it reads the network description to define the sequence of those states and the conditions that govern them. The code fragments, in whatever simulation language, are then used to modify the standard generic model in some way or other.

One suitably parameterised, the generic model is run by a simulation executive, which sequences and schedules the tasks that define the application model. Like the generic model, the simulation executive is also hidden from the user. Hence, the form of the simulation executive is rarely revealed by the VIMS vendors, who are keen to present their software as easy to use and also powerful. How the simulation runs is deemed to be of little concern to the users.

2 SIMULATION EXECUTIVES AND WORLDVIEWS

Hollocks (2004) provides a fascinating account of the early days of discrete simulation in UK industry. It seems that the idea of simulation executives and worldviews emerged from intelligent thinking about practical experiences.

The first discrete simulations were executed by hand, with game boards and people acting as if they were the machines and entities of the simulation. This permitted the modelling of relatively simple applications and the simulations were slow to execute and, therefore, allowed limited experimentation and replication. Before too long, people replaced these games with simple computer programs that were written using the only available software technology – the flipping of switches and, slightly later, the writing of machine code and assembler. Needless to say, such programming was slow and error-prone and required detailed knowledge that was in short supply. Eventually, general purpose programming languages appeared and they eased the task of program development.

At some point in this process, the simulation application developers and programmers realised that many of their simulations had the same structure – entities changed state through time, as enabled by available resources. The nature of the entities and resources and the sequence of state changes would vary from application to application, but underneath were entities, resources and state changes through time. This is analogous to the need of busy people to organise their lives with a diary system in which future commitments can be entered and, periodically, checked to see what commitments need to be met. Thus, the concept of simulation worldview (Kiviat 1969a, 1969b) became of great concern to developers of simulation software.

Hence there emerged the separation of function that characterises most well written discrete event simulation programs. As shown in Figure 2, there is

1. A general purpose simulation executive that can be called by
2. An application program that complies with the rules of the executive

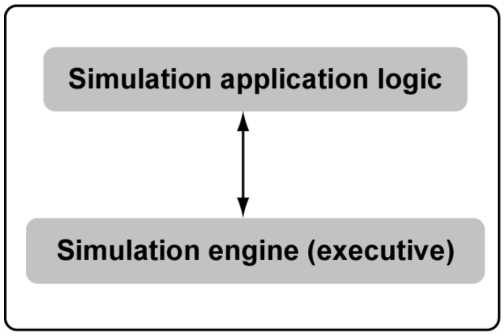

Figure 2: Executive and Application

The executive's job is to keep track of any future commitments in its diary and to remind the application program when, in simulation time, a particular state change is due. The rules by which the application communicates with the executive represent and comprise what has come to be known as a simulation world-view. In the lexicon of contemporary computing, they define the architecture that allows the application components and executive components to interoperate. Without a clearly specified architecture, the interoperation is impossible.

3 THREE PHASE SIMULATION

It can be confusing when simulation worldviews are discussed in general and very abstract terms; hence this section includes a brief review of one such worldview – the three phase approach.

Pidd (2004) provides a detailed account of the workings of the three phase worldview developed by Tocher (1963) and his colleagues in the UK steel industry. Sadly, many US-based writers do not understand how a three phase simulation operates and confuse it with what is usually known as activity scanning. Hence, a short summary is needed here. I can present cogent arguments about why a three phase approach is preferable to the other worldviews, but I suspect that my real preference is based on my initial simulation education as well. Hence I shall make no real attempt to argue that a three phase approach is best – though it is!

In essence, the three phase approach recognises that a simulation entity will change state if, and only if, defined conditions are met. These conditions can be divided into two groups: those that depend only on the passage of time and those that depend on other conditions in the simulation.

3.1 Bs

Some operations have a starting or finishing time that can be predicted in advance. These can therefore be scheduled as if they were appointments being entered into a diary, and these are known as Bs. Originally they were called B activities, which was an abbreviation for Book-keeping activities or Bound activities, the term 'bound' indicating that they were bound to happen at some specified time, and the term 'book-keeping' that they might used for keeping regular records (for example of queue lengths). Some people refer to B events instead, and to avoid confusion they are called Bs here. As a general rule, to which there are some exceptions, when a simulation includes an activity that takes some defined time, the end of that activity is modelled by a B. Hence, the usual effect of a B is to release resources and entities. For example, when the machining of a part is finished, this is modelled as B, which releases the part for the next stage of its route and releases the machine for its next task.

Because these Bs can be directly scheduled, the simulation executive can precisely control when they will occur by ensuring that they are executed when the simulation clock reaches the correct time. Hence, each B must have an entry in the event calendar which serves as a reminder to the executive to take some action. A suitable analogy for this would be a central heating controller in which the start time for the heating is set at, say, 6.30 am. When the clock of the controller reaches 6.30 am, it triggers the central heating system into life.

3.2 Cs

Operations that are not Bs are regarded as Cs. This was originally an abbreviation for Conditional activity or Co-operative activity, the idea being that such activity is not dependent on the simulation clock but must wait until the conditions are right or until some other entity is ready to co-operate in the task. As with Bs, some people refer to C events instead, and so the term C will be used here. The simulation executive has no direct say in when these Cs will occur, for this will depend on the states of the entities and resources in the simulation. Obviously the executive has some indirect control, since the main effect of the Bs (which it does control) is to release entities and resources.

3.3 A Typical Three Phase Executive

The flowchart of Figure 3 shows the operation of a typical three phase executive. In the A phase, the simulation clock is moved to the next event time by checking all the Bs that are currently scheduled. Those Bs that are now due are executed in some defined sequence so as to release resources. The third phase is a C scan in which the Cs are attempted in some defined priority sequence.

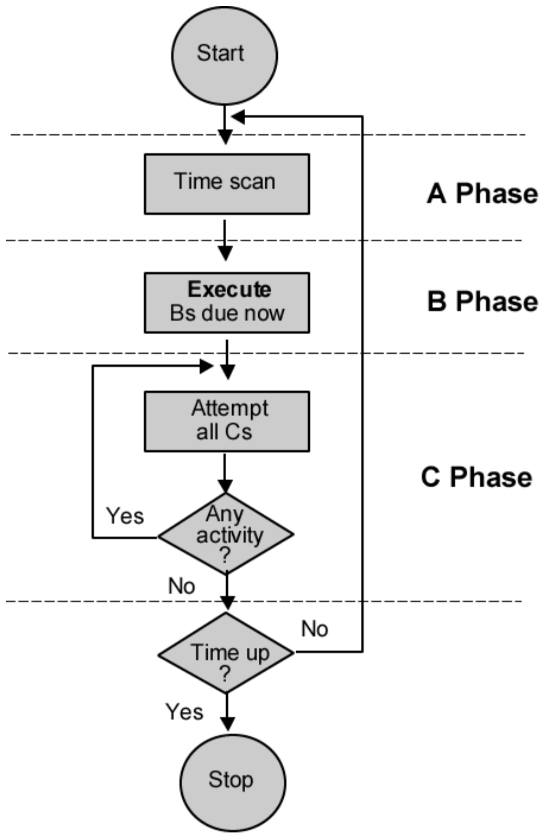

Figure 3: A Three Phase Executive

It is important to realise that a three phase executive processes all the Bs due at some simulation time and then attempts all the Cs. That is, no Cs are attempted until all the resources that can be released by the Bs are released. The effect of a C may be to commit resource and this is done when the full position of all resources is known – at the end of the Bs. This avoids the deadlock problems that can plague discrete simulations in which there is resource contention.

4 DOES ALL THIS MATTER?

Since few simulation models are built in ways that require the modeller to know about the simulation executive and its worldview, does any of this matter? The answer is 'probably not' if we only need to build a simple model or if we are trying to sell a VIMS on the basis that it will help solve of particular problem. However, I can conceive of several situations in which it does matter and will briefly explore each.

4.1 Large-Scale, Hand-Coded Models

As mentioned earlier, some large-scale models are still developed by writing code in a general purpose programming language and it is clear that the developers of these simulations do need to attend to their simulation worldviews. Nothing more will be said about this.

4.2 Simulation Education

Discrete event simulation is taught to a wide range of groups at several levels. Audiences include students of engineering, business and operational research and they may be undergraduates or graduate students, or may be taking a professional post-experience programme. Clearly many of these need only a superficial introduction to simulation approaches, but some do need rather more than that.

An analogy will perhaps help illustrate the point. A car driver does not need to know much about how a car, its engine, its transmission and other features work. Instead, most people merely learn how to operate a car with some degree of safety. However, some drivers do need to know much more than this and other drivers may wish to know more out of curiosity and doing so may even make them better drivers. Firstly, of those whom need to know more, are those whose job will involve them advising others about the purchase and use of a car. They don't need very detailed knowledge, but do need to know more than how to safely drive a car. Secondly there are those who will become mechanics and will service cars and fix them when they go wrong. Finally, there is a small group who will go on to design new cars and they need very detailed knowledge indeed.

In the simulation world, most people will only learn roughly how a simulation operates and a few things about what can go wrong. This probably applies to most business students, who may never even become competent 'drivers' of the simulation packages but may later become clients for a simulation study. The second group of 'mechanics' are those who may need to use a VIMS to solve some particular problem. Hence they learn how to drive the VIMS, but also about how to conduct a simulation study and how to design and analyse simulation experiments. Finally, there is a small group – probably those specialising in computer science or operational research, who need to know how to make simulation software sing and dance. This final group do need to understand about simulation worldviews, for then they will understand what can go wrong in a simulation model and why.

4.3 Simulation Research and Development

Of the final group mentioned above, there is a yet smaller subset who will be the designers and implementers of new simulation software. If they do not understand the options open for the internal design of discrete simulation software, they are doomed to devise a rather limited offering. As it happens, the ways in which a simulation model can be organised and communicate with its executive (its worldview) are few and none are that difficult to understand. However, the insistence of software vendors on hiding the internal operation and on charging high prices for their products tends to reinforce the view that the internal operation of discrete simulation software is open only to the super-intelligent.

4.4 Simulation Model Reuse and Interoperation

Finally, some sectors – notably defence, but others may follow – have such large investments in existing simulation models that they wish to squeeze yet more value out of them. One way of doing so is to treat existing models as components that interoperate, possibly as an HLA federation. Whether this is a good idea or not, is another question discussed in, for example, Robinson et al (2004). To do so, though, requires detailed knowledge in several areas.

The first, and most relevant to this paper, is the internal workings of the simulation and, in particular, the ways in which simulation events and activities are processed. Mathewson (1974), writing about program generators, was surely correct in pointing out that a simulation application can be equivalently modelled in any of the common simulation worldviews. However, without appropriate skills and knowledge, this equivalence cannot be guaranteed. Hence, it would seem wise for anyone attempting model interoperation to have a detailed understanding of the different simulation worldviews – just in case!

Linked to this, and indirectly relevant to the paper, is the need to provide adaptor wrapper components (Oses et al 2004) to enable the interoperation of models that were not designed with such reuse in mind. Terms such as 'adaptor' and 'wrapper' have an easy feel about them, which may hide some considerable complexity. It seems reasonable to argue that correct external modification is more likely if due heed is paid to internal operation. Hence, this group, too, requires detailed knowledge of simulation worldviews and how they are implemented.

5 CONCLUSIONS

The conclusions are probably obvious. In today's world, in which most discrete simulations are implemented using VIMS, it is not sensible to insist that everyone learn about simulation worldviews when learning about simulation. Indeed, it is unreasonable to argue that the vast majority of users need this.

However, there are people who need this knowledge, typically those specialising in computer science and operational research. They have the time, the inclination and the likely career need to gain this knowledge. Some if them will develop the next generation of simulation software and others will need to tweak existing software. If they are convinced, as I am, that a three phase worldview is best, so much the better.

REFERENCES

Gordon G. 1969. *System simulation*. Prentice-Hall, New Jersey

Hollocks B. W. 2004. Still simulating after all these years. Reflections on 40 years in simulation. *Proceedings of the OR Society Simulation Workshop*, Birmingham, March 23-24, Operational Research Society, Birmingham.

Kiviat P. J. 1969a. Digital computer simulation: computer programming languages. *Rand Corporation, AD 684 124*, Santa Monica, CA.

Kiviat P .J. 1969b. Digital computer simulation: modeling concepts. *Rand Corporation, RM-5378-PR*, Santa Monica, CA.

Law A. M. and W. D. Kelton. 2000. *Simulation modeling and analysis* (3rd edition). McGraw-Hill, Boston, MA.

Mathewson S. C. 1974. Simulation program generators. Simulation, 23, 6, 181-189.

Melao N. and M. Pidd. 2004. Using component technology to develop a simulation library for business process modelling. Under revision for *Euro Jnl Opl Res*.

Oses N., M. Pidd and R. J. Brooks. 2004. Component-based simulation. In Press, *Simulation Practice and Theory*.

Pidd M. 2004. *Computer simulation in management science* (5th edition). John Wiley & Sons, Chichester.

Robinson S., R. E. Nance, R. J. Paul, M. Pidd and S. J. E. Taylor. 2004. Simulation model reuse. Definitions, benefits and obstacles. In Press, *Simulation Practice & Theory*.

Tocher K. D. 1963. *The art of simulation*. English Universities Press, London.

AUTHOR BIOGRAPHY

MIKE PIDD is Professor of Management Science and Associate Dean (Research) in Lancaster University Management School. He is known for his work in 2 areas: computer simulation and the complementary use of soft and hard OR. His text 'Computer simulation in management science's is now in its 5th edition and his books on complementarity include 'Tools for thinking' (in its second edition) and 'Systems modelling: theory and practice'. All are published by John Wiley. For the whole of 2004 he is a Research Fellow in the UK's Advanced Institute of Management Research, funded by the ESRC, examining performance measurement in the public sector. Email to `<M.Pidd@lancaster.ac.uk>`, website at `<http://www.lancs.ac.uk/staff/smamp/>`.