Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

MODEL DEVELOPMENT AND HCI

Mike Pidd
Department of Management Science
The Management School
Lancaster University
Lancaster LA1 4YX
UK

ABSTRACT
Much of the progress within discrete simulation has gone hand-in-hand with general developments in computing. Recent years have seen software developers putting great efforts into improving the user interfaces of discrete simulation systems. This too parallels developments elsewhere. This paper considers what further benefits there might be for users and developers of simulation software from more careful attention to interface design.

1 INTRODUCTION

As computers have grown more powerful, so have users' expectations of how they might be used. Today's school students now assume that computers will be fast, friendly and easy to use. Many of them, certainly in the USA and Western Europe, are video game experts, connoisseurs who are easily bored with experiences that do not live up to their expectations. (See Microserfs, Coupland (1996), for amusing examples of this). These people will be the next generation of simulation modellers and users and they may not be satisfied with the interfaces available in today's discrete simulation software. Thus, it seems important that the designers and vendors of simulation software start to take the question of user interface design very seriously. This paper presents the basic requirements for simulation software that stem from a particular view of modelling that is argued elsewhere (Pidd, 1996a, 1996b). These are linked to current theories and ideas about HCI (Human Computer Interaction or the Human Computer Interface) to present desirable features of future discrete simulation software.

1.1 Historical Background

It is probably true to say that developments in discrete simulation software have proceeded hand-in-hand with general developments in computer software. There have been a few cases in which discrete simulation software has led the field, examples being object orientation in Simula (Dahl and Nygaard, 1966) and the introduction of interactive program generators by Clementson (1991) in his ECSL system. However, in most cases, discrete simulation software developers have been willing to pick up and use whatever tools and ideas appear within the computing community. Examples of this being the use of animated graphics for visualisation by Hurrion (1976) and the use of source level debuggers to support program development.

The purpose of this paper is to examine some of the links between simulation modelling and what is now known as HCI. Given that the idea of simulation is to use a computer as a dynamic model of some situation or system, then it would seem that the question of appropriate HCI is an important one. This paper begins by considering the different ways in which discrete simulation software is provided, looks at the different roles which people play in developing and using simulation models and then speculates on how things might be improved. The question at issue is, what can simulation users and developers learn from the field of HCI? In addition, is there anything that the HCI community can learn from discrete simulation?

2 THE USE OF DISCRETE SIMULATION SOFTWARE

2.1 Types of Discrete Simulation Software

Different authors have their own ways of discussing the various types of discrete simulation software that are now available. Law and Kelton (1991) divide this software into two main groups, those which require programming and the rest. The latter they describe as simulators. The former category includes general purpose programming languages and libraries such as C++ or FORTRAN, as well as dedicated simulation programming languages such as the SIMSCRIPT family. A rather more finely drawn categorisation is found in Pidd (1992), who has no less than seven categories, ranging from 'do-it-yourself' (in a general purpose programming language), through to Visual Interactive Modelling Systems (VIMS) such as Witness (Lanner Systems, 1996) and Microsaint (Micro Analysis and Design, 1992), via flow diagram or block diagram systems such as HOCUS (Szymankiewicz, McDonald and Turner, 1988) and GPSS.

This paper uses a simple tripartite classification as follows.

1. Software that requires all users to develop some true program code. This includes general purpose programming languages as well as simulation programming languages. It also includes those languages that provide support for visualisation and those that do not.
2. Visual Interactive Modelling Systems which are primarily based around some kind of visual motif for virtually all of their functions. In these systems, examples of which include Witness, Microsaint, ProModel and AutoMod, a visual interface is used for model building, controlling simulation runs and for interaction as a simulation run proceeds.
3. Layered systems in which the user can operate at a number of different levels, such as visual interactive modelling, automatic program generation, direct coding and low level bit twiddling. A few systems currently available incorporate some of these facilities and an example would be ARENA.

2.2 Groups of People Involved in Simulation Projects

Though this conference primarily focuses on those people who conduct the technical aspects of simulation projects, there is quite a range of people who may be involved across the set of activities that make up such projects. These may be different people or might be different roles that are conducted by one or more people. Examples include the following.

1. *Modellers*: whose main role is to understand the system being simulated and to capture that understanding in a model that is implemented in some software system or other. Thus the software interface must offer support to modellers as they develop, usually in a stepwise manner, their abstract and symbolic models.
2. *Programmers*: who have the task, in some projects, of taking model conceptualisations and realising them in program code. In an ideal world this would enable the programmers, though coding, to operate at a level which relates to the problem being modelled, rather than just the computing aspects of the work.
3. *Project managers*: who may be responsible for ensuring that the project meets the needs of the clients, is on-time and is properly conducted in a technical sense. This group of people may wish to establish standards that enable projects to be properly managed and consistently presented to clients.
4. *Customers*: who are footing the bill for the project and whose questions are being addressed in the project itself. This concern of this group may be to ensure that they get value for money and this may be aided by suitable HCI considerations.
5. *Users*: who may not be the same as the customers, but who may need to use a simulation model on one or more occasions to address issues of interest. This group may not be expert in simulation and may have limited expertise in computing. Thus the interface needs to be very supportive and should build on their previous experience and expertise.

In addition, of course, there may be software vendors keeping an eye on what is happening.

2.3 Discrete Simulation Modelling

A computer simulation involves experimentation on a computer-based representation of some system or other. This particular conference is interested, mainly, in discrete event simulation which is suited to systems made up of objects that change state and display dynamic behaviour. That is, those which change state through time. Thus, to build a discrete event model, the analyst must attempt to tease out and represent a number of features of the system being modelled. These are as follows.

1. *The main entities of the system*: that is, those classes of object whose behaviour seems important to the operation of the system. Note that word *seems*, if the modeller were absolutely sure of the important entities and if this were not a subject for thought, debate and argument, then it might not be necessary to model and simulate the system at all.
2. *The logical operations of these entities*: that is, the significant behaviour in which the classes of object engage as they persist, or appear, or disappear within the system being modelled. Once again, the previous sentence includes a weasel word, *significant*. This suggests that the modeller must make judgements about what features of the entities' behaviour deserves inclusion within the model.
3. *The logical interactions of the entities*: systems would be very simple to model and to simulate if the objects within them did not engage with one another. It is these complicated interactions through time that determine how the system itself will appear to behave and some of these interactions may be hard to tease out or may occur only rarely.
4. *The statistical distributions*: the entities will engage in activities, alone or whole other object classes

and these activities may occur at irregular intervals and for time periods that cannot be precisely determined in advance. It is normal to model these durations and intervals using statistical distributions that are believed to be good representations of the observed behaviour.

Thus an important HCI task is to ensure that the interface offers support on this model building, especially that it eases the gradual development of a model.

3 BASIC IDEAS OF HUMAN COMPUTER INTERACTION/INTERFACE

3.1 Interface Design

The main concern of HCI is to ensure that computer software and hardware is well suited to its users and to the tasks that they wish to perform. Early work on HCI tended to focus on the specialised needs of people such as pilots, engineers running large, complex systems and others such as air traffic controllers who worked with computers in situations of great pressure, risk or danger. More recently, the emergence of the Apple MacIntosh and other windowing-type operating systems has brought some of the benefits of the this early work to users of general purpose computer systems. HCI includes the design of hardware as well as software, but for the purposes of this paper, HCI is restricted to the software that enables users to interact with computer systems. This software interface should reflect the characteristics of the people who wish to use the system and the tasks and functions that they have in mind as they approach the computer system.

Lansdale and Ormerod (1994) suggest that the development of good software interfaces requires the co-operation of two groups of people, designers and psychologists. The designer must ensure that the software is able to achieve the tasks which the user may require of it. The psychologist must focus on the user to try to understand how the user might wish to conduct these tasks. This co-operation occurs during task analysis which aims to understand the logical structure of complex tasks, to identify the information needed by the users to carry out these complex tasks, to understand the activities conducted by users with the interface and to appreciate the conditions that may affect their performance.

On the same theme, Norman (1990) suggests that the interface designer must bear three things in mind if the interface is to be well-designed.

1. The types of people who may wish to use the system.
2. The tasks that these people wish to accomplish.
3. The tools that are most appropriate in enabling them to do those tasks.

This is an extension of what Norman (1988) terms 'user-centered design'.

3.2 User-centered Design

Norman (1988) suggests that designers should take account of seven principles if they wish to achieve a user-centered design.

1. *Use both knowledge in the world and knowledge in the head.* Knowledge in the world is that which is available externally, especially visibly and which will suggest what the function of some system component may be. Knowledge in the head is that which the user is able to internalise as they become practised and proficient and for which few cues are needed. Hence, at its simplest level, there should always be enough clues on-screen for the naive user to know what to do and yet the expert user should also feel comfortable and should not be frustrated by a having to step through a mind-numbing routine of protocols to achieve some simple end.
2. *Simplify the structure of tasks.*: This requires the designer to take account of the limitations of human memory. This is often conceptualised as *short term memory*, which is often believed to be capable of holding 7±2 items and *long term memory*, which seems to develop conceptual frameworks within which new experiences are considered. Thus, software must not overload short term memory, and it should be a good fit with the users' conceptual frameworks. Software that clashes markedly with a user's likely conceptual framework is likely to be difficult to learn and will be error prone in use.
3. *Make things visible.* Designers should struggle to ensure that sufficient information is visible to the user to enable them to do what they wish, within the limitations of the software. This clearly relates to the first principle of using knowledge in the world - it requires the designer to ensure that suitable cues are available to the user.
4. *Get the mappings right.* This principle requires the designer to ensure that the user can understand the links between the options, the information presented and the tasks that they wish to perform. The links between the users' intentions and the actions open to them in the software need to be made very clear. It also relates to the need to ensure that users have appropriate conceptual models, otherwise they may misunderstand the function of the system and its components.
5. *Exploit the power of constraints*: Constraints are those are restrictions that are deliberately built into the software to ensure that dangerous or bizarre actions

cannot be attempted. This might mean, on occasions, frustrating the user who wishes to cut corners.

6. *Design for error.* This principle requires the designer to assume that the user may make mistakes. It is vital, therefore, to ensure that, if mistakes and slips occur, the behaviour of the software is safe and helpful to the user. It is unreasonable and downright stupid to assume that even experts never make mistakes.

7. *When all else fails, standardise.* The idea of this final principle being to limit the amount of learning that the user must achieve in order to accomplish their required tasks.

Clearly, these principles are inter-dependent and overlap with one another. But their overall message is very clear. Focus on the user and make things easy for them. This requires the designer to know who the likely users will be and to understand what they may wish to do.

3.3 Some Principles of Learning

Kay (1990) builds on Bruner (1962) in suggesting that human cognition is made up of three interlinked aspects as follows, from the simplest through to the more abstract. As learning occurs, an individual moves through these levels.

1. *The enactive mentality.* In this, there is a focus on direct experimentation, on trying things out. It relates to very concrete thinking. Expressed in computer interface terms, this requires the user to be aware of exactly where they are in the system and exactly what is open to them at that point. Given that Bruner (op cit) argues that this is the way in which many of us start to learn about new things, this suggests that the interface be designed to ensure that its exploratory use can do no harm.

2. *The iconic mentality.* In this, the individual tries to recognise, compare and configure different aspects of their surroundings. This is rather more abstract than an enactive approach in that it suggests that a person may wish to change the world in some way or other and to make simple inferences. In computer interface terms, this means that the user needs recognisable images and objects whose function is clear. This should then reduce the need for learning by enactment and experimentation.

3. *The symbolic mentality.* In this, the individual tries to tie together chains of reasoning in a way that may be very abstract. In essence, the person has conceptual models which are applied to the task at hand. There is thus a need to see the connect between different symbols and to understand how they may be linked to achieve particular tasks. In discrete simulation terms, the interface should support modelling.

In these terms, simulation modelling operates with a symbolic mentality, but Kay(op cit) points out that this will not be achieved unless the other two levels are in place. That is, the ability to conduct symbolic reasoning sits on iconic and enactive foundations. Thus, useful discrete simulation software must make provision for all three.

3.4 The Effect of the Interface

Finally, it is important to bear in mind the comment made by Lansdale and Ormerod (op cit). Though they are insistent that good, user-centered design must be based on a good fit between task, technology and the user, they argue that one other aspect is of crucial importance. This is that the users' goals may not be static through time. There are two reasons for this, one of which may not be obvious.

The obvious point is that the task itself may change for reasons quite unconnected with the software. Its scale might increase (e.g. air traffic controllers might need to deal with a much more crowded airspace) or other changes may take away much of the task (e.g. decisions on the sharing of air space amongst national traffic controllers). Thus a sensible task analysis needs to look at possible changes in the task.

The less obvious point, and the one that matters for discrete simulation, is that the nature of the interface may change the task itself. To use a non-simulation example, consider word processors. They have two main virtues. The first is that they enable any trained user to produce work that is well-presented, hence this conference can provide presentation guidelines that enable authors to meet their requirements for camera-ready copy. The second is that it changes the nature of the task by allowing people to adopt a different approach to their writing. Writing with pen and ink requires a sentence, a paragraph even, to be composed in the head before it is committed to paper. This forces the pen-and-paper writing style to be thoughtful and considered (in the opinion of Stoll (1995)) or ponderous, in the opinion of others. Using a word processor, especially one with a good outliner, allows the user to hammer away at the keyboard with a stream of ideas that can later be expressed as lucid prose. Hence the software, and its interface, can change the nature of the task.

4 IMPLICATIONS FOR DISCRETE SIMULATION SOFTWARE

4.1 User-centered Design

Norman's seven principles can, perhaps, be reduced to four when viewed within the use of today's window-centric world. These are as follows.

a) Simplify the structure of tasks.
b) Get the mappings right.
c) Exploit the power of constraints.
d) Design for error.

This is not to say that today's windowing operating systems are perfect, far from it. However, adherence to their norms at least helps to enforce Norman's other three principles. A user who is familiar with other windowing software will at least have useful knowledge in the head and will know where to look for knowledge in the world. Such systems are also essentially visual (which presents severe problems for some users of computers with disabilities) and they provide some form of standardisation.

Principles a) and b) above relate to Kay's point about symbolic, iconic and enactive mentalities. That is, they relate to the need to ensure that the conceptual model of the software is abundantly clear in its interface and for it to fit with that assumed by the user. Those of the user are provided, in part at least, by the conceptual frameworks of their long term memory. The interface and its metaphors need to be consistent, clear and must make sense to the user, but this is not easy.

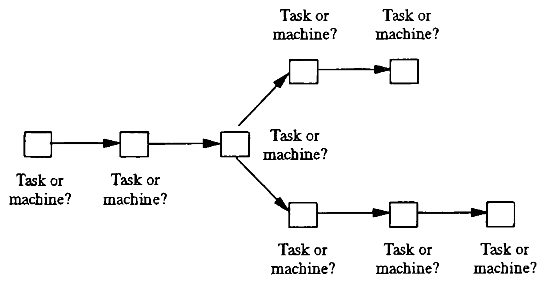

Figure 1: Task network or machine network

Consider for example, the metaphor that is usually employed in Visual Interactive Modelling Systems (VIMS). In these, the user develops a model by placing icons on-screen and by linking them with directed arcs in a network of some kind, see Figure 1. Though it may not be obvious, there are at least two ways in which this can be done - and this will not be obvious to the naive user.

1. Make this a task network. This is the approach employed in software such as Microsaint (op cit). Each node represents a task and each of these tasks may employ several resources which are committed during the duration of the task. Thus, for example, a medical consultation may require the co-operation of a patient, a doctor, a nurse and some specialised equipment. These resources are freed at the end of the task. One of these resources is modelled as the system entity which flows through the task network.
2. Make this a machine network. This is the approach commonly taken on manufacturing-oriented VIMS such as Witness (op cit). Each node represents a system resource that will be occupied as a system entity (usually an item being manufactured) passes through it. The node represents a machine that may well be capable of performing several tasks.

These distinctions will be not be obvious to a naive user from the interface nor may they be obvious from simple examples. The user may suddenly be brought to the jarring realisation that their conceptual model is a task network (or vice versa) and the simulation software assumes a machine network (or vice versa).

Principles c) and d) relate to the need for safety. They aim to minimise the risk that the user may make slips (accidental errors and abuses) or errors (deliberate actions that turn out to be misconceived). The interface and its metaphors must be clearly designed with this in mind. In the case of discrete simulation software, if designers wish to take these ideas seriously, then serious issues about various statistical aspects of the model and the simulation need to be taken into account. As a simple example, most VIMS permit the use of Normal distributions, but even sophisticated users may sometimes forget that Normally distributed variates range from plus to minus infinity. In most cases, users and modellers are probably assuming bounded Normal distributions. Whether this is what they will get may not be clear. If it is what they get, the bounds may not be stated anywhere that is easily accessible to the user.

4.2 Needs of Different Groups

It is probably true to say that early simulation software systems concentrated on the needs of modellers and programmers. The idea being to give maximum support to the technical aspects of the task in hand. That this effort has been successful is evident in the widespread use of VIMS and the growing acceptance of layered systems and approaches. Though the early, first-generation user interfaces of these systems now seem somewhat crude, the latest generation does seem to have taken some of the lessons of user-centered design to heart - assuming, that is, that the technical people are the users.

But what of the other groups of people involved in a simulation project? When a simulation model is used or viewed by someone who is not the developer, the user is modelling - comparing their conceptual model of the system with the perception of what the simulation model is doing. How can this be supported?

The most common approach has been to improve the quality of graphics and visualisation. Welcome though this is, it does carry some problems. Last year the author visited a state-of-the-art engineering company who were planning the development of a highly expensive flexible manufacturing line. As part of this, they developed a discrete simulation which they were keen to show off. The graphics and visualisation were stunning with 3 dimensions, correct proportions, panning and zooming ..etc.. But, the simulation itself was, in analytical terms, probably a waste of time. Simple calculations showed where the bottlenecks would occur and, given the discrete nature of the system components, it was pretty obvious what design would be needed. The author has a strong suspicion that this simulation may have hindered a sensible decision making process rather than supporting it. Though graphics and visualisation are important, it must be remembered that their role should be a support to people in their thinking. Sometimes a simple representation may be a better support to thinking than a complex one.

As with user-centered design for the analyst, so too for the other groups, the basic questions concern the tasks to which the different users wish to put the system. It may be that this implies that well designed systems should offer support to these users as they experiment with simulation models. If Bruner and Kay are correct, then these users will wish to enact scenarios in an attempt to learn about the model and to make inferences about the system being simulated. If appearances are deceptive, perhaps superb visualisation is not the be-and-end-ali for non-technical users?

REFERENCES

Bruner E. (1962) *On knowing: essays for the left hand.* Harvard Univ Press, Cambridge, Mass.

Clementson A.T. (1991) *The ESCL Plus system manual.* AT Clementson, The Chestnuts, Princes Road, Windermere, Cumbria, UK.

Coupland D. (1995) *Microserfs.* Flamingo, London.

Dahl O. and Nygaard K. (1996) SIMULA - an Algol-based simulation language. *Comm ACM,* 9, 9, 671-678.

Hurrion R. (1976) The design, use and requirements of an interactive visual computer simulation language to explore production planning problems. PhD Thesis, University of London.

Kay A. (1990) Interview. In Laurel B. and Mountford S.J. (1990) *The art of HCI design.* Addison-Wesley, Reading, Mass.

Lanner Systems (1996) *WITNESS User manual.* Lanner Systems, Redditch, Worcs, UK.

Lansdale M.W. and Ormerod T.C. (1994) *Understanding interfaces: a handbook of human computer dialogue.* Academic Press, New York.

Law A.M. & Kelton W.D (1991) *Simulation modelling & analysis,* second edition. McGraw-Hill, New York.

Micro Analysis & Design (1992) *Getting started with Micro Saint for Windows.* Micro Analysis & Design Simulation Software Inc., Boulder CA.

Norman D.A. (1988) *The psychology of everyday things.* Basic Books, New York.

Norman D.A. (1990) Interview. In Laurel B. and Mountford S.J. (1990) *The art of HCI design.* Addison-Wesley, Reading, Mass.

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

Pidd M. (1996a) *Tools for thinking: modelling in management science.* John Wiley & Sons Ltd, Chichester.

Pidd M. (1996b) Five simple principles of modelling. In *Proceedings of the 1996 Winter Simulation Conference,* San Diego CA.

Stoll C. (1995) *Silicon snake oil: second thoughts on the information superhighway.* Doubleday, New York.

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

AUTHOR BIOGRAPHY

MIKE PIDD is Professor of Management Science in the Management School of Lancaster University in the UK. He has written two books on simulation, of which the best known is Computer Simulation in Management Science (John Wiley). He has just completed a new book for the same publisher (Tools for thinking: modelling in management science) which addresses the issues often raised by critics of rational modelling. His current interests in computer simulation include object oriented methods.