Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

THE KEY TO OBJECT-ORIENTED SIMULATION: SEPARATING THE USER AND THE DEVELOPER

Pete Ball

DMEM
University of Strathclyde
Glasgow, G1 1XJ, UNITED KINGDOM

Doug Love

Mech & Elect Eng
Aston University
Birmingham, B4 7ET, UNITED KINGDOM

ABSTRACT

Simulation tools should be both easy to use and applicable to a wide range of problems. In practice, however, a compromise exists giving rise to a range of software from simulators to simulation languages. Object-oriented techniques have the potential to overcome this compromise; the ability to reuse and extend software could enable the development of a simulator that would be extended over time. The use of object-oriented techniques to date has mainly resulted in the development of powerful but difficult to use libraries. Ideally the skills of manufacturing engineers should be concentrated on building models of manufacturing systems whilst the skills of software developers should be concentrated on adding new functionality. This paper presents a mechanism whereby the roles of engineer and developer are clearly split to provide an easy to use simulator with a potentially very wide range of application.

1 INTRODUCTION

A range of simulation tools exists for use in modelling systems. At one extreme are simulation languages which are flexible but relatively difficult to use and at the other extreme are data driven simulators which are quick and easy to use but lack the range of potential application (Shewchuk & Chang 1991). Between these two extremes lie a large number of packages which are essentially a compromise between ease of use and range of application. This paper examines how this compromise should be overcome.

Ideally simulation tools should be quick and easy to use yet offer sufficient modelling power to allow wide application. Ease of use is a significant issue as it can dictate whether typical manufacturing engineers could use the simulation software and whether the software could be applied within the time scales permitted. Some software systems attempt to combine the two requirements by combining the concepts of data driven simulator with some sort of programming interface. Whilst this overcomes the limitations of simulators it results in a tool that is more difficult and time consuming to use.

One approach is to use domain specific simulators (see Pidd 1992), e.g. for modelling low technology batch manufacturing systems, and modify the software when new needs arise. This approach is difficult to adopt in practice due to the need to have access to the source code and have the skill to understand and modify it. Such modifications are likely to give rise to an increase in complexity and a reduction in the overall robustness. With each modification the task of modifying and maintaining the software becomes progressively more difficult.

The use of object-oriented (OO) techniques has the potential for developing software that is relatively easy to maintain, can be re-used and contains close abstractions of real world concepts (Graham 1994). The use of OO techniques could assist in overcoming the ease of use vs. range of application compromise. Whilst there has been significant activity in the application of OO techniques to simulation software currently there are few, if any, systems that demonstrate benefits other than to skilled users.

This paper examines the roles of different types of user in the creation and use of simulation software. The use of object-oriented techniques will be examined and their use in the creation of simulation software will be discussed. The work that was carried out is concentrated in the area of manufacturing systems. The issues discussed in this paper relate to the creation and use of the software rather than its application in manufacturing and therefore the principles will be relevant to other application areas.

2 OBJECT-ORIENTED SIMULATION

2.1 Object-Oriented Libraries

Since the development of the Simula simulation language there has been increasing interest in object-oriented (OO) concepts. OO concepts applicable to simulation software development are:

- design and analysis techniques for the abstraction of real world concepts in software classes;
- design and analysis techniques for the development of the overall software design;
- programming techniques for the implementation of the designs.

These points are illustrated in Figure 1.

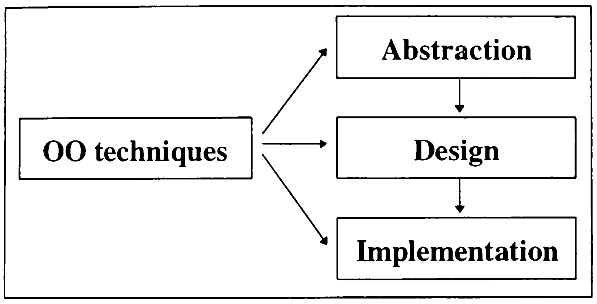


Figure 1: Role of OO Techniques

As Nof (1994) observes, there are underlying relationships between manufacturing and object-orientation which make OO particularly appropriate.

Using OO techniques for developing software results in the creation of a library of classes. For example a manufacturing system class library would include operators, machines, parts, routings, trucks and storage areas. Examples of libraries include BLOCS/M (Glassey and Adiga 1990) and a highly reusable library (Bhuskute et al. 1992). The use of the library concept allows great flexibility for modelling. Different classes can be used for building models as and when required. As new modelling needs arise new classes can be developed.

The use of such libraries alone is inefficient. Software needs to be developed to support the libraries. Creation of this software is time consuming and requires skill; users need to be able to develop complex software, use language syntax, understand detailed abstract simulation logic as well as understand and use OO techniques. This latter point is significant as it is generally accepted that there is a significant learning curve associated with the use of OO techniques.

2.2 Object-Oriented Simulators

The application of OO techniques to data driven simulators would seem to offer significant benefits. End-users would be able to build simulation models easily taking advantage of the close mapping between objects in the simulated and real world. If the need to add new functionality to the system arose then the OO characteristics should lead to a reduction in the development time and risks involved (Figure 2).

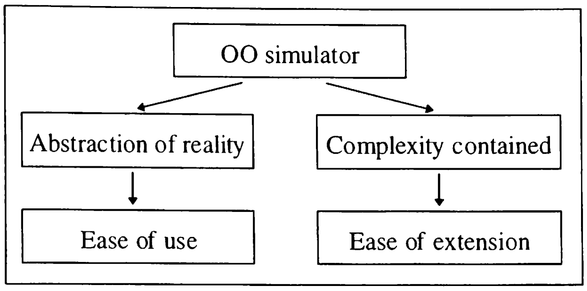


Figure 2: Benefits of an OO Simulator

A number of systems have been developed that have a data driven interface and have utilised object-oriented programming techniques or principles in their construction. Examples include the object-based Arena (Collins & Watson 1993), ProModel (Harrell & Tumay 1991) and Simple++ programmed using OO and the object-oriented simulator SIMMEK (Borgen & Strandhagen 1990). Arena and Simple++ have demonstrated a capability to expand to adapt to new requirements. Arena uses a series of templates to develop models. The range of templates can be extended by an experienced user. The limitation of this approach is that the extension mechanism is functionally-based, not object-based, and therefore does not take full advantage of encapsulation or inheritance. Similarly in Simple++ inheritance would appear to be functionally based and thereby fail to support polymorphism and containment of complexity.

An important issue in the use of object-oriented techniques is whether the user actually benefits. If there is no significant change in the way the user sees or interacts with the system or in the potential benefits offered then the choice of programming style is largely irrelevant. One of the benefits that could be claimed from the use of OO techniques is the generation of a user interface that has a close correspondence with reality and is therefore easy to use. Whilst this may be so, ATOMS (Love & Bridge 1988) was developed using traditional software construction techniques and yet its user-interface has a close correspondence with manufacturing systems which it was designed to model.

Object-oriented techniques have significant benefits to offer but these must be utilised effectively. Unless the underlying software architecture provides some potential advantage to the user then the use of the programming technique has no relevance. For example, unless the power of OO techniques is to be utilised to provide new functionality the choice of programming technique is simply for developer convenience rather than any long term advantage.

3 TWO TYPES OF USER

3.1 Combining OO and Simulators

Object-oriented techniques have the potential for replicating real world concepts and terminology within software. The techniques can be used to develop software that is robust, reusable and extendible. As new needs arise new functionality can be developed. Applied correctly, the use of OO techniques has the potential to provide significant benefits to the user.

The concept of data driven user-interfaces enables models to be built quickly and easily and allows the users to concentrate on the modelling task at hand rather than the abstract software development. A tool that has both a data driven user-interface and an underlying (well designed) software architecture will be very powerful. The tool could be readily expanded and would be quick and easy to use.

3.2 Separation of Users

Whilst the concepts of OO and data driven could be combined into a single tool it is not necessary for the tool to be applied by a single type of user. The developer of OO simulation functionality does not necessarily have to be the same person who applies the simulation tool. Indeed it could even be desirable to enforce this separation. We might call these two types of user the 'developer' and the 'engineer', (Figure 3).

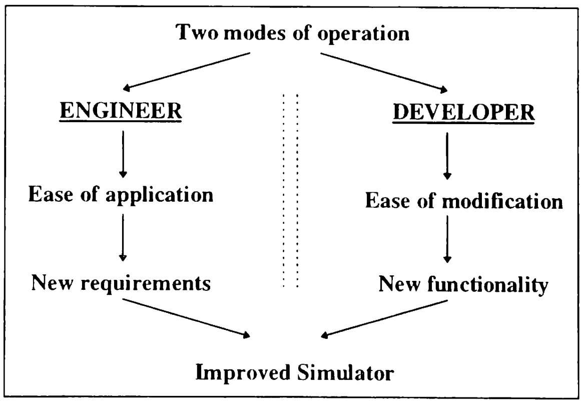


Figure 3: Two Modes of Simulator Use

Individuals able to develop object-oriented simulation software are able to develop simulation classes relatively easily. The developers need to be familiar with software design, language syntax, simulation techniques and the construction of the particular class libraries used by the system. In addition they need to have an appreciation of the target applications area, otherwise the allocation of functionality across the class structure is likely to be incorrect. For example, failure to include operators in a library or only to develop operators as resources to be acquired and released presents problems when the need arises to model operators as proactive entities. Familiarity with these areas takes a considerable amount of time to acquire.

Manufacturing engineers wishing to use simulation software will understand, in general terms, the behaviour of the manufacturing system they wish to examine. Whilst they will not require any knowledge of the internal structure of the simulator, they will need to understand the properties of the objects, the way a model is constructed and how the simulation experiments should be set up and analysed. The engineers will understand the problems with the system and will have ideas about possible solutions. Engineers will benefit from using simulation both from the process of building the model as well as obtaining results from it.

With the type of simulation tool proposed, confining individuals into two distinct roles of developer and engineer has a number of advantages:

- the engineer does not have to understand the underlying software architecture, software engineering techniques and detailed simulation techniques;
- the engineer does not develop poorly written software that has limited application and affects the robustness of the system;
- the engineer is able to spend more time on generating and interpreting results and less on model building;
- the engineer applies engineering skills to engineering problems;
- the developer applies programming skills to software development;
- the developer has the time and skill to create robust, variant functionality quickly to meet the engineers' needs;
- the developer can consider the needs of a wider audience rather than focusing only on the needs of a specific project.

3.3 Combination is Powerful

The two types of user would complement one another in their activities. Each type of user expends their time and skills on appropriate tasks. The software developers can develop new functionality (classes) as new user requirements evolve, see Figure 4. The new functionality can be distributed to the various users. The engineers in turn would apply the new functionality to create models more easily and more representative of the system being modelled.

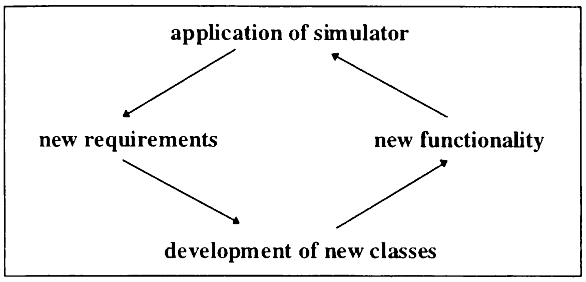


Figure 4: 'Reuse' of an OO Simulator

This approach combines the ease of use and flexibility into a single tool but *does not place an additional burden on the users*. The engineer is able to apply the tool easily and as new needs arise it is the responsibility of the developers to meet them. This prevents the engineer's task becoming more difficult and means that when the new functionality is developed it can be used by many rather than just one.

In this context the term 'engineer' applies to anyone able to build models using data driven simulation software. The term 'developer' describes individuals able to develop object-oriented simulation software and is therefore not restricted to the original developer.

4 THE ARCHITECTURE

This section develops the idea of combining ease of use and flexibility into a single tool but maintaining separate users. The approach to the simulator design will be described along with the way in which it is used.

4.1 Ease of Extension

For any software architecture to deliver flexibility it must be easy to extend. As new needs arise developers extend the software by adding new functionality. Ease of extension covers the issues of how much knowledge of the whole software is required to add new functionality and how much of the existing software must be modified.

In the ideal situation it would be possible to add new classes (functionality) to the simulator without any knowledge of the existing software construction. Whilst this is not possible the amount of knowledge required can be kept to a minimum. For example to add a new machine class knowledge would be required of existing machine classes, batches and operators. Knowledge of how the object would be displayed on the screen, scheduled on an event list or results collected is not necessary. The design should therefore ensure many of the latter mechanisms are provided automatically. One way of doing this is to place the mechanisms in abstract classes which the new class inherits, see Figure 5.

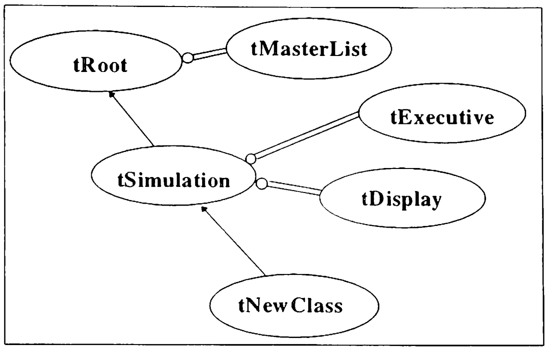


Figure 5: Inheriting Simulation Mechanisms

The second issue of software extension is the degree of modification of existing software. If a large amount of existing code must be modified then this complicates the process and, more significantly, the software. As more classes are added more modifications are required. An example would be the addition of a new type of machine requiring the modification of all those classes likely to interact with it, including operators, repair people and batches.

As each modification is made the software becomes more complex and more difficult to extend. Ideally a new class should be added by a simple process of registration of the new class with the system. Thus classes could be added without increasing the complexity of the system, see Figure 6.

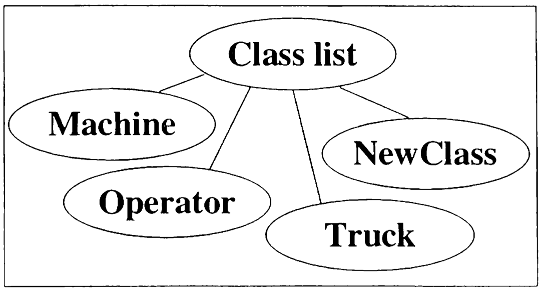


Figure 6: Registering Classes

Simply adding a new class to a hierarchy is not sufficient to enable interaction with other classes. Some means must be sought to enable interaction but to prevent increase in complexity. For example the operators must be made aware of a new machine type without requiring modification of the operator code. One possible mechanism for this is to use message passing. Here the term "message passing" would not be a substitute for the term "method" but would be a separate, distinct mechanism. Each class is assigned one method for receiving messages, see Figure 7.

```
tNewClass = OBJECT(tExistingClass)
   PUBLIC  { accessible by other classes }

      CONSTRUCTOR Init(.....)
      DESTRUCTOR  Done; VIRTUAL;
      PROCEDURE   ReceiveMsg(pMsg:tMsgPtr);

   PRIVATE  { only for use by this class }

      PROCEDURE   Load(...);
      PROCEDURE   Run(...);
      PROCEDURE   Stop(...);
      PROCEDURE   etc.

   END; { Definition of class : tNewClass }
```

Figure 7: Access Only Via ReceiveMsg Method

This is in contrast to approaches in which the list of messages a class can receive actually refers to a list of public methods.

Using this approach, one object would send a message (actually an object itself) to another object. Since all classes have the same receive message method, any object can send a message to any other object even if the target object is a newly added class. Hence when a new machine class is added, operator objects can be assigned to the new machine at run time and send messages to it without any prior knowledge of it being a new or old machine, see Figure 8.

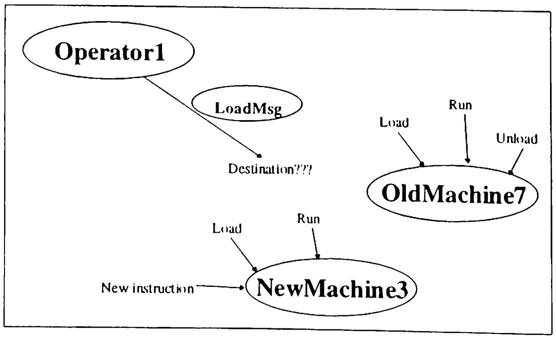


Figure 8: Messaging New or Old Objects Without Modification

Mechanisms that provide general features necessary for all simulation models (such as results collection, graphical displays, event list management and load and save management) exist but are distinctly separate from the modelling functionality. This approach allows any type of functionality to be added and still benefit from the use of the general features.

All additions to the simulation software would be made by a software developer.

4.2 Ease of Use

A data driven user-interface can be employed to provide ease of use. The interface should possess features and terminology that are familiar to the user. This type of user-interface would not be too dissimilar from other simulator interfaces such as ATOMS (Love & Bridge 1988) or ProModel (Harrell & Tumay 1991).

Whilst menus and dialogs are available for entering data and selecting logic there is no programming interface. The user is unable to conceive, implement and refine logic and thereby the speed and ease of use is preserved. If new requirements arise the developers can work to meet them.

Elements of the user-interface would fall into two categories: general and functionality specific. The general features (such as results display) would be created initially and would require little, if any, modifications over time. As new functionality is added menus and dialogs would be created to view and edit the data, e.g. a dialog to edit the data of a new machine.

5 PROTOTYPE SOFTWARE

Demonstration of the principles introduced can be found in the Advanced Factory Simulator (AFS) (Ball & Love 1994). AFS has been developed as an object-oriented, data driven simulator. The use of OO principles has been applied throughout the construction of the software giving data driven capabilities combined with ease of extension.

5.1 OO Simulation Architecture

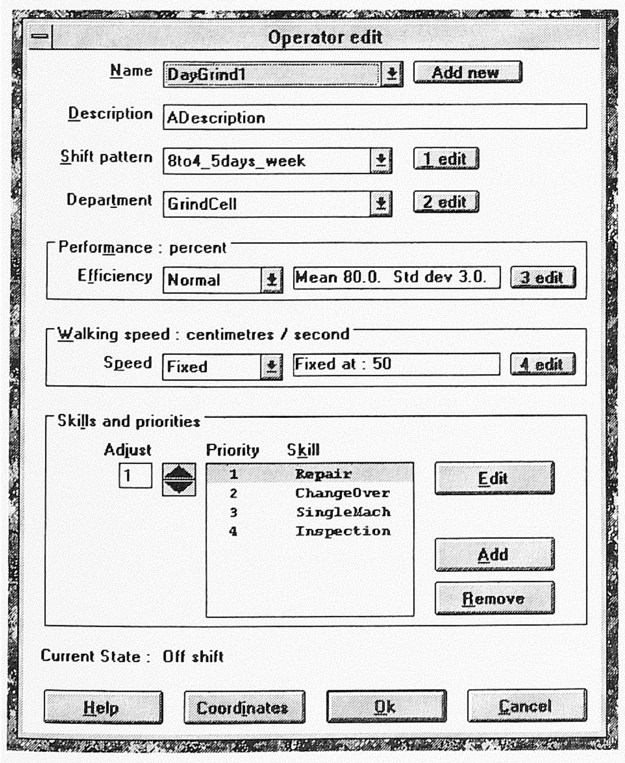
The simulation architecture refers to the significant element of software that is required to support the application classes, such as the machines and operators. The architecture includes the load and save management, the results collation and display, the simulation executive, the graphical displays, the reset mechanisms, support for editing the interaction of objects, etc.

The simulation architecture was a key consideration and was designed and developed in tandem with the application classes. By adopting this approach the architecture design was such to ease the task of adding new functionality. One of the prime design criteria was that the addition of new application classes should not increase the complexity of the whole software. The complexity refers to the interaction between the application classes and between an application class and the supporting architecture.

The process of adding new classes is extremely simple. The class and a class manager are created, invariably inherited from existing classes. The class manager's role is to provide a link between the user-interface and the objects of the particular class. The class manager is then added to a registration unit that simply involves placing an object of the class manager on a list. This process is all that is required to add a new class. The difficult part, the part that requires the skill of a software developer, is to develop the newly added class to abstractly model the behaviour of some aspect of the real world. This is the time consuming element that is undesirable for the engineer to become involved in.

5.2 Data Driven User-Interface

The required properties of the user-interface have been described already; data driven in style and employing terminology and concepts from the problem domain. The Advanced Factory Simulator (AFS) uses an interface typical of Microsoft Windows. A typical dialog is shown below in Figure 9.



Name: DayGrind1
Description: ADescription
Shift pattern: 8to4_5days_week
Department: GrindCell

Performance : percent
Efficiency: Normal — Mean 80.0 Std dev 3.0

Walking speed : centimetres / second
Speed: Fixed — Fixed at : 50

Skills and priorities
Adjust: 1

Priority	Skill
1	Repair
2	ChangeOver
3	SingleMach
4	Inspection

Current State : Off shift

Figure 9: Typical AFS Dialog

The data requirements are typical of the information available within a factory, for example shift patterns, routing data, reliability data and order data. There is no access either to the underlying software code or a high level simulation programming language.

5.3 Advantages of Approach

The advantages of this approach have already been described; engineers can use the software without the need to program whilst developers can provide a greater range of functionality. In addition to this key combination there are a number of other benefits.

In addition to creating variant shop floor classes, such as new types of machine tools, more radical additions can be made. Since the architecture will support any type of functionality then the functionality does not have to be directly related to the shop-floor. A number of significant additions have been made. One group of additions is a set of production planning and control classes (Boughton & Love 1994). These classes have been added to the core system with minimal increase in complexity and can be used to simulate different production planning and control system configurations. Modelling can either be stand alone or include the shop floor entities.

The user-interface can be readily modified to include new types of dialogs. The process simply involves designing and developing the dialog followed by its registration within the user-interface. This allows variant dialogs to be created. Dialogs can be significantly different to those in existence. For example, a "wizard" has been added for the development of "U"-shaped cells (Ball & Love 1995). This wizard uses the existing functionality to guide the engineer through the process of designing and modelling such cells. The dialog prompts the user at each stage for additional data and provides feedback on the likely performance of the cell prior to simulation.

6 SUMMARY AND CONCLUSIONS

This paper identified the benefits and drawbacks and discussed the development of object-oriented software. It was asserted that current approaches do not make full use of the potential of object-oriented techniques. The roles of "engineer" and "developer" were examined. It was argued that whilst it is beneficial to combine data driven and object-oriented techniques it is undesirable to combine the engineer and developer roles. A mechanism was described that allows the engineer and developer roles to be kept separate whilst at the same time combining the benefits of the data driven and programming tools.

Using this approach it is possible to build up a library of functionality. Because the software is created by developers, not users, the functionality is applicable to a wider range of situations and therefore can be distributed to other users. The ability to support an ever increasing range of functionality removes the limitations typically associated with simulators.

ACKNOWLEDGEMENTS

The authors would like to thank Lucas Engineering & Systems Ltd. for its support.

REFERENCES

Ball, P.D. & Love, D.M. 1994. Expanding the capabilities of manufacturing simulators through the application of object-oriented principles. *Journal of Manufacturing Systems*, 13 (6): 412-423.

Ball, P.D. & Love, D.M. 1995. Development of simulation techniques for the design of nagare flow production systems. In *Proceedings of the International MATADOR conference*, 111-116. UMIST, Manchester, UK.

Bhuskute, H.C., Duse, M.N., Gharpure, J.T., Pratt, D.B., Kamath, M. & Mize, J.H. 1992. Design and implementation of a highly reusable modelling and simulation framework for discrete part manufacturing systems. In *Proceedings of the Winter Simulation Conference*, eds. J.J.Swain, D.Goldsman, R.C.Crain, & J.R.Wilson, 680-688. Institute of Electrical and Electronic Engineers, Arlington, Virginia, USA.

Borgen, E. & Strandhagen, J.O. 1990. An object oriented tool based on discrete event simulation for analysis and design of manufacturing systems. In *Optimization of Manufacturing Systems Design: International Conference Proceedings*: North-Holland Publishing Company: 195-220.

Boughton, N.J. & Love, D.M. 1994. An object-oriented approach to modelling manufacturing planning and control systems. In *Proceedings of the National Conference on Manufacturing Research*, 426-430. Loughborough University, Loughborough, UK.

Glassey, C.R. & Adiga, S. 1990. Berkeley Library of Objects for Control and Simulation of Manufacturing (BLOCS/M). In *Applications of Object-Oriented Programming*, Eds. L.J.Pinson & R.S.Wiener, New York, USA: Addison-Wesley Publishing Company.

Graham, I. 1994. *Object-Oriented Methods*. Wokingham, UK: Addison-Wesley.

Harrell, C.R. & Tumay, K. 1991. ProModel tutorial. In *Proceedings of the Winter Simulation Conference*, eds: B.L.Nelson, W.D.Kelton, & G.M.Clark, 101-105. Institute of Electrical and Electronic Engineers, Pheonix, Arizona, USA.

Love, D.M. & Bridge, K. 1988. Specification of a computer simulator to support the manufacturing system design process. In *3rd International Conference on Computer-Aided Production Engineering*, 317-323. Ann Arbor, Michigan, USA.

Nof, S.Y. 1994. Critiquing the potential of object orientation in manufacturing, *International Journal of Computer Integrated Manufacturing*, 7 (1): 3-16.

Pidd, M. 1992. Guidelines for the design of data driven generic simulators for specific domains. *Simulation*, 59 (4): 237-243.

Shewchuk, J.P. & Chang, T.C. 1991. An approach to object-oriented discrete-event simulation of manufacturing systems. In *Proceedings of the Winter Simulation Conference*, eds: B.L.Nelson, W.D.Kelton, & G.M.Clark, 302-311. Institute of Electrical and Electronic Engineers, Phoenix, Arizona, USA.

AUTHOR BIOGRAPHIES

PETER D. BALL is a lecturer in the department of Design, Manufacture and Engineering Management (DMEM) at the University of Strathclyde, Scotland. He holds a BEng in mechanical engineering and a PhD in manufacturing simulation from Aston University. His research interests cover manufacturing system design and the development and application of simulation tools. He is an associate member of IEE (Manufacturing Division) (UK).

DOUG M. LOVE is a lecturer in the department of Mechanical and Electrical Engineering at Aston University, England. He holds a BSc in mechanical engineering from Manchester University and a PhD in the dynamic behaviour of manufacturing systems from Aston University. He has worked in production control, as a product manufacturing manager and as an industrial engineer designing group technology cells. His research interests concentrate on the design and operation of cellular manufacturing systems; he has published papers on Distributed MRP (DMRP) systems, simulation in manufacturing systems design, and the development of computerised component coding systems. He is a member of IEE (Manufacturing Division) (UK).