Proceedings of the 1997 Winter Simulation Conference
ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson

THE FOUR PHASE METHOD FOR MODELLING COMPLEX SYSTEMS

Hamad I. Odhabi
Ray J. Paul
Robert D. Macredie

Centre for Applied Simulation Modelling (CASM)
Department of Information Systems and Computing
Brunel University
Uxbridge, Middlesex UB8 3PH, UNITED KINGDOM

ABSTRACT

This paper investigates an attempt to combine different simulation tools in order to build a simulation environment that can be used to model complex system behaviour. The component tools used in the project are: the four phase method, a simulation world view derived from the three phase approach especially for Object-Oriented and iconic representation; iconic representation that represents the actual system components and logic through using icons and arcs; Object-Oriented Programming and the MODSIM simulation library. Related aspects of design and implementation are presented and critiqued through an investigation into the development of a model for an existing problem. Different modelling approaches are used to model the same problem and, as such, the advantages of the four phase method are highlighted.

1 INTRODUCTION

Discrete-event simulation modelling offers people the chance to develop an understanding of their problem domain by building a simulation of the problem space in which they are interested. Three broad perspectives on how to approach the development of simulation models are easily identifiable from the literature:

The first perspective focuses on using a graphical user interface (GUI) that allows the user to build the model on the screen, connect the components by arcs to represent the model logic, and run the simulation (Drury and Laughery 1994). In most cases, due to the limitation of the simulation program under use, some simplifications and approximations to the model may be required. Such simplifications or approximations can be very costly.

The second perspective is underpinned by the belief that no simulation program is able to model all types of systems behaviour without making some simplifications or modifications (Joines 1994). This suggests that models should be developed from scratch by using a simulation modelling language. This approach influences the modelling development life cycle by increasing the production time and may divert the modeller into concentrating on the programming challenge rather than on developing an understanding of the system under study.

The third perspective, on which this paper will focus, concentrates on using a GUI that is able to automatically generate code, with the modeller making changes to the generated code to match the system needs (Hlupic and Paul 1994). This approach is popular because it can reduce the model production time and provides the user with access to low level programming structures where needed. However, modifying existing code is not easy, and many issues may have an impact on the code's comprehensibility. In this research we are concerned with two of these basic issues.

The first issue concerns the modelling approach. Several programming approaches, often known as 'simulation world views', have been designed for discrete-event simulation modelling. The aim of any approach used should be to aid the production of a valid, working simulation at minimum cost and/or in the shortest time (Pidd 1992a).

The second issue that may affect the cost of modifying the generated code is the programming methodology. Specific methodologies, reflecting particular programming paradigms, may support simplified model code comprehension, and therefore maintenance. Object-Oriented Programming (OOP), for example, has become popular in simulation modelling (Kienbaum and

Paul 1994b), with a claim for relative ease of maintenance being made for the approach.

This research directly addresses both issues of modelling approach and programming methodology. We will introduce a new simulation world view termed the Four Phase Method (FPM), and discuss its importance in the context of iconic representations and the automatic generation of model code. We will argue that the approach combines the simplicity of the three phase method with the power of the process-based approach when modelling complex system behaviour. The programming methodology that has been used in this research is OOP because it increases productivity and speeds up the software development life cycle. Another important factor which makes OOP attractive is that it is a natural paradigm for simulation modelling: the problem space will be composed of objects and this can naturally be captures and reflected in the model development.

The aim of this research is to attempt to combine a new simulation world view (the four phase method), OOP, and iconic representation to construct a simulation environment for the development of discrete-event simulation models. The modelling environment should be able to model complex system behaviour, provide the user with a simple iconic representation to 'drive' the model design, and generate understandable code.

In the following section we will define terms and notations which are used in the paper. This will lead us on to a broad description of the four phase method (section 3), which will provide the background to our presentation of the main characteristics of the modelling environment, along with a description of its design and implementation (section 4). Section 5 will discuss the use of the four phase method in building a simulation model for a manufacturing system. The section will also reflect on the insights gained from the practical use of the environment, particularly focusing on the advantages of the FPM over other, established modelling approaches when applied to the same modelling problem. Section 6 will present conclusions drawn from the research.

2 DEFINITIONS

Before we explain our research, we will define terms and notations that will appear frequently in the rest of this paper:

- A process is a set of actions that a node has to perform. For example, the process of the *source* node consists of the following actions: generating token(s), initialising token(s), and sending tokens to the next node.
- A node represents a process or a hierarchically arranged group of processes.
- *A Delay node* represents a process that requires a period of time (simulation time) to be completed.
- *An UnDelay* node represents a process that happens in a simulation time = 0.
- A hierarchical node contains other nodes that may themselves be hierarchical nodes.
- A token represents an entity moving through the system. For example, a customer in a bank.
- Resources represent domain elements that constrain the performance of the system, such as manpower, a communications link, or machines.

3 THE FOUR PHASE METHOD

Discrete computer simulation programs can become complicated, making it important to use a well-structured approach when writing or modifying them. There are at least four widely used programming/modelling approaches (world views) for discrete-event simulation modelling (Pidd 1992a):

- The three phase approach.
- The activity approach.
- The process interaction approach.
- The event approach.

A common factor is that all of these modelling approaches work as structuring devices to specify a simulation model's steps to the computer. It is important that a modelling approach should enhance the conceptualisation of the simulation problem and the understanding of the system behaviour. It should also reduce the time taken to produce working, valid simulations and ease the task of the programmer.

As we shall explain in section 5.2, the development of iconic representations such as hierarchical activity cycle diagrams HACD (Kienbaum and Paul 1994a) and extended activity cycle diagrams X-ACD (Pooley and Hughes 1991), have demonstrated that the three phase and the activity approaches do not have sufficient support for all types of icons/nodes (especially *UnDelay Nodes*). By contrast, the process-interaction approach has a complex executive, which makes the enhancement of an existing program a more elaborate task. The event approach usually requires the user to become directly involved in computer programming.

The modelling approach used throughout this research is termed the Four Phase Method (FPM) as it was first described in Odhabi and Paul (1995). This new approach has many advantages in meeting the needs of the analysis and model building phases when using an Object-Oriented approach. It has been designed

especially to be used in conjunction with OOP and iconic representation, although it does not require a particular iconic representation. A program written using FPM represents the model life cycle in four separated steps/phases. Each phase describes a set of actions to be undertaken. All of these phases occur in the same simulation time. FPM is underpinned by two general considerations:

- each icon (node) in the iconic representation has an internal queue. When the node receives an entity, it holds it in the internal queue.
- a distinction is drawn between two classes of nodes. The common characteristic of all nodes in the first class, called *Delay Nodes*, is that they can delay entities for some time. By contrast, the common characteristic of the second class is that they do not delay entities when they are required by another node in the model. This class is called *UnDelay Nodes*.

Fig. 1 shows the general structure of a simulation model based on the FPM. The model starts by initialising the system. In the initialisation, the model creates the nodes, defines the system variables, etc.

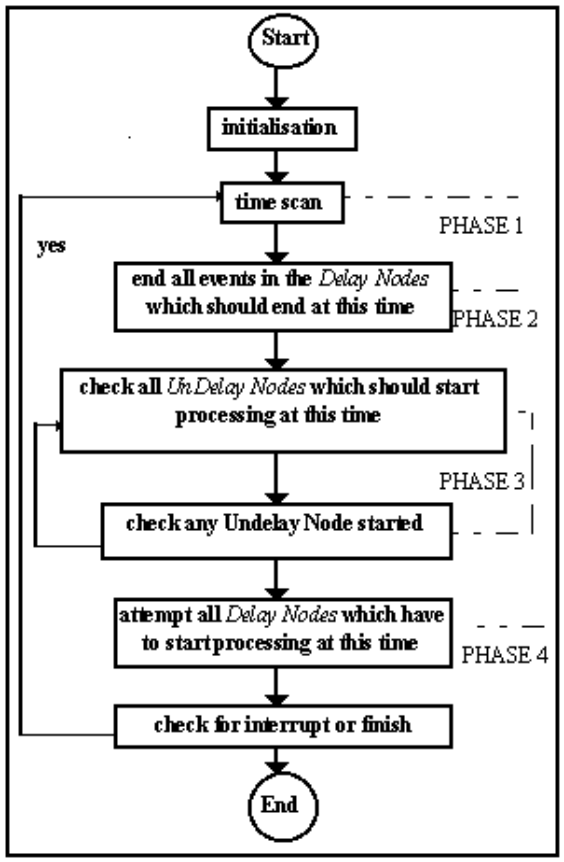

Figure 1: The Four Phase Method

After initialisation, the phases of the FPM must be executed sequentially in a cycle. The starting point in this cycle is Phase1. Phase 1 searches the activity list to find the earliest activity finishing time, and advances the simulation clock to this time. For this purpose, Phase 1 keeps information regarding current activities in the system. After Phase 1 completes its trial, Phase 2 stops the activities that have been scheduled to be completed by this time, and moves the relevant entities into the internal queue. However, Phase 2 only works on *Delay Nodes*. Phase 3 which works only on *UnDelay Nodes*, checks all the *UnDelay Nodes* to find those which are able to start processes at this time and performs the relevant processes (with duration time zero). Phase 3 must be repeated until there is no Undelay Node with processes to start. Phase 4 starts processing the relevant *Delay Nodes*, calculates the finishing time, and records this time. When all relevant *Delay Nodes* have been processed, Phase 4 checks for an interrupt or a specified finishing condition.

4 DESIGN AND IMPLEMENTATION

Fig. 2 illustrates the general conceptual design of the FPM and object-oriented simulation in the context of our research. The modelling tools presented in Fig. 2 are the minimum requirements for demonstrating the use of the FPM. Other tools may be added or existing ones modified depending on the needs of the modeller.

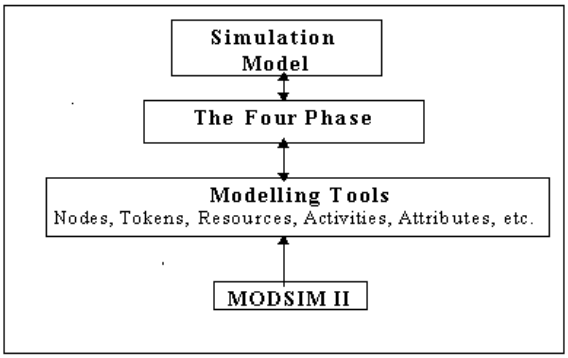

Figure 2: The General Conceptual Design of the Four Phase Approach in Object-Oriented Simulation Language (MODSIM II)

The lowest 'level' in this design is MODSIM II (1992), which includes the language structure and the library elements. A particular simulation model is at the highest level. The concepts at each level are 'encapsulated' so that a simulation model user, for instance, need not be concerned about the concepts at a lower level (Joines 1994). The four phase objects are placed next to the highest level to work as an interface

between the simulation model and the lowest levels. When the model invokes any phase to start, for example Phase 1, the phase deals immediately with the relevant nodes and library in order to perform the process.

This section will investigate the conceptual design presented in figure 2 in more detail, discussing the four phase objects and the modelling tools.

4.1 The FPM Objects

The implementation of the four phases as objects in their own right is illustrated in Fig. 3.

```
FourPhaseObj = OBJECT;
  EndPhase  : BOOLEAN;
  ActiveManager : ActivityListManagerObj;
  StopManager   : StopActivityListManagerObj;
  ASK METHOD InitActiveManager;
  ASK METHOD InitStopManager;
  ASK METHOD SetEndPhase( IN TheEnd : BOOLEAN);
END OBJECT;

Phase1Obj = OBJECT(FourPhaseObj);
  CurrentTime : REAL;
  ASK METHOD InitCurrentTime;
  ASK METHOD SendCurrentTime : REAL;
  ASK METHOD TransfareActive;
  ASK METHOD StartPhase1;
END OBJECT;

Phase2Obj = OBJECT(FourPhaseObj);
  ASK METHOD StartPhase2;
  ASK METHOD CurrentActivity : ActivityObj;
END OBJECT;

Phase3Obj = OBJECT(FourPhaseObj);
  UndelayNode : QueueObj;
  ASK METHOD Init;
  ASK METHOD AddToUndelayList(IN Node : IconObj);
  ASK METHOD NodeStarted(IN Bo : BOOLEAN);
  ASK METHOD StartPhase3;
END OBJECT;

Phase4Obj=OBJECT(FourPhaseObj);
  DelayNode : QueueObj;
  ASK METHOD Init;
  ASK METHOD AddToDelayList(IN Node : IconObj);
  ASK METHOD StartPhase4;
END OBJECT;
```

Figure 3: The Definition of The Four Phase Objects

An abstract object, FourPhaseObj, has been defined. This object is the ancestor (parent) of the actual four phases' objects. FourPhaseObj has three fields. The first is EndPhase which is a boolean used to announce the end of the current phase. If EndPhase is FALSE, the phase must repeat the process. The second field is ActiveManager which is a queue based object. ActiveManager is used to store a group of activities. It is important to note that any Activity object 'knows' the nodes and entities that it belongs to. The third field is StopManager which is another queue based object. Unlike ActiveManager, StopManager stores only the activities that must be stopped by the current simulation time.

In addition to these three fields, there are three 'methods'. The first and second methods are used to initialise ActiveManager and StopManager. The last method is used to set EndPhase to either TRUE or FALSE, according to the system situation.

Phase1Obj, Phase2Obj, Phase3Obj, and Phase4Obj are instances of FourPhaseObj; as such, they inherit all its fields and methods.

Phase1Obj defines an extra field (CurrentTime) and four methods. These new methods are used to initialise the simulation time, to send information concerning the simulation time to the model objects, to transfer activities from ActiveManager to StopManager, and to start the phase.

Phase2Obj starts the process by asking the StopManager to remove the first activity from the queue, asking the activity owner (Delay Node) to finish the process. Phase 2 repeats this process until there are no activities left in StopManager. Phase2Obj provides information to the system concerning the current activity being processed.

Phase3Obj keeps information concerning any UnDelay node that is holding entities in its internal queue in the field UndelayNode. Phase3Object starts the process by setting EndPhase to TRUE and asking UndelayNode to ask every node in its queue to attempt to start its process. The node checks which processes can start; starts appropriates processes and asks Phase3Obj to set the EndPhase to FALSE. Phase3Obj repeats this process until there is no Undelay node holding entities able to start processes at this time. In summary, Phase3Obj has defined four new methods: the first is the initialisation of the phase that includes creating the UndelayNode; the second allows Phase3Obj to add any Undelay node to the list; the third informs Phase3Obj about the situation after asking the node to start processing; while the last method starts the phase.

Phase4Obj is concerned with *Delay Nodes*, so it keeps information about *Delay Nodes* that are available in the model, using a field called DelayNode. DelayNode is a queue based object, meaning that it can store any type of object.

4.2 Nodes in the FPM

Fig 4. illustrates the hierarchical structure and the inheritance mechanism of nodes in the FPM. IconObj is the ancestor of all nodes in this implementation. It is a dynamic image object: it can be drawn, moved, rotated, scaled, etc. IconObj uses an internal queue to hold the entities; it 'knows' the next node(s) by making use of the field NextIcon. This general node initialises itself, accepts entities, pushes entities to the next nodes, starts new processes, finishes old processes, and adds links to relevant nodes in the model.

Two groups of nodes based on IconObj have been created. Fig. 4 is a high level hierarchical structure which illustrates these groups.

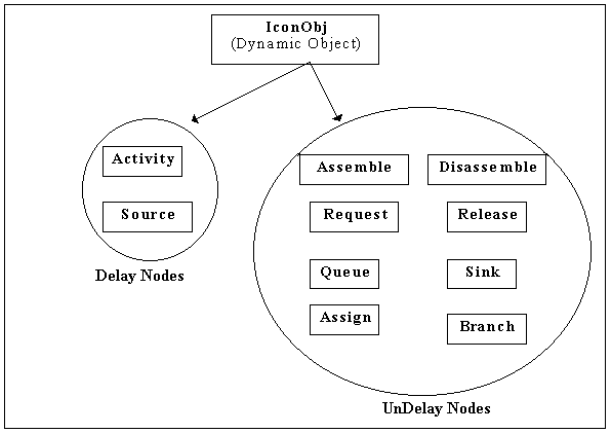

Figure 4: The Hierarchical Structure and Inheritance of The Model Nodes

The first group is the *Delay Nodes* which includes the Source node and the Activity node. The Source node creates tokens of a particular token category. The Activity node delays tokens for some amount of time before sending them on their way. The Activity node is not capacity limited, meaning that it can process any number of tokens at the same time.

The second group is the *UnDelay Nodes*, which includes the following nodes types:

- Assemble node: assembles or joins tokens belonging to different categories.
- Disassemble node: disassembles tokens of different categories.
- Request node: allocates adequate units of a resource to tokens.
- Release node: releases some of the resources owned by a token.
- Queue node: creates tokens in the initialisation step and delays a token until a suitable message is received from the system.
- Sink node: terminates tokens which arrive at it, acting as an 'exit door' for the system.
- Assign node: changes the values of token or system attributes.
- Branch node: responsible for aspects of routing decisions; the decisions depend on the user's needs (for example, when the user wants to route a token to different arcs depending on the token category or its attributes).

For a detailed description of these nodes the reader should refer to Kienbaum and Paul (1994a) or Pooley and Hughes (1991).

4.3 Other Elements in the FPM

The remaining modelling elements in the FPM are tokens, resources, activities, and attributes. Tokens represent the system entities that will be generated by the source, are engaged in some processing activities, and then finally leave the system. Each token can carry any type of objects and hold information. Activities are used to hold information such as the 'owning' node, a list of tokens that are involved in the activity, and its finishing time. Attributes are used to store information concerning tokens, resources, and system variables.

5 FPM IN ACTION: MODELLING A FLEXIBLE ASSEMBLY SYSTEM

This section explores the FPM through the crafting of a simulation model for a manufacturing system. The model layout and problem description will be introduced. The characteristics of using different modelling approaches will be discussed to highlight the advantages that we see in using the FPM.

5.1 Model Layout and Problem Description

The problem being investigated is not of principal importance to this work, since it is mainly concerned with the modelling process in its own right. It could be an investigation of production quantity or the calculation of the average time that is required to produce a product. The model to which we refer has been discussed in (Carrie 1988). Fig. 5a shows level 1 of the system which consists of a loop conveyor serving three workstations, at each of which machines perform some assembly task. Each workstation has a buffer position at which material may wait for the machines to complete the operation on its previous assembly. There are three types of assembly, each of which requires a different type of material, and therefore visits a different set of workstations.

At WORKSTATION 1, materials of type 1 and 2 (M1 and M2) will be assembled to produce a new material

(M12). By contrast, at WORKSTATION 2 a new material (M34) will be produced by assembling M3 and M4. WORKSTATION 3 produces the final product by assembling M12 and M34. The Conveyor has 30 carriers on it and it keeps moving on all times.

Several interesting nodes have been used to describe the logic of this model, in particular, a 'conditional assemble' node. Such a node does the assemble process when a specific condition has been satisfied. CASM1 is an example of a conditional assemble node.

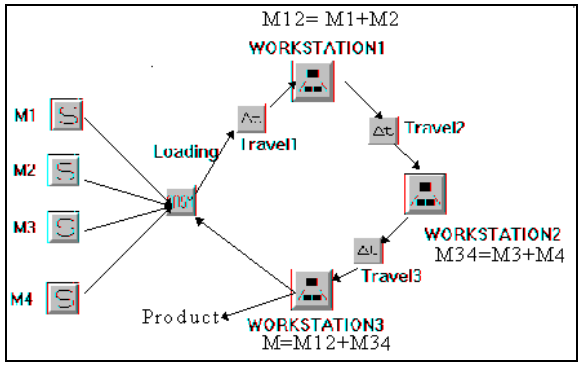

Figure 5a: A Flexible Assembly System Using Iconic Representation Level 1

CASM1 works as follow: when an empty carrier arrives at the node, it will do the assemble process (loading) only if there is any material queuing; otherwise, the carrier travels empty on its way to WORKSTATION 1. Unlike the conditional assemble node, the assemble node adds the entity to the internal queue until the required entity arrives. WORKSTATION is an example of a hierarchical node. The hierarchical node is used, usually, to simplify the graphical representation. If necessary, the user can specify the internal structure of any hierarchical node. Travel1, Travel2, and Travel3 are activity nodes, which represent the travel activities between the different workstations. The difficult part in this model is that the conveyor moves at all times and no loading can be done until the empty carrier arrives at the loading point. The conveyor takes fifteen seconds to travel from the first loading point to WORKSTATION 1, and takes ten seconds to travel between any two sequential workstations.

Figure 5b shows the internal structure of the hierarchical nodes (WORKSTATION 1, WORKSTATION 2, and WORKSTATION 3) that appeared in Figure 5a. When a carrier arrives at WORKSTATION 1, the branch node (B1) will check it. If B1 has not found any material in the queue of the carrier, the carrier will travel on its way to the conditional assemble node (CASM2). Otherwise, the carrier goes to the branch node (B2) which is responsible for identifying the types of material. At B2, Materials type 1 or 2 are sent to the disassemble node (CASM1), which will unload it from the carrier. Any other types of material are sent to the activity node, Travel2. B3 sends the materials, which must be type 1 or 2, to station 1 and sends the carrier to CASM2.

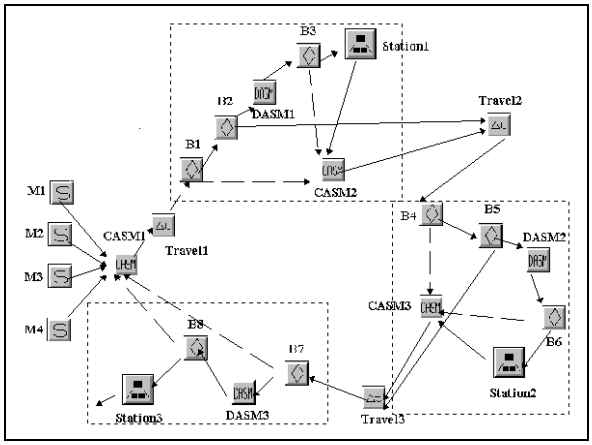

Figure 5b: A Flexible Assembly System Using Iconic Representation Level 2

CASM2, which is a conditional assemble node, works like CASM1. Station 1 is a hierarchical node which represents the assemble process to produce material type M12. The same logic can be used in describing the internal structure of WORKSTATION 2.

WORKSTATION 3 is a little different. Any carrier arriving at this workstation must be either empty or carrying materials type M12 and M34 that have been produced by WORKSTATION 1 and WORKSTATION 2. At WORKSTATION 3, both types of materials (M12 and M34) are required, which means there is no need to check the type of materials. After assembling M12 and M34 to produce the final product (M), WORKSTATION 3 sends the final product (M) to the collecting point using another conveyor.

5.2 Model Analysis and Discussion

In this research, iconic representation has been used in both the analysis and model building phases. This iconic representation is computer-based rather than paper-based. It serves as a framework with which the simulation user can analyse and conceptualise the problem. It can also be used as a communication medium among the people involved in the project. As mentioned above (and outlined in Fig. 5) new nodes have been introduced to improve the iconic representation.

The implementation of this iconic representation using an Object-Oriented language and any existing modelling

approach faces many difficulties. The four established modelling approaches introduced in section 3 have been used in this investigation. The attempt to use the three phase method faced three major problems. First, the three phase method uses the activity as the main building block. Entities move to the activity from the previous queue(s) to engage in some process and after finishing the process they move to the next queue(s). As such, the three phase method does not support the *UnDelay Nodes* such as assemble, disassemble, assign, branch, etc. Second, the three phase method does not contain any detail to support Object-Oriented programming. Third, the three phase approach forces the use of Activity Cycle Diagrams (ACDs) or similar diagrams to specify the simulation model's steps to the computer. In our view, this makes it difficult to model complex system behaviour using the three phase approach.

The process based (process-interaction) approach seems to be attractive, since it is similar to a production description. The process is a sequence of operations through which an entity must pass during its life cycle within the system. During the simulation time, entities will be created as members of different classes. When an entity is created, it takes the process of its class as a template for its future life, and the system then keeps track of how far the entity has moved through its process. A control program, named 'executive', supervises the progress of each entity through its process template, passing control to each process in turn, and trying to move the corresponding entity as far as possible through its template at a given point in simulation time. However, the approach does have two disadvantages. First, it requires a rather complex executive. More importantly, the interacting processes are more complex to program and the executive increases in complexity in line with the model's complexity. This makes enhancement of an existing program more difficult.

The Event-Scheduling approach concentrates on events and their effect on systems state. However, an event's routines can get rather complex, making enhancement and debugging somewhat tortuous (Pidd 1992a). Another disadvantage is that the event approach usually requires the user to become directly involved in computer programming.

The last modelling approach that has been used in this investigation is the Four Phase Method (FPM). This approach supports iconic representation. The approach pays special attention to *Delay Nodes* and *UnDelay Nodes*. However, the user can introduce any new node that has been found necessary. The most important advantage of this approach is that its executive remains simple irrespective of the model's complexity. This makes the enhancement of existing code possible. Another advantage of this approach is that it reduces the model run time by recognising which nodes have been affected by the previous process, and asks only those nodes to start.

6 CONCLUSIONS

This paper has attempted to combine a new simulation world view (the FPM), Object-Oriented Programming (OOP), and iconic representation to build a simulation environment for application in discrete-event simulation modelling. Both the modelling and implementation phases of discrete-event simulation have been addressed. The aim has been to build the simulation model gradually and uniformly from the analysis to final simulation model implementation. Such an approach might encourage a non-specialist developer to use simulation.

This research suggests that the FPM has the simplicity of the three phase method and the power of the process-interaction approach in modelling complex system behaviour. The FPM appears to be a simple, high-level system and enhances the conceptualisation of simulation problems and the understanding of system behaviour. At the same time, it reduces the time to produce working, valid simulation models by supporting the task of the modeller.

REFERENCES

Carrie, A. 1988. *Simulation of manufacturing systems*. London: Wiley.

Drury, C.E., and K.R. Laughery 1994. Fundamentals of simulation using *MICRO-SAINT*, in: *Proceedings of the Winter Simulation Conference*, ed. J.D. Tew, S. Manivannan, D.A. Sadowski, and A.F. Seila, 546-551. SCS, New York,USA.

Fishwick, P.A. 1994. Simulation model design, in: *Proceedings of the Winter Simulation Conference*, ed. J.D. Tew, S. Manivannan, D.A. Sadowski, and A.F. Seila, 173-175. SCS, New York, USA.

Hlupic, V., and R.J. Paul 1994. Simulating an automated paint shop in the electronics industry, *Simulation Practice and Theory* 1**:** 195-205.

Joines, J.A. 1994. Design of object-oriented simulation in C++, in: *Proceedings of the Winter Simulation Conference*, ed. J.D. Tew, S. Manivannan, D.A. Sadowski, and A.F. Seila, 157-165. SCS, New York, USA.

Kienbaum, G., and R.J. Paul 1994a. H-ACD: Hierarchical activity cycle diagrams for object-oriented simulation modelling, in: *Proceedings of the Winter Simulation Conference*, ed. J.D. Tew, S. Manivannan, D.A. Sadowski, and A.F. Seila, 600-610. SCS, New York: USA.

Kienbaum, G., and R.J. Paul 1994b. H-ACDNET: An object-oriented graphical user interface for simulation modelling of manufacturing systems, *Simulation Practice and Theory* 2**:** 141-157.

Macredie, R.D., and H.I.A. Odhabi 1995. A graphical user interface for discrete event simulation., *International Journal of Manufacturing System Design*, 2(2): 97-104.

MODSIM II 1992. *The language for object oriented programming (reference manual).* La Jolla, CA.: CACI Product Company.

Odhabi, H.I.A., and R.J. Paul 1995. Accessible simulation modelling for manufacturing system design., *International Journal of Manufacturing System Design*, 2(2): 145-151.

Pidd, M. 1992a. *Computer simulation in management science* 3rd ed, Chichester.: John Wiley & Sons.

Pidd, M. 1992b. object orientation and three phase simulation, in: *Proceedings of the Winter Simulation Conference*, ed. J.J. Swain, D. Goldsman, R.C. Crain, and J.R. Wilson, 600-610. SCS, San Diego, CA.

Pooley, R.J., and P.H. Hughes 1991 Towards a standard for hierarchical process oriented discrete event simulation diagrams, Part II: The suggested approach for flat models, *Transactions of Society for Computer Simulation* 8(1): 21-31.

AUTHOR BIOGRAPHIES

HAMAD I. ODHABI is a Ph.D. student in the Department of Information Systems and Computing, Brunel University. He received a B.Sc. degree in Physics from King Suad's University, Saudi Arabia in 1988, and he received an M.Sc. degree in Simulation Modelling from Brunel University in 1994.

RAY J. PAUL holds the first U.K. Chair in Simulation Modelling, at Brunel University. He previously taught Information Systems and Operational Research at the London School of Economics. He received a B.Sc. in Mathematics, and a M.Sc. and a Ph.D. in Operational Research from Hull University. He has published widely in book and paper form (two books, over 100 papers in journals, edited books and conference proceedings), mainly in the areas of the simulation modelling process and in software environments for simulation modelling. He has acted as a consultant for variety of United Kingdom Government departments, software companies, and commercial companies in the tobacco and oil industries.

ROBERT D. MACREDIE is a Lecturer in the Department of Information Systems and Computing, Brunel University. He received a B.Sc. in Physics and Computer Science and a PhD in Computer Science from Hull University. His research interests are in human-computer interaction, simulation modelling, and virtual environments/virtual reality. He has published widely in these areas, and is also executive editor of the international journal *Virtual Reality: Research, Development and Applications*.