Proceedings of the 1998 Winter Simulation Conference
D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.

MAKING SIMULATION MORE ACCESSIBLE IN MANUFACTURING SYSTEMS THROUGH A 'FOUR PHASE' APPROACH

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

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

ABSTRACT

This paper will describe an approach to the development of computer simulations - the 'four phase' approach - which aims to be more accessible than established approaches to non-specialist developers in manufacturing system design. This paper will briefly review the traditional 'three phase' approach and highlight its potential drawbacks. This paper will then go on to suggest that the benefit of the 'four phase' approach over the more established three phase approach that it is more suited to simulations developed through iconic representations. Such iconic representations are seen as central to the spread of simulation modelling into application domains such as manufacturing system design. The work reported here also suggests that complete modelling environments can be built around those iconic representations which allow the user the opportunity to concentrate on the manufacturing system's behaviour rather than on developing computer code to support the model. This is achieved by the automatic generation of the code from the iconic representation. Modelling environments that provide such a focus are likely to be more usable by those without a specialist simulation background.

1 INTRODUCTION

Computer simulations offer opportunities to develop models of problem domains and activities which can be used to assess the prospective effectiveness of particular solutions. Manufacturing system design is an ideal arena for the application of simulation, since it is often vital to assess the implications of particular solutions prior to their acceptance. One (non-simulation) approach is to develop partial solutions - based on discussion and experience - for assessment, but their cost can often be very high, and the resulting solution may not meet expectations. An alternative is to develop (or model) a computer simulation of the problem domain. Such simulations provide the opportunity for those involved in the application domain to explore their problem area and develop simulated solutions which can form the basis for decisions about the physical solutions that are required. The cost and convenience of developing simulations as an aid to decision making can make them an attractive proposition in manufacturing system design.

Over the past years, the simulation modelling community has developed approaches to modelling which have been aimed at making the modelling process more accessible to non-specialists. A key development has been iconic representations which can be used to specify the logic of the simulation model. Here, icons are used to represent physical elements and activities of the system being modelled. These environments are underpinned by the development of graphical user interface techniques, more general advances in computer graphics, and the development of object-oriented languages on which to base the modelling environment.

A general problem, however, is that iconic representations cannot easily be used with the 'three phase' method (Tocher 1963) which is widely used in simulation. The 'three' in three phase refers to the number of phases that are executed in each cycle of the simulation. Three phase methods are generally simplistic formalisms which cannot suitably model complex behaviour of systems.

Three phase methods have an 'activity' as their basic building block; the building block has two events that can describe it - the start of activity event and the end of activity event. Whilst relatively simple to understand, the limited components of three phase methods (such as the popular Activity Cycle Diagrams, or ACDs) mean that they are not very good for accurately modelling the complex behaviour often associated with manufacturing system design. Iconic representations, by contrast, tend to be richer, offering the modeller increased opportunity to model complex systems behaviour through an extended set of building blocks.

This paper will describe a possible solution to this problem: the development of a 'four phase' method which supports existing iconic representations and provides a way to implement and execute simulation models developed using them. The four phase method is used within a modelling environment that is currently being developed. The environment supports the development of an iconic representation of the system model, and then aims to automatically generate the simulation program. This removes the need for direct programming by the model developer and has the potential to open up the development of simulations to non-specialists. This offers experts in fields such as manufacturing system design the scope to develop and explore their own simulations of complex systems.

Before moving on to discuss the four phase approach, this paper will develop the central arguments through a discussion of the three phase method and ACDs, and their limitations for modelling complex system behaviour. This will be used as justification for the development of both iconic modelling environments and the four phase approach that we propose.

2 THE THREE PHASE APPROACH TO SIMULATION MODELLING

As it names suggests, the three phase method for simulation modelling divides the execution of the model into three phases (see Paul 1993; Paul and Balmer 1993 for more detailed discussions), which can be summarised as follows:

Phase 1: Determine when the next event in the simulation model is due. This event will usually be the completion of a current/on-going activity. Advance the simulation clock to the time of this event.

Phase 2: Execute the events identified in phase 1, this will usually involve moving the entities from the activities that have just completed into appropriate queues in the model.

Phase 3: Attempt any events in the model that are conditional in turn, and execute those for which the conditions are satisfied. Repeat this process until no more conditional events can take place (i.e. no more activities can start).

These phases are repeated until the simulation is complete - when a terminating condition is reached, for example.

2.1 Activity Cycle Diagrams (ACDs)

Activity Cycle Diagrams (ACDs) are an example of a three phase method. ACDs, which are based on (Pidd 1998 and Tocher’s 1963) idea of stochastic gearwheels, are used to create a model of the interaction of system objects, and are especially useful when modelling systems which have strong queuing structures. ACDs are simple in that they use only two symbols to describe the life cycle of objects, or entities, in the system. Entities are either idle in some form of 'queue', or active through engagement with other entities in time consuming activities. After specifying the system through an ACD, the ACD would be executed, or run, using the three phases described in section 2.

Whilst they are generally a paper based approach, ACDs can be used to underpin iconic models for computer based simulations: the two symbol types (queue and activity) having icons associated with them and being used to describe the life cycle of the system’s objects or entities. 'Arcs' are, where appropriate, used to join activities and queues, so that the entities may flow through the model.

2.2 Extended Activity Cycle Diagram (X-ACDs)

The simplicity of ACDs, and their associated limitations for developing computer based simulations of complex systems, led researchers such as Pooley to suggest extensions to enhance the use of ACDs for specifying process based discrete event models (Pooley 1991a, 1991b). Pooley proposed an extended set of symbols (shown in figure 1) to describe processes in simulations. This symbol set is used to develop Extended Activity Cycle Diagrams (X-ACDs) which can accurately model more complex system behaviour than simple ACDs.

As Figure 1 shows, the symbol set can be divided into two parts: symbols which control the flow; and resource and queue symbols. The main characteristics of these different groups will be briefly discussed here.

The symbols which control the flow in the model are similar in function to those used in conventional flows charts. There are symbols to control the start and termination of processes in the model and to 'hold' the model. 'Holds' represent activities whose duration have known properties. The different symbols are linked by vectors which indicate the flow of control in the model. The algorithmic description of a process is contained in a direction graph made up from these symbols.

The second group of symbols model various types of queuing activities. The group includes 'resource' and 'bin' symbols. As their names suggest, the symbols introduce (or resource) the model with a particular entity and remove (or bin) entities respectively. Resource symbols can, for

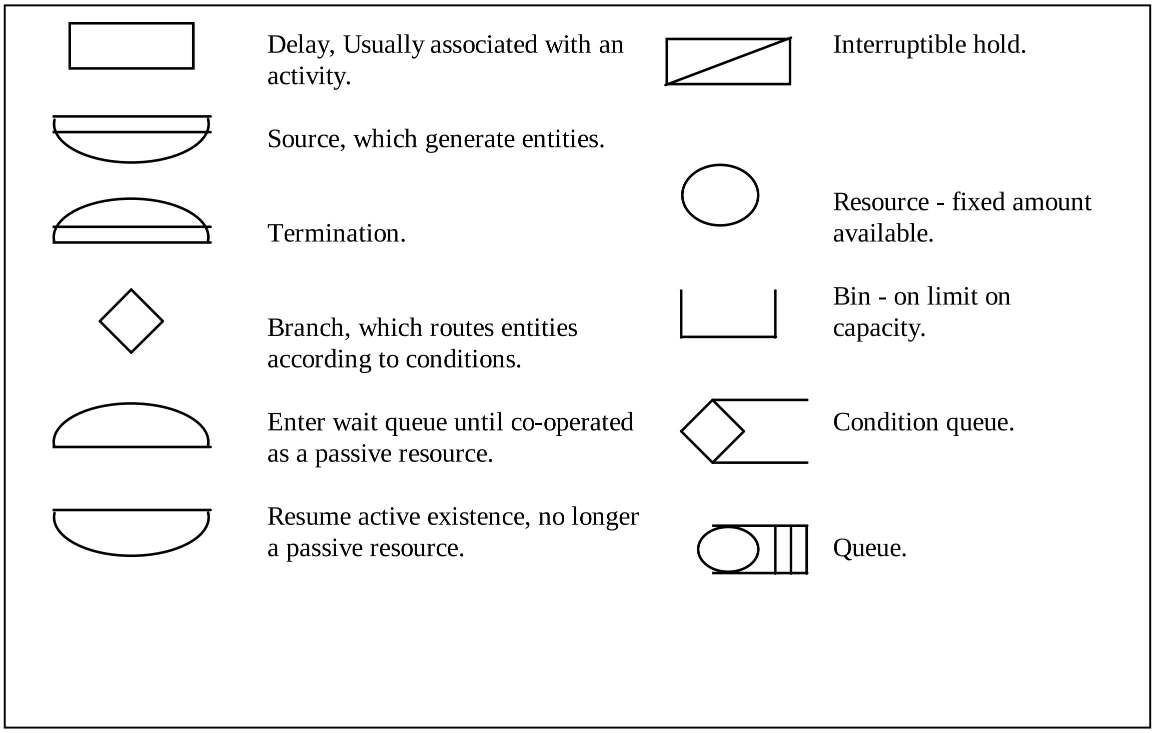
Delay, Usually associated with an activity.
Source, which generate entities.
Termination.
Branch, which routes entities according to conditions.
Enter wait queue until co-operated as a passive resource.
Resume active existence, no longer a passive resource.
Interruptible hold.
Resource - fixed amount available.
Bin - on limit on capacity.
Condition queue.
Queue.

Figure 1: The X-ACD Symbol Set (Adapted from Pooley and Hughes 1991)

example, be used to model the availability of a raw material used in the manufacturing process. A resource is limited by initial conditions, such as tonnes of raw material available. Bins, by contrast, can receive unlimited amounts of material. This makes them suitable for modelling the most general cases of producer-consumer process interaction.

The remaining symbols in the second group are used to model different types of queue. The conditional queue is a hybrid, combining the attributes of a resource and a decision box. A process whose flow of control reaches a conditional queue is blocked until the associated conditions are satisfied. This can be thought of as a 'wait until' construct (which is also offered by some simulation languages).

Pooley and Hughes (1991) note that the complete X-ACD symbol set has proved rich enough to allow the description of a large class of models. Accordingly, the set seems a suitable candidate for incorporation and development into an iconic modelling environment.

2.3 Hierarchy Activity Cycle Diagrams (H-ACDs)

Kienbaum and Paul (1994a) propose some modifications to the X-ACD symbol set, calling the resulting approach Hierarchical Activity Cycle Diagrams (H-ACDs). The objective of their work is to advance the presentation of the X-ACD diagramming technique, and illustrate how it could be used to create a usable iconic modelling environment. A key consideration of Kienbaum and Paul's work was to ensure that their modelling environment was computer based.

Kienbaum and Paul (1994b) suggest that the H-ACD development of the ACD technique has many advantages in meeting the needs not only of the analysis, but also of the modelling and design phases of object-oriented discrete event simulation projects.

Figure 2 shows the H-ACD symbol set, representing different types of processes, synchronisation, queuing and resource blocking mechanisms that can be used to model systems. The process symbols in H-ACD include the types 'source' and 'sink' which are used to represent the arrival (generation) and departure (termination) of transactions through the border of the system component being modelled.

Another symbol incorporated into H-ACD is the 'interruptable hold'. This symbol executes when a suitable interrupt signal is received. The 'transform' symbol describes a transformation process, which takes one category of entity as input and produces a different category as output. The 'assemble' symbol is concerned with the pre-requisite conditions necessary for an activity to start. H-ACD also supports a 'disassemble' symbol, used to disassemble the entities when an activity finishes.

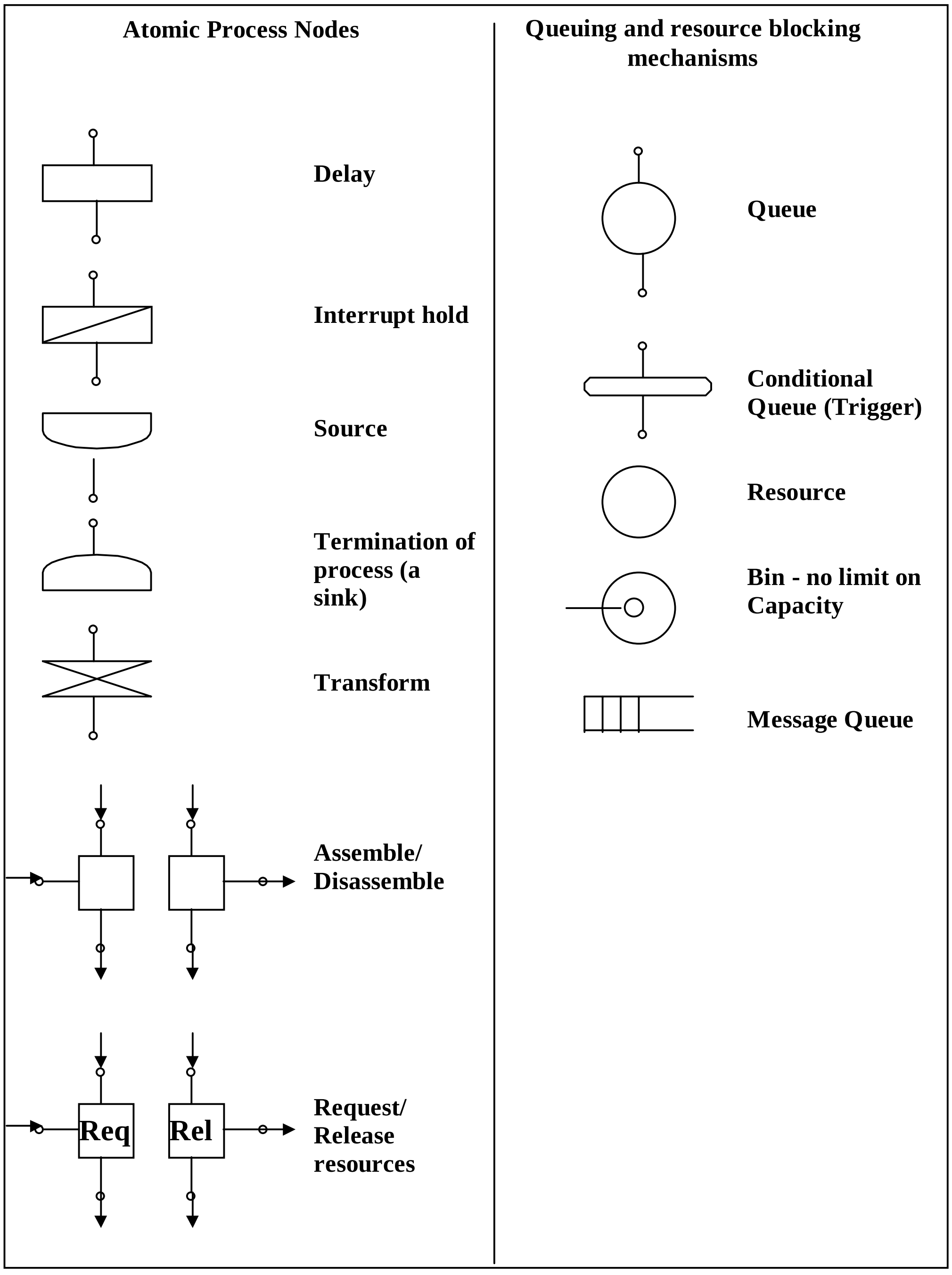
Atomic Process Nodes
Queuing and resource blocking mechanisms
Delay
Interrupt hold
Source
Termination of process (a sink)
Transform
Assemble/ Disassemble
Req
Rel
Request/ Release resources
Queue
Conditional Queue (Trigger)
Resource
Bin - no limit on Capacity
Message Queue

Figure 2: The H-ACD Symbol Set (Adapted from Kienbaum and Paul 1994a)

The H-ACD symbol set contains simple queues of various sorts, which service processes in the model through its life cycle. Queues are also used to represent build ups in the model as a result of a block in the flow of entities.
The conditional queue that was presented in the X-ACD set, is represented in H-ACD, but is called the 'trigger' symbol. As with the comparable X-ACD symbol, a process whose flow of control reaches the trigger symbol is blocked until the associated conditions are satisfied. The message queue provides a mechanism through which trigger symbols are informed of waiting entities. Resource and bin symbols have the same basic characteristics as their X-ACD counterparts.

The H-ACD symbol set has been used to successfully model complex systems behaviour within a computer-based modelling environment. However, the expanded symbol set of H-ACD (and X-ACD) are not compatible with the three-phase approach to model execution. This presents an interesting situation for the modeller. The modeller can either use a simple formalism based on the three phase method (such as ACDs) which has limited opportunities to model complex system behaviour, or make use of an extended iconic representation and look for a suitable method of execution. The development of a suitable method of execution will be discussed in the remainder of this paper.

3 THE FOUR PHASE APPROACH TO MODEL EXECUTION

The alternative approach that is reported in the remainder of this paper is called the 'four phase' approach. The approach is underpinned by the following two general considerations:

(i) Each symbol in the H-ACD set has an internal queue. When the symbol has completed its function it can hold any relevant entities in this internal queue until the relevant time for them to be released.

(ii) Distinctions are drawn between different symbols, with two important groups being identified. The first, called *Delay Nodes*, includes the activity and source symbols, which both have the common characteristic of being able to delay entities for some period of time before sending them on to the internal queue. The second group are the *UnDelay Nodes*, which includes the assemble, disassemble, request, release, branch, assign, queue, and trigger symbols. The common characteristic of these symbols is that they do not delay entities when they are required by another symbol in the model.

A diagrammatic representation of the four phase method is shown in figure 3. We can describe the four phases of model execution as follows:

Phase 1: Check the finish times of all the Delay Nodes currently in progress. Find the earliest of these, and advance the clock to this time.

Phase 2: For the Delay Nodes, finish all the processing scheduled to be completed at this time, and move the relevant entities into the internal queue.

Phase 3: Check all of the UnDelay Nodes identifying all those which should start processing at this time. Perform the relevant processes (with duration time zero). Repeat the check until there are no UnDelay Nodes with processes to start at this time.

Phase 4: Start the processing of any relevant Delay Nodes. Calculate when the Delay Node will finish its processing, and record this time. When all relevant Delay Nodes have been processed, check for an interrupt or any specified finishing conditions. If the model is not due to terminate, return to phase 1 and being the execution cycle again.

The implementation of the four phase approach is at an early stage. We have used an object oriented approach to its implementation. This is appropriate as it provides a strong mapping between the concepts and the actual implementation of the icons, or symbols, which are used to make up a simulation model. Each symbol can be implemented as an individual instance of the relevant class of object. For example, modelling a particular queue in the simulation would use an instance of the queue object. The approach that we are currently exploring takes this approach but also views each of the four phases as an object. This approach requires some supporting objects to be defined for use in housekeeping activities. The four most important supporting objects are defined as:

(i) **DelayActiveListManager**: The 'DelayActive-ListManager' maintains a list of the Delay Nodes in the simulation that are currently active, along with the time that each of the activities of the Delay Nodes are to complete. On completion of an activity, the DelayActiveListManager should delete the information concerning the activity (and therefore the relevant Delay Node) from the list.

(ii) **StopActiveListManager:** The 'StopActive-ListManager' object contains only the names of the Delay Nodes that stop particular activities during the execution of the simulation. Where a Delay Node has stopped all of its activities at a particular time, the StopActiveListManager deletes this Delay Node from its list.

(iii) **UnDelayNodeListManager:** This object simply maintains a list of all of the model's UnDelay Nodes.

(iv) **DelayNodeListManager:** Similarly, this object maintains a list of all of the model's Delay Nodes.

The implementation of the four phases as objects in their own right can be described as follows:

Phase1 Object: 'Phase1 Object' is responsible for scanning through time, looking at the simulation to find the earliest finishing time for an activity and setting the simulation clock to that time. 'Phase1 Object' begins by asking DelayActiveListManager to scan its list to find the earliest finishing time. 'Phase1 Object' also initiates StopActiveListManager, ensuring that all Delay Node names in the simulation are added.

Phase2 Object: The 'Phase2 Object' interrogates the StopActiveListManager to compile a list of Delay Nodes that have activities which should be stopped at this time. 'Phase2 Object' then ensures that these activities are stopped.

Phase3 Object: The 'Phase3 Object' integrates the UnDelayNodeListManager to find the names of all of the UnDelay Nodes in the simulation. 'Phase3 Object' asks all the UnDelay Nodes to start their respective processes. This request is repeated until no Delay Node is able to start processing.

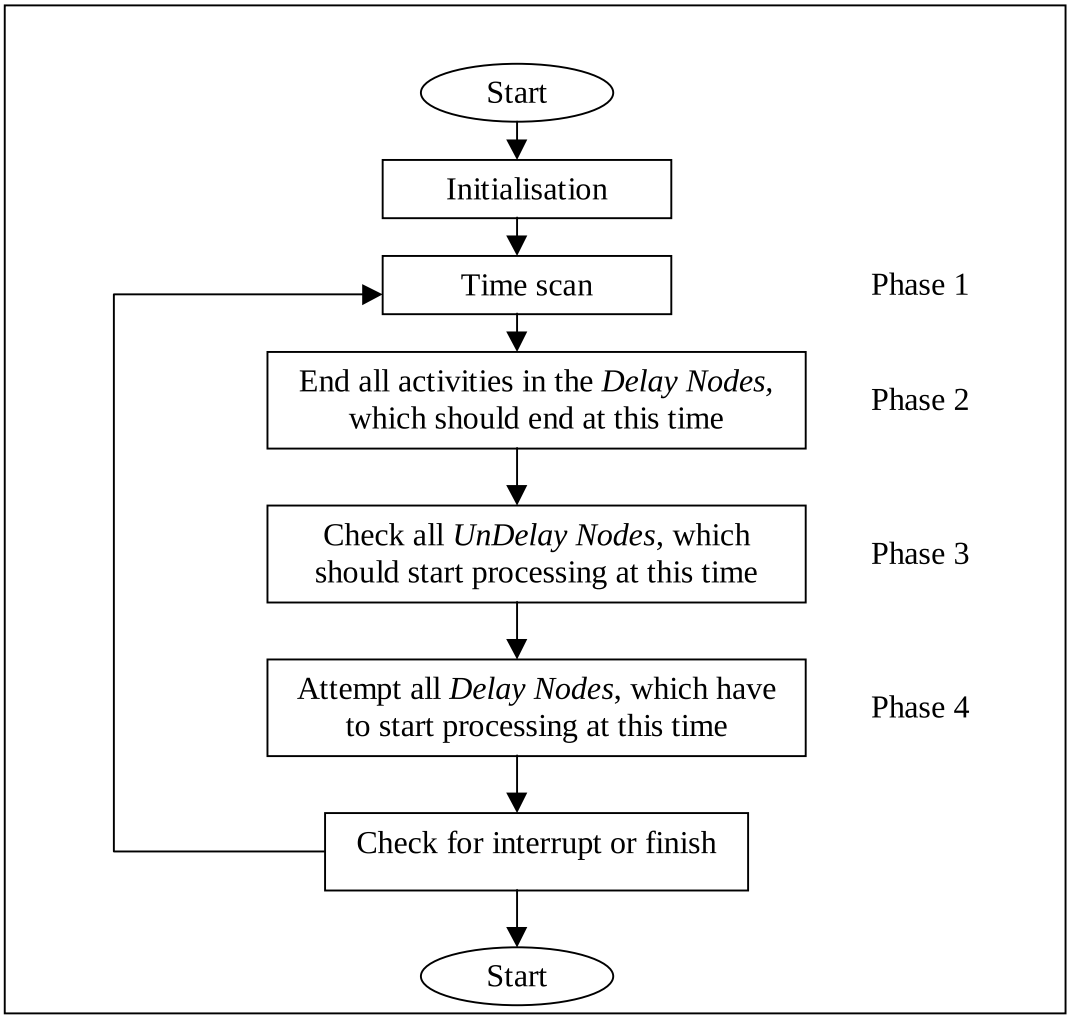
Start
Initialisation
Time scan
Phase 1
End all activities in the *Delay Nodes,* which should end at this time
Phase 2
Check all *UnDelay Nodes*, which should start processing at this time
Phase 3
Attempt all *Delay Nodes*, which have to start processing at this time
Phase 4
Check for interrupt or finish
Start

Figure 3: An Illustration of the Four Phase Method

Phase4 Object: The 'Phase4 Object' uses the DelayNodeListManager to keep a list of all Delay Nodes in the simulation. 'Phase4 Object' then asks all Delay Nodes to attempt to start their respective activities. If a Delay Node is able to start an activity, 'Phase4 Object' asks DelayActiveListManager to add the Delay Node's name, the activity identification number, and the stopping time to its list.

4 AUTOMATING SIMULATION THROUGH THE FOUR PHASE APPROACH

Our on-going research is looking at developing a complete modelling environment which takes an iconic representation similar to that offered by H-ACD and builds it into a graphical front end that can be used by non-specialists in simulation. The modelling environment aims to provide a usable graphical interface through which the user can develop the model of their system. The environment takes the iconic description of the system and automatically generates the computer code to support it. This moves the emphasis for the user away from the complexities of developing code to represent the model of their system, and allows them to focus on the central issue of effectively modelling the potentially complex behaviour of their system.

This also offers flexibility to users, since they can develop their model, exploring the effects of different changes and setups. This may support more effective final solutions, since a higher resource can be allocated to developing the behaviour of the system simulation rather than the code to support it.

5 CONCLUSION

There are many advantages to using iconic representations for discrete event simulation modelling, especially for modelling highly complex behaviours associated with many manufacturing systems. Centrally, these include the accessibility of the modelling approach to the user. Iconic representations can have a close conceptual mapping to the physical elements of the system being modelled, representing them more closely than other approaches.

One drawback of iconic representations is that they do not fit with the three phase approach to simulation that has been central to much of the simulation work undertaken since the 1960s. This is because the three phase method uses only two symbols - as characterised by ACDs - as opposed to the extended symbol set that characterises iconic representations, such as the symbol sets of X-ACDs and H-ACDs.

To support the implementation of iconic representations, this paper has proposed a four phase method for simulation execution which can be used for the H-ACD symbol set. This approach is currently being developed into a modelling environment using an object oriented language, with each phase modelled as a distinct object. The modelling environment is used to iconically develop a model of system behaviour using the icons of the symbol set. The computer code to support this model is automatically generated by the modelling environment.

It is hoped that the accessibility and flexibility that the modelling environment offers will provide an important opportunity for those modelling complex systems behaviour, and that it will be a useful tool in manufacturing system design.

REFERENCES

Kienbaum, G. and R. J. Paul (1994a). H-ACD: hierarchical activity cycle diagrams for object-oriented simulation modelling. In *the Proceedings of the Winter Simulation Conference* (IEEE Cat. No. 94CH35705), edited by Tew, J. D., Manivannan, M. S., Sadowski, D. A., Seila, A. F. (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.

Paul, R. J. (1993). Activity Cycle Diagrams and the three phase method. *In the Proceedings of the Winter Simulation Conference* (Cat. No. 93CH3338-1), edited by Evans, G. W., Mollaghasemi, M., Russell, E. C., Biles, W. E. (New York, USA).

Paul, R. J. and D. W. Balmer (1993). *Simulation Modelling* (Lund, Sweden: Chartwell Bratt).

Pidd, M. (1998). *Computer Simulation in Management Science* (4th edition) (Chichester, UK: John Wiley and Sons).

Pooley (1991a). Towards a standard for hierarchical process oriented discrete event simulation diagrams. Part I: a comparison of existing approaches. *Transactions of the Society for Computer Simulation*, 8(1): 1-20.

Pooley (1991b). Towards a standard for hierarchical process oriented discrete event simulation diagrams. Part III: aggregation and hierarchical modelling. *Transactions of the Society for Computer Simulation*, 8(1): 33-41.

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

Tocher, K. D. (1963). *The Art of Simulation* (London: English University Press).

AUTHOR BIOGRAPHIES

HAMAD I. ODHABI is a researcher in the Department of Information Systems and Computing, Brunel University. He received a B.Sc. degree in Physics from King Saud’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 200 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 redaer 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.*