Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

H-ACD: HIERARCHICAL ACTIVITY CYCLE DIAGRAMS FOR OBJECT-ORIENTED SIMULATION MODELLING

Germano Kienbaum
Ray J. Paul

Department of Computer Science
Brunel University
Uxbridge, Middlesex UB8 3PH
United Kingdom

ABSTRACT

This paper describes a graphical representation formalism, Hierarchical Activity Cycle Diagrams (H-ACD). H-ACDs are based on more abstract modelling concepts formulated in Pooley's extended Activity Cycle Diagrams. The abstraction mechanism expressed more loosely in his ideas of *configuration diagrams* has been realised to its full extent to create a detailed graphical representation form to be used for object-oriented modelling and design of simulation programs. The diagrams are currently been used as the basis for a Graphical User Interface to support automatic program generation of simulation models for manufacturing systems applications. The diagrams nevertheless can be used generally as a tool to assist analysis and design of simulation systems, enabling a hierarchical, modular and uniform view throughout the whole model development process of a wide range of discrete event simulation programs.

1 INTRODUCTION

In the analysis phase of a simulation study, the graphical representation formalism serves as a very useful framework with which the modeller can analyse and conceptualise the problem and as a communication medium among the people who are involved in the project. It can also become a communication means to specify simulation models to the computer. It is therefore desirable that the graphical representation formalism be a simple, high-level system abstraction so that it enhances conceptualisation of simulation problems and understanding of the implications of different modelling strategies. At the same time, it should have a sufficient number of powerful conceptual features to be able to model all possible situations (at least with respect to a certain class of problems being studied). That is, on the one hand it should contain all the conceptual features that are essential to represent all structural and behavioural aspects of simulation objects that may exist in a simulation problem class. On the other hand, the number of graphical building-blocks representing these concepts should be small and high-level enough for easy human comprehension and use.

This work proposes some modifications (termed H-ACD) to a well-known formalism of this type named the Activity Cycle Diagrams (ACD). It is based on another different extension of the original ACD diagramming technique presented by Pooley for specifying process based discrete event models (Pooley and Hughes, 1991). Our claim is that this version 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 in a manufacturing system environment.

Pooley's approach was developed initially based on the conventions of Activity Cycle Diagrams used to describe models for the DEMOS package, a primer for the SIMULA language (Birtwistle, 1979). The extensions were made to make it applicable for use with most process based discrete event simulation languages and to allow a description of a much wider range of models.

An additional work (Pooley, 1991b) was dedicated to formulating how a hierarchy could be introduced into the modelling process through the use of black boxes or *configuration diagrams* to further abstract parts of the

model for better understanding and for the specification of complex models. Although the author mentioned the possibility of using the formalism to cover other application areas than computing, as for example Flexible Manufacturing System, and to use it as a tool to help perform object oriented modelling of systems, he preferred to leave it as abstract as possible to keep its generality.

The objective of this work is to move forward in the presentation of this diagramming technique, from the point where it was left by Pooley, and show how it can be used to create a concrete formalism and a graphical user interface to support object-oriented discrete event simulation modelling. A graphical user interface with these characteristics for the automatic generation of simulation programs of manufacturing systems is described in Kienbaum and Paul (1994).

The present work is organised as follows. Section 2, subsection 2.1, makes a brief presentation of the standard ACD form to fix the basic ideas behind the use of the technique. The subsequent two subsections describe the concepts which led to the creation of the H-ACD representation formalism. The first step, described in subsection 2.2, consists of the modifications and additions to the basic set of symbols presented by Pooley to make them more appropriate for describing manufacturing processes. The second step, contained in section 2.3, consists of the application of a similar abstraction process as the so called *configuration diagrams* (Pooley, 1991) to add hierarchy to the model description. A significant change at this stage is introduced with the idea of creating hierarchical nodes and defining input and output pads attached to the nodes, inspired by the authors previous experience with the SIMPLEX-II simulation system (Schmidt, 1991).

Section 3 presents a simple example developed in a stepwise manner, evolving gradually from the standard ACD format to the extended ACD technique proposed by Pooley and finally coming to the presently proposed H-ACD format for the object oriented specification of simulation systems. A detailed discussion of the modelling difficulties encountered as well as of the experience gained so far by using the approach are also presented.

Section 4 makes further remarks on the use of H-ACD and on the full potentialities of the modelling technique. This section justifies our claim that the modifications made are much more than just a change in format to the previous diagrams. They incorporate a conceptual change in the way the systems analysis and modelling is performed, making them a real tool for object oriented modelling and design of simulation systems.

Finally section 5 recapitulates and points out the directions for further research using the graphical formalism.

2 GRAPHICAL REPRESENTATION FOR OBJECT-ORIENTED PROCESS BASED SIMULATION MODELLING

2.1 ACD Diagrams In Their Standard Form

The original symbols used in Tocher's ACD version (Tocher, 1963) showed only a circle to represent a *dead state* (generally associated with a *queue*) and a rectangle to represent a *delay* (also known as an *activity*), linked by arrows representing the flow of entities through their life cycles.

Figure 1 shows the primary elements of an ACD in its simplified version.

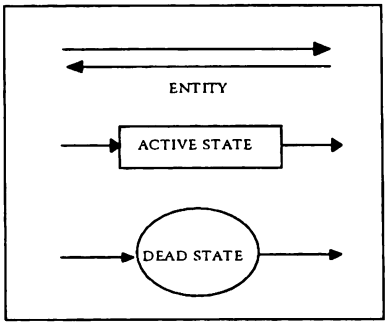

Figure 1: Primary Elements Of ACD

An *active state* or *activity* usually involves the co-operation of different classes of entity. The duration of an active state can always be determined in advance-usually by taking a sample from an appropriate probability distribution if the simulation model is stochastic.

A *dead state* involves no co-operation between different classes of entity and it is often thought of as a *queue*. Therefore, the length of time that an entity spends in a dead state cannot be determined in advance.

The diagram itself is a map which shows the life history of each class of entity and displays graphically their interactions. Each class of entity is considered to have a lifecycle which consists of a series of states. The entities move from state to state as their life proceeds (Pidd, 1992). The example of the Pub model described in Section 3 is shown using a standard ACD in Appendix A.

2.2 H-ACD's Atomic Processes

This first step consisted of some modifications and additions to Pooley's extended set of symbols to make them more appropriate for describing manufacturing processes. This step comprised also a change in the graphical appearance of the original symbols proposed in his version of the diagrams. Although not always necessary, these changes were made to make them more readable for use in a graphical hierarchical interface for interactive model building. Whenever possible the correspondence between the original and modified form is emphasised by making use of a similar type of icon. Using the original version also proves useful as an intermediary step in the modelling process, due to their more abstract nature. This allows a stepwise description of models with an increasing gain of understanding of the model behaviour until the final representation in the H-ACD format proposed.

Figure 2 shows the derived symbols in the new representation form. They are also named atomic nodes, which represent different types of processes, synchronisation, queuing and resource blocking mechanisms.

The process nodes, additionally to *activities* (or *delays*), now include the types *source* and *sink* nodes. They are used to represent the arrival and departure of transactions through the border of a system's component being modelled.

Another type of node is the *interruptable hold*, which contains a clause for execution in case an interrupt signal is received while it is being performed.

The *Transform* node describes a transformation process, in which the incoming entity ceases to exist and another different class of entity is produced at the output.

The *Assemble/Disassemble* nodes describe the co-operation between processes, when the object class described by one of them is co-opted as a passive resource by the other or when it is again released to resume its active life. In manufacturing terms they represent operations such as assembling and disassembling of product parts.

The *Request/Release* nodes simply describe the acquiring/releasing of a resource needed by an entity to perform any subsequent activity in its lifecycle.

The second set of symbols contains simple queues of various sorts, which the process needs to draw things from or to put things into, before continuing its life cycle.

Queues are used to represent proper queues which build up in the model as a result of a block in the flow of entities. Queues used only as dummy queues in the original ACD form and in the X-ACD form are not represented in the model. This grants the allocation of entities to the appropriate physical queues existing in the system and will facilitate the gathering of statistical data related to the permanence of entities in some specific systems locations.

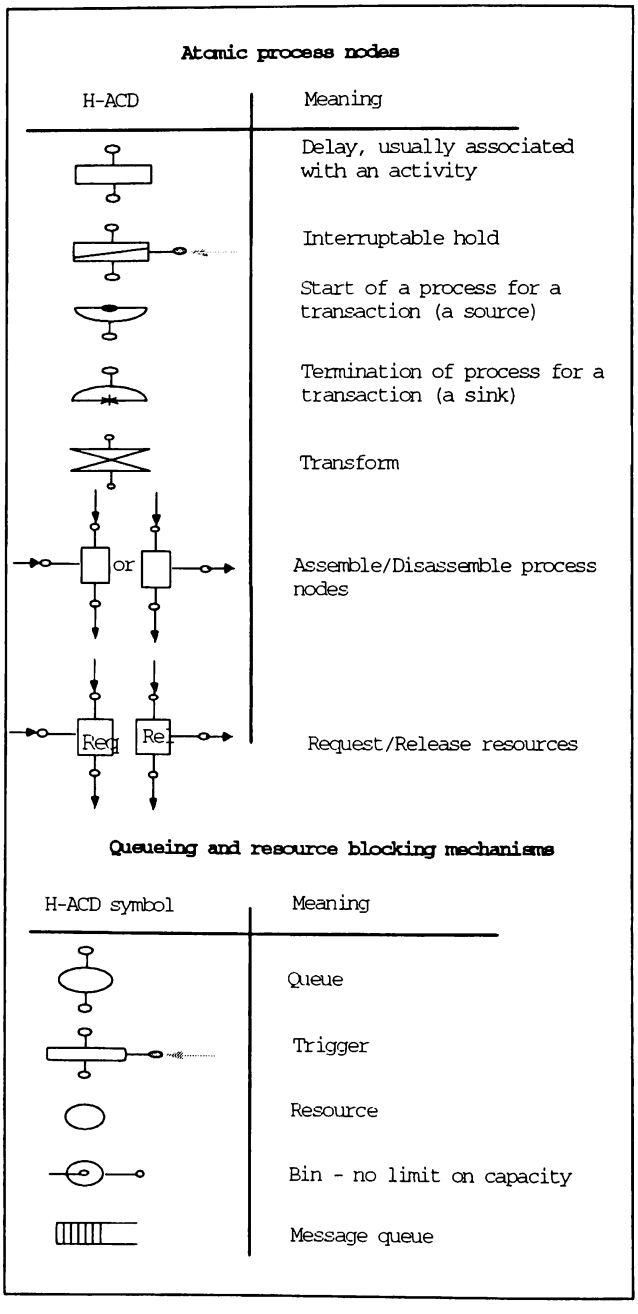

Figure 2: The set of process and queue node types

The *Trigger* is a hybrid between a queue and a decision box. A process whose flow of control reaches

this node is blocked until the associated condition is satisfied, in an equivalent manner to the *wait until* constructs of some simulation languages.

The *resource* and *bin* has a type and an amount, but its individual units differ from the queues for entities because they are anonymous and have no attributes. A *resource* is always limited to its initial quantity. On the other hand, a *bin* can receive unlimited amounts of new tokens, which can be created by a process and does not have to be acquired first (as do resources and entities). A *bin* is used to model the most general cases of producer / consumer relationships.

The *message queue* can contain messages with a data attribute structure. Thus it can be used to model objects which have an individual identity, flowing among processes. This allows the representation of aspects of the objects' behaviour dependent on data or information flow and the collection of statistics on these changes.

The set of symbols shown are similar to those proposed by Pooley and Hughes (1991) to logically describe flat simulation models in a computing environment. The main difference is that in this new set each element describes an actual physical element of the system been modelled. The diagrams no longer represent solely a logical description of the model, they represent more closely the objects of the real system. Symbols such as the *Waitqueue* and the start of a process either through generation or through scheduling by another process were suppressed. New symbols, such as *Assemble/Disassemble* and the *Transform* processes were added. The *Trigger* symbol is introduced in place of the *conditional queue*, and it becomes the node in charge of all the synchronisation mechanism between the entities. Additionally, for different types of application, particular symbols may be added to better reflect the type of environment the modeller is addressing. In a Graphical Interface any new symbol can be treated exactly in the same manner as the existing ones, by performing an operation on input tokens and showing the result as output tokens. This operation may be described in any sort of algorithmic description, including a separate piece of code written in a high level language. The *queues* in the model act also as automatic statistics collection mechanisms, and are therefore a necessary element, which was not shown in the previous form of X-ACD diagrams.

2.3 Hierarchical modelling with H-ACD.

To provide for hierarchical object oriented model construction the basic set of atomic nodes have further to be abstracted and seen as sub-classes of a basic type process node. A second class of process nodes, the hierarchical nodes, which share many characteristics in common with the atomic nodes through their parent class, is defined.

Then a whole set of symbols (*pads*), which describe the interaction and synchronisation between the nodes, is defined and are represented as input and output controllers for the node to which they are attached. Four basic types of *pads* are allowed: *Signal/Interrupt, Send/Receive, Release/Acquire* and *Routing*. Figure 3 shows the symbols used for the hierarchical nodes and the pads, together with their meaning in the model representation.

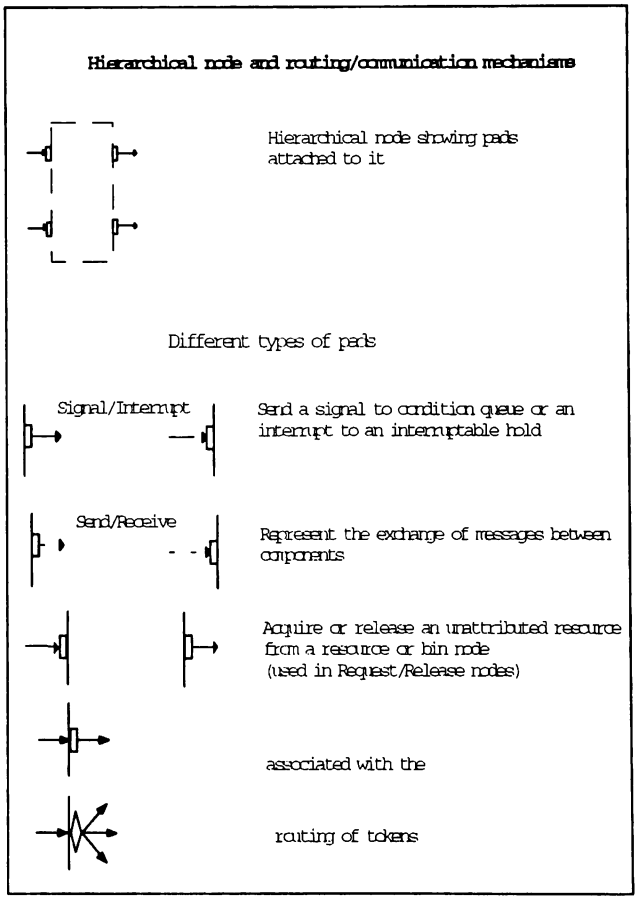

Figure 3: The hierarchical node and the synchronisation/communication pads

The atomic process nodes may also show pads attached to them, which are linked by arrows indicating the flow of information or material between them.

The flows of *tokens* and their routing mechanism through the use of *pads* are concepts neatly stated in the

new type of notation H-ACD as opposed to the X-ACD format.

The *tokens* represent signal, message or entity flows. Tokens may also represent new types of entities created at a node (material flow) or data structure (information flow) inside the model. The entities they represent may be added to other tokens in a *Assemble* node, losing their individuality in a similar way to the *Transform* process for one single entity.

The *pads* model an input or output controller for the node to which it is attached. When a node needs to route a token, it passes the token to one of its pads and asks the pad to route the token.

This separation of the processing behaviour (which belongs to the node) and the routing behaviour (which belongs to the pad), allows the inclusion and change of routing strategies without any further changes to the nodes.

3 THE PUB EXAMPLE

3.1 Problem Description

Customers arrive in the Pub with an exponentially distributed inter-arrival time with a mean of 10 minutes. Each customer is then assigned the number of drinks he/she wishes to consume before leaving. This number is distributed uniformly between 1 and 3 drinks. The customer then waits for a clean glass as well as the barmaid to be idle. If these conditions are met, the barmaid pours the drink into the glass and the customer starts to drink. Drinking is uniformly distributed between 5 and 8 minutes. Once the customer has finish his/her drink, the glass is put to the pile of dirty glasses waiting to be washed. If he/she has consumed the allocated number of drinks, the customer leaves the pub, otherwise another drink is ordered by waiting for both an empty glass as well as the barmaid to be available. The task of serving a customer takes precedence. The dirty glasses are taken back to the barmaid by a waiter who enters the bar every 30 minutes, collects all the empties, and places them on the bar top for the barmaid. He then retires to perform his other duties. The time taken to collect the glasses is proportional to the number of glasses using a constant of 0.2. The glasses in the dirty pool are washed if the barmaid is idle. Washing takes 5 minutes, after which the glass is put on the pile of clean glasses.

3.2 Model Analysis And Experience Gained Using The Approach

In this presentation we make use of all three type of diagrams, the standard ACD, the extended set of symbols proposed by Pooley (termed X-ACD), as well as the further modified version H-ACD in a step-by-step modelling process. Once the interface has been built there will be no provision for the user to do this stepwise modelling interactively. But he can do it as a separate modelling exercise, which proves very helpful to obtain a gradual improvement in the understanding of the system being modelled, as is demonstrated here.

Appendix A shows the Pub problem modelled using the original ACD graphical representation form. This type of diagram is the most conceptual of them all, showing the logical pattern of interaction of the entities and the conditions for the start of each activity. Being less 'cluttered' it allows many conceptual misunderstandings to be found more easily. It tends also to have one 'unique' solution for each model and it may be developed in sectors, although its biggest disadvantage soon become clear, with the size of the diagrams reached.

The second step is shown in Appendix B using Pooley's original notation. Some interesting aspects are highlighted during this transformation. First it should be noticed how we have taken the arbitrary decision to make all model elements entities (customer, barmaid, waiter, and glasses). A perfectly valid solution would be to have glass, customer and waiter as entities, and barmaid as a resource. In this second case we would still have to decide to which of these entities we would give the responsibility for activities *pour* and *drink*. This affects directly the relation 'who is coopted by whom' or, in other words, 'who is the master and who is the slave' in the cycles of the entities, as seen in the example for barmaid and customer. In this case the responsibility for activity *pour* was given to the barmaid and, as a result, customer is the one coopted by the barmaid, which led to the definition of *Barq* as a *Waitq*.

Two other important remarks need still to be made. The first is that it is not necessary for an entity to possess activities of its own, which is shown by glasses in the example. The other one is that resources are thought as non attrributable elements, what means that we could not model glasses as resources, because in this case we would have to distinguish between dirty and clean glasses through an attribute.

The definition of *idleq* as a *conditional queue* is made necessary because activity pour can only start after a complex set of conditions is met, and *conditional queue* is the modelling element used to describe this sort of situation. Entity glasses have their cycle consisting only of the three queues *cleanq*, *dirtyq* and *emptyq*. The queues *qready* and *qfull* need not be represented because they are dummy queues and the entities proceed to their

following activities without any delay after finishing activity *pour*. In the following we examine in detail the description of each entity's life cycle in turn.

The entity customer starts its activity by scheduling another arrival after a time interval sampled from an exponential distribution. The desired number of drinks is also sampled and the customer enters a loop to perform activities *pour* and *drink* for as many times as the desired number of drinks. But activity *pour* is not executed by entity customer directly, thus it is placed into a pool to wait to be coopted by entity barmaid before it can go on. After the customer is picked up by entity barmaid and finished *pour* 'on board' of it, it will continue its life cycle by entering *drink* (this is assured by the *schedule* mechanism in the barmaid class description). The customers recover their active life cycle to be subsequently held for the duration time of activity *drink*. They are released at the end of this time, but they are kept in a loop repeating its life cycle as many times as the number of desired drinks assessed at the beginning.

The barmaids are set initially sleeping in *idleq* until a certain condition is met and then they start their activities *pour* and *wash*, whereby *pour* was given priority by means of a branching test. To perform activity *pour* they coopt a customer and take a unit of resource glass from queue *cleanq*. If the main initial condition is met, but there is either no client waiting to be served or no glass in the queue *cleanq* then it's because a dirty glass is waiting to be washed in queue *dirtyq* and that activity is performed. As a result a clean glass is liberated to the resource pool *cleanq* and the barmaid restarts its cycle again.

The lifecycle of entity waiter starts by collecting the empty glasses from queue *emptyq*. The duration of this activity will be made proportional to the number of glasses by a constant of 0.2 time units for each empty glass collected (these details are hidden from the diagram). The waiter then proceeds immediately to its next activity *stayaway*, but before this he puts the collected glasses in queue *dirtyq* and signals the barmaid entity that it has work to do. Notice that this reactivation or warning signal triggers a check of the conditional expression in *Idleq*, acting thus differently to the reawakening mechanism seen before for the customer class (co-option by another class).

Entity glasses have a curious life cycle, since they are always coopted by the other types of entities to perform their common activities. Nevertheless their individuality is preserved by the fact that they are proper entities. They start in the *cleanq*, are coopted by customer to perform activity *Pour*, then they are released by customer after *Drink* and proceed to be *collected* by the waiter. He places them in queue *dirtyq*, from where they are again coopted by the barmaid to perform activity *Wash*, whenever the proper condition in the trigger becomes true.

The third step is shown in Appendix C. This step adds some fine details and the important concept of hierarchy. It terms of notation the diagrams do not change much. They change much more in terms of conceptualisation. Each of the atomic or hierarchical nodes may now be interpreted as an object to be implemented as such in an Object Oriented Programming Language (OOPL). It serves therefore to realise the practical consequences of some modelling decisions in the actual design of the object oriented code of the final simulation program.

As a general rule, the complexities derived from the begin or end of an activity are now shown together in the variety and sequencing of the input and output pads surrounding each icon.

One aspect not clearly enough stated in the model is the question of set handling in the case of the manipulation of entities which follow complex priority rules. This will be dealt with within the modelling elements definition, and added to all those elements which can be used to represent physical queues of the real system.

4 FURTHER REMARKS ON THE USE OF H-ACD FOR THE OBJECT ORIENTED MODELLING OF MANUFACTURING SYSTEMS

The general idea is to separate the purely structural aspects from those related to the behaviour for each object being modelled. It was mentioned before how the internal operation of a subcomponent can be associated with any sort of algorithmic description, even by writing a separate piece of program code to represent it. For each real object being modelled there is an entity object, which represents its static structure, and a machine/operation object, which represents its dynamic behaviour. The customer for example is the resultant of an attributed token and a dynamic description. This dynamic description is nevertheless not only algorithmic, as in most types of diagrams (including the previous X-ACD version). This is because they can be broken down into finer components, which can in its turn be seen as machines/operations acting on tokens. Each of these machines can thus be further detailed, either using the same type of atomic nodes or other types of similar machine/operation descriptions.

If a service order for a part in a manufacturing system is represented by a token and the part's production plan is known by the token, the nodes will

decide the correct routing for the part at any point in time based on its production plan, the actual position of the token and the model's state.

Although the diagram now proposed has started from rather generic descriptions of the component machines (objects), one could easily imagine how specific machine types could be implemented and saved for reuse. *Reusability* is brought into the modelling in a quite natural way, by the effective realisation of the hierarchy concept. This is so well perceived from the diagrams that one might wonder if this is not generally applicable to other types of software system's design.

The set of extensions suggested has the effect of making the diagrams more easily applicable for describing large models using a stepwise uniform modelling technique throughout the whole model life cycle. A model's comprehensiveness and visibility is no more a matter of a trade-off. With a powerful computer graphical interface it is easy to leave entire submodels as abstract components to be defined later. And if these modules are available from other applications the modelling process can be sped up enormously.

H-ACD and object orientedness seem to be the ultimate generalisation of the process world view. We no more need to talk of individual processes of particular entities, but of hierarchical processes of hierarchically decomposable components.

H-ACD might well be a small improvement in the notation of a diagramming technique, but it is also certainly a major achievement in the understanding of the simulation modelling process.

5 CONCLUSIONS

Many unexpected benefits in an understanding of the modelling process, design and implementation of simulation systems can be derived by the use of this notation. For example, the gradual modelling technique presented demonstrates the equivalence of models represented in the three types of notations. But, since this last notation can be implemented using a typical queuing network notation, of nodes, arcs and tokens, another subproduct of the modelling technique using H-ACD is to demonstrate the equivalence of queuing network systems representation with the ACD system representation. Maybe equivalencies with other types of simulation diagramming techniques, such as SLAM or Petri Nets could also be established, in a similar way as has been done for Petri Nets and SLAM diagrams (Taqi et al., 1992).

Another benefit is that many other types of misunderstandings concerning the use of object oriented programming to write simulation systems can be easily dismissed by the diagram notation. For instance, it has been stated that model specification does not benefit at all from the property of inheritance, that it forbids the use of encapsulation and that the communication only by means of messages would not allow for a proper model specification (Schmidt, 1990). A quick look at the H-ACD diagram shows how the model specification benefits from inheritance, since the basic class nodes can define many of the characteristics to be inherited by the subtypes hierarchical and atomic nodes. The complexity of each atomic process is handled through further decomposition and not by making them unique complex units of code.

The use of encapsulation and messages enhances the modelling process rather than places any sort of limitation on it. The encapsulation and message passing mechanism (by message one could understand any type of token flowing between the submodels or atomic nodes) is taken to its ultimate boundaries to allow a neat, well defined and unlimited (at least conceptually) decomposition of models. Furthermore, the mechanism of communication is done in an elaborated way using the input and output controllers for each node, which separates strategic routing decisions from the actual component behaviour. This is generally a highly desirable characteristic when creating models of manufacturing systems, which need to be submitted to a variety of configuration and strategic changes to assess the best machine composition or sequencing of operations for a production plan.

H-ACD graphical representation is introduced here as a tool for specifying manufacturing systems models to be implemented using an object oriented approach. The authors' view is nevertheless that they would be benefitial also for modelling of other types of discrete event systems to be implemented using other types of simulation environments SIMPLEX-II for instance).

To realise all the potentialities of the model building technique suggested by the diagrams it is nevertheless necessary that all further steps of implementation using a simulation system follow the modular approach reflected by its interface. So, if the individual models can be compiled and run separately, a further gain would be the immediate execution of the simulation runs and their animation in cases when pre-compiled components are assembled and defined using parametrisation.

ACKNOWLEDGEMENTS

This research is supported by CNPq, under reference no. 201082/90 - 0, and INPE, Instituto de Pesquisas Espaciais, Brazil.

APPENDIX A: PUB'S ACD DIAGRAM

Figure A.1 shows the standard Activity Cycle Diagram representation of the Pub.

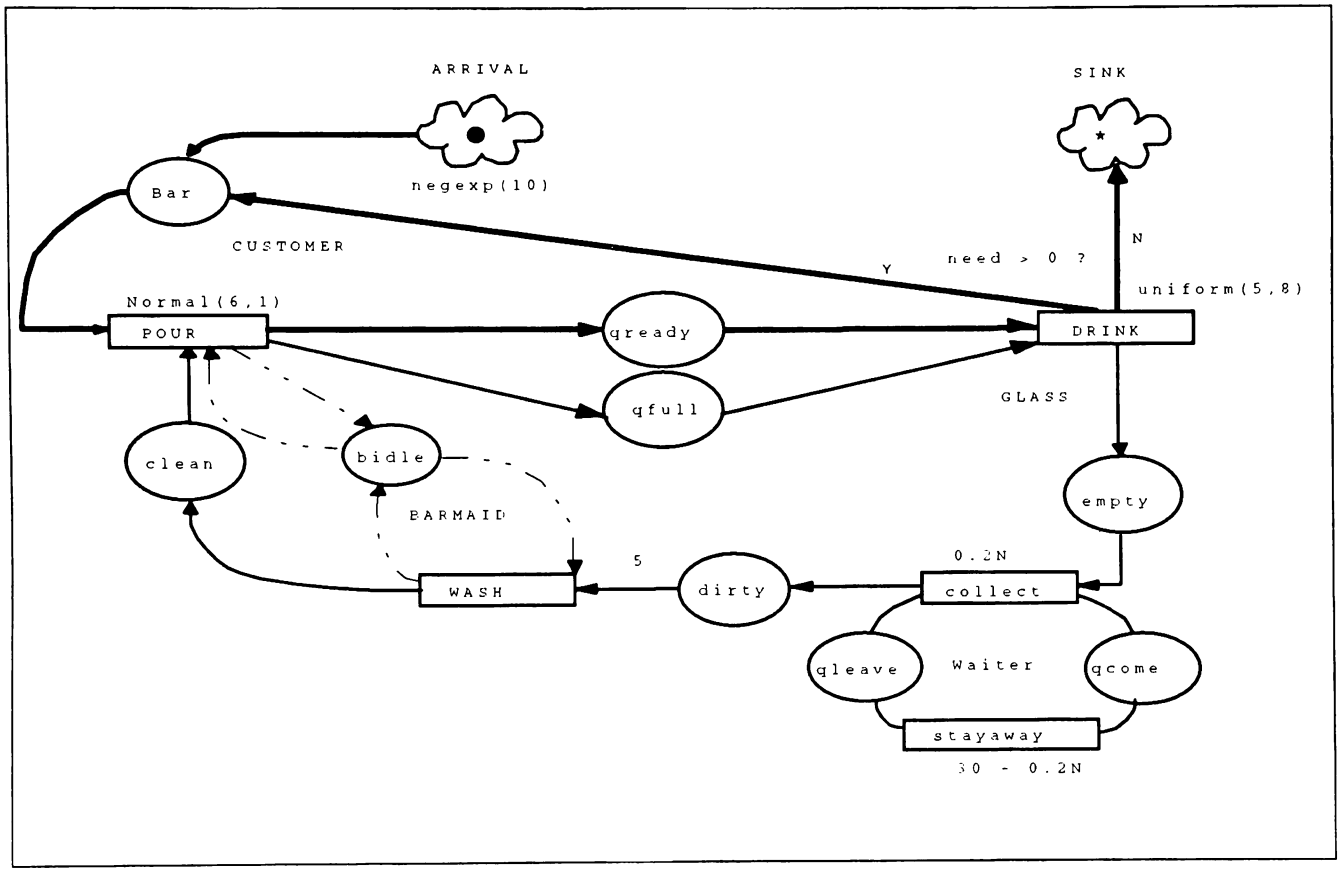

Figure A.1: Pub In Standard ACD Graphical Representation

APPENDIX B: PUB'S X-ACD DIAGRAM

Figure B.1 shows the eXtended Activity Cycle Diagram representation of the Pub problem.

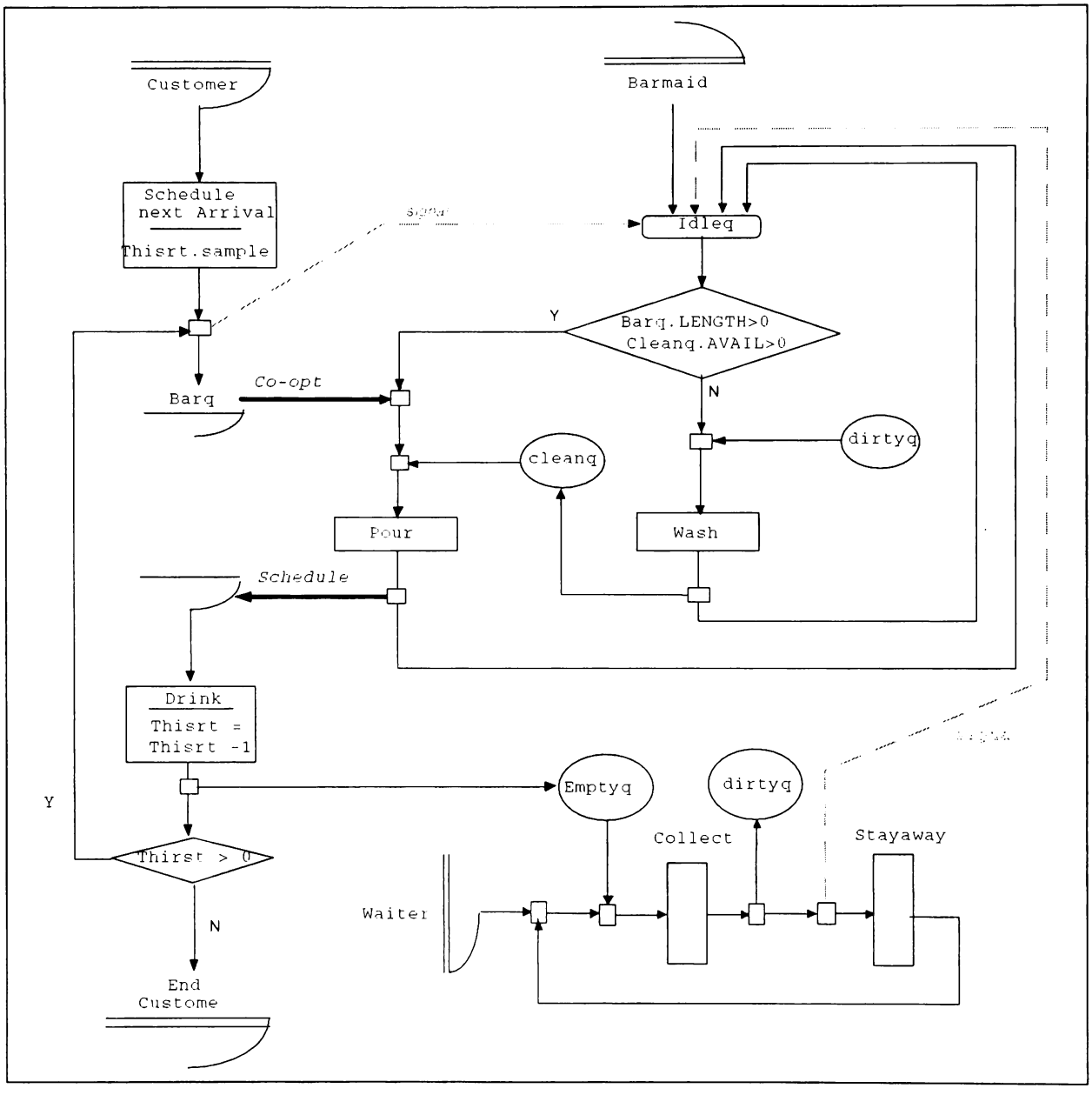

Figure B.1: Steelworks In X-ACD Representation Form

APPENDIX C: PUB'S H-ACD DIAGRAM

Figure C.1 shows the Hierarchical Activity Cycle Diagram representation of the Pub problem.

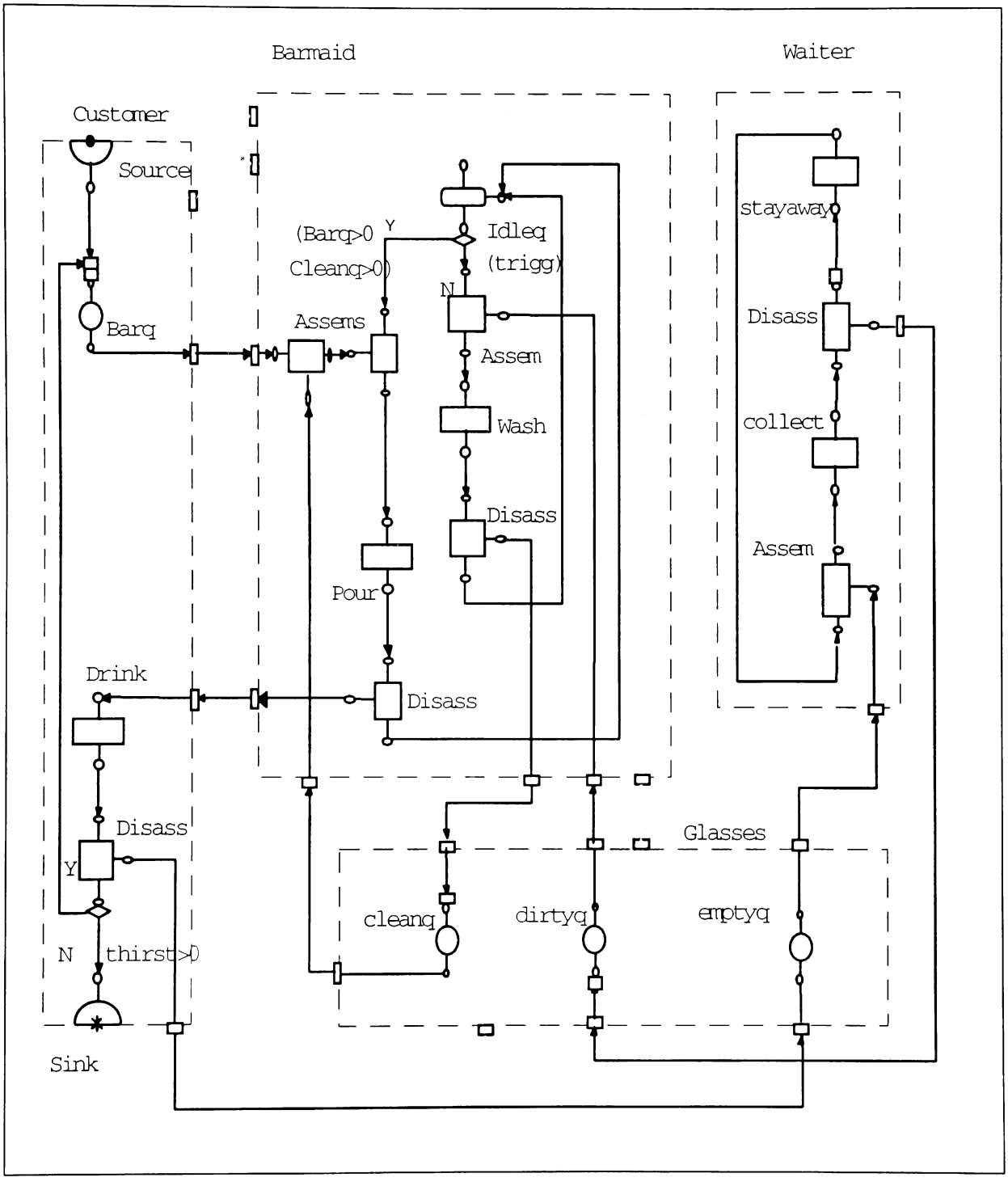

Figure C.1: Pub in H-ACD Representation Form

REFERENCES

Birtwistle, G. M. 1981. DEMOS Reference Manual. Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4.

Kienbaum, G. S., and R. J. Paul 1994. H-ACDNET: An Object-Oriented Graphical User Interface for Simulation Modelling of Manufacturing Systems. Department of Computer Science, Brunel University (Working paper).

Pidd, M. 1992. *Computer Simulation in Management Science*. 3d ed. Chichester: John Wiley & Sons.

Pooley, R. J. 1991. 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, 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 the Society for Computer Simulation,* **8**(1):21-31.

Schmidt, B. 1990. Simulation und Objektorientierte Programmierung. Universität Passau, Lehrstuhl für Operations Research.

Schmidt, B. 1991. Simulationsysteme der 5. Generation. Universität Passau, Lehrstuhl für Operations Research.

Taqi, A. A. Q; A. J. Al-Sammak, A. A. Khan, and N. Ahmed. 1992. A comparative study between Petri Nets and SLAM. *Simulation*, **59**(5):339-344.

Tocher, K. D. 1963. *The art of simulation*. Princeton: Van Norstrand.

AUTHOR BIOGRAPHIES

GERMANO S KIENBAUM is undertaking PhD research in the Department of Computer Science at Brunel University. He received his B.Sc. in Aeronautical Engineering from Instituto Tecnologico de Aeronautica (ITA), Brazil, and his M.Sc. in Operations Research from Instituto de Pesquisas Espaciais (INPE), Brazil. He is a researcher at the Associated Laboratory for Mathematics and Applied Computing at INPE, currently on leave of absence at the Department of Computer Science, Brunel University. His research interests are in the automatisation of the simulation modelling process and in the use of simulation to the manufacturing systems application area.

RAY J. PAUL holds the first U.K. Chair in Simulation Modelling at Brunel University after teaching Information System and Operational Research for 21 years at the London School of Economics. He received a B. Sc. in Mathematics, and a M. Sc. and Ph. D. in Operational Research from the Hull University. He has published widely in book and paper form (two books, over 70 papers in journals, books and conference proceedings), mainly in the area of simulation modelling process and in Software environments for simulation modelling. He acts as a consultant for a variety of U.K. government departments, software companies, and commercial companies in the tobacco and oil industries. His research interests are in methods of automating the process of discrete event simulation modelling, and the general aplicability of such methods and their extensions to the wide arena of information systems. Recent research results have been in automatic code generation, color graphics modelling interfaces, dynamically driven icon representations of simulation models, machine learning applied to model specification and to output analysis, object oriented approaches, and information systems paradigms. He is Head of the Computer Science Department at Brunel University. He has recently instituted the first M.Sc. in Simulation Modelling, a one year course starting in October each year at Brunel University.