Computer Simulation Modelling Using Hypercard

Grace Au[1] and Ray J. Paul[2]

[1] Department of Business Information Systems, The Hong Kong University of Science & Technology, Hong Kong
[2] Department of Computer Science, Brunel University, U.K.

Discrete event simulation modelling mimics a real world problem numerically in order that real world alternatives can be tested and evaluated. Such problems are often ill-defined, since their complexity is usually the source of the problem itself. There is a need, in such cases, for a flexible method of model specification. This paper addresses the issue of providing a rapidly adaptable environment for such modelling, so that the computer model can be easily, safely and quickly updated to meet current problem understanding. In particular, diagrammatic methods for expressing the model are discussed, and their implementation using HyperCard described. The research system developed is called HyperSim. Such a prototyping system enables rapid model development using either graphical or textual editing or both. This is illustrated using a popular simulation example. It is suggested that HyperSim should act as a front end to a complete simulation environment, and proposals to that end are outlined.

Introduction

Simulation model specification is one of the most important stages of the modelling process. Whilst computer based simulation modelling has a relatively extensive past, compared to the use of computers, there is a variety of proposed specification methods. Ceric and Paul (1992), for example, list fourteen such methods, with research still ongoing into what makes a good or poor method. Since model development is a function of specification, and some specifications are often under continuous review, it would appear that one characteristic desirable of a specification system is that it should be flexible to use. This paper describes a HyperCard implementation based on the graphical method known as activity cycle diagrams. HyperCard was selected because of its potential to offer a rich and robust environment for a flexible specification system.

This research work centres around a formal model description known as an *Activity Cycle Diagram* (ACD). An ACD represents, in a concise clear form, the flow of control within a simulation model. Since ACDs have proved to be a reasonable, if not comprehensive, method of describing the formal logic of the simulation model, we have developed an ACD based diagramming system called MacACD, described by Au & Paul (1993). It was found that the merits of ACDs diminish as the complexity of the model logic increases (see El Sheikh et.al. (1987) for example). An ideal simulation system would allow the user to combine graphical and textual specification of the model. This paper describes the results of research into such a flexible simulation specification environment using HyperCard on the Apple Macintosh.

In the next section, a brief description of the type of simulation modelling that this research is focused on is outlined. A definition and description of activity cycle diagrams is then discussed. The main section of the paper describes how research into a flexible graphical/textual based specification systems has been pursued by building a particular application. This application is called HyperSim, developed from HyperCard on the Apple Macintosh computer. An example of the way in which HyperSim achieves its task is illustrated in the appendix using the ubiquitous simulation example of a pub or bar. The detailed description of the example enables the reader to see the power and flexibility of HyperCard for development of such graphical/textual systems.

The penultimate section of the paper discusses the advantages the Macintosh HyperCard approach provides for this particular form of research. The paper concludes with the lessons learned from this research and discusses future research into graphical model specification and methods of achieving it.

Discrete Event Simulation modelling

Computer Simulation involves the careful planning of a model of a real world environment or system of interest. A computer is used as a means to develop an artificial or hypothetical model which imitates the systems behaviour when subject to a variety of operating policies. The computer model incorporates as many details as necessary to provide a realistic representation of the real world system. The computerised model allows the user to perform a variety of experiments, say, by changing certain conditions in the system, or by using different combinations of resources, or by applying different operating policies. In general, the computerised model acts as a vehicle for experimentation, often in a trial-and-error way, to demonstrate the likely effects of various conditions and policies. The results of the analysis may then be used to provide assistance for man-agement decisions.

A model based on a discrete system (as opposed to a continuous system) changes at specific points in time and is only concerned with state changes at these events. In this research, we are only concerned with discrete system modelling. Continuous systems are usually represented by differential equations or some equivalent approximation, and consequently they have an entirely different mode of solution.

In the modelling process, the formulation of the problem and the definition of the model logic can be specified by means of a flowchart, or an ACD (Pidd, 1992), or using special symbols as in GPSS (Shriber, 1991). The model specification can be translated into a text file via an interactive simulation program generator (ISPG). The program generator, making use of this data file, writes the simulation program using some software subsystems (Crookes et. al., 1986; Paul & Chew, 1987). This model, under the control of the analyst, is then ready to be run and output produced. Visual screens of the model can be created to represent the dynamics of the system during a visual simulation run (Bell 1991). The output can be used to determine correctness of the model logic, and of the computer program. Representational graphics (Bell and O'Keefe, 1987), for example histograms and time series, are often produced to emulate the simulation model output dynamically.

It has been an ideal since the 1970's to provide systems within which a problem owner would be able to build and use a model without the intervention of a specialist, or with the specialist having only light involvement (Crookes, 1992). Nevertheless, simulation is an ongoing process. Real world systems always involve participation from different groups of interests. Since the problem is user-defined, the end product is very much a compromise for these groups. Poor communications and different opinions between different groups often make the definition process more complicated. This fact is further emphasised in Paul (1992). There is therefore a need for flexible and easy-to-use systems for building prototypes in order to accelerate the model definition stage.

Simulation is often regarded as a decision-aiding tool. A less established role that today's simulation modelling plays is its power of helping the problem owner and the specialist to understand the problem and the system more thoroughly. It also helps to narrow the communication gaps between different parties of interests who are involved in the system since they are given a chance to understand what other components of the system do. Because of the complexity of the definition stage, a flexible specification system is necessary for allowing efficient updating of data in the model. It is the purpose of the research described in this paper to achieve such a flexible specification system.

This research is based upon the simulation environment as described by Balmer and Paul (1986) and Paul (1992). Figure 1 shows an overview of this environment.

The gap between an understanding of the model purpose by the problem-owner and the specialist is a common obstacle to a successful use of a simulation model. An attempt to bridge this gap is proposed by introducing Artificial Intelligence (AI) systems to aid the analyst in formulating the problem and experimenting with the model with

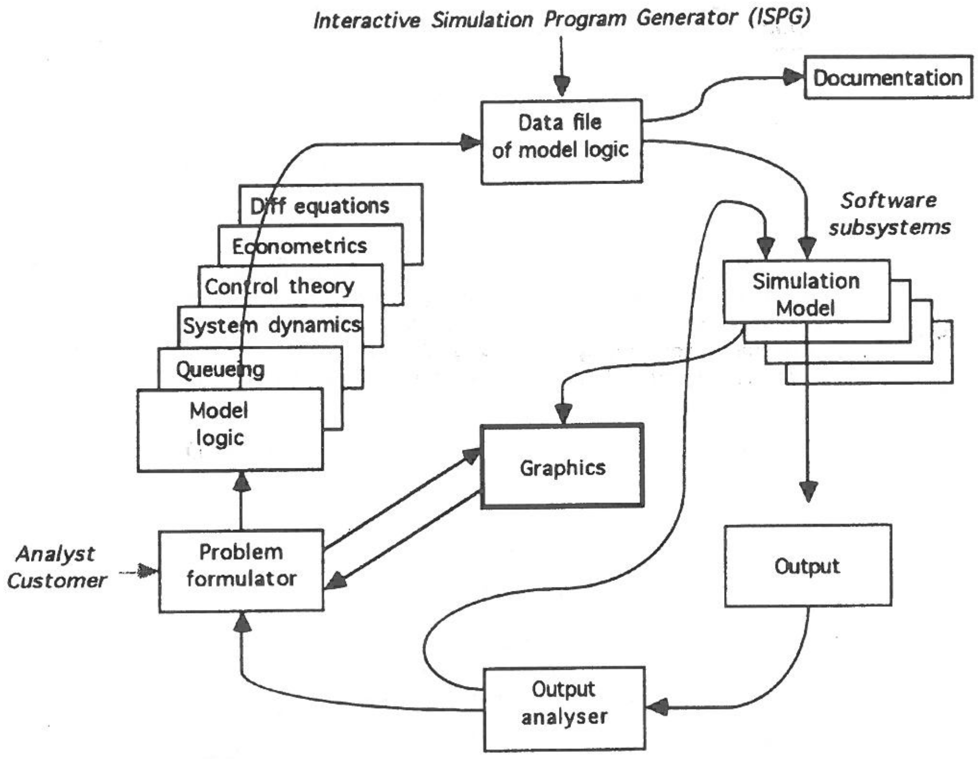


Figure 1: CASM's View of a Simulation Environment

the customer. This enables model development in small, easily checked stages, model correction in the light of program output, and determination of the running conditions and the run lengths of the simulation model. The main benefit is that the customer can also participate in the modelling process.

Graphics have historically been seen as a tool for emulating the simulation model output dynamically. However, graphics might be used in conjunction with a problem formulator. By means of a graphics screen, the problem can be described and thereby formulated, with the rest of the system driven as before. The simulation model would run the screen dynamically over time, with interrupt-and-amend capabilities. With graphics editing, the process of development, correction and obtaining model confidence is greatly enhanced. The use of graphics enables the end-user to see what is going on in the system and to respond to it dynamically, rather than to occasional events as presented by the analyst. In the following section, we will discuss a diagramming technique that is commonly used in formulating simulation models.

Activity Cycle Diagrams (ACD)

An activity cycle diagram (ACD) provides the means of describing the logic of a simulation model. It is a way of modelling the interactions of system objects and is particularly useful for systems with a strong queuing structure. The graphics represent the model in terms of the life cycles of the entities or objects it comprises. ACDs consist of activities (rectangles), queues (circles) and life cycles of entities (using arrows). A summary of the symbols used in an ACD is shown in figure 2.

An Example of ACD : The Pub

A simple pub or bar can be seen to comprise of the entities : customers, glasses, barmaids and a door. The way the entities behave in the pub can be represented by the ACD in figure 3. From this model logic, plus the duration of each activity, a program can be constructed and then run to simulate the behaviour of the pub over time. Parameters, such as the number of entities, may be varied. The behaviour and output of the model can then be compared to find, say, the maximum number of barmaids needed to maximise

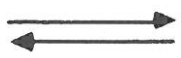
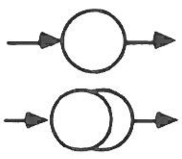
Summary of the symbols used in an Activity Cycle Diagram	
	An Entity is any component of the model which can be imagined to retain its identity through time. Entities are either idle in queues, or active, engaged with other entities in time consuming activities. Their life cycle is represented by arrows.
	An Activity usually involves the cooperation of different classes of entity. It is represented by a rectangle. The duration of an activity can be determined in advance, usually by sampling from a probability distribution for stochastic models.
	A Queue involves no cooperation between different classes of entity and is a state in which the entity waits for something to happen. The queuing time cannot be determined in advance as it depends on the conditions for the next activity to start. A queue is represent by a circle in an ACD. A Source/Sink queue, where a temporary entity is created and destroyed, is represented by two overlapping circles.

Figure 2: Summary of the symbols used in an ACD

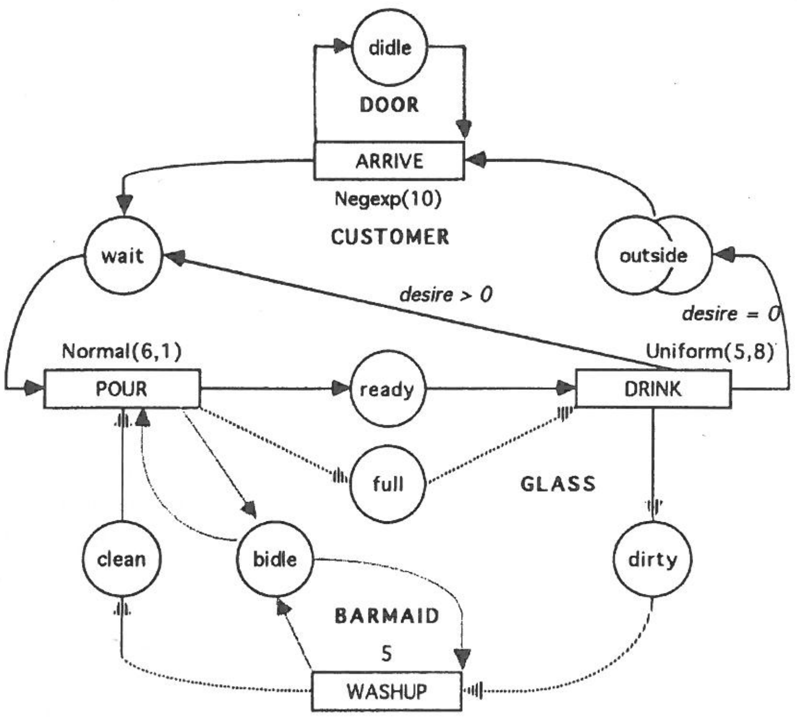


Figure 3: An Activity Cycle Diagram of the Pub

throughput. More extensive descriptions of ACDs are given in Pidd (1992) and Szymankiewicz et. al. (1988).

Hypercard & Hypersim

This section gives an introduction to HyperCard and HyperTalk (Apple, 1988; Shafer, 1988). An overview of the simulation system, HyperSim, and its system architecture are also described.

HyperCard and HyperTalk

HyperCard is a user-friendly object-ori-ented application generator and viewer. It may be described as an information organiser and its main benefits are power and simplicity. Written for the Macintosh and utilising its famous bit-mapped graphics capabilities, HyperCard organises data and activities around logical "Card Stacks". A card can contain textual, numeric and graphical data and also instructions (scripts) written in

HyperCard associated programming language - HyperTalk.

The fundamental concept of HyperCard is that each card in the same stack has similar structure and functions. Each card can contain card buttons, card fields and at least one background which can also contain background buttons or fields which are common to all cards in the stack. Programming HyperCard can be achieved by writing scripts in the object-like language HyperTalk. Objects that exist in HyperTalk are stacks, cards, backgrounds, buttons and fields; each of which can send and receive "messages". A script is associated with an object enabling it to respond in a specific manner to a message, depending on the instructions given by the user.

An Overview of HyperSim

HyperSim is a system which was built so as to allow constant redefinition of the model specification by text, graphics, or a mixture of both. Whichever method of input is used, both a graphical and a textual specification are held by the system. These descriptions are structured into a number of stacks. These stacks contain information concerning for example, the model entities or objects, the activities they engage in, the queues they rest in whilst waiting for an activity start, the assignment of entity attributes representing numerical, textual or logical characteristics of the entities, and icons used for visual display of the entities. HyperSim contains a stack which allows the user to specify the simulation model by drawing an activity cycle diagram. Each object in the diagram is linked with a specification card which the user can access by clicking on the object. Moreover, HyperSim allows the user to generate a simulation program based on a three-phase modelling structure from the specification given by the user. This program can then be modified, linked with the simulation library (called MacSim.Lib on the Macintosh) and run under Turbo Pascal on the Mac.

HyperSim contains a set of conventional buttons which are local to the system, appearing in different stacks. A summary of the name of the buttons is shown in figure 4.

The "New Entity" and the "Delete Entity" buttons are used for creating and deleting an entity type respectively. The "New Activity" and the "Delete Activity" button are used for creating and deleting an activity respectively. Other common buttons include the "GotoACD" button which takes the user to the ACD stack, the "GotoIcon" button which takes the user to the Icon stack, the "Help" button which takes the user to the "Reference" stack and the "Home" button which takes the user to the Home stack. These common buttons contribute to the flexibility of the HyperSim system, i.e. the user can do anything - create new data or edit old data of the model - at any time during the specification of the model.

System Architecture of HyperSim

HyperSim is made up of nine stacks. A summary of each of the stacks is given in this section. The relationship between these stacks is shown in figure 5.

1. Reference Stack is a tutorial stack designed to help the user to understand some of the basic principles of computer simulation modelling and to use the HyperSim system. The user can choose a topic he wants to view by clicking on the corresponding button.

2. *Model Stack* is the heart of the specification system which allows the user to create new models and edit old models. The user should go to this stack first and select a model that he would like to work on. There is a table showing the names of the entities, activities and queues the model possesses. The user can go to any entity, activity or queue by clicking at its name in the table. This stack also allows the user to go to the Icon stack and the ACD stack. The Code stack can only be accessed via this stack.

3. *ACD Stack* contains a graphical description of the model. The simulation model can be specified by means of an activity cycle diagram in this stack. An ACD card is automatically created by the sys-

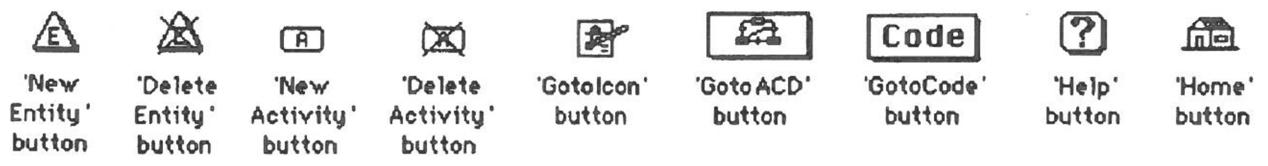


Figure 4: A Summary of the Buttons inside HyperSim

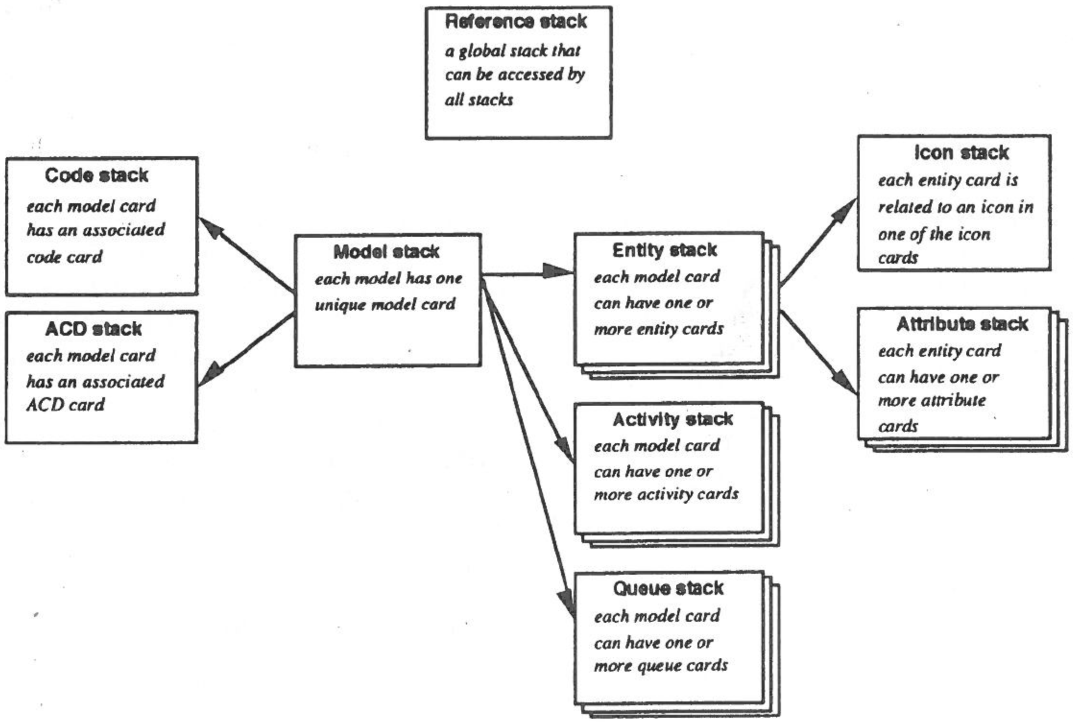

Reference stack
a global stack that can be accessed by all stacks
Code stack
each model card has an associated code card
ACD stack
each model card has an associated ACD card
Model stack
each model has one unique model card
Entity stack
each model card can have one or more entity cards
Activity stack
each model card can have one or more activity cards
Queue stack
each model card can have one or more queue cards
Icon stack
each entity card is related to an icon in one of the icon cards
Attribute stack
each entity card can have one or more attribute cards

Figure 5: Relationships between different stacks in HyperSim

tem whenever a new model is created. There is an iconic menu in this stack which helps the user to develop an ACD for the model and to input the data for the entities, activities and queues. Each object that is created on the screen has a card associated with it.

4. *Entity Stack* contains information about each entity type defined by the user - the name, the number, attributes involved, whether it has a source queue (for temporary entities) or a common queue (for facility entities), the queues involved, and the activities involved. It also shows the life cycle of the entity type and allows the user to add in any conditional paths during its life cycle. When a new model is created, a System entity is automatically created for the model. This system entity is a device for enabling the user to specify attributes for the system. These are global counters or variables that might be used throughout the model.

5. *Icon Stack* contains cards of icons and allows the user to create new or to edit old icons. An icon can be selected for an entity by clicking the "Select Icon" button and then by clicking anywhere inside a chosen icon rectangle. Once an icon is chosen for an entity, the icon will appear in the right hand corner of the corresponding entity card in the Entity stack.

6. *Activity Stack* contains information about each activity defined by the user - the name, the formula for the activity duration, the entities involved and the assignments involved in the activity.

7. *Queue Stack* contains information about each queue automated by HyperSim - the name, the entity type that it belongs to, the number of entities in the queue, the assignment of any attributes, and histograms that are recorded in the queue.

8. *Attribute Stack* contains information about each attribute defined by the user - the name, the entity type that it belongs to, the assignments of the attribute, and information about the histograms that are recorded for the attribute.

9. *Code Stack* allows the user to generate a three-phase simulation program from the specification of the model. The user can modify the generated program which is shown on the screen. The program can be exported as a text file by using the "Export" button. The "Turbo" button takes the user to the Turbo Pascal application where the program can be run when linked with the simulation library unit MacSim.Lib on the Macintosh.

Using HyperSim

HyperSim is a flexible visual interactive modelling system which allows the user to enter at any place to carry out any task. The appendix gives an example of how to use this system using the above Pub or bar example. The order of steps listed in the appendix is not the only way of specifying a model. The system allows the user to specify the model according to his own logical way of thinking.

Lessons Learned from Hypersim

This section outlines the lessons learned from using HyperSim in simulation modelling.

Advantages

HyperSim achieves its user-friendliness by making use of the Apple Macintosh WIMP (windows, icons, mouse and pull-down menu) interface and consistent authoring tools across different stacks. The use of bit-mapped graphics allows a marriage of graphics and text, and the ability to manipulate both on the same display. This gives tremendous flexibility in how that text is presented, in terms of size, style, and font design, and in mixing text with graphics. In addition, any of these elements can be changed and redisplayed on the screen countless times.

HyperSim supports the use of icons. Using icons is an ideal way of representing an entity on the screen. Because of its high resolution, the movement of the entity can be very smooth and well-presented. Applications which allow a visual simulation to be run on the screen are more appropriately developed in this type of environment.

HyperCard has always been well recognised as a powerful organiser tool. The idea behind HyperCard is simple - to allow the user to develop useful applications easily by creating buttons and fields in backgrounds or cards. HyperCard suggests a new way of file management. Instead of organising data in files, cards which contain similar information are grouped together in a stack. Moreover, stacks can be linked together so that large applications can be easily developed.

Mobility between stacks is facilitated by using buttons and fields. Different stacks can be linked together to allow easy access for the user just by a click of the button. New stacks can be continuously added to the application without extensive alterations to the rest of the stacks in the system. This allows large applications to be developed very easily. Good graphics facilities allows pictures to be drawn at a reasonable speed. The developer can make use of the graphics facilities available in HyperCard and is not required to write a set of graphical routines. HyperTalk programming is simple and easy to learn. Its object-oriented nature allows the user to develop applications and prototypes quickly.

Disadvantages

The main disadvantage of using HyperCard in developing applications like HyperSim is a delay when data is being updated between stacks. For such a system to be efficient, large amounts of memory space are required. A danger in developing applications using HyperCard is a tendency to create too many buttons on the screen which might become traps for the user. Therefore, a set of conventional buttons should be adapted throughout the design of the system, i.e. buttons that perform the same function should have the same appearance so as to avoid confusion and misunderstanding. HyperTalk programming is simple but it is easy to lose control. In order to maintain flexibility and mobility between different stacks, one has to keep track of global variables manually. Moreover, it does not support dynamic memory allocation, i.e. pointers or handles. For large models, code generation is relatively slower than program generators written in high-level languages like Pascal or C. HyperTalk is not an ideal programming language for building a simulation library that can be used as an interpreter. In order to use HyperSim as an interpreter for running a simulation model, we have to develop the simulation routines in a high level language and link these with HyperSim as external functions. The external functions can then be called from within the HyperSim application during a simulation run.

Future Research

Simulation models require extensive runs to determine the result of what is essentially a stochastic experiment. HyperCard can not provide the speed of execution required for such experiments. Hence, HyperSim is used as the basis for an automatic simulation program generator.

This generated HyperSim simulation program can be run in conjunction with the simulation library MacSim.Lib using Turbo Pascal on the Macintosh. Such a generated problem enables the desirable rapid processing of a simulation experiment. Storing, browsing and searching the contents of the output from a simulation model is laborious using current methods. Therefore, it is planned to return the experimental results to HyperSim to gain the advantage of a HyperText system for these purposes.

Conclusions

This paper has demonstrated two main points. The first concerns the ability to produce a flexible simulation model specification using the diagramming method of activity cycle diagrams. The second concerns the user of Hypercard in developing such simulation systems. Simulation graphics is an important part of simulation modelling, as can be seen with the fast growing popularity of Visual Interactive Modelling techniques (Bell & O'Keefe, 1987; Bell, 1991; Paul, 1989). Diagramming is a form of language which can provide for clear thinking and for human communication. Interactive diagramming on a computer screen has the advantages of speeding up the process greatly and enforcing standards. In particular, the ability to rapidly modify the model to represent the latest understanding of the problem enables the analyst to remain in touch with the customer. The computer system can apply many logic checks during creation. It can automate the documentation process. The computer system enforces discipline and permits cross-checking and validity checks that human beings do not apply consistently. Such a graphical interface can be used as the front end to an automatic program generator (Paul and Chew, 1987), and thereby a simulation modelling environment can be created. The reader is referred to Paul (1992) and to Au (1990) for examples of how this can be achieved.

The HyperSim application has demonstrated the power of modern graphical/textual based interfaces, as provided by the Apple Macintosh, for inputting graphical specifications for simulation models. Further work in this area has been undertaken (Au, 1990) and current research is looking at better diagramming techniques for specification. The aim is to produce a graphical interface that enables the analyst to formulate a clients problem with the client. This would be more than a specification system, which would be a sub-component of the analyst-client system. Such a graphical interface would have the versatility to enable, in a non-technical way, the problem to be described in the users terms. This would form the basis for the specification to be written in simulation terms.

References

APPLE Computer Inc. (1988), HyperCard Script Language Guide. Addison Wesley, USA.

AU G. (1990), A Graphics Driven Approach to Discrete Event Simulation. Unpublished Ph.D. Thesis, University of London, England.

AU G. and PAUL R.J. (1993). Graphical Simulation Model Specification using Activity Cycle Diagrams. Under revision for *Computers and Industrial Engineering* .

BALMER D.W. and PAUL R.J. (1986), CASM - The Right Environment for Simulation. *Journal of the Operational Research Society* **37** : 5, 443-452.

BELL P.C. (1991), Visual Interactive Modelling : "The Past, the Present, and the Prospects." European Journal of Operational Research **54** : 3, 274-286.

BELL P.C. and OKEEFE R.M. (1987), Visual Interactive Simulation - History, Recent Developments, and Major Issues. *Simulation* **49** : 3, 109-116.

CERIC V. and PAUL R.J. (1992), Diagrammatic Representations of the Conceptual Simulation Model for Discrete Event Systems. *Mathematics and Computers in Simulation,* Vol.34, Nos.3-4, pp.317-324.

CROOKES J.G. (1992), "Simulation in 1991 - Ten Years On", *European Journal of Operational Research* vol. **57** : 3, 305-308.

CROOKES J.G., BALMER D.W., CHEW S.T. and PAUL R.J. (1986), A Three Phase Simulation Systems written in Pascal. *Journal of the Operational Research Society* **37** : 6, 603-618.

EL SHEIKH A.A.R., PAUL R.J., HARDING A.S., BALMER D.W. (1987), A Microcomputer-Based Simulation Study of a Port. *Journal of the Operational Research Society* **38** : 8, 673-681.

HOLDER R.D. and GITTINS R.P. (1989), The Effects of Warship and Replenishment Ship Attrition on War Arsenal Requirements. *Journal of the Operational Research Society* **40** : 2, 155-166.

PAUL R.J. (1989), Visual Simulation : Seeing is Believing. In *Impacts of Recent Computer Advances on Operations Research*, Publications in Operations Research series Vol.9 (R.Sharda, B.L.Golden, E.Wasil, O.Balci, and W.Stewart, Eds.) North-Holland, New York. 422-432.

PAUL R.J. (1992), The Computer Aided Simulation Modeling Environment: An Overview. In the *Proceedings of the 1992 Winter Simulation Conference* . (J.J. Swain and D. Gainsman, Eds.). (13-16 December 1992, Arlington, Virginia). Society for Computer Simulation, San Diego.

PAUL R.J. and CHEW S.T. (1987), Simulation Modelling using an Interactive Simulation Program Generator. *Journal of the Operational Research Society* **38** : 8, 735-752.

PAUL R.J. and DOUKIDIS G.I. (1986), Further Developments in the Use of Artificial Intelligence Techniques which Formulate Simulation Problems. *Journal of the Operational Research Society* **37** : 8, 787-810.

PIDD M. (1992), Computer Simulation in Management Science, 3rd edn., John Wiley, Chichester.

SHAFER D. (1988), HyperTalk Programming. Hayden Books.

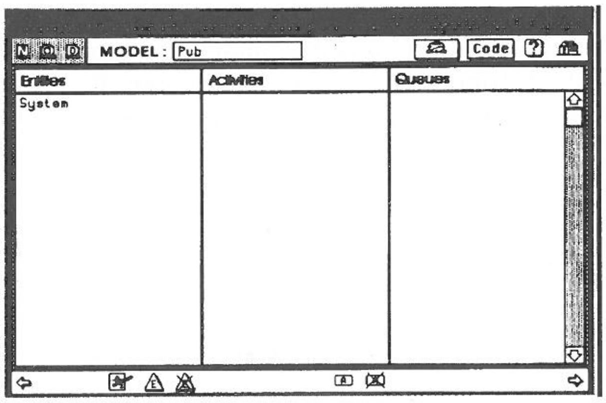
SHRIBER T. J. (1991), An Introduction to Simulation Using GPSS/H, John Wiley, New York.

SZYMANKIEWICZ J., MCDONALD J. and TURNER K. (1988), Solving Business Problems by Simulation. 2nd edn., McGraw-Hill, London.

WILLIAMS T.M., GITTINS R.P. and BURKE D.M. (1989), Replenishment at Sea. *Journal of the Operational Research Society* **40** : 10, 881-887.

APPENDIX

The first stack that the user should go to is the Model stack (figure 6) where a new model can be created or an existing model can be selected.


MODEL: Pub

Entities	Activities	Queues
System		

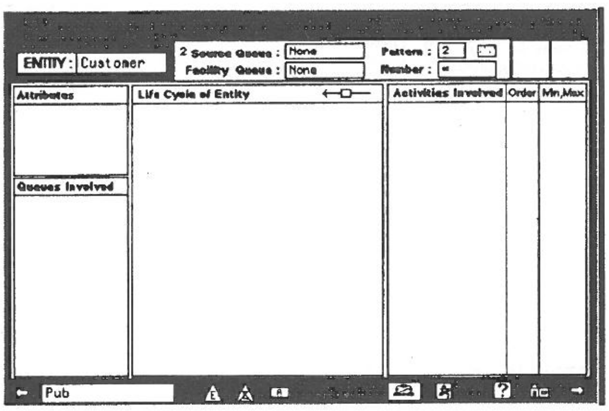
Figure 6: The New Model Card for the Pub

Creating a New Model

A new model can be created by clicking at the "**N**" button on the left-hand corner of the card. After the user types in the model name, a new model card will be added to the Model stack. Moreover, an ACD card in the ACD stack and a system entity card in the Entity stack will be automatically generated for the new model. An existing model can be opened by clicking the "**O**" button and then selecting a model by clicking at the name of the model in the model list table. The "**D**" button is used for deleting an existing model.

Creating Entities

An entity type can be created anywhere in the system by using the "New Entity" button. A new card in the Entity stack will be created where the user can enter any relevant information about the entity type. Figure 7 shows the Entity card when the entity type "Customer" is created.


ENTITY: Customer
2 Source Queue: None
Facility Queue: None
Pattern: 2
Number: ∞

Attributes | Life Cycle of Entity | Activities Involved | Order | Min,Max
Queues Involved

Pub

Figure 7: Creating an Entity type "Customer" in the Pub Model

In the Entity stack, the user can create attributes (see the following section *Creating and Assigning Attributes*), select an icon or create a source/sink or facility queue for an entity type in the Entity stack.

Since "Customer" is a temporary entity in the Pub model, we can create a source/sink queue for this entity type. Figure 8 shows the source/sink queue for the entity type "Customer". Similarly, a facility queue can be created for a facility entity, for example, "Door".

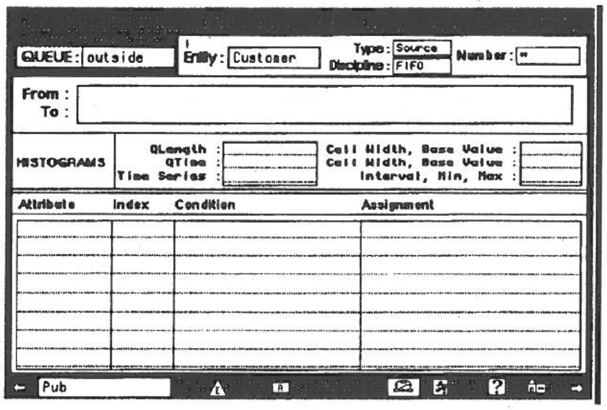

QUEUE: outside
Entity: Customer
Type: Source
Discipline: FIFO
Number: ∞

Figure 8: The Source/Sink queue "outside" for entity "Customer"

The top-right rectangle in an entity card is reserved for placing an icon in the Entity stack from the Icon stack. By clicking once anywhere inside the rectangle (the "Select Icon" button), HyperSim will take the user to the icon library - the Icon stack. The user can either create a new icon or edit an existing icon in any one of the 135 rectangles in an icon card. Figure 9 shows the entity card for "Customer" after an icon is created.

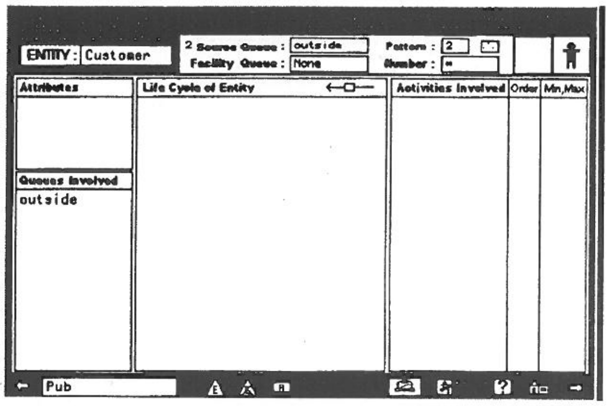

ENTITY: Customer
2 Source Queue : outside
Facility Queue : None
Pattern : 2
Number : ∞
Queues Involved: outside

Figure 9: The Entity Card "Customer" after an icon is selected

Creating Activities

An activity can be created anywhere in the system by using the "New Activity" button. A new card in the Activity Stack will be created where the user can enter any relevant information about the new activity. Figure 10 shows an example of the activity card. Only if the user creates an activity in the ACD stack will HyperSim go back to the ACD stack after an activity is created. This enables the user to draw the ACD before entering information for an activity.

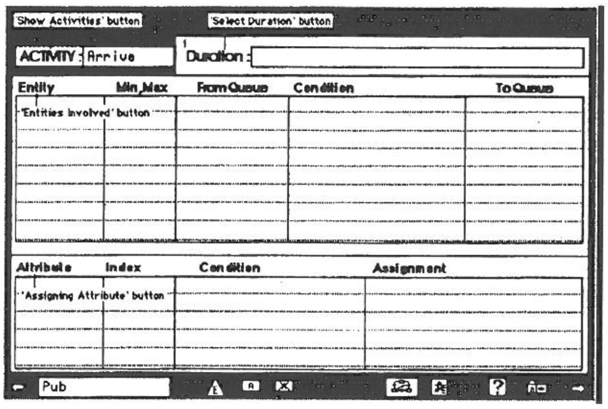

ACTIVITY: Arrive

Figure 10: An Activity Card

The formula for the activity time can be entered in the "Duration" field. If the user clicks at the word "Duration", a table containing all the available distribution formulae will appear. The user can select any one of them by clicking at its appropriate line in the table and then enter the required parameters. Attributes can be evaluated in the Activity stack (see the section *Creating and Assigning Attributes* below).

Drawing Life Cycles of Entities

HyperSim offers the user three ways of specifying the model logic. Firstly, by entering the life cycle of individual entity types in the Entity stack. Secondly, by selecting entity types that are involved in an activity in the Activity stack, and finally by drawing an activity cycle diagram in the ACD stack. Queues are automated in the first and the last method whereas in the second method, the user has to enter the names of the queues manually. Future development of HyperSim is working towards total flexibility among these three methods of specification, i.e. allowing the user to specify a model by any combination of the three methods described. Meanwhile, any one method used to define the model logic will automatically update the corresponding information in different stacks.

In this section, we assume that the following entity types for the Pub model have been created - System, Customer, Door, Barmaid and Glass. We also assume that activities - Arrive, Pour, Drink and WashUp have been created.

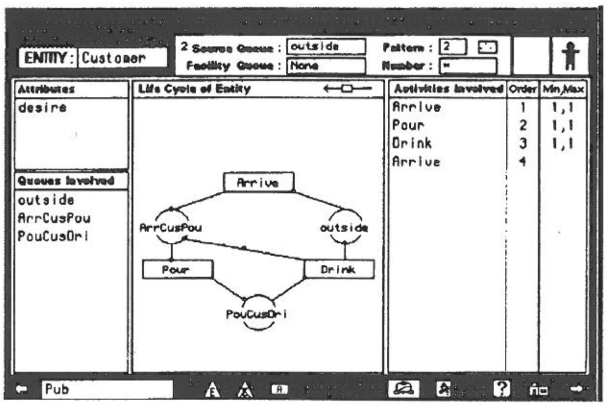
(i) Using the Entity Stack

This is the simplest method of all, but is only appropriate for a simple simulation model where the life cycle of individual entity types can easily be identified. This method does not allow the user to see an overall view of the model during the construction of the model logic.

For example, to define the life cycle of entity type "Customer", click once at the word "Activities Involved" in the "Activities Involved" table and select "Arrive" from the activity list table. The word "Arrive" will then appear on the first line of the "Activities Involved" table with "1" and "1,1" in the same line under field "Order" (order of the activity) and "Min,Max" (the minimum and maximum number of entities of that entity type required for the activity) respectively. Similarly, activity "Pour" can be added to the "Activities Involved" table. When activity "Pour" is added, a queue "ArrCusPou" connecting "Arrive" and "Pour" is automatically created for "Customer". This queue is added to the "Queues Involved" table in the entity card for the customer. Finally, when activity "Drink" is added, a queue "PouCusDri" connecting "Pour" and "Drink" is created. To indicate that the cycle is complete, select "Arrive" again. The user can then click at the "Entity Life Cycle" button to see a life cycle diagram for the "Customer". The conditional path for a customer to go to activity "Pour" after "Drink" can be specified by selecting the "Conditional Arrow" button (an arrow with a box in the centre). Figure 11 shows the completed entity card for entity type "Customer".

The life cycle for a facility entity can be defined in a similar manner except that the user does not have to close the cycle by selecting the starting activity again. All he has to do is to select the activities that the facility entity is involved in and, when finished, click the "Entity Life Cycle" button.

Information on this specification of life cycles of entities is automatically updated in the "Entities Involved" table in the Activity stack. The user can then, if desired, re-specify the conditions in the "Condition" field of a path in the appropriate row.



ENTITY : Customer

2 Source Queue : outside
Facility Queue : None

Pattern : 2
Number : -

Attributes
desire

Queues Involved
outside
ArrCusPou
PouCusDri

Activities Involved	Order	Min,Max
Arrive	1	1,1
Pour	2	1,1
Drink	3	1,1
Arrive	4	

Figure 11: The Entity card "Customer"

(ii) Using the Activity Stack

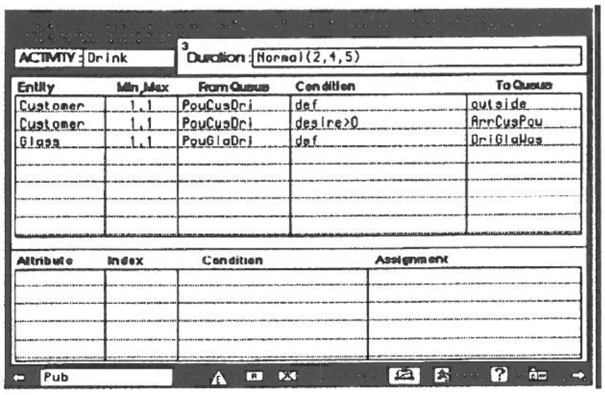
This method allows the user to define the model logic by selecting the entities that are involved in each activity. Again, this method does not allow the user to obtain an overview of the model during the construction of the model logic.

For example, to define activity "Arrive", click on the word "Entities Involved". The user can then select the entity type "Customer" from the entity list table and enter the name of the queue where the customer comes from (queue in) before "Arrive" begins, and the name of the queue where the customer goes to (queue out) after commencing "Arrive". The user can continue to select the facility entity type "Door" for activity "Arrive".

In order to assist the user in keeping track of the queues associated with an activity for an entity type, the default queue for an activity is set to be the queue out from the previous activity which the entity is involved in. For example, if activity "Pour" is defined next, and entity "Customer" is selected, then the default queue in would be "wait". Figure 12 shows the completed activity card for activity "Drink".

(iii) Using the ACD Stack

This method allows the user to build up the model logic by drawing an activity cycle diagram. There are three types of objects that can be drawn inside the drawing area - rectangles (activities), circles (queues) or straight lines (life paths of entities).



ACTIVITY : Drink

3 Duration : Normal(2,4,5)

Entity	Min,Max	From Queue	Condition	To Queue
Customer	1,1	PouCusDri	def	outside
Customer	1,1	PouCusDri	desire>0	ArrCusPou
Glass	1,1	PouGlaDri	def	DriGlaWas

Attribute	Index	Condition	Assignment

Pub

Figure 12: The Activity card "Drink"

Figure 13 shows the complete ACD card for the Pub model.

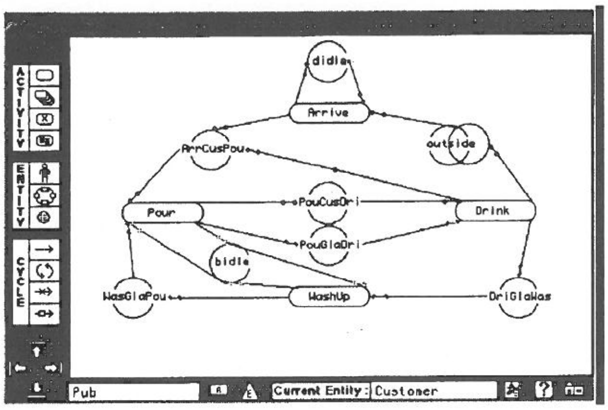


Figure 13: The ACD card for the Pub Model

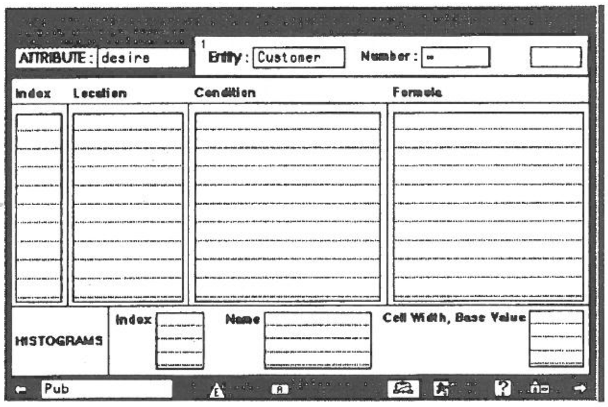
The user can either select an activity to be placed (using the "Select Display Activity" button) or display all the activities (using the "Display All Activities" button) in the drawing area. Objects that have been placed can be moved around in the window.

Each of the displayed activity rectangles is linked with an activity card in the Activity stack. The user can go to the appropriate activity card to enter information by first clicking at the "Activity Information" button and then by selecting the desired activity. Similarly, each of the displayed queue circles is linked with a queue card in the Queue stack. The user can go to the appropriate queue card to enter information by first clicking at the "Queue Information" button and then by selecting the desired queue.

Before drawing life paths for an entity, an entity type must be selected by using the "Select Entity" button. Drawing a life cycle for a facility entity is simple. For example, to define the life cycle for the entity "Barmaid", select the "Facility Path" button. First click anywhere in the drawing area where you want the facility queue for "Barmaid", "bidle", to be placed. The queue "bidle" will then be drawn automatically. If however the user has forgotten to define a facility queue for "Barmaid" and has selected the "Facility Path" button, he will be prompted for the name of the facility queue. If he types "bidle" and clicks "OK", a queue card called "bidle" will be created. The user can then select activity "Pour". Two paths - one going from "bidle" to "Pour" and one from "Pour" to "bidle" are then drawn. Since "Barmaid" is also involved in activity "WashUp", select the "Facility Path" button again. This time click anywhere inside the queue "bidle" circle and then select activity "WashUp". Similarly, the paths are automatically drawn.

Creating and Assigning Attributes

An attribute can only be created inside the Entity stack by clicking at the word "Attribute" at the top of the attribute table in an entity card. An attribute card will be created in the Attribute stack. Figure 14 shows the "desire" attribute card for the entity type "Customer".



ATTRIBUTE : desire

1 Entity : Customer Number : -

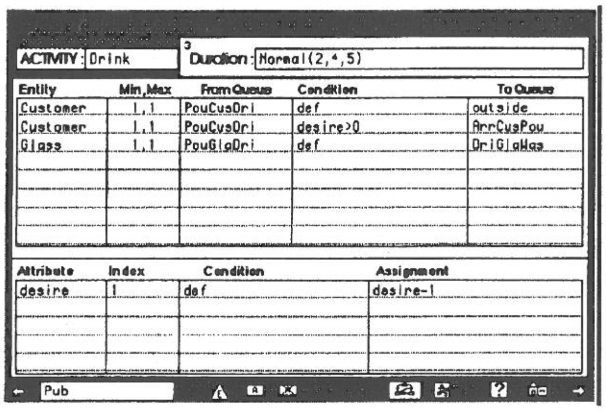
Index	Location	Condition	Formula

HISTOGRAMS Index Name Cell Width, Base Value

Pub

Figure 14: The Attribute card "desire" for entity "Customer"

An attribute can be evaluated either in an activity or in a queue. To assign an attribute in an activity, go to the appropriate activity card. Click once at the word "Attribute" and select an attribute from the attribute list table by clicking at its name. The user will then be prompted for the index value of the attribute (this is used to order multiple attribute assignments). After typing in the value, click "OK". HyperSim will set the condition for the assignment to "def", i.e. default, and the user can type in a formula for the assignment. Figure 15 shows the activity card for "Drink" after the attribute "desire" is assigned. Similarly, attributes can be assigned in the same way in a queue.



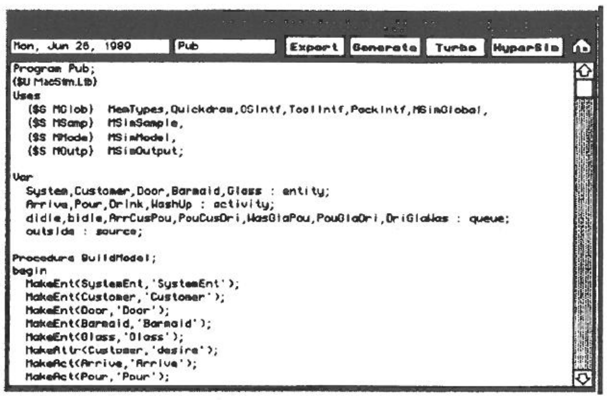
ACTIVITY: Drink   3 Duration: Normal(2,4,5)

Entity	Min,Max	From Queue	Condition	To Queue
Customer	1,1	PouCusDri	def	outside
Customer	1,1	PouCvsDri	desire>0	ArrCusPou
Glass	1,1	PouGlaDri	def	DriGlaWas

Attribute	Index	Condition	Assignment
desire	1	def	desire-1

Figure 15: Assignment of Attribute "desire" in Activity "Drink"

be achieved by editing the program text in the card. The user can either export the generated code as a text file or go directly into the Turbo Pascal program. Figure 16 shows the generated code for the pub example.



```
Program Pub;
($U MacSim.Lib)
Uses
  ($G MGlob)  MemTypes,Quickdraw,OSIntf,ToolIntf,PackIntf,MSimGlobal,
  ($S MSamp)  MSimSample,
  ($S MMode)  MSimModel,
  ($S MOutp)  MSimOutput;

Var
  System,Customer,Door,Barmaid,Glass : entity;
  Arrive,Pour,Drink,WashUp : activity;
  didle,bidle,ArrCusPou,PouCusDri,WasGlaPou,PouGlaDri,DriGlaWas : queue;
  outside : source;

Procedure BuildModel;
begin
  MakeEnt(SystemEnt,'SystemEnt');
  MakeEnt(Customer,'Customer');
  MakeEnt(Door,'Door');
  MakeEnt(Barmaid,'Barmaid');
  MakeEnt(Glass,'Glass');
  MakeAttr(Customer,'desire');
  MakeAct(Arrive,'Arrive');
  MakeAct(Pour,'Pour');
```

Figure 16: The Code Stack

Generating a Simulation Program

After the specification is completed, the user can return to the model card "Pub" and select the "Code" button. The system will take the user to the Code stack where a simulation program can be generated. The built-in program generator then translates the data file into lines of code. The program will be shown in a scrollable text field on the card. Modifications to the code can

Received: August 1, 1992
Accepted: February 25, 1993

Address for correspondence:
Professor Ray J. Paul
Professor of Simulation Modelling
Deputy Head of Department
Department of Computer Science
Brunel University
Middlesex UB8 3PH
U.K.
phone: 0895-274000
fax: 0895-232806

Grace Au received an undergraduate degree in management sciences, a masters degree in operational research and a Ph.D. in information systems from the London School of Economics (LSE). She is currently a Lecturer in the Department of Business Information Systems at the Hong Kong University of Science and Technology.

Ray Paul is Professor in Simulation Modelling at Brunel University. Previously he taught operational research and information systems at the London School of Economics. He has published more than three dozen papers and research articles in various international journals. Professor Paul is a consultant to several UK government departments and many private companies. He has many international research contacts, in particular with Brazil, Croatia, Hong Kong and the U.S.A. He is co-author of two books, Simulation Modelling, and Artificial Intelligence in Operational Research.