A GRAPHICS DRIVEN APPROACH TO DISCRETE EVENT SIMULATION

GRACE AU

The London School of Economics

Thesis submitted in fulfillment of the requirement

for the degree of Doctor of Philosophy at

the London School of Economics,

University of London.

April 1990

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

ABSTRACT

This thesis investigates the potential of computer graphics in providing for a graphics driven specification system that gives sufficient structure and content to form the simulation model itself. The nature of discrete event simulation modelling, the diagramming method of activity cycle diagrams which underpinned this research, the three phase simulation model structure, and the trend of visual simulation modelling are discussed as the basis for the research. Some current existing simulation languages and packages are reviewed, which gives insight into the essential features of an ideal computer simulation environment.

The basic research method adopted was to build systems that exemplified the state of thinking at the time. The purpose of this method was to enable ideas to be developed, discarded and enhanced, and for new ideas to emerge. The research has undergone a series of application developments on the Apple Macintosh to examine the advantages and limitations of such systems.

The first system developed during the research, MacACD, provides the basis for proposals concerning the enhancement of the ACD diagramming method in a computer-aided environment. However, MacACD demonstrated the limitations of an ACD interface and the need for a more flexible specification system. HyperSim, a simulation system developed using Hypercard, has all the power of interconnectivity demonstrated as a need by MacACD, but has severe limitations both in terms of security of system development, and an inability to provide a running model directly due to lack of speed. However, the power of an icon-based interconnected textual and diagrammatic based system were demonstrated by the construction of this system during this research, and led to the development of the final system described in this thesis : MacGraSE. The development of this system during this research incorporates many innovations. The main input device is a picture representing the problem, including a background display. This system allows for dynamic icon based visual model running, as well as code generation for complete model embellishments, interactive report writing, and representational graphics outputs.

ACKNOWLEDGEMENTS

I would like to thank *Dr. Ray Paul*

for his excellent supervision and constant encouragement

throughout this research.

I would like to thank *Lizza Domingo* and *Jean Mak*

for their wonderful friendship and loving care.

I would like to thank *my parents,*

whose love is the force behind everything I work for.

Thank God for all the blessings and gifts.

Thanks for being there to watch over me.

CONTENTS

CHAPTER 1

INTRODUCTION

" Simulation is a two-edged sword. It can cut both ways. In the wrong hands it can do great damage. How can we assure that it is used only for good ? "

The Society for Computer Simulation Journal

Simulation is a broad subject. In this research, we are only concerned with dynamic systems, i.e. those that change with time. And the type of simulation we are interested in is *discrete-event simulation modelling* (Adelsberger et. al. 87, Pidd 84, Pidd 88). A model based on a discrete system changes at specific points in time and is only concerned with state changes at these events. Discrete systems are the opposite of continuous systems. The latter systems are usually represented by differential equations or some equivalent approximation, and consequently they have an entirely different mode of solution.

Simulation is becoming more and more widely used in the area of operational research and will continue to be one of the most useful techniques for helping analysts and users understand their system better and to make decisions for improving their system. Here is an abstract of a simulation article by McLeod (88) :

> On 11 April 1970, Apollo 13 was launched. At first the mission went flawlessly. Injection into moon orbit took place as planned. Then, on the evening of the third day, there was a loud "bang." Working with a preflight simulation, the NASA ground crew not only diagnosed the problem as an explosion of one of two oxygen storage tanks in the service module, but devised and transmitted to the flight crew instructions as to how to cope with the situation and make a safe return and landing on the earth !

Simulation can be defined as the use of models as surrogates for the study of existing or hypothesized systems. It is the process of building up an artificial model which resembles the real world system. This model is then used to perform experiments to find out the relationships between the different factors of production and the system output. For example, a simulation model can be built for a steelworks to investigate the number of each type of resource which is required to increase the throughput of the system.

This is an introductory chapter to the thesis. The nature of computer simulation modelling and the process of simulation are discussed in section 1.1. Much of this research is focused on a diagramming technique called *Activity Cycle Diagrams*. Its methodology is discussed in section 1.2. In section 1.3, we look at different approaches to modelling structure, with a detailed description of the three phase method in particular, and explain why we have chosen this method in our research. Visual interactive simulation modelling (an essential component of this research) is discussed in section 1.4. The research objectives and the outline of this thesis are given in sections 1.5 and 1.6 respectively.

1.1. SIMULATION MODELLING

Simulation modelling is a powerful method for modelling problems, and one which would be more widely used if it were cheaper and easier to use (Balmer & Paul 86). Simulation is a powerful tool for understanding, which is a prerequisite for solving any kind of problem, and a catalyst by which new policy rules have been articulated. Understanding is often necessary in order to know what the problem is.

There are many ways to build up a simulation model. Since a popular tool for studying dynamic systems is by using computer models, we are only concerned with computer simulation modelling (section 1.1.1) in this research. The general process of

simulation is discussed in section 1.1.2 and a more detailed life cycle of a simulation study is described in section 1.1.3.

1.1.1. Computer Simulation Modelling

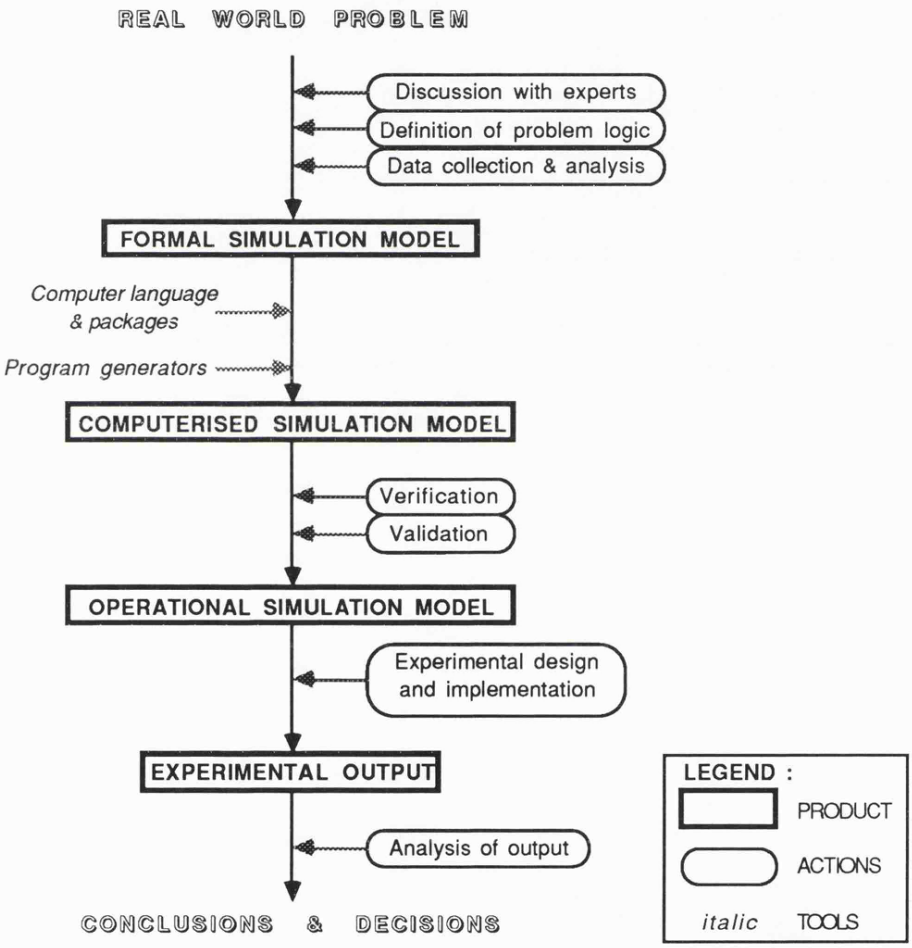
Computer Simulation involves the careful planning of a model of a real world environment or system of interest, for example, a steelworks (Paul & Balmer 89) or a port (El Sheikh 87). A computer is used as a vehicle to develop an artificial or hypothetical model which imitates the system's behaviour when subject to a variety of operating policies. The computerized model incorporates as much detail as necessary so as to provide a realistic representation of the real world system. This model also 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 computerized 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 management decisions. The nature of simulation modelling is further discussed in Paul & Balmer (89), Pidd (84, 88), and Szymankiewicz et. al. (88).

Digital computers are now fast enough to simulate most real systems in real time, and computational results can be made accurate to any desired degree. Modern digital computers and software are excellent for handling communications with the modeller, with the customer, and with other digital systems, local or remote. Icons can be pictured on the screen of a digital work-station and moved with a pointing device across the screen to construct a pictorial representation of the simulation model. Then the computer will program itself to simulate the real world system. To communicate with the customer who wants to study the operation of a system, a diagram of a job-shop, a traffic intersection, or other kind of system can be dynamically displayed to show what is going on with the simulation picture in real time or faster, or slower.

1.1.2. The Process of Computer Simulation

The process of computer simulation modelling includes at least three phases : Modelling, Programming and Experimentation (Paul & Balmer 89, Pidd 88). Figure 1.1 presents the general simulation process.

Figure 1.1. The Process of Simulation



The Modelling Phase

This phase can be subdivided into two stages - building up an informal model and building up a formal model.

An informal model formulation requires the analyst to discuss with the customer and the system experts the different aspects of the system, and to examine

written accounts of historic or proposed system operation. It is important that the analyst should understand how the real world system operates and the requirements of the model. Statistical analysis of numerical data on the system and some further investigation of the system may be involved.

A formal model is the use of a standard notation and a set of rules by the analyst to depict the problem logic in a systematic way. The formulation of the problem and the definition of the model logic can be specified by means of flow charts, an Activity Cycle Diagram (Paul & Balmer 89, Pidd 88), an Activity Diagram (Davies & O'Keefe 89) or using special symbols as in GPSS (Shriber 74). In this research, we are mainly concerned with activity cycle diagrams, which are further discussed in section 1.2. Attempts have been made to use expert systems in the simulation formulation process (Doukidis 85, Doukidis & Paul 86).

The Programming Phase

The formal model is used as the backbone for a simulation program to be developed. This phase is sometimes accelerated by the use of generic software systems, for example PROPHET (Manufacturing Management Ltd.) or HOCUS (Szymankiewicz et. al. 88), which takes data defining the user's particular problem as the basis for running the model. Code generators can also be used in this phase (Chew 86, Paul & Chew 87). After verification and validation tests (Balmer 85a) on the computerized model, the operational simulation model is ready for the next phase.

The Experimentation Phase

In this phase, experiments are carried out to discover, reinforce or quantity appropriate management action, subject to the needs of the system users. The output

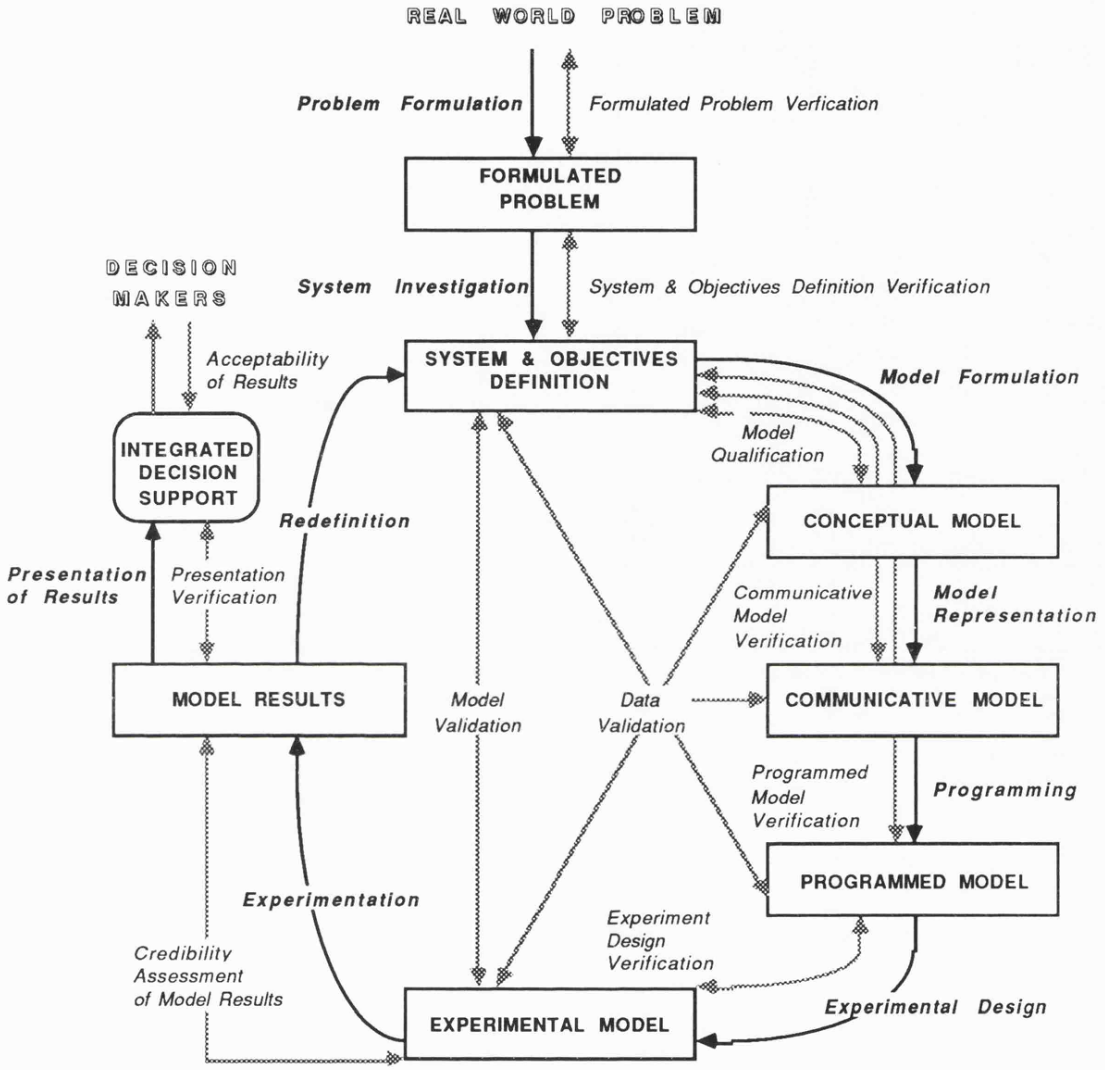
can be used to determine correctness of the model logic, and of the computer program. Representational graphics, for example, histograms and time series charts, are often produced to emulate the simulation model output dynamically. Careful planning of experimental design is required in this phase (Balmer 86, Balmer 87b).

The simulation process described is applicable to systems where the problem and the model logic can be easily identified. Recent research and practice in simulation modelling suggests that as the complexity of the model increases, more steps have to be taken and the above process seems to be too simple and "one-way". Simulation is an on going process. Real world systems always involve the participation of 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 makes the definition process more complicated. This fact is further emphasised in Paul (88c). A more detailed simulation study is discussed in the next section.

1.1.3. A Detailed Life Cycle of a Simulation Study

Simulation is often regarded as a decision-aiding tool. A less established role that today's simulation modelling plays is its power in helping the customer and the analyst understand the problem and the system more thoroughly. It also helps to narrow the communication gap between different parties of interest 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 and logic in the model. The aim of our research is to design and implement such a flexible system. A diagram of the life cycle of a simulation study which originated from Balci (86) is shown in figure 1.2.

Figure 1.2. The Life Cycle of a Simulation Study (Balci 86)



For large and complex systems, it is often the case that an agreement on the specification of the model must be obtained from different groups of interests that are involved in the real world system. New information about the model logic may be explored. Existing information may be found to be incorrect during the definition process or the actual building up of the computer model. Hence, an immediate path between the specification model and the computer model is desirable for an efficient simulation system. Constant redefinition of the model can be vital to the analyst and the customer.

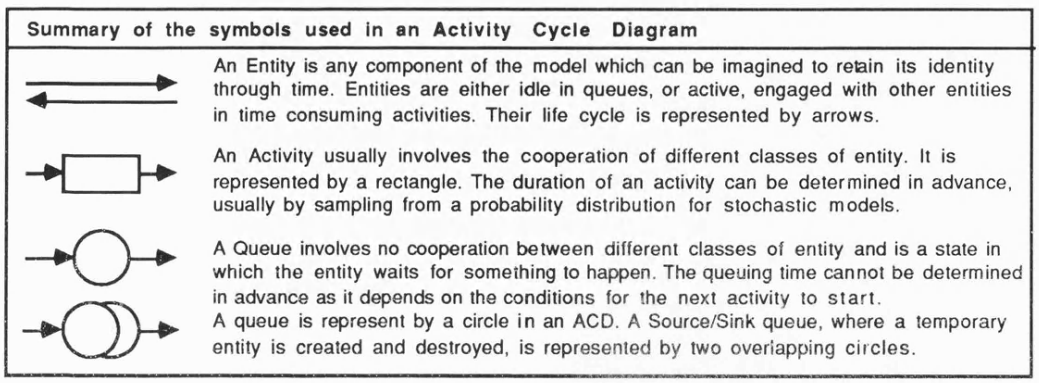
1.2. ACTIVITY CYCLE DIAGRAMS

Activity Cycle Diagrams (ACDs) are presented here, as they are an important component of this research. ACDs are a popular diagramming technique used in specifying a simulation model. An ACD provides the means of describing the logic of a simulation model. The methodology of ACDs is given in section 1.2.1 with examples of the pub model, the launderette, the steelworks model and the port model described in sections 1.2.2 to 1.2.5 respectively. The advantages and the disadvantages of the ACD method are discussed in section 1.2.6.

1.2.1. The ACD Methodology

A formal representation of a simulation model can be specified by an ACD, which is a 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 1.3. More extensive descriptions of ACDs are given in Pidd (88) and Szymankiewicz et. al. (88).

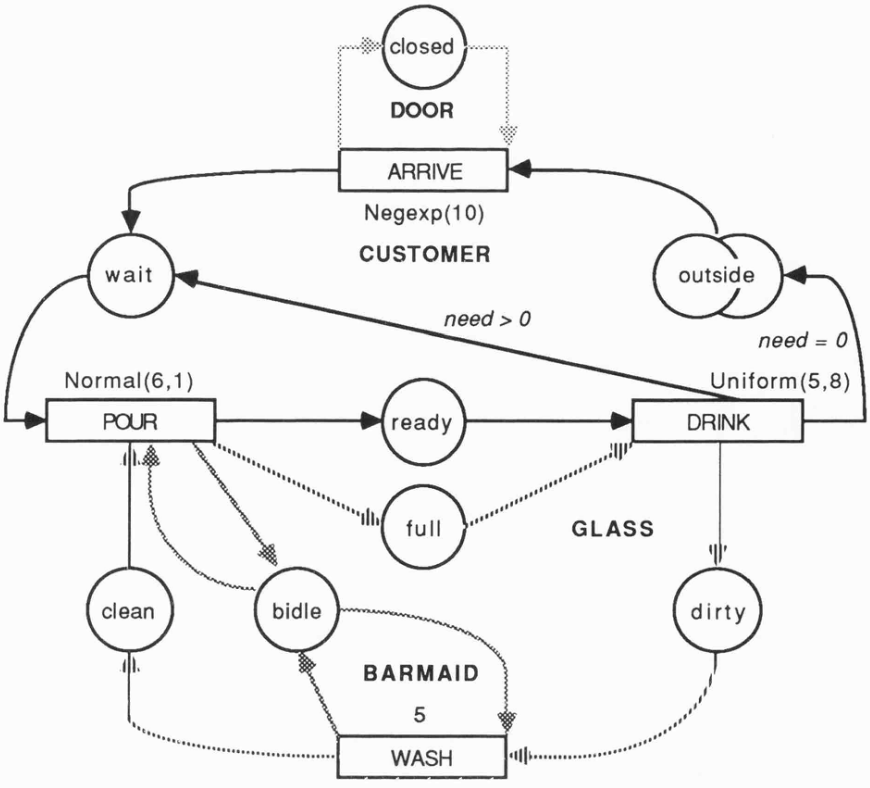
Figure 1.3. Summary of the symbols used in an ACD



1.2.2. The Pub Example

A pub comprises the following entities : customers, glasses, barmaids and a door. A customer enters the pub through the door (activity 'Arrive') and waits for a barmaid to pour a drink for him with a clean glass (activity 'Pour'). After he finishes his drink (activity 'Drink'), he either leaves the pub or waits for another drink to be served, depending on his desire for drinks. This desire for drink (need) is an *attribute* belonging to the customer. The barmaid either serves a customer with a clean glass (activity 'Pour') or cleans a dirty glass (activity 'Wash'). The way the entities behave in the pub is shown by the ACD in figure 1.4. From this logic model, plus the duration of each activity, a program can be constructed and 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 maximize throughput.

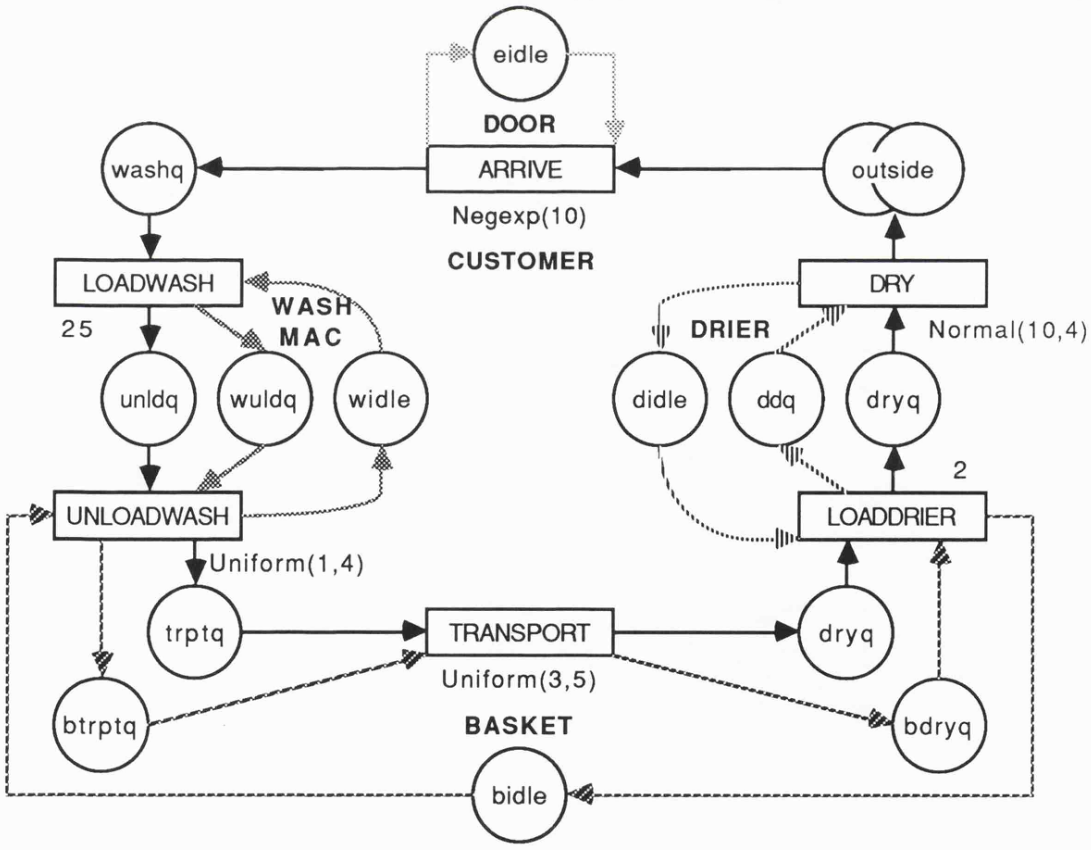
Figure 1.4. Activity Cycle Diagram of the Pub



1.2.3. The Launderette Example

The launderette comprises of the following entities : customers, washing machines, driers and baskets. After the customer arrives in the launderette (activity 'Arrive'), he starts washing the clothes when there is a washing machine available (activity 'LoadWash'). After washing, the customer unloads the machines contents into a basket (activity 'UnloadWash'). He then transports the basket to the drier (activity 'Transport') and waits to load the clothes into the drier (activity 'LoadDrier'). The customer must wait for the drier to do its job (activity 'Dry') and then leave the launderette. A typical objective of the simulation is to measure the average time the customer spends in the launderette. Figure 1.5 shows the ACD of the launderette.

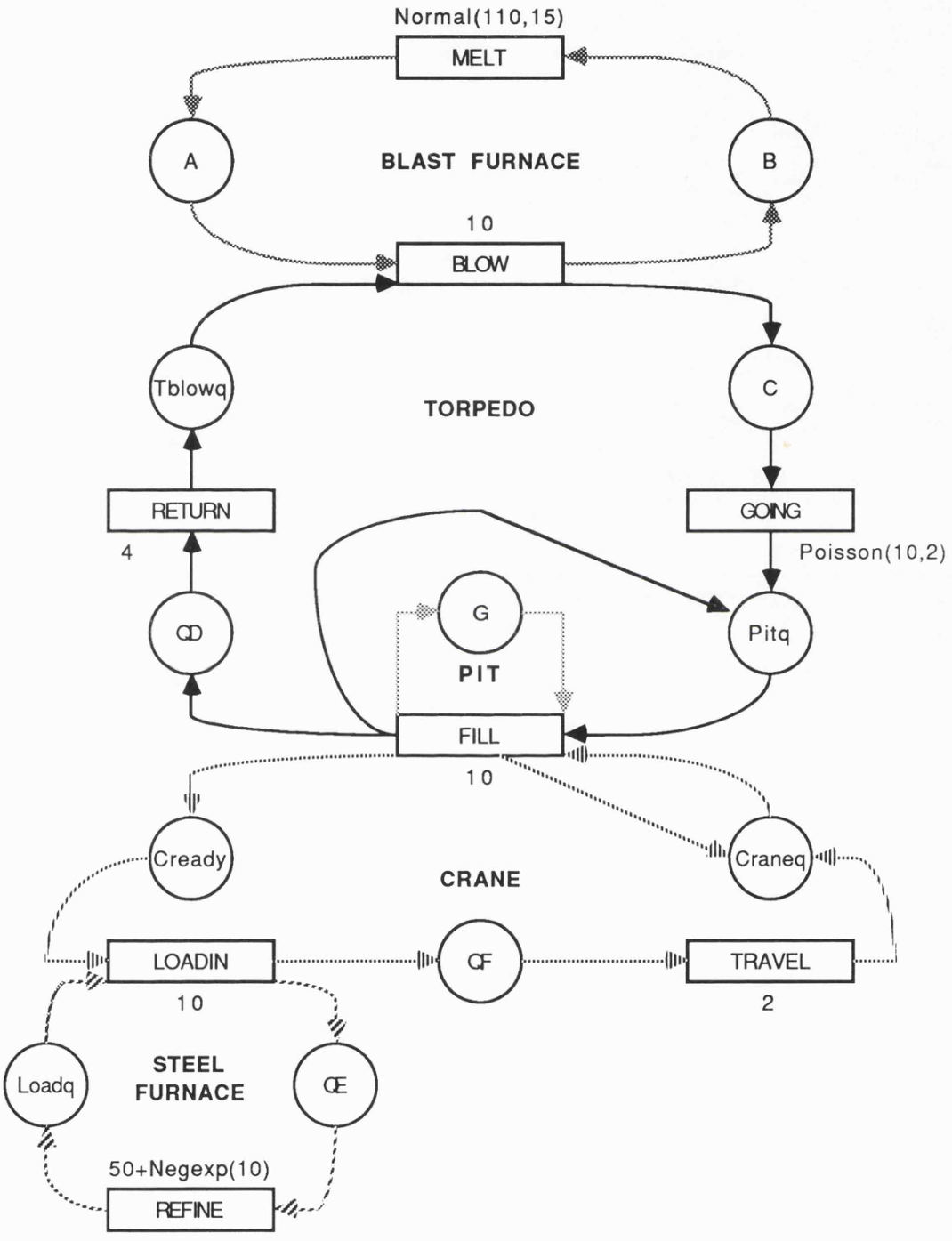
Figure 1.5. Activity Cycle Diagram of the Launderette



1.2.4. The Steelworks Example

The steelworks comprises the following entities : blast furnaces, torpedoes, cranes, steel furnaces and a pit. The steelworks commences with the blast furnace producing molten iron (activity 'Melt'). A 'cast' is the amount of molten iron it produces and is sampled from a normal distribution. The blast furnace then empties its cast into the minimum number of torpedoes required (activity 'Blow'). Each torpedo can hold up to 300 tonnes of molten iron. If the number of torpedoes cannot accommodate the full amount of cast produced during activity 'Blow', the remainder is considered as 'waste'. This waste is accumulated during the modelling process and its minimisation provides an objective for the analyst. After activity 'Blow', the blast furnace begins activity 'Melt'. All torpedoes containing molten iron, including partially full torpedoes, then proceed to the queue 'pitq'. It takes 4 minutes for a return journey (activity 'Return'). When a torpedo is in the queue 'pitq', a gantry crane's ladle may be filled from one torpedo at a time (activity 'Fill'). The crane can hold up to 100 tonnes of molten iron. If the contents of a torpedo ('tcast') are not emptied after filling, then the torpedo returns to the front of the queue 'pitq' and waits for another fill. The torpedoes then start their return journey (activity 'Return'). After 100 tonnes of molten iron have been loaded into the crane, it proceeds to the queue 'cready' and when a steel furnace is available, it transfers its load ('ccast') into the steel furnace (activity 'Loadin'). 'Travel' is the activity when the crane travels back to the pit. The ACD of the steelworks is shown in figure 1.6.

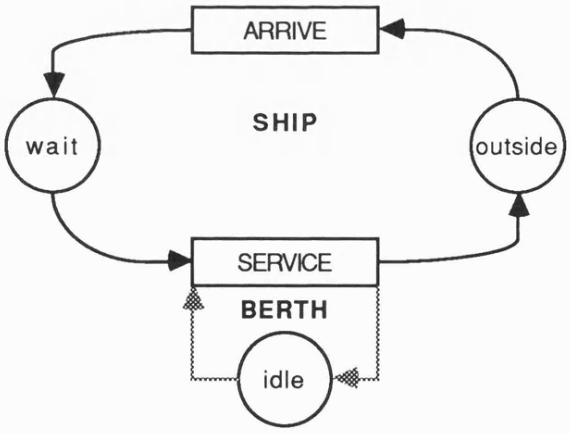
Figure 1.6. Activity Cycle Diagram of the Steelworks



1.2.5. The Port Example

The port (El. Sheikh et. al. 87) is comprised of the following entities : ships and berths. The port considered has a total length of ship-berth space which can accommodate between 20-25 ships at any one time, depending on the ships' lengths. Part of this berth space is dedicated to a number of single categories of cargo such as coal, iron ore, etc. In addition to general cargo ships, there were five other clearly identifiable categories of ships that loaded at the port, and 21 categories of ships that discharged at the port. The arrival-time and service-time patterns for the various ship categories are different and the ships are clearly restricted to one or two berths owing to their cargo. Many ships categories can only use a subset of the berths. Within this subset of berths, some ship categories may have preferences for some berths over others if there is a choice. The ACD of the port is shown in figure 1.7. The objective of the simulation is to estimate the number of berths required in the short and medium term, and to examine the impact of proposed handling improvements.

Figure 1.7. Activity Cycle Diagram of the Port



1.2.6. Advantages and Disadvantages of ACDs

The main advantage of ACDs lies in its simple structure which allows the analyst to depict the model logic of the system of interest. The technique can be easily understood by the user of the system. In the pub and the launderette examples, the ACD is an ideal tool to show all the model details.

However, it can be seen from the above examples that the ACD is not capable of handling all the model details in a structured way. In the steelworks example, the ACD cannot depict the conditional assignments of attribute values. As a model's complexity increases, the ACD becomes more limited in providing a full description. In the port example, the ACD cannot show the matching of ships and berths in the activity 'Service'. The inadequacy of the ACD method has led us to investigate other diagramming techniques which can be used to specify the model. Chapter 3 of this thesis summarizes this investigation and suggests the use of a computer-aided ACD so as to increase the depth of model specification that an ACD can achieve.

1.3. SIMULATION MODELLING STRUCTURE

There are three major approaches to simulation modelling structure (Paul 89a, Pidd 88) - the process flow method, event-based method and the three phase method. The following sections contain a brief description of the process flow method and the event-based method. The three phase method is described in detail since it is the modelling structure that has been adopted for this research.

1.3.1. The Process Flow Method

This method has characteristics that are similar to object oriented

programming. The program code emulates the flow of an object through the system. This flow describes in sequence all the states the object can be in the system. Each of the processes in the system is programmed as a separate and independent routine in program code and the processes communicate through an executive which controls their execution. The process structure of SIMULA (Birtwhistle 79) was the original implementation. Later versions of SIMSCRIPT (CACI 83) also have this 'Process' feature.

The main advantage of the process flow method is its simple structure so that a non-technical user can easily understand it. However, it requires careful thinking by the analyst about the model structure so as to get the interrupts and delays correctly registered. Model amendments also tend to affect all parts of the system, which tends to make such amendments slow and expensive.

1.3.2. The Event-based Method

This method is more widely used in the U.S.A. than in the U.K. Time is advanced to when something next happens and activities are examined to see if any can now start as a consequence. Mathewson (89a) describes the procedure as to : 'Identify the next event and complete all changes that are dependent on that event. Review the subset of activities who use resources released by that event, and where possible schedule future events using the newly available resources.'.

Computer efficiency (in terms of CPU time) is the main advantage of this method. However, a problem occurs when several events end at the same time. This method allocates the resources that are released by each event in turn before one knows what other resources are going to be released by the other events that occur at that time. Moreover, complications occur in the model structure when there is some priority in allocation of resources within the system. As for the process flow method, model

amendments tend to affect all parts of the system.

1.3.3. The Three Phase Method

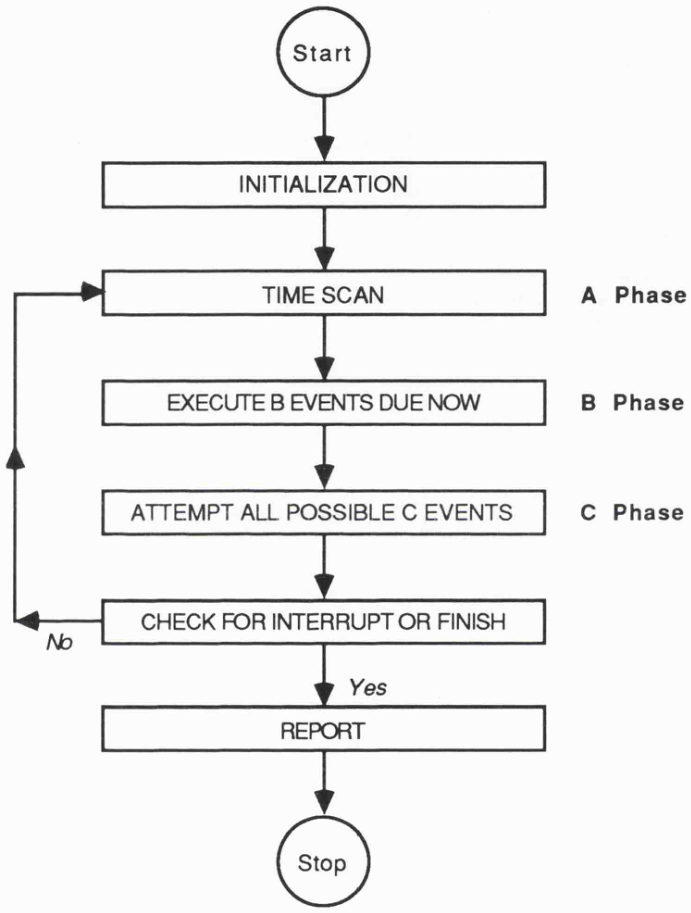
The first phase is time advance. Time is advanced until there is a state change in the system or until something next happens. At this time the system is examined to find out all the events that take place at this time. Hence, the second phase is to release those resources scheduled to end their activities at this time. Only when all resources that are due to be released at this time have been released, is the reallocation of these resources into new activities started in the third phase of the simulation, i.e. to start activities given the global picture about the resource availability. A three phase executive is shown in figure 1.8.

The three phase approach is the most commonly used in Britain. This method is further discussed in Crookes et. al. (87), Paul (89a), Pidd (84), and Pidd (88). The advantage of this method is that it gives maximum control of the model, the experimental tool for simulation, to the analyst. Decisions as to priority over resource allocations are more readily made within this structure. Moreover, it offers a well-defined and transparent program structure, and is particularly robust to changing specifications. These advantages of the three phase method are the reasons that we have adopted this method in our research instead of the other two methods. On the other hand, the disadvantage of this method is that in computing terms it can be slightly inefficient to run. Every time there is a time advance, resources are released and then one searches through all possible activity starts. However, with recent technology, this problem can be overcome by using more advanced computer hardware. Software solutions have also been proposed (Spinelli de Carvahlo & Crookes 76). The accuracy of the modelling structure is more important in the building of a good simulation model.

In this research, we are using a three-phase structure simulation library called

MacSim.Lib (see Appendix B), written in Turbo Pascal on the Apple Macintosh. A three phase skeleton simulation program and the program listings of the three example models discussed in sections 1.2.2 to 1.2.4 are shown in Appendix A of this thesis.

Figure 1.8. A Three Phase Executive



1.4. VISUAL INTERACTIVE SIMULATION

Visual Interactive Simulation (VIS) is an important area of this research. It is an increasingly popular method of problem solving. The recent, and increasing, use of computer graphics is having a significant impact on simulation modelling. Animated computer graphics is commonly an integral part of a simulation project. The nature, the use and the future of VIS are discussed in sections 1.4.1, 1.4.2 and 1.4.3 respectively.

1.4.1. The Nature of VIS

Different methodologies on the use of simulation graphics have emerged from North America and the UK. Visual Interactive Simulation (VIS), discussed by Hurrion & Secker (78) is the method that has become common practice within the UK. VIS refers to the use of graphics to animate a model as it is running and at the same time allows the user to interact during the execution of the model. The advantage of this technique is apparent during the initial stages of a project when both the client and the analyst are searching for a suitable simulation model to represent the original problem.

There are a wide variety of graphical symbolisms, movement representations and screen layouts available. Bell (86) described two types of modelling using graphics - representational and iconic graphics. Representational graphics are used to summarise and display data from a mathematical model or component (e.g. histograms, charts, time series). Iconic graphics, which are the main interest of this research, are used to describe the modelled system using pictures. The display is intended to represent the modelled system.

Hurrion (89) further identified two types of iconic graphics animation used for discrete event simulation - character graphics and high resolution bit-mapped graphics. The former refers to the use of keyboard characters to represent model objects where the repeated drawing and erasing of text at slightly different locations will give the impression of elements moving. With bit-mapped graphics, the display screen is made up of pixels (a Macintosh screen is made up of 175,104 pixels, 512 across and 342 down) in which each pixel can be displayed by one colour, depending on the colour configuration of the hardware system. The cost of using bit-mapped graphics is sometimes higher than that of character graphics in terms of programming time and hardware specification. However, a more sophisticated animation picture can be obtained by using high-resolution bit-mapped graphics.

1.4.2. The Use of VIS

Visual interactive simulation is like a voyage of discovery (Fiddy et. al. 81). The client has his own perception of the target system. The analyst, who is employed to build a formal model will establish his own perceptions of the problem via interviews and observations. The VIS model thus acts as an interpreter between these different cultures. The major contribution that interactive graphics has is the ability to improve the communications and language barriers which exist between different management and professional staff for an application, so that both have an improved understanding of the problem.

Paul (89b) emphasised that : "The advantage of visual simulation is further enhanced if the systems used are flexible and fast, so that rapid visual prototyping in collaboration with the customer becomes a possibility. Although visual modelling is a powerful complement to an analyst's problem solving capabilities, it has new problems to overcome as well. The problems of visual simulation includes the fact that vision is interpreted by the brain which does not remember all the visual detail. Moreover, the customer is required to understand the simulation in order to understand what the visual simulation represents. Also, visual simulation is time consuming and it is impossible to test all model interactions visually for a complex model. The most important potential benefit of visual simulation is the increasing ability to help a decision maker by working together in a collaborative effort."

Visual simulation should not be misused or overused. For large and complex models, a simulation picture running on the screen might cause misunderstandings unless the monitor is large and clear. It might mean absolutely nothing to the user unless he understands what the picture represents. Visual simulation is a very useful tool to help the customer understand the system if explained properly. It is also a convincing tool to show that the model is working correctly according to the final user-defined specification. Once the customer is convinced that the computer model satisfies

his needs, his interest will switch from an impressive running picture on the screen to the results that are produced from performing experiments on the model. Therefore, any system that provides the user with visual simulation should also allow the user to switch off this option and choose to extract only results from the simulation run since running a picture on the screen is time consuming.

1.4.3. The Future of VIS

Hurrion (89) pointed out that the main difficulty of the method is the fact that an interactive simulation model will be used by an experienced manager but there is no method within the current visual interactive framework by which this expertise can be retained by the model for future use. The addition of an expert interactive component to a VIS model hence forms the next generation of visual interactive model. This is further described by Flitman and Hurrion (87), and Taylor and Hurrion (88).

Visual interactive simulation modelling will continue its role of improving communications between the analyst and the user, if used properly. With the advent of powerful personal computers and the decline of the price of computer hardware, it is now cost-effective to use graphics in the form of an animated display to show the dynamics of a simulation model. The trend of having better graphical displays at a lower cost will undoubtedly be continued.

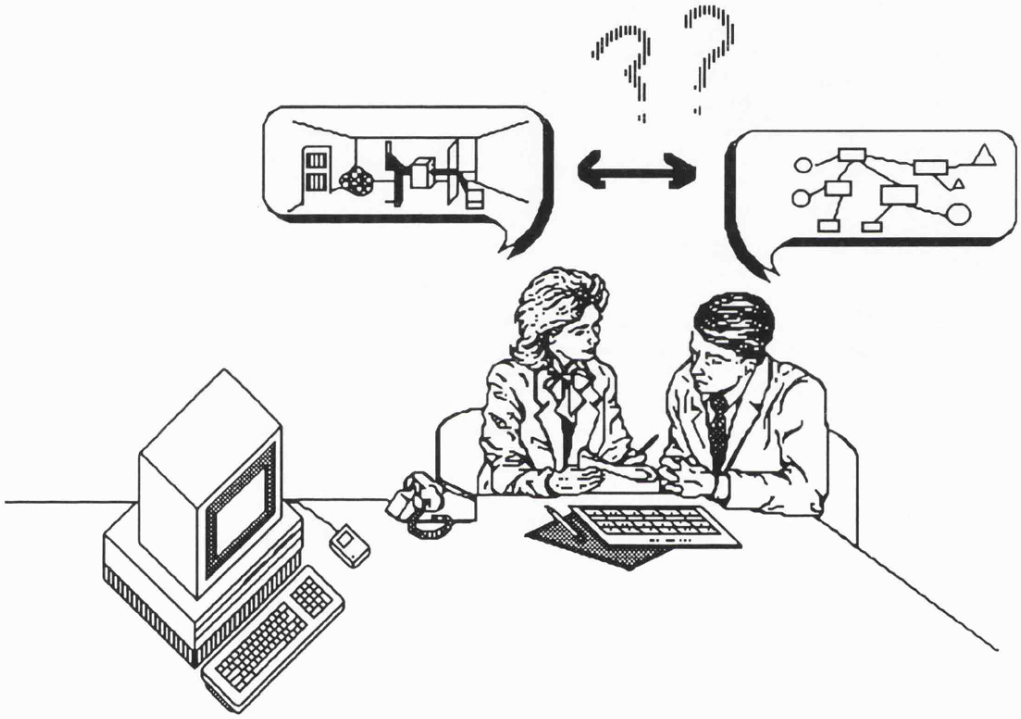
1.5. RESEARCH OBJECTIVES

The main objective of this research is to investigate the role of computer graphics in simulation modelling, especially in the specification stage; and to design and implement an ideal graphics based simulation system.

Consider the case where a client (with limited experience in simulation

modelling techniques) who has her own perception of the target system, is employing an analyst to build a simulation model for aiding her decision-making process. Through interviews and observations, the analyst will translate the client's verbal description of the model into a structural form, very often in terms of a diagramming technique in which the analyst is familiar with. This is shown in figure 1.9.

Figure 1.9. Between Client and Analyst



The heart of this research is to design and implement a visual interactive simulation system which will encourage the client and analyst to build up a model in a collaborate effort. The concept is to allow the user to define their problem by drawing a picture and to be able to run this picture directly on the screen. The pictorial specification of the model will allow the user to reconstruct the model logic constantly throughout the modelling process. The client and analyst can test run the model, repeatedly change the model specification and test run again until the final formal model is obtained. Once they are happy with the model, they can then switch off the option of a visual run and choose to design their output screen on which the results are displayed

during a simulation run.

The work of the CASM (Computer Aided Simulation Modelling) research team at the L.S.E., further discussed in Chapter 2 of the thesis, has formed a very strong basis for such a flexible graphical simulation system to be developed. However, whereas most of the recent work of the team is done on the IBM/PCs machines, this particular simulation system is developed on the Apple Macintosh. The main virtue of using the Apple Macintosh in this research lies in the remarkable graphics facilities and its user-friendly interface.

1.6. OUTLINE OF THE THESIS

There are seven chapters in this thesis.

Chapter 1. Introduction : is a brief introduction to computer simulation modelling and the process of simulation. The Activity Cycle Diagram methodology and the simulation modelling approaches are explained. The use of Visual Interactive Modelling in simulation is discussed. The research objectives and the outline of the thesis are given in this chapter.

Chapter 2. Literature Survey : discusses general simulation environments and describes the philosophy and work of the Computer Aided Simulation Modelling (CASM) Research Team at the L.S.E. A general view of the role of graphics in simulation modelling is given. Furthermore, the current trends in simulation package development are also discussed.

Chapter 3. MacACD, Diagramming Techniques and Computer-Aided ACDs : gives an overview of MacACD and its mechanism. MacACD was the first system developed during this research, and which enabled experimentation with the basic ACD ideas. A study of other existing diagramming techniques is presented. Finally, a

proposed computer-aided ACDs methodology is suggested in the chapter.

Chapter 4. Hypercard and HyperSim : discusses the flexible simulation model specification system HyperSim which is based on ACDs, and which was developed during this research using the Hypertext system - Hypercard. This chapter outlines the philosophy behind such a flexible specification system and discusses how this gives us an insight into the final part of this research.

Chapter 5. Design of the Graphical Simulation Modelling Environment : discusses the design principles behind our ideal graphics-driven simulation environment. A Macintosh application called 'MacGraSE' which incorporates these design principles, developed during this research, is presented in this chapter.

Chapter 6. Implementation of the Graphical Simulation Modelling Environment : gives the internal data structure and the mechanism inside the application MacGraSE.

Chapter 7. Conclusions and Further Research : contains a summary and review of the findings of this research, the three developed simulation systems, and suggests further research opportunities.

CHAPTER 2

LITERATURE SURVEY

This chapter presents a literature survey for the research. The view of an ideal simulation environment and some of the important research projects of the Computer Aided Simulation Modelling (CASM) research group at the L.S.E. are discussed in section 2.1. In section 2.2 we look at the role of graphics in simulation modelling, which forms the theme of this research. Some of the current commercially available simulation languages and some of the modern simulation systems are discussed in section 2.3 and 2.4 respectively. A summary of this chapter is given in section 2.5.

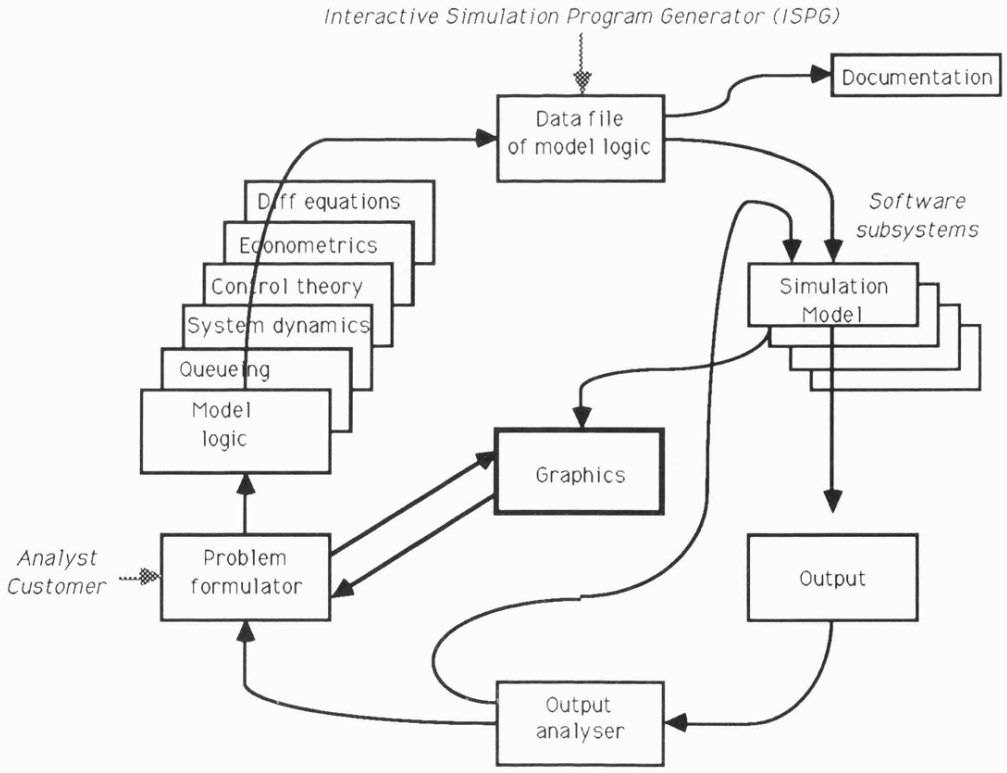
2.1. CASM AT THE L.S.E.

The Computer Aided Simulation Modelling (CASM) Research Project Group at the London School of Economics have been researching into discrete event simulation modelling since 1982 (Balmer & Paul 86). The CASM research objectives are to develop computer based systems that act as tools for the analyst in the process of simulation modelling. So for example, the research group have produced an interactive simulation program generator called AutoSim (Paul & Chew 87) which enables a specification of the simulation model to be turned into program code automatically. The group has also worked on a variety of methods that assist in problem formulation (Doukidis & Paul 85, Paul & Doukidis 86). Current on going projects include the use of formal methods in simulation specification (Domingo & Paul 90) and the use of system dynamics in discrete event simulation modelling (Mak & Paul 90). Section 2.1.1. discusses one of the CASM's views of a simulation environment, and a brief description of the research work done by the team is given in section 2.1.2.

2.1.1. CASM's View of a Simulation Environment

One of CASM's views of a simulation environment is seen in figure 2.1. Activity cycle diagrams (Pidd 88), flowchart or symbols in GPSS (Shriber 74) can be used in the formulation of the problem and the definition of the model logic. The model specification can be fed into an interactive simulation program generator (ISPG) from which a data file representing the logic is produced. The use of an ISPG has two main potential benefits : speed and transparency. It is an important aid in the formulation of a complex model. Further discussions and developments of ISPGs can be found in Balmer & Paul (86), Chew (86), Crookes (87), Paul (88c) and Paul & Chew (87). The program generator, making use of the data file, writes the simulation program using some software subsystems. This model is then ready to be run and output produced. The output, including representational graphics (e.g. histograms, time series, etc.) can be used to determine the correctness of the model logic, and of the computer program.

Figure 2.1. CASM's View of a Simulation Environment



An attempt to bridge the gap between an understanding of the model proposed by the end-user and the analyst is by using an artificial intelligence system (Balmer 85b, Doukidis 85, Doukidis & Paul 85, Doukidis & Paul 87b). With the introduction of an Artificial Intelligence (AI) system to aid the analyst in formulating the problem and experimenting with the model with the customer, the simulation environment is depicted as a continuous loop of activity. 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 was originally seen as a tool for emulating the simulation model output dynamically. However, as shown in figure 2.1, 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 respond to it dynamically, rather than to respond to occasional events as presented by the analyst. Note that this environment assumes that the problem to be solved needs to be formulated as a simulation model. The objective of this research is to look into such a graphical simulation environment.

2.1.2. CASM Research

The CASM team adopted a three-phase set of Pascal simulation routines developed at the University of Lancaster and described by Crookes et. al. (86). These routines were originally written for an Apple II microcomputer. They have been amended and extended to other microcomputers, such as IBM and Apricot as well as to

a VAX computer. These routines are collectively known as the extended Lancaster Simulation Environment (eLSE), and have been used to solve simulation problems (Holder & Gittins 89, Williams et. al. 89).

AUTOSIM

An important development of the CASM team is the ISPG AUTOSIM described by Chew (86), and Paul and Chew (87). AUTOSIM uses a library of Pascal routines called LIBSIM to generate a Pascal simulation program written in a three-phase structure. It requires as input a model specification based on an ACD, and produces a simulation program written in Pascal and supported by LIBSIM. AUTOSIM is currently available on the VAX and IBM/PCs. AUTOSIM uses an interactive session to obtain information from the user by means of a description of the problem ACD. Alternatively, it can accept data files of certain formats specified for the generator.

LIBSIM

LIBSIM, originally described by Crookes et. al. (86), is a later development of a suite of Pascal routines that were created at Lancaster University and subsequently modified at the L.S.E. Some representational graphics routines (for example, routines for producing histograms in the simulation output) are also included in LIBSIM.

SPIF

A first attempt at using an expert system as a problem formulator was described by Doukidis and Paul (85). However, it was found that expert systems were inappropriate for handling the natural language understanding problem. The research

was then directed towards a natural language understanding system (NLUS). A NLUS called SPIF was developed (Doukidis 85). The NLUS captures the essential structure of an ACD for the problem, with the participation of the decision-maker, and under the guidance of the analyst (Doukidis 87).

Database Systems

Research was undertaken into the production of a simulation system within the relational database package INGRES (El Sheikh 87, El Sheikh & Paul 88a). A system called INGRESSIM (El Sheikh & Paul 88b) was produced. This system allowed the analyst to specify the problem, set the initial conditions, and run the simulation all within INGRESSIM. The system was extended by Mashhour (89), who showed how INGRESSIM will be used to specify a model, to which a program generator would then be applied. The resultant Pascal program, when run, would automatically return results to INGRESSIM. This package was called DBSIM.

MacACD

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, the CASM group have spent some of their research effort on developing an ACD based software system which can be used with AUTOSIM. This research was done on the Apple Macintosh since the graphics facilities on the Macintosh provide an ideal environment for producing a graphics orientated simulation system. MacACD, described by Au (87) and Au & Paul (89), is a visual interactive interface for specifying a simulation model logic by drawing an ACD on the screen. A text file using the AUTOSIM data file format can be generated by MacACD. This application is further discussed in Chapter 3 of the thesis.

HyperSim

HyperSim is an attempt to develop a flexible specification interface for a simulation program to be generated using Hypercard on the Apple Macintosh (Au & Paul 89). This application is further discussed in Chapter 4 of the thesis.

2.2. GRAPHICS IN SIMULATION MODELLING

The concept of linking graphics with discrete event simulation began in the mid-1970s and became common practice by the mid-1980s. Today, computer graphics assist in all phases of a simulation project, from its initial problem specification to final project implementation (Hurrion 89). There are now numerous graphical systems both for the development of the models and the presentation of results. Graphics are generally applied in three main areas of simulation modelling - model specification, visual simulation and output display. We discuss how graphics are used in each of these areas in the following sections.

2.2.1. The Use of Graphics in Specification

Most of the specification methods require the use of a diagramming technique. The most commonly used in three phase modelling is activity cycle diagrams (see section 1.2). A variation of the ACD method, called the activity diagram is discussed in Davies and O'Keefe (89). The successes of HOCUS (Szymankiewicz et. al. 88) and SLAM (Pritsker 79) illustrate that flow diagram systems have a broader appeal than text-based ones.

Not all the current existing simulation packages have a graphical front end for model specification. Most data-driven simulation packages require the user to translate

the image of his perceptual model into some form of structured diagram before the actual model building process on the computer. The input session of DRAFT (Mathewson 87) is a series of responses to prompts. The model expressed as an entity cycle diagram is input via a terminal. DRAFT/DRAW then allows the user to display the input entity cycle diagram either in its original form, with the software managing the layout, or in a representational form which mimics the perceived system and omits artificial devices used for programming purposes.

A less established area in using graphics in simulation specification is the direct application of pictures to describe the model logic. This research is aiming to achieve a graphics driven interface for the user to draw the invisible image of the real world system inside his brain during the construction process and to be able to model the logic within such an environment.

2.2.2. Visual Simulation

Visual simulation is commonly considered as an essential part of a simulation system. Its importance can be seen from the fact that nearly all the latest versions of the current simulation packages have added a visual simulation component to their systems. Visual simulation allows the user to watch the dynamics of a model unfold in the form of a semi-pictorial silent film. Not only can visual graphics narrow the gap between the client and the analyst, it also helps the user to gain confidence in the use of the model.

Most simulation packages require the user to pre-specify the model in a text descriptive format and provide the interface for developing a visual picture for the model from this formulation. The animation represents a final polish to the model-building process. PCMODEL (Simsoft 86) on an IBM/PC uses the colour monitor display to provide a direct character-based, graphics animation of the model. A graphics editor is supplied to build the overlay. The model is set up with statements describing

the route of the modelled object and its delays. The logic of the conditional transaction movement is specified as a mnemonic.

This research aims to design an environment which allows the user to combine the specification of the model and the display of a visual representation of the model in one step. The specification picture is by itself a picture which can be used for running a visual simulation.

2.2.3. Graphical Output

Computer graphics are now replacing the more traditional printed report as a presentation medium. Business graphics use charts, pie diagrams, graphs and histograms to display information, while in education and entertainment computer-generated animation files are becoming common (Hurrion 89). Presentation of model results is an important stage of simulation modelling. The popularity of existing sophisticated presentation packages, for example, Freelance on the IBM/PC and PowerPoint on the Apple Macintosh, show that there is a requirement for high standard graphical output by top management. Nevertheless, it is always the final simulation output that will reach the decision-making management team directly, and which is the product of the simulation study on which decisions will be based.

Three-dimensional graphics are popular but should not be overused. Structured graphics in terms of charts or graphs must be clear and precise since over-emphasised fancy graphics might be misleading.

2.3. SIMULATION LANGUAGES

Mathewson (89a) describes some of the most commonly used simulation languages. The simulation languages adopt either the three-phase, event-based or the

process simulation structures. The use of object-oriented simulation languages is becoming more and more popular in the simulation industry.

ECSL (Clementson 85) and SIMON (Mathewson 77) are three-phase based simulation routines. ECSL is written in FORTRAN but the user is not required to know FORTRAN, nor is the code translated into FORTRAN. The authors claim that it successfully implements the 'cell structure' developed at Lancaster University (Spinelli de Carvahlo and Crookes 76) and also incorporates visual interactive modelling. SIMON provides facilities for queue handling and for activity scheduling and event identification, including analysis tools for variance reduction and interval estimation. The most recent library SIMONG (Mathewson 85) has been extended to incorporate graphics drivers for model animation.

SIMSCRIPT II.5 (CACI 83) and GASP II (Pritsker and Kiviat 69) are event-based simulation routines. SIMSCRIPT is represented as having a number of levels which enable it to form an easy introduction to computer programming. The language consists of two extensions - PROCESSES which permits sequences of events to be specified and SIMANIMATION which allows the user to add graphics animation commands to the SIMSCRIPT II.5 text. GASP II was the first well-documented release, followed by GASP IV which provides mixed discrete/continuous simulation. Both systems provide a library of FORTRAN routines for writing a FORTRAN model.

SIMULA (Birtwhistle et. al. 79) is the original language which implemented the process modelling approach. The language is operated by defining system 'classes' which provide event scheduling and queue manipulation. Other process type simulation languages in Pascal includes Micro-PASSIM (Barnett 86), PASCAL-SIM (O'Keefe 86), SIMPAS (Bryant 80) and SIMTOOLS (Seila 86).

2.4. MODERN SIMULATION SYSTEMS

A survey of the range of methodologies and systems available in simulation is given by Mathewson 89b. Paul 89a pointed out that simulation support environments can be broadly classified into four groups - data driven systems or generic models, program generators, pseudo-code or macro-statements, and bespoke modelling or handcrafting. Some simulation packages consist of a combination of the features of these different groups. Each type of environment is discussed in the following sections.

2.4.1. Generic models

These packages are general purpose simulation systems which take data defining the user's particular problem as the basis for running the general model. The advantage is that no programming is required by the user. The disadvantage is that one can only do whatever the package is designed to do. There is usually no program code for the model to be edited.

HOCUS, Hand Or Computer Universal Simulator (Szymankiewicz et. al. 88) is a generic data driven system based on activity cycle diagrams from which the data is input into the package in a structured way. There is no generated code from HOCUS. The system automatically generates a trace of the operation of the model by calling and combining standard FORTRAN subroutines. These are then compiled and executed but they are not available for modification by the analyst. The package is supported by a textbook by Szymankiewicz et. al. (88). The advantage of HOCUS is that it is simple to understand and requires no knowledge of computing. However, HOCUS provides only the model description program. The model is expressed completely within the structure of the diagram and therefore suffers from severe restrictions in the scope of the detail which can be represented. As the user is not allowed to edit the FORTRAN code, some complexities are likely to be difficult to model.

WITNESS (Istel Ltd.) is a data-driven system originally written in SEE-WHY. This is a menu-driven simulation created for the study of manufacturing systems. WITNESS can also be integrated with SEE-WHY submodels if its internal capabilities are insufficient for a particular task. The modelling elements of WITNESS are related to the factory - parts, machines, buffers, conveyors and labour. The interface design can progress in parallel and independently of the parent model. It is therefore attractive to the management of system design houses, as it gives them more flexibility in the provision of enhancements to the system. It is possible to add code to the system. The disadvantage is that the modeller is required to learn how to use the package and it is not an easy package to learn. It is also an expensive package.

SIMFACTORY (CACI Ltd.) is a general purpose simulation system for factory design and production analysis. Models are designed by entering factory layout information & production parameters. The user interface is provided as a menu-driven editor. There are twelve functional menus with on-line help. Through these menus the data set can be assembled. The display is created by selecting elements and positioning them under cursor control. An animated representation of the factory and an interrupt facility is provided. The generic model is written in SIMSCRIPT II.5 and the application-specific features are selected by input data.

Other examples are EPSIM (Epsim Ltd.) and PROPHET (Manufacturing Management Ltd.). The former is a discrete event generic simulation model of a manufacturing environment with financial analysis and the latter is a data driven generic system models batch manufacturing processes controlled by material requirements.

2.4.2. Program Generators

The user specifies his problem either in data or graphical form. The system then automatically produces program code in a programming language which the

analyst can access. It allows the analyst to have greater flexibility in what sort of model might be produced. However, the analyst must be able to understand this generated program code. The advantage of using a program generator is that it shields the user from the intrinsic difficulties of general simulation languages.

CAPS (Clementson 85) is an interactive simulation program generator for discrete event simulation modelling. It is the front end to ECSL and generates ECSL code. CAPS is designed to run interactively and it conventionally prompts the user in a way that allows the user to define a simulation model based on an activity cycle diagram. After the definition of a model, an ECSL source program is generated.

DRAFT (Mathewson 87) is an automatic program generator producing FORTRAN code. DRAWER permits computer assisted development of interactive animation for models produced by DRAFT. SSIM uses a pictorial activity diagram input as a preprocessor for the program generator. Although the structure of DRAFT has been carefully designed, the diversity of information in simulation models represented by their activity cycle diagrams has not been fully attended. Therefore, DRAFT is only suitable for complete entry of very simple cycles. More complicated cycles need much time and effort to modify their DRAFT generated programs to represent the system fully. With DRAFT, the objects within an ACD are to be numbered in order to go through the interactive input session.

2.4.3. Pseudo code

These packages use pseudo code or macro statements to enable the analyst to write a model in shorthand form. The analyst has greater control over the model representation since the model is represented in code form. The disadvantage is that one has to learn another programming language in order to write code in the system. Some packages include a graphics interface to aid the analyst in formulation of the problem.

One example of this type of packages is ECSL. The production of ECSL code is the major limitation of CAPS/ECSL. ECSL is a special purpose language and has its own neat and concise style. A skilled ECSL programmer is not easy to find. As a non-compiled language, ECSL is relatively slow running and does not offer the benefit of an interpreted language. Partial programs cannot easily be run for testing because of the compilation and execution time delays. Moreover, it does not have a dynamic entity type due to its static storage management.

Another commonly used simulation system which allows the user to write macros is SIMSCRIPT II.5 - a macro programming language with English-like program code which allows the user to read the simulation program like a description of the system that is being studied. Typical applications include military planning, manufacturing, communications, logistics and transportation. SYSMOD (Systems Designers Scientific) is a language which is similar to Pascal & Ada in syntax and which is used for defence simulation models.

2.4.4. Handcrafting

This implies writing the simulation model from scratch in some programming language. Although this seems to be a weak method to undertake, its popularity is possibly due to the freedom and complete control the analyst has over the simulation model, and, to a certain extent, because simulation packages are too expensive to use.

2.5. SUMMARY

'A Simulation environment provides tools for specifying the processes in a simulation and generating debug and production models automatically from these specifications.' (Birtwhistle 85). The CASM team at the L.S.E. is a research program

which aims to provide aids for problem formulation, program generation and output analysis in simulation modelling. The CASM research project has provided a very strong backbone to this piece of research.

This chapter has described current trends in simulation systems, and the popular methods of assisting the process. The following conclusions are directly derived from this literature search and the proposed needs for simulation modelling set out in Chapter 1.

A data-driven or generic package should be user-friendly so that data can be updated very easily. If a program generator is provided with the package, the code should be easy to understand so that the user can easily find the part of the code to be edited during the modification of the program.

Graphics should be the front end to the formulation of the problem and not as an additional way of representing the model logic from the textual description. Visual simulation should be a vital part of the package so that animation of the model can be seen on the screen and picture running should be available. The user should be able to design his own output screens for use in a simulation run.

Graphics and modern software interfaces will enhance the process of translating the formulation into code. Subsequent modifications to problem definitions will be readily facilitated and interface design will ensure the logical rigour of the definition. Furthermore, graphics may be used to supplement or replace the traditional problem definition techniques. The use of graphical input techniques may provide a means for automating the formulation process. Colour graphics are and will continue to play an important role in model construction and development.

CHAPTER 3

MACACD, DIAGRAMMING TECHNIQUES AND COMPUTER-AIDED ACDS

The first section of this chapter discusses the first development of this research, the MacACD application (Au 87, Au & Paul 89) on the Apple Macintosh computer. Experience gained from developing MacACD is given in section 3.2. Since most CASM research is based on a diagramming technique known as activity cycle diagrams, other existing diagramming techniques are also examined. The results of this study are given in section section 3.3. With the developing experience of MacACD and the study of other diagramming techniques, the computer-aided activity cycle diagrams methodology is proposed in section 3.4. A summary of this chapter is given in section 3.5.

3.1. MacACD

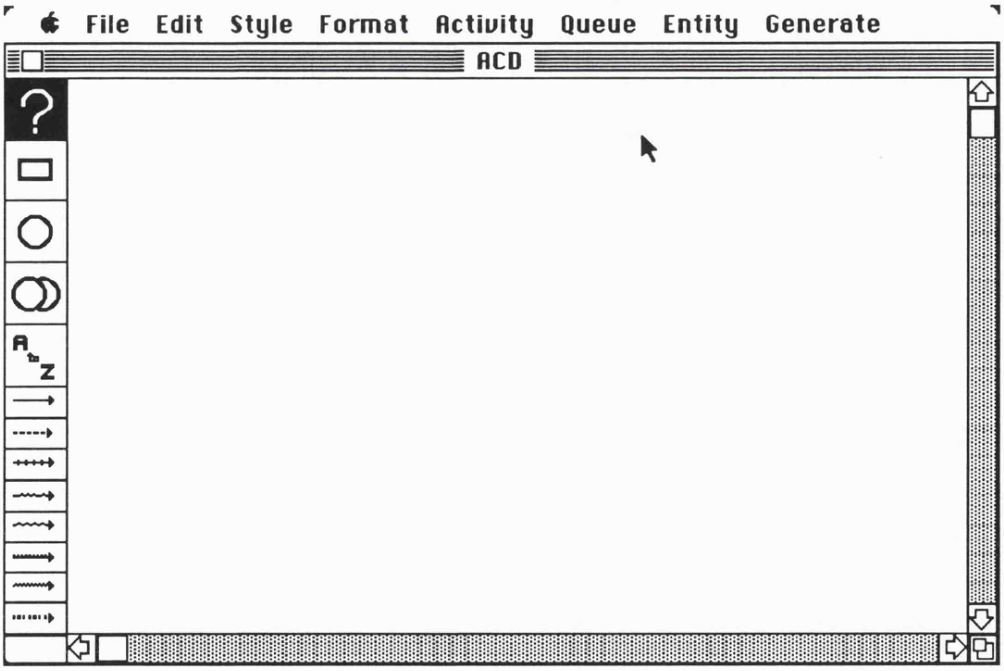
MacACD is a simulation specification system that uses the graphical methods of activity cycle diagrams, developed on the Apple Macintosh computer. This system can be combined with other simulation tools developed by members of the CASM research group to enable the automatic generation of a simulation program with a visual iconic display. An overview of the software is given in section 3.1.1. and some of the design aspects are discussed in section 3.1.2. The experience gained from this research project are discussed in section 3.2. An example of the way in which MacACD achieves its task using the ubiquitous simulation example of a pub (described in section 1.2.2) is given in Appendix C.

3.1.1. An Overview of MacACD

MacACD is a user-friendly application which allows the user to specify a simulation model by drawing an activity cycle diagram on the computer screen. The user can enter the details of the parameters of the model by means of dialogue boxes. These dialogue boxes are evoked by the action of the user. MacACD allows the user to generate a text file from the ACD specification. This text file can be passed down to VAX and read by a program generator AutoSim (Chew 86) so that a three-phase simulation program can be generated.

A more detailed description of how to use the MacACD application is given in Appendix C of this thesis. Figure 3.1. shows the appearance of the interface of MacACD.

Figure 3.1. The MacACD Application



3.1.2. The MacACD Interface Design

MacACD consists of nine pull-down menus and an iconic menu with thirteen palette choices. The pull-down menus handle all the file management and data recording whereas the iconic menu mainly deals with the creation of graphics on the screen.

The application is designed so that the user can choose to do anything (for example, creating an entity type, creating an activity, editing an existing activity, creating a queue, or drawing part of the life cycle of an entity type, etc.) at any time during the model construction process. The aim was to produce a user-friendly interactive environment for the user to build up the logic of a simulation model.

Objects on the screen

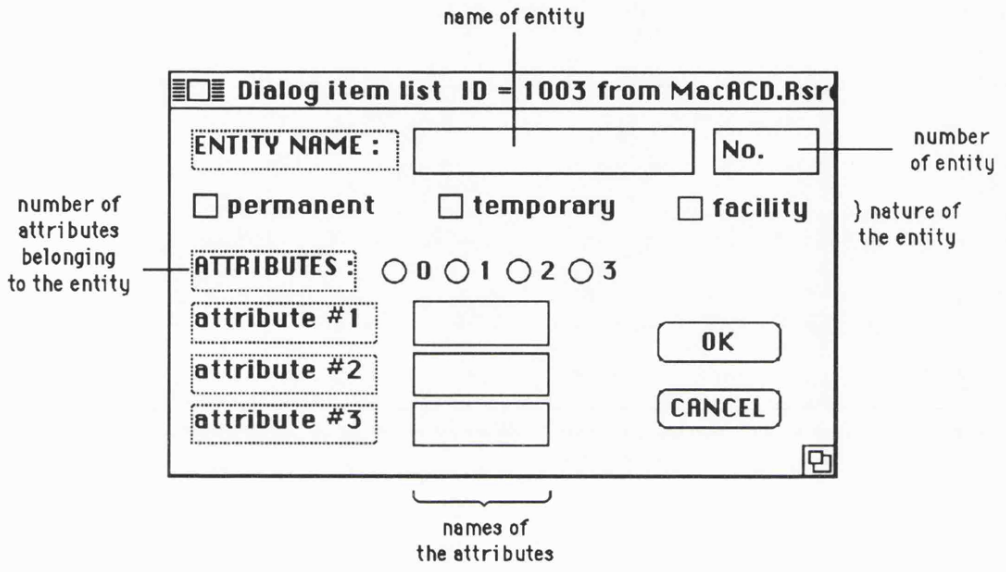
There can be three types of objects on the screen. Firstly, rectangles which represent activities. Secondly, a circle or two overlapping circles which represent queues and source/sink queues respectively. Finally, line arrows which represent life paths of an entity type. These objects can be created using commands in the iconic menu. For example, to create an entity type, just click at any one of the line arrow mode boxes. MacACD will then prompt the user for information about this entity type by means of an entity dialogue box as shown in figure 3.2. The objects on screen can be resized and moved within the main window so that the user can easily arrange the ACD in a structured way.

Entities

The information about an entity is obtained by means of an entity dialogue box as shown in figure 3.2. MacACD distinguishes three types of entities - temporary,

permanent and facility entities. Attributes can be created for an entity type. An entity can have up to three types of attribute. The user can just click at the desired number of attribute types that are to be created and type in the attribute name in the appropriate text box.

Figure 3.2. The Entity Dialogue Box

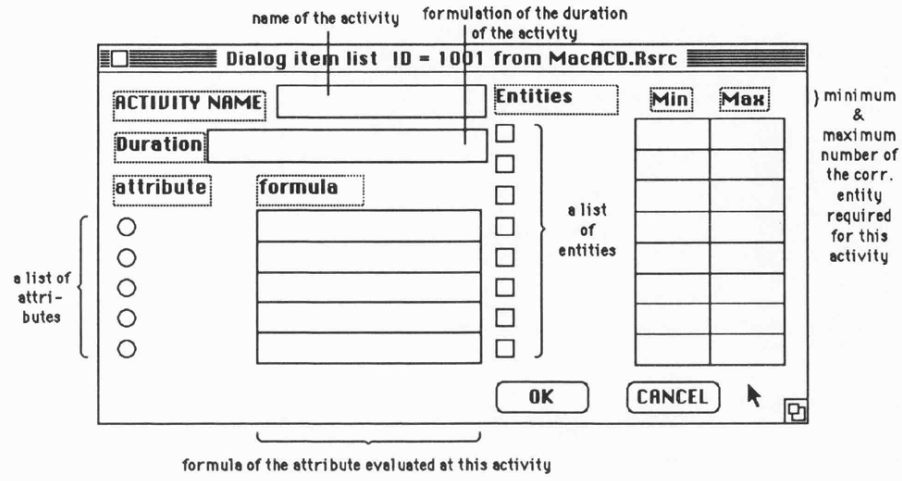


The creation of an entity type is not complete until the user clicks the 'OK' button. The name of the entity is then appended to the Entity pull-down menu. The user can subsequently select any entity from the menu and edit its information.

Activities

The information about an activity is obtained by means of an activity dialogue box as shown in figure 3.3. The dialogue box enables the user to choose the entities that are required in order to start the activity. The Entities column displays the full list of entities declared by the user. The Min and Max columns are for recording the minimum and maximum number of the corresponding entity type that is required in the activity. The default is set to one for each Min and Max entry.

Figure 3.3. The Activity Dialogue Box



Since attributes can have different formulae in different activities, the list of attributes declared by the user is always displayed. The user can type in a formula for the appropriate attribute that is being evaluated in that activity.

The creation of an activity is not complete unless the 'OK' button is clicked. The name of the activity is then appended to the Activity pull-down menu. The user can subsequently select an activity from the menu in order to look at a summary, or to edit the relevant information for the activity.

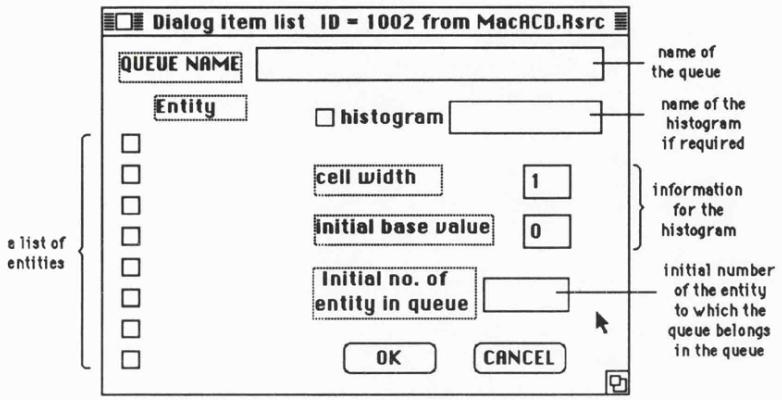
Queues

MacACD distinguishes three types of queue - a source/sink queue, a normal queue and a dummy queue. A source/sink queue is represented by two overlapping circles while the normal queue by a circle. The dummy queue is not drawn on the diagram.

The information for a queue is obtained by means of a queue dialogue box as shown in figure 3.4. The dialogue box displays a list of entities which the user has

declared and the user can choose the entity that the queue belongs to. The user can also define a histogram to record the queue lengths of the queue during a simulation run.

Figure 3.4. The Queue Dialogue Box



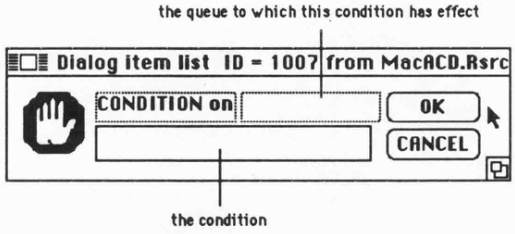
The creation of a queue is not complete until the user clicks the 'OK' button. The name of the queue is then appended to the Queue pull-down menu.

Life Cycles of Entities

When an entity is created, a line pattern is assigned to it. The user can then just click at the line arrow mode box which represents the selected entity type and draw its life path in the ACD. Moreover, MacACD uses straight lines for representing the life paths of the temporary and permanent entities and arcs for that of the facility entities.

The life paths of a permanent or temporary entity can be drawn by moving the mouse from an activity to a destination queue, or from a queue to a destination activity, or from an activity to another destination activity. Whenever MacACD detects two activities being linked together, it will automatically produce a dummy queue. Whenever MacACD detects that there is an activity which goes to more than one queue, a condition dialogue box (figure 3.5) appears. The user is prompted for the condition that is required to place the entity in the first queue that the user selected.

Figure 3.5. The Condition Dialogue Box



For a facility entity, the user only needs to move the mouse to the activities that the entity type is involved in. However, the user is required to create an idle queue for the facility entity before the life paths can be drawn.

3.2. EXPERIENCE GAINED FROM MacACD

This section looks at the advantages of MacACD in model specification, and the limitations of such an application. The potential of using the Apple Macintosh in developing a graphics-orientated application is also discussed.

3.2.1. Advantages of MacACD

The MacACD application has demonstrated the power of modern graphics based interfaces as provided by the Apple Macintosh for inputting graphical specifications for simulation models.

The application is event-driven, i.e. the user has full control over the model building exercise. Moreover, the direct input of a diagram minimises the errors that can occur if the user is required to translate the diagram into a textual description. Diagrams are always a more appealing communicative media than text.

The translation from the diagram to a data file which represents the model logic is the first step towards the generation of code.

3.2.2. Limitations of MacACD

One of the limitations of MacACD is that it does not allow the user to run the computer simulation model directly from the flow diagram. The lack of colour on a Macintosh plus screen might also discourage the use of such a system.

An ACD is insufficient to model all complex real world systems. New ways of formulating the model logic are constantly being explored. Recent research on graphics is concentrated on providing an interface which allows the user to draw the entities on the computer screen, and to describe the movement of the entities directly on the screen. There are a lot of diagramming techniques such as decision trees or semantic networks which might be appropriate in simulation modelling. The result of an investigation into these is given in section 3.3.

3.2.3. The Apple Macintosh

Most CASM developments have been on an IBM/PC. Au (87) and Au & Paul (89) outline the general differences between the Macintosh and the IBM/PC, as well as their different graphics facilities. The main differences include the use of mouse, the graphics facilities, the structure of the ROM and the user interface.

A graphics-based visual system does require some sort of pointing device, and the mouse works as well as, or better than, most. The advantage of having a mouse, as shown by the success of the Macintosh, has led to its rapid acceptance on other systems.

The bit-mapping graphics display (512 pixels each line, 342 lines) on the Macintosh enables the marriage of graphics and text and the ability to manipulate both on the same display. On an IBM/PC machine the screen is made up of 25 lines of 80 characters each, and one has to choose between a text or graphics mode. One of the disadvantages of the Macintosh family is that the Plus and SE range only come with monochrome graphics. Only the Macintosh II has colour graphics facilities.

Another important difference between a Macintosh and an IBM/PC is in the ROM of the machine. The IBM/PC puts some low-level portions of the system in ROM and then loads in more of the operating system PC-DOS when the system is started up. The Macintosh has relatively more of the operating system, the Macintosh Operating System (MOS), in the ROM. The MOS takes up only a fraction of the ROM. There is a User Interface Toolbox (UIT) in the ROM which consists of hundreds of callable routines that are used to implement the standard Macintosh application interface.

The Macintosh provides a standard visual user-friendly interface based on menus, icons, windows and a mouse as the input device. A similar interface can be produced on an IBM/PC, but the developer has to write all the procedures to incorporate these features. This might cut down the running speed of the application tremendously, and a much greater amount of memory would be required by the software.

The main advantage in using graphics on a Macintosh is its speed in drawing. Quickdraw, the magician artist in the ROM of the Macintosh allows one to draw complicated graphics at a very high speed. Another advantage concerns its ability to store resources separately from the the application code. Also, Macintosh supports the use of icons. Using icons is an ideal way of representing an entity on the screen. Because of the high resolution, the movement of the entity can be very smooth and well-presented. Moreover, the capability of having multiple-windows is another important aspect of the Macintosh. This allows the ACD to be divided into a hierarchy

of levels. A complex system might have a lot of activities and queues in which some of them may be grouped into an overall activity in general. Therefore, inside this activity, there will be another ACD.

3.3. DIAGRAMMING TECHNIQUES

Diagramming is a form of language, beneficial both for clear thinking and for human communication. It is desirable to provide the analyst with a set of diagramming techniques in which to conceptualize, analyze, and design. These techniques can act as an aid to clear thinking.

The role of diagramming techniques in software engineering is discussed in section 3.3.1. Section 3.3.2 gives a brief overview of the current existing diagramming techniques and their main functions. A summary table of the different areas where these diagramming techniques are applicable is given in Appendix D. Some commonly used computer graphics tools are discussed in section 3.3.3, conclusions concerning the properties of a good diagramming technique are given in section 3.3.4.

3.3.1. Diagramming Techniques and Software Engineering

Diagramming techniques are one of the system specification methods and can be used in conjunction with other specification methods, for example, mathematical logic. They are also widely used in system design. When an analyst is designing a system, the use of diagrams can be an aid to clear thinking. When a number of people are working on a system, the diagrams becomes an essential communication tool. A formal diagramming technique is needed to enable the developers to interchange ideas and to make their separate components fit together with precision. Clear diagrams are also an essential aid to system maintenance since they enable the programmers to

understand the consequential effects of changes they make. They are a highly valuable tool for understanding how the programs ought to work, and tracking down what might be wrong. The diagrams should be user-friendly, and should be designed to encourage user understanding, participation and sketching.

3.3.2. Current Diagramming Techniques

This section, following from Martin & McClure (85), contains a brief description of some of the diagramming methods most commonly in use.

Decomposition diagrams are a simple means of diagramming the structure of organizations and complex processes. The user can easily understand and draw decomposition tree structures.

Data flow diagrams (DFDs) are tools for understanding the flows of documents and data among processes. The technique is simple and easy to learn. DFDs are not ideal for drawing program architectures and should be tightly linked to data models.

Dependency diagrams are a replacement for data flow diagrams with a similar ability to represent the flow of data among processes, but designed to be automatically convertible to action diagrams. There are notations to represent optionality, conditions, cardinality, mutual exclusivity and inclusivity.

Entity-relationship diagrams are tools which provides a logical overview of data needed for running an operation. They are an essential part of strategic data planning. The user can be quickly taught to read, check and draw entity-relationship diagrams. A data structure diagram is an expansion of an entity-relationship diagram into detail showing data items.

Data navigation diagrams are simple tools for designing database navigation and are also useful with file systems.

Structure charts are used for showing program hierarchy used in conjunction with data flow diagrams and pseudocode. The chart itself does not give complete control structure information and prevents rather than assists in automated verification.

HIPO diagrams show the input, output, and functions of a system or program. Data flow diagrams or dependency diagrams give a much more compact and easy-to-read view of the flows of data than HIPO diagrams. A high-level HIPO diagram does not give complete control structure information whereas a detail-level HIPO diagram is limited to defining procedural components.

Warnier-Orr diagrams are used for showing functional decomposition and hierarchical data structures. They are easy to read, draw and change. The chart shows sequence, selection and repetition, but not conditions, control variables, case or loops. They do not show input and output data for procedural components and do not facilitate automated checking. There is no direct link to a data model or a data dictionary and they are not database oriented.

The Michael Jackson diagram methodology requires the user to design the data first and then derive the program structure from them. They are not user-friendly and are the most difficult of the methodologies to learn and use correctly. They are oriented to file, and not database operations. They show sequence, selection and repetition, but not conditions, control variables, case of loops. Only hierarchical data structures are represented.

A flowchart is not a structured technique and it leads to unstructured code which is difficult to maintain. It should be avoided in favour of structured diagramming techniques.

Nassi-Shneiderman charts only show detailed logic and not program architecture nor functional decomposition. Although they are easy to read and teach, and are graphically appealing, they are too time-consuming to draw and change. The chart is not linked to a data dictionary or data model and shows neither high-level program structure nor low-level degeneration into code.

Action diagrams are a simple and elegant technique designed to overcome many deficiencies in earlier techniques. They are quick to draw, easy to read, teach and computerize. It is a technique that extends all the way from the highest-level functional decomposition to the lowest-level logic and coding. Action diagrams can be decomposed into executable code. They enforce correct control structures and show nonprocedual database operations.

Decision trees and decision tables are used to represent complex sets of conditions or rules and the resulting actions. The users can be taught to check decision tables for complex sets of rules or conditions.

State-transition diagrams are used for certain types of complex logic where multiple transitions among states occur.

HOS charts are mathematically based so that designs which are provably correct are created. Program code can be automatically converted. They are not user-friendly and require a commitment to learn a technique substantially different from traditional techniques.

3.3.3. Computer Graphics Tool

Interactive diagramming on a computer screen speeds up the drawing process greatly. The advantages of using computer graphics tools include significant cost and time savings, great reduction in work needed to redraft graphics during development

and maintenance effort, elimination of both proof-reading and the potential for error introduction on diagram updates, and the capability to produce very large diagrams. Computer-aided drawings enforce standard and discipline. Checks can be applied to what is being created by the user. Different types of cross-checking, calculations and validity checks can be applied. Computer-aided drawings replace volumes of text with powerful computable symbolic design.

General drawing applications includes PCPaint on the IBM/PC, MacDraw II (Claris UK Ltd.), MacPaint 2.0 (Claris UK Ltd.), PixelPaint 2.0 (Principal Distribution Ltd.), SuperPaint (Persona-TMC), and Studio/8 (Electronic Arts) on the Apple Macintosh. These applications allow the user to draw any static picture on a computer screen, without any built-in mechanism. Tools for dynamic drawings include Excelerator (Index Technology) and computer-aided design (CAD) software like ClarisCAD (Claris UK Ltd.) and ArchiCAD (Desktop Engineering Systems). Dynamic drawing defines linkages between components or relationship between icons. MacCadd (Logica) on the Apple Macintosh is an application which incorporates some of the existing methodologies, for example data flow diagrams, structure charts and Jackson diagrams. Use-it (High Order Software Inc.) is one of the new methodologies designed to take advantage of the computer facility. The application Visual Interactive Programming (The MacSerious Company) on the Apple Macintosh is an example of a tool for generating executable code from graphics design. This type of application may be linked to an interpreter or optimizing compiler, or may create code in a fourth generation language which has its own interpreter or compiler.

3.3.4. Properties of a Good Diagramming Technique

Martin & McClure (85) outline the properties of a good diagramming technique. In general, a good computer-aided diagramming technique should possess the following functions :

- It should be an aid to clear thinking.

- It enables precise communication between development team members.

- It can be manipulated easily on a computer graphics screen.

- The end-users can learn to read, critique and draw the diagrams quickly, so that the diagrams form a good basis for communication between users and analysts.

- Whilst hand-drawn diagrams are designed for speed of drawing, computer-aided diagrams can have more lines and elaboration.

- The technique should use constructs that are obvious in meaning, and avoid mnemonics and symbols that are not explained in the diagram.

- The diagram can be printed on A4-sized paper.

- Complex diagrams are structured so that they can be subdivided into easy-to-understand components.

- An overview diagram can be decomposed into detail.

- The diagram should reflect the concept of structured techniques.

- It should be an aid to teaching of computer methods.

- There must be a consistency of notation among all different types of diagrams that an analyst needs.

- The technique should be a basis for computer-aided design and code generation.

This advice certainly appears pertinent to simulation as well as to general systems development.

3.4. COMPUTER-AIDED ACDS

ACDs have many of the characteristics required of a diagramming technique. Having examined the conventions and methodologies of the current existing diagramming techniques, some enhancements can be made to the activity cycle diagram method in a computer-aided environment.

3.4.1. Activities

The existing diagramming techniques distinguish the use of round-cornered and square-cornered rectangles. Data-entity types or record types are normally drawn as square-cornered boxes while functions, processes, procedures, or activities in general are drawn as round-cornered rectangles. Therefore, an activity should be represented by a round-cornered rectangles. Moreover, besides containing just the activity name inside the activity box, the user should have the option of displaying the formula for the activity duration in the box.

Matching of entities sometimes occurs when testing the possibility of an activity start. This can be indicated by putting an asterisk '*' besides the activity name. The rules of the matching process, which is usually achieved by comparing certain attributes of different entity classes, can be described in text form.

3.4.2. Queues

The advantage of displaying queues in an ACD is that the picture is more clearly and logically represented, since, after an activity has finished, the entity should be placed somewhere in the system so as to wait for the next activity to start. However, for a system which involves a large number of different entity classes, the ACD will be

full of queues and the diagram looks more complicated and less precise. Also, some queues inside the system are dummy queues which do not need to be displayed. These queues are imaginary since the entities can start the next activity immediately after the finishing the current activity.

Therefore, the user should have the option of whether or not to display the queues belonging to an entity class in the computer-aided ACD. In computer-aided diagrams, the queues should be automated. This can be achieved by tracing the movement of the user. When he links one activity to another, a queue is automatically generated for the selected entity class.

3.4.3. Life Paths of Entities

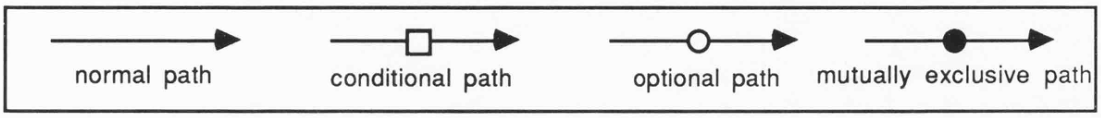
A line arrow is used to represent a life path of an entity class. In a monochrome system, different line patterns can be used for different entity types, whereas in a colour system, different colours can be used. For complex systems which have different groups within an entity class, this would traditionally be regarded as having different classes of entity. However, in modern graphics environments, this grouping classification can be represented by using one colour for the entity class, but with different patterns for different groups within the class.

A conditional line path can be introduced for indicating that an entity only uses this path when certain conditions in the system are satisfied. The appearance of a conditional line path is identical to an ordinary line arrow except that there should be a square somewhere along the arrow.

Optionality is necessary when the entity has a choice whether to go through an activity or not at some point in time during its life cycle. This can be indicated by a small empty circle on the line arrow which goes to this activity. Mutual exclusivity is necessary when the entity has the option of going to one of different activities at some

point in time during its life cycle. This can be indicated by putting a dot on the line arrow which branches out to different activities. A summary table of different arrow conventions is given in figure 3.6.

Figure 3.6. A Summary Table of Different Arrow Conventions



3.4.4. Layered ACD

For large and complex systems which involve a lot of activities, some activities can be refined into another ACD. In a computer-aided environment, this representation can be depicted by using a shaded activity box which indicates that there is another ACD within this activity. A simple ACD can be regarded as a first level or a skeleton of a simulation system. Within this skeleton, some activities might contain another ACD; and within some activities involved in these individual ACDs, there might be another ACD; and so on. This layering of ACDs can be clearly structured in a computer-aided environment by using multiple windows.

3.4.5. Code Generation

The computer-aided ACD represents the model logic and contains information about the parameters of the simulation. However, the variety of human endeavour suggests that an all-embracing specification method is not completely achievable. It is desirable, therefore, to allow the user to generate a simulation program directly from a graphical description of the problem and thereby make the necessary small but intricate amendments to derive the final model.

3.4.6. Iconic Representation of an ACD

An entity type should be associated with at least one icon. An iconic display of an ACD can be easily translated. If the system allows visual simulation, then the picture can be used for a graphical simulation run.

3.5. SUMMARY

In this chapter, we have introduced the MacACD application which is a specification system that allows an activity cycle diagram to be drawn on the computer screen. The parameter input is via dialogue boxes within a windowing environment, with pull-down menus and an iconic interface. The automatic generation of code is achieved by the generation of a text file which can be fed into a program generator.

After looking at some current existing diagramming techniques and experimenting by applying some to simulation modelling, we concluded that activity cycle diagrams are an appropriate method for illustrating the logic of a discrete-event simulation model, although they become unable to incorporate the relevant detail as the model becomes more complex. Some enhancements to the activity cycle diagrams methodology are therefore prepared for a computer-aided environment. The most important ideas are the possibility of layered activity cycle diagrams and automatic code generation.

The Apple Macintosh has proved to be an ideal computer system for a graphics oriented application to be developed. The main merits are its user-friendly interface and the speed of drawing. The development of MacACD has given an insight into the need of a flexible simulation system which allows the user to specify a simulation model either in graphical or textual format, or a combination of both. Chapter 4 presents the development of this flexible system on the Macintosh.

CHAPTER 4

HYPERCARD AND HYPERSIM

Simulation modelling is often applied to ill-defined problems, so that the model specification is constantly under review. There is a need, in such cases, for a flexible method of model specification. The previous chapter proposed the characteristics and advice of such a system. Given the strong interlinking connections between the components of a simulation model, it was decided to attempt an implementation of these ideas using the Hypertext approach. This chapter describes a mixed graphical textual system, HyperSim, based on activity cycle diagrams using Hypercard. Such a system enables rapid model development using either graphical or textual editing or both.

A brief history of the Hypertext concept and the Hypercard application is discussed in section 4.1. In section 4.2, an overview and the design of the HyperSim specification system is given. Section 4.3 discusses the advantages and disadvantages of this approach. Section 4.4 summarises the result of this research development and describes how it gives insight into the development of a complete graphical simulation system.

4.1. HYERTEXT AND HYPERCARD

Bush (45) envisaged and described a virtual system which relied heavily on the intricate web of connections which can be made by the human mind - an information system which would allow non-linear access for the user and become 'an enlarged intimate supplement to his memory'. These ideas were taken up by people like Douglas Engelbart, Ted Nelson, Alan Kay and others during the 1960s but the

mushrooming of interest forseen by Nelson (72) was only evident during the last six months of 1988.

Some of the basic concepts behind Hypertext are looked at in section 4.1.1. A brief introduction to the Apple Hypercard system is given in section 4.1.2.

4.1.1. The Concepts of Hypertext

Hypertext, at its most basic level, is a DBMS (database management system) that lets the user connect screens of information using association links. At its most sophisticated level, Hypertext is a software environment for collaborative work, communication, and knowledge acquisition. Hypertext products mimic the brain's ability to store and retrieve information by referential links for quick and intuitive access.

Hypertext programs, and the free-flowing databases that are their trademark, have been adapted for electronic publishing, project management, systems analysis, software development, and CAD. Typical Hypertext software consists of a text editor, a graphics editor, a database, and a browsing tool for three-dimensional viewing. Bit-mapped displays, a mouse, windows, icons, and pull-down menus are all standard Hypertext tools. Some of the terminology used in the Hypertext environment includes *nodes*, *links*, *pointers* (or buttons) and *browser* are discussed below.

Nodes

To use a Hypertext system, the user must get used to parsing the information into small discrete units, or nodes, which consist of a single concept or idea. In theory, nodes are both semantically and syntactically discrete. The information contained in a node can usually be displayed on one computer screen. Nodes can come in two

varieties : typed and untyped. An untyped node is a box for information. It has no label or descriptor, so that it can be filled with anything. A typed node is labelled, and the description helps to determine the style of information contained in the node. Types help in classifying nodes and defining specialized operations. They are also helpful in browsing through a database looking for a particular area of interest.

Composite nodes can be formed by combining nodes. These are composed of related subnodes that can be handled as a single object or broken out into individual elements. Icons can be created to reflect the contents of a composite node for easy access. Subnodes can also be rearranged if needed.

Links

In general, links are used to connect the nodes. They are the mode of transportation in a Hypertext network. One follows them to move about between various nodes. One can usually embed them in text and then edit and review them to ensure that they are valid. Link attributes can also be created, deleted or changed. Links must have two qualities : the computer must be able to trace or follow them, and they must be able to transport the user quickly from one node to another. However, links can do more than just connect two nodes. Depending on the Hypertext system, links can connect annotations to a document and provide organizational information, such as where the text fits in a table of contents or where it originated. Therefore, links can help define the node's relationship to other nodes within the database. Links may also clarify the contents of charts and graphics by connecting the graphics to explanatory information such as longer descriptions. Links usually originate at a single point, called a link *reference*. Their destination, called a link *referent*, is usually a node, a chunk or region of text.

Points and Buttons

A point is a single character, token, or icon that points out a link in a document. It is usually identified by either the name of the destination node, the link or an arbitrary string, and by whether it is a source or destination point. Hypercard refers to points as buttons. Buttons can trigger the display of additional information, traverse a link, or activate a program. They can be represented by text or icons.

Browser

The graphical browser is a node that contains a structural diagram of a network of nodes. Browsers usually supply a global map of the network. The browser can be used to orient yourself or to move directly into an area of interest by selecting the appropriate point on the screen with a mouse. While not all systems provide a graphical browser, most attempt to provide some type of overview system that helps you stay oriented in the network and visualize how information is linked.

4.1.2. Hypercard as a Hypertext System

This section gives a brief introduction to Hypercard and Hypertalk. More details of Hypercard can be found in Apple Computer (87, 88) and Shafer (88).

Hypercard, available for the Mac II, the Mac SE and the Mac Plus, is a personal organization tool and a simple database manager. It is also a commercial software developer's tool and is in use in some corporations as a front-end to the mainframe database. This system uses screen-size cards (or window-size cards on the Mac II) organized into topic-related stacks to create simple databases. One card is

displayed at a time. Touching the mouse cursor to a button on a card executes a script written in Hypertalk, Hypercard's programming language.

Hypercard is also a user-friendly application generator and viewer. It may be described as an information organizer and its main benefits are power and simplicity. Written for the Macintosh and utilizing its famous bit-mapped graphics capabilities, Hypercard organizes data and activities around logical "Card Stacks". Each card in a stack has similar structure and functions. It can contain card buttons, card fields and at least one background which can also contain background buttons or fields that are common to all cards in the stack. A card can contain textual, numeric and graphical data and also instructions (scripts) written in Hypertalk. Objects that exist in Hypertalk are stacks, cards, background, 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. The user can browse through already-created stacks, create new cards and stacks, and write and edit Hypertalk scripts.

4.2. THE HYPERSIM SYSTEM

HyperSim is a flexible simulation specification system developed using the Apple Hypercard system. An overview of the HyperSim application is given in section 4.2.1. and the design of the system is discussed in section 4.2.2. An illustrative example of using HyperSim to build up a simulation model is given in Appendix E.

4.2.1. An Overview of HyperSim

HyperSim is a system which allows 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 the three-phase modelling technique 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 Macintosh.

4.2.2. The Design of HyperSim

The main aim of developing HyperSim is to provide the user with a graphical and textual specification option so that he can choose to define the model logic either textually, graphically, or a mixture or both. This is achieved by allowing the user to move around different stacks and constantly update the information that is being entered by the user. If the user makes any changes in one stack, the corresponding entry in any other stacks is automatically updated.

HyperSim maintains a set of global variables which is not seen by the user. It is this set of global variables that enables the user to add any information at any time and place during the building up of the model, with constant updating of the rest of the system in which the new additional piece of information is concerned.

HyperSim contains a set of conventional buttons which is local to the system, appearing in different stacks. Here's the list of buttons which HyperSim supports :

The triangular shape button with a letter 'E' in the middle is called 'New Entity' button and is used for creating a new entity type in the model. The same shape button but with a cross in front is the 'Delete Entity' button which allows the user to delete an entity type specified in the system. Similarly, the rectangle shape button with a letter 'A' in the middle is called 'New Activity' button and is used for creating a new activity in the model. The same shape button but with a cross in front is the 'Delete Activity' button.

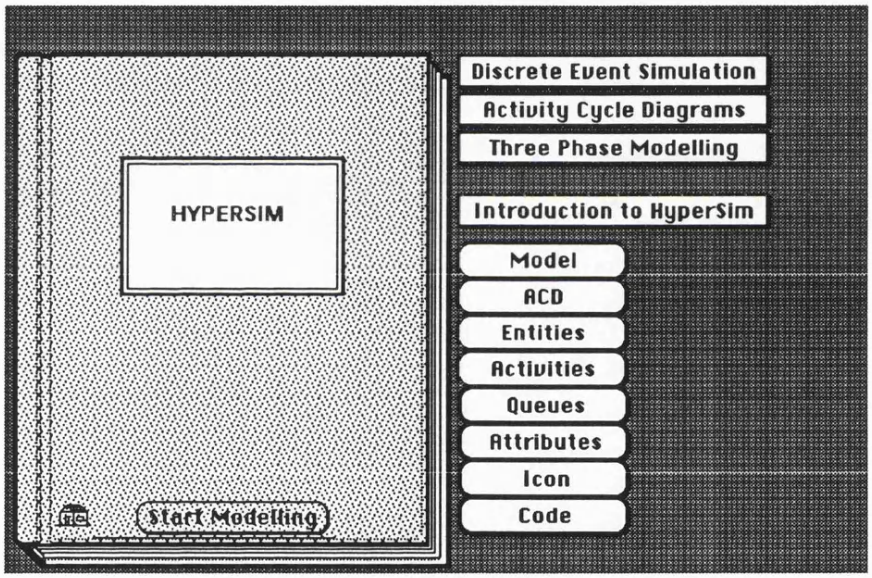
Other common buttons include : the 'GotoACD' button which takes the user to the ACD stack; the 'GotoIcon' button which takes the user to the Icon stack; the 'GotoCode' button which takes the user to the Code stack where a simulation program can be generated; the 'Help' button which takes the user to the 'Reference' stack; and the 'Home' button which takes the user to the Home stack of Hypercard. These common buttons contribute to the flexibility of the HyperSim system, so that the user can do anything - create new data or edit old data of the model - at any time during the specification of the model.

The HyperSim system is made up of nine stacks - the Reference stack, the Model stack, the ACD stack, the Entities stack, the Icon stack, the Activities stack, the Queues stack, the Attributes stack and the Code stack. Here is a summary of the purpose of each of the nine stacks in HyperSim :

Reference Stack

This 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 function of the Reference stack is to offer an on-line help system to assist the user. The user can choose a topic he wants to view by clicking on the corresponding button. If he clicks the 'Start Modelling' button, HyperSim will take him into the Model stack. Figure 4.1 shows the appearance of the Reference stack.

Figure 4.1. The Reference Stack

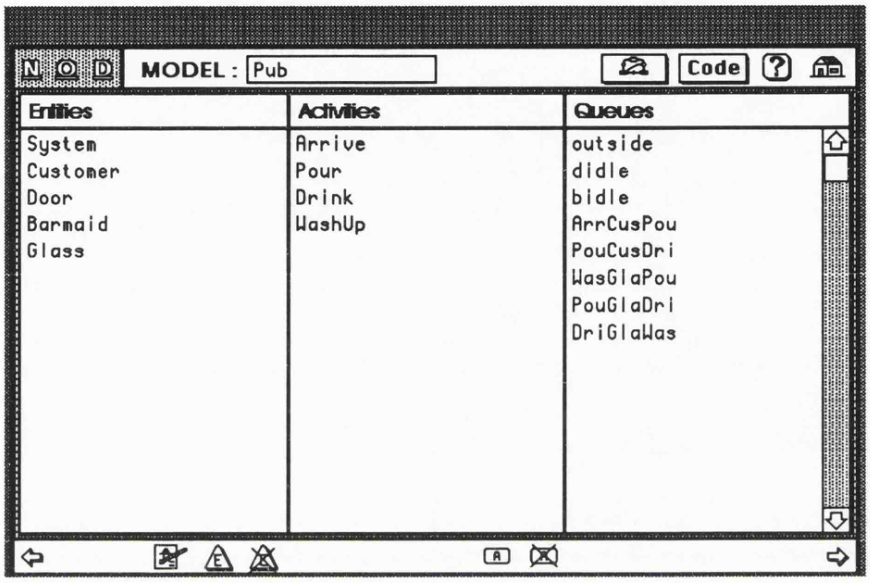


Model Stack

This 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 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. Figure 4.2 shows the appearance of the Pub model card in the Model stack.

Figure 4.2. The Model Stack



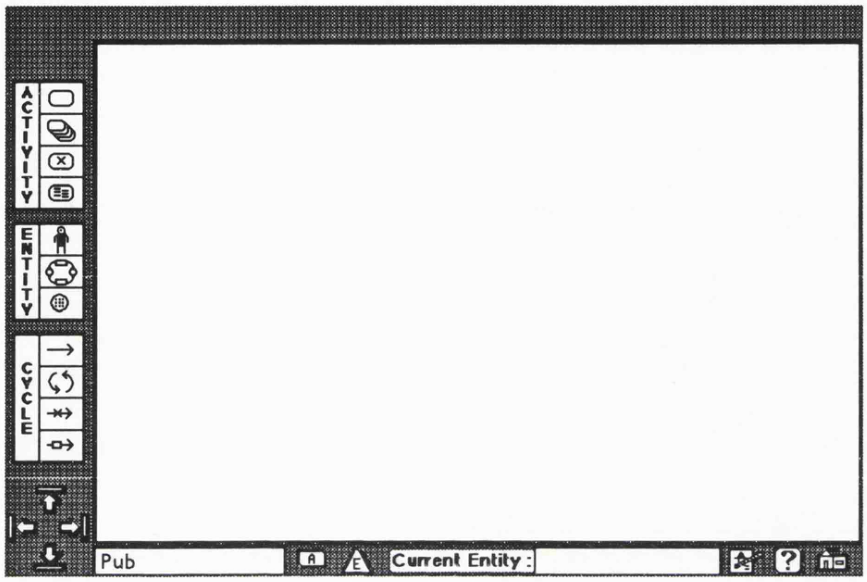
ACD Stack

This 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 system 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 (see figure 4.3).

There are four buttons in the Activity iconic menu. The first button is the 'Select Display Activity' button in which the user can select any one of the activities in the activity list to be displayed on the screen. The second button is the 'Display All Activities' button which allows the user to display all the activities in the model on the screen. The third button is the 'Delete Display Activity' button. The last button is the

'Activity Information' button in which the user can click at the desired activity in the ACD and go to the appropriate Activity card in the Activity stack.

Figure 4.3. The ACD Stack



There are three buttons in the Entity iconic menu. The first button is the 'Select Entity' button which allows the user to import the entity icon onto the screen. The second button is the 'Import Life Cycle' button which is used for importing the life cycle of an entity type which has previously been defined in the system. The last button is the 'Queue Information' button. When this button is selected, all the queues in the model are displayed on the screen. The user can then select any one by clicking on the queue circle and thereby going to the appropriate Queue card in the Queue stack.

There are four buttons in the Cycle iconic menu. The first button is the 'Path' button which is used for drawing life paths of an entity type. The user can click at the activity from which the entity starts and click again at the activity where the entity goes to. A path will be formed and a queue joining these two activities is automatically created by the system. The second button is the 'Facility Path' button which is used for drawing life paths of a facility entity. The user is required to click once at where he

wants to place the idle queue for the facility entity, and then select which activity or activities that this entity type is involved in. The third button is the 'Delete Path' button. The last button is the 'Conditional Path' button, which is used for creating a conditional path for an entity type. The user will be prompted for the condition which is required for the path to be chosen.

Each object that is created on the screen has a card associated with it. For example, a rounded-corner rectangle is associated with an activity card in the Activity stack, and a circle is associated with a queue card in the Queue stack.

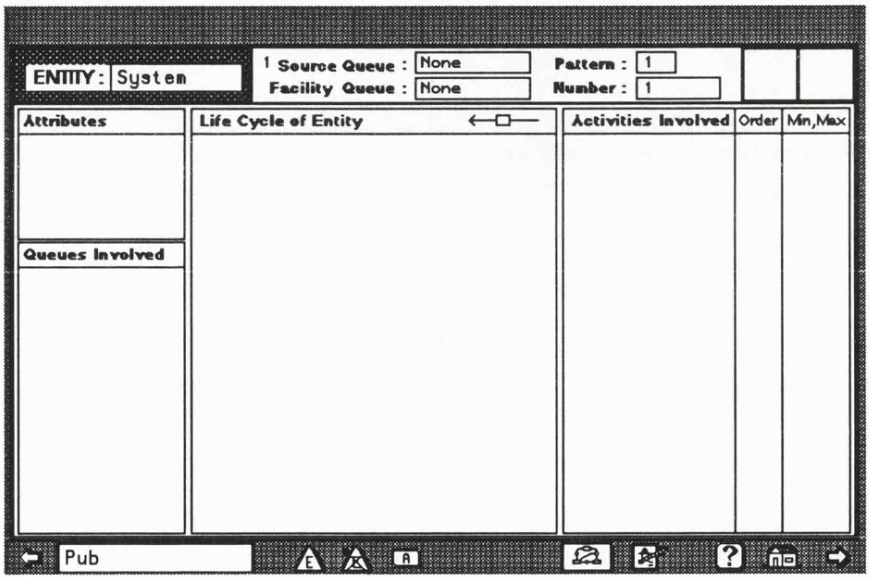
Entities Stack

Each card in this stack contains information of an 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 within 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. Figure 4.4 shows the appearance of the Entities stack.

This stack can be used to define the life cycle of an entity type in the model. If the user clicks at the 'Activities Involved' button on the right-hand side of the entity card, HyperSim will show the list of activities that have previously been defined by the user. The user can then select the initial activity in which the entity is involved. This selected activity will then appear in the 'Activities Involved' table. The chain of activities can be added to until the user selects the initial activity again to close the cycle.

When the list of activities for the entity is completed, the user can click at the 'Life Cycle of Entity' button in which the entity's life cycle will be drawn automatically on the screen. The 'Conditional Arrow' (an arrow with a rectangular box in the middle) button can be used to add a conditional path in the entity's life cycle. This can be done by first clicking at the queue where the conditional paths starts from, and then clicking at the activity where the path goes to. HyperSim will then prompt the user for the condition which is required for the path to be chosen. The path will then be shown in the diagram when the user clicks 'OK' from the condition dialog box.

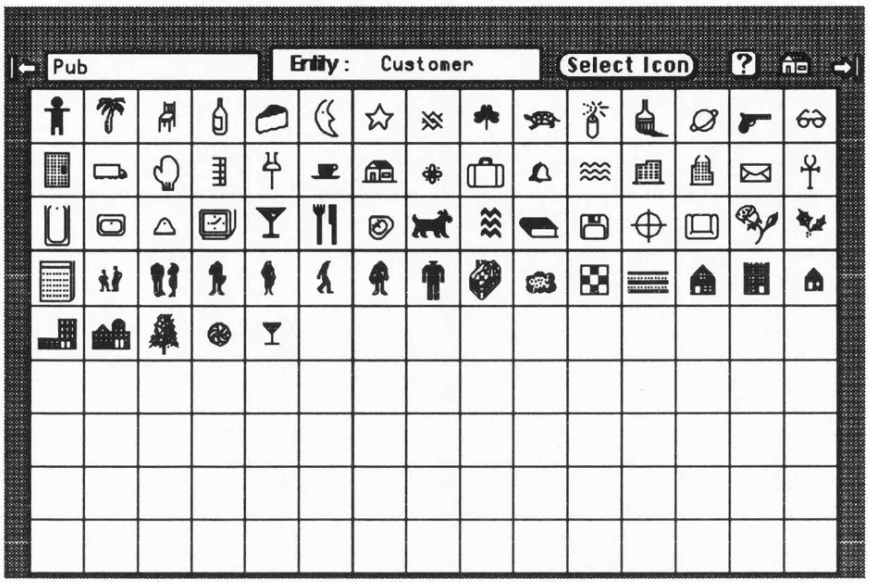
Figure 4.4. The Entities Stack



Icon Stack

This stack contains cards of icons and allows the user to create new or 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. Figure 4.5 shows a card from the Icon stack.

Figure 4.5. The Icon Stack



Activities Stack

Each card in this stack contains information about an activity defined by the user - the name, the formula for the duration, the entities involved and the assignments involved in the activity. Figure 4.6 shows the appearance of the Activities stack.

The model logic can be defined by using the Activities stack. However, this method does not support the automation of queues. The user is required to create queues when an entity is selected for the activity. If the user clicks at the 'Entities Involved' button, then a list of entities that have previously been defined by the user will appear. After the user has selected the entity that is involved in the activity, HyperSim will prompt the user to enter the name of the queue where the entity comes from, and the name of the queue where the entity goes to after the activity is finished. In this stack, an entity can be selected more than once since there might be more than one possible path that the entity can go to after the activity is finished, depending on the

condition. The system will prompt the user for the condition and the queue that the entity goes to when it detects that an entity is selected more than once.

Figure 4.6. The Activities Stack

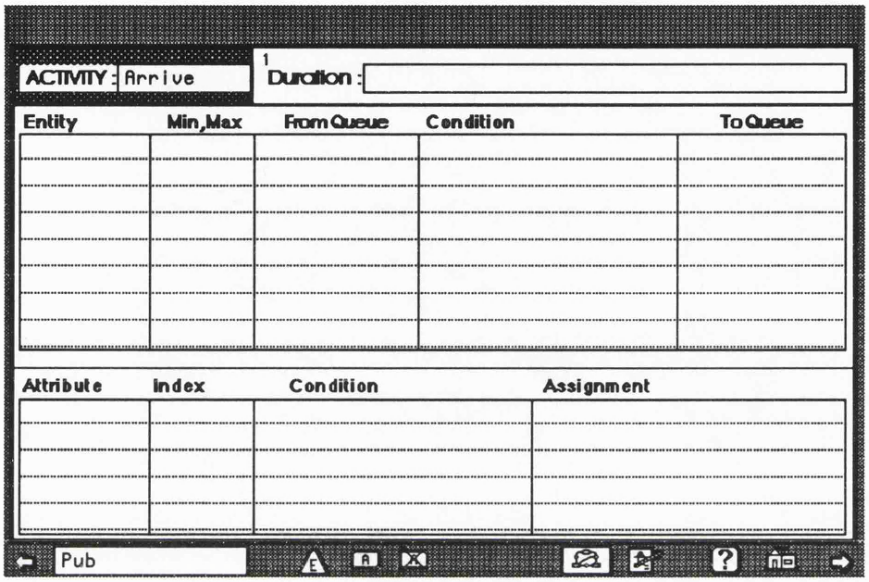
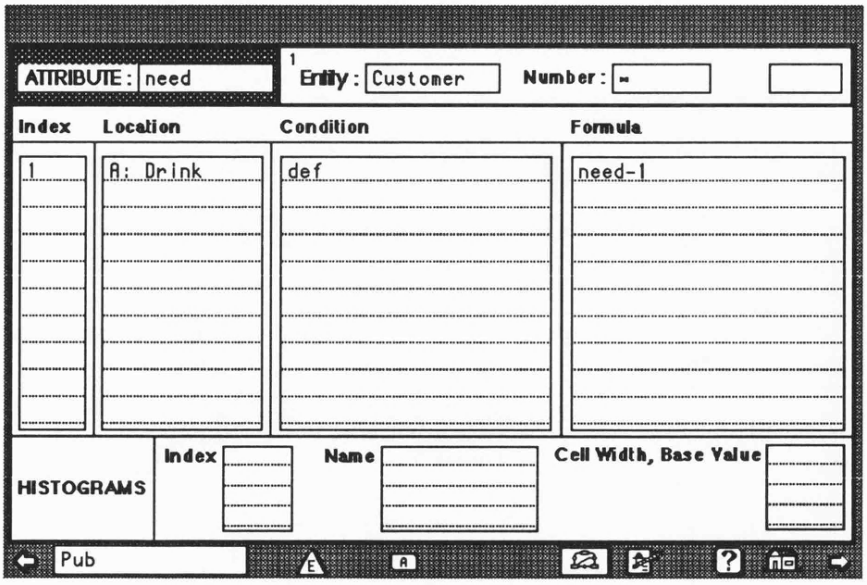


Figure 4.7. The Attributes Stack



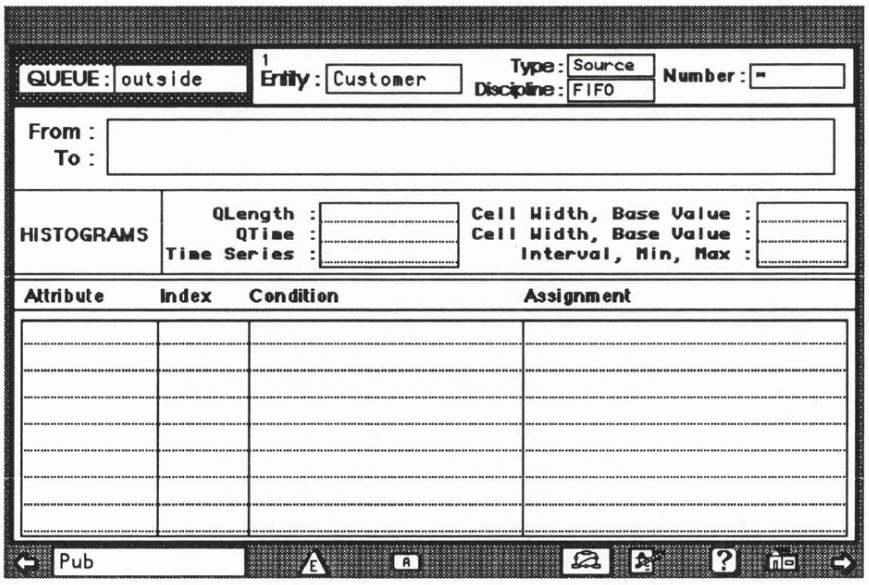
Attributes Stack

Each card in this stack contains information about an 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. Figure 4.7 shows the appearance of the Attributes stack.

Queues Stack

Each card in this stack (figure 4.8) contains information about the queues which are automatically produced by HyperSim - the name, the entity type that it belongs to, the number of entities in the queue, the activity from which an entity comes, the activity to which an entity in the queue goes, the assignment of any attributes, and the histograms that are recorded in the queue.

Figure 4.8. The Queues Stack



Code Stack

This stack allows the user to generate a three-phase simulation program from the specification of the model by using the 'Generate' button. 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. Figure 4.9 shows the appearance of the Code stack.

Figure 4.9. The Code Stack

4.3. EXPERIENCE GAINED FROM HyperSim

This section discusses the advantages of the HyperSim system (section 4.3.1) and its limitations (section 4.3.2).

4.3.1. Advantages of HyperSim

The flexibility of allowing the user to specify the model interactively either by textual or graphical description, or a mixture of both, is the main advantage of HyperSim. The user can look at the model logic from different points of view. He can either look at the overall structure of the model, or at information on individual entities, or on individual activities.

The user can change information in any place at any time, and the system will automatically update the corresponding information in different stacks in which this piece of information is concerned.

HyperSim is so user-friendly that it could take a new user less than an hour to master the system. The building up of the model is entirely dependent upon the user's action. However, the system constantly carries out checks which aids the user in providing the correct type of information in the corresponding entry.

The entry of the model logic is not required in a systematic way. The user can define new entities and new activities at any point during the building up of the model in the system. He can also edit existing information in any stack at any time.

HyperSim is very useful for developing a first trial model in a very short period of time. This trial model can then be easily updated until the user is satisfied with the model logic and the data entered. The code generator in HyperSim allows the user to generate a simulation model by just the touch of a button. This generated program can be modified inside HyperSim, and transported into Turbo Pascal on the Macintosh. It can be compiled with the simulation library MacSim.Lib so that simulation runs can be performed.

4.3.2. Limitations of HyperSim

The main disadavantage of using HyperSim is its lack of robustness when data is being updated between stacks. This is a general deficiency in all Hypercard applications. A Hypercard application runs more efficiently on the more advanced Macintosh models - SE/30 and IIci. Moreoever, for such a system to be effective, large amounts of memory space are required.

A danger in using the HyperSim system is that the user might easily get lost in the system if he accidentally clicks at the wrong button. There are many buttons on the screen which might become traps for the user. In order to minimise this error, a set of conventional buttons is adopted throughout the design of the system, i.e. buttons that perform the same function have the same appearance in different stacks so as to avoid confusion and misunderstanding.

Another disadvantage of Hypercard is its limitation on the size of the card. The screen is non-scrollable. To create an activity cycle diagram that is larger than the size of a card, special programming is required so as to link different cards together to form a big picture.

Hypertalk programming is simple but it is easy to lose control. In order to maintain the flexibility and mobility between different stacks, one has to keep track of global variables manually. Moreover, it does not support dynamic memory allocations, i.e. pointers or handles. Hence, for large models, searching through records and code generation is slow.

The HyperSim system does not allow the user to run the simulation model directly from the specification. In order to achieve this, external procedures and functions must be linked to the HyperSim system so that the simulation model can be run at a reasonable speed.

4.4. SUMMARY

Hypercard is well recognized as a powerful organizer 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 method of file management. Instead of organizing 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.

There are a lot of advantages in using Hypercard to develop a system like HyperSim. Firstly, a high degree of mobility between stacks is easily achieved by using buttons and fields. Different stacks can be linked together to allow easy assess for the user just by a click of the button. Secondly, 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. Thirdly, good graphics facilities allow pictures to be drawn easily 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. Lastly, Hypertalk programming is simple and easy to learn. Its object-orientated nature allows the user to develop applications and prototypes very quickly.

However, Hypertext is an immature technology with many problems yet to resolve. Perhaps the most difficult part of creating a Hypertext system is not the building of the user interface, but the creation of sound underlying data models that can be maintained. Since Hypertext systems need to be maintained, we have to watch for uncontrolled linkages which will become maintenance problems. Just as large software programs with many patches can turn into spaghetti code, so a Hypertext system can turn into a morass of meaningless, obscure connections and references. Another problem for some users is that some Hypertext systems give you control, when in fact, one might need guidance.

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.

On the other hand, simulation models require extensive runs to determine the result of what is essentially a stochastic experiment. Hypercard does not provide the speed of execution required for such experiments. Hence, HyperSim forms the basis for an automatic simulation program generator. This generated HyperSim simulation program can be run in conjunction with the simulation library 'MSim.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 theoretically possible to return the experimental results to HyperSim to gain the advantage of a Hypertext system for these purposes. This latter work has not been implemented since a new system, described in the following chapter, and based on the experience of developing HyperSim, has replaced HyperSim.

The main advantage of using computer graphics is to break the communication gap between the analyst and the end-user. It also provides a means of understanding the simulation model and easy interpretation of the results. This research has given an insight into the production of a graphical interface that enables the analyst to formulate a client's problem with the client. This would be more than a specification system. A specification would be the product of the analyst-client system session. But the graphical interface would have the versatility to enable, in a non-technical way, the problem to be described in the user's terms.

CHAPTER 5

DESIGN OF THE GRAPHICAL SIMULATION MODELLING ENVIRONMENT

With the rapid development of computer hardware in recent years, the possibility and the flexibility which allows the development of intelligent user-friendly graphics oriented software has genuinely increased. The increasing use of computer graphics is having a significant impact in the area of simulation. Visual modelling is now regarded as a powerful component to an analyst's problem solving capabilities.

In this chapter, we discuss the design of an ideal graphical simulation system which allows the user to formulate the problem in a pictorial format. The design objectives of our graphical simulation system are given in section 5.1. The components of the proposed system are outlined in sections 5.2. In order to demonstrate our proposed intelligent graphical simulation system, we have developed an application called MacGraSE (**Mac**intosh **Gra**phical Simulation **E**nvironment) on the Apple Macintosh. This application is introduced in section 5.3. Section 5.4 summarises this chapter. An example of how to use the MacGraSE application using the steelworks model (discussed in section 1.2.4) is given in Appendix F of this thesis.

5.1. DESIGN OBJECTIVES

The existing simulation specification methods normally require a translation process inside the human brain from the modeller's visualisation of the real world system to a logical representation of the model (for example, in terms of diagramming techniques or macro languages). This research aims to design a simulation environment

which ideally allows the client and the analyst to build up a simulation model in a collaborate manner. Such a system enables constant reconstruction of the model and, at the same time, aids the user in creating a more clear picture of his problem during the model building process.

The concept behind the proposed system is simple. This is an evironment which allows the user to draw what they think directly on the computer screen during the building up of the model, without the need to translate the modeller's thoughts to an initial logical structured form. However, the system should be able to help the user to organise their information in a structured format during the construction process.

The objective of the system is to allow the user to draw a pictorial representation of the real world system and then formulate the model logic within such an environment. The system should enable the user to update his specification of the model constantly, so that he can repeatedly test run it and change the appropriate data, until the final experimental model is obtained. The pictorial description of the model can be used for running a visual simulation. The user should be able to select either a visual run or a text run. He should be able to design his desired output screens (in terms of pictures, graphs or charts) so that he can watch the dynamics of the model during a simulation run on selected parameters.

5.2. COMPONENTS OF THE GRAPHICAL SIMULATION SYSTEM

In this section, the term 'user' refers to the one who is using the graphical simulation system to build a simulation model. The role of the user here includes the role of the modeller and that of the end user of the system. A general view of the design components of this proposed simulation environment and their functions are given in this section.

One way that the human mind tries to model a real world system is by a

dynamic picture, i.e. an imaginative image of how entities move and interact in the system. The user attempts to translate this image into a diagrammatic or textual description. This description is then used as a means of communication between parties of interest who are involved in the model building project. For a large and complex system, the user might have to go through a great number of amendments to the formulation description before the final model definition sychronizes with the image in his mind. If the user can see the image of the real world system in front of the computer screen, he might easily draw the dynamic picture that is inside his mind with the right software support tools.

The following sub-sections describe the components of a system that allows the user to draw such a picture.

5.2.1. Background picture

Any picture will probably have a fixed background that provides context for the dynamic picture. Figure 5.1 shows two types of background picture that can be drawn on the screen. Figure 5.1a is a three-dimensional picture of the pub and figure 5.1b is a factory layout diagram of the steelworks.

The facility of having a background picture in our proposed graphics driven simulation system is to enable the user to depict what is inside his mind visually on the screen. Once a pictorial description of the real world system is drawn, it is like a background that is being stuck on the back of the computer screen. The user can then easily define the movement of entities that are involved in the system by moving the icons which represents individual entity types on the screen, but without altering the appearance of the background picture. On the other hand, he can redraw the background picture in any way he wants, without altering the model definition that he has previously described.

Figure 5.1a. The Pub Picture

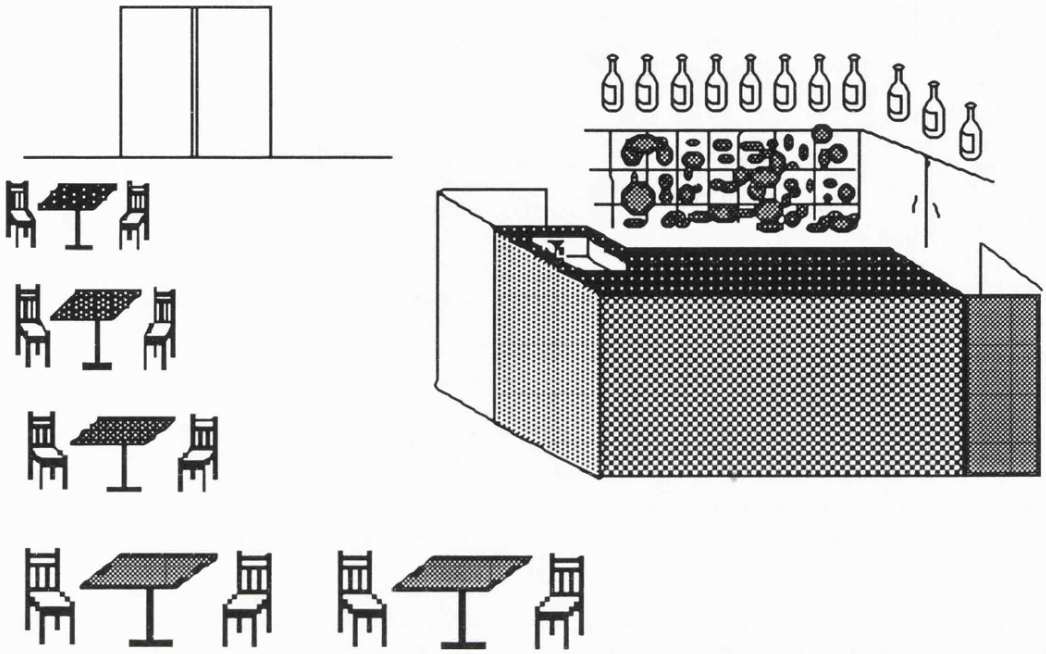
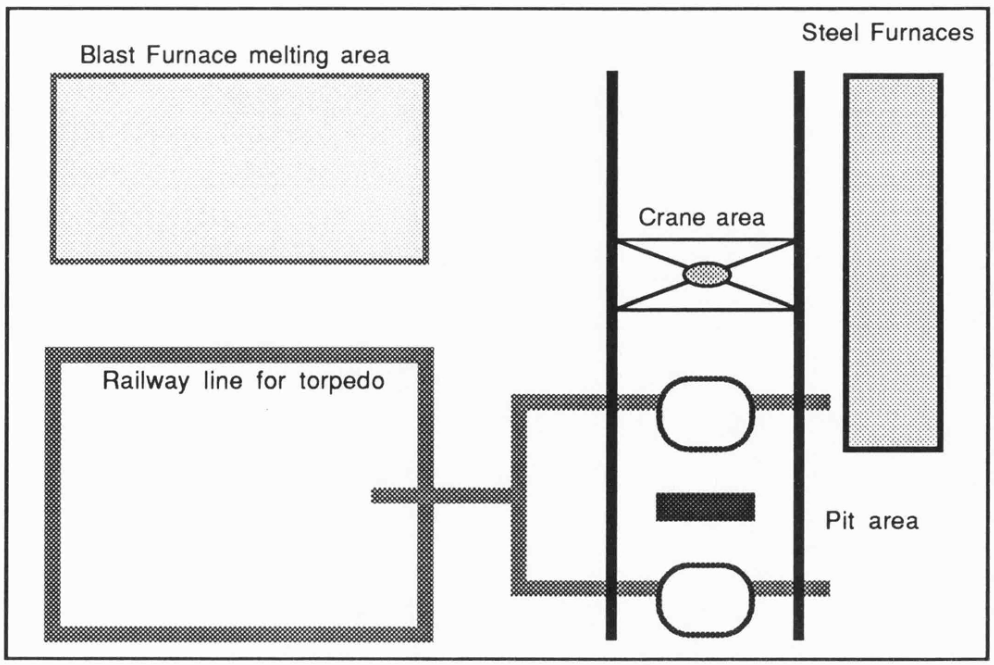


Figure 5.1b. The Steelworks Picture



5.2.2. Objects on Screen

There should be at least two types of objects on screen - icons which indicate entities and rectangles which indicate activities. The display of queues and attributes are optional. Queues are represented by circles. Attributes can be represented by means of any polygon. In our case, we use a cloud symbol to represent attributes.

Entities refer to any component of the model which can be imagined to retain its identify through time. In other words, they are the main input resources of the system. The way entities move about in the system forms the basic model logic. In order to make the pictorial description look as close possible to the real life picture of the system, each entity type should have its own iconic representation in the model. The model logic can then be defined by using these icons on the screen. The use of icons makes the visual simulation more vivid and is a better representation than character graphics.

An activity involves the interaction of different entity types. It is represented by a rectangle on the screen. The user should be able to expand or shrink the size of the rectangle so that entity icons can be placed within the rectangle to indicate that they are involved in the appropriate activity. An activity can also be represented by an action picture which can be used during a simulation run.

A queue represents the idle state of an entity type during its life cycle in the system. It is represented by a circle on the screen. The automation of queues is desirable since it minimises input error and guarantees that all of the life cycles are closed. A dimmed icon of an entity type can also be used to indicate its idle state. The user can select the option of displaying the iconic idle state during a simulation run.

If attributes are represented on the screen, the user will be able to see how the assignments of attributes flow through different activities of the system. This option is useful when the simulation model is heavily attribute-based.

5.2.3. Generation of ACD

Although an activity cycle diagram is incapable of depicting all the details for large and complex system, it is a most precise way of displaying the model logic for most discrete event simulation models. The ideal system should support the automatic generation of an activity cycle diagram from the pictorial description of the model. The generated ACD can be used to verify the model logic and act as a means of parameter input. Moreover, the user should be able to run a simulation model in its flow diagram form, if preferred.

The option of generating an ACD shows that a structured diagram can be directly translated from the pictorial description of the model. It is worthwhile to mention that it is possible to generate other structured diagrams, for example, entity cycle diagrams (Pidd 88) and activity diagrams (Davies & O'Keefe 89), from this pictorial model definition. Each diagramming techniques has its own methodology, definition and emphasis. However, the pictorial description is common to human thinking since it represents what is happening in the real world system.

5.2.4. Logical Description of Data

The structured form of the model formulation should be automatically generated by the system so that the user can view the logical form of the data input at any time during the construction process. For example, a flow diagram showing how an attribute is evaluated throughout the system among different activities. Alternatively, the user might want to see a structured diagram which shows how different attributes are evaluated within an activity. The system should thereby provide this kind of data display so that the user can enter formulae or evaluation rules in a structured fashion, if required. This component is particularly useful when the system is large and complex.

5.2.5. Program Generation

The automatic coding of a model from a system specification reduces the need for error-checking. However, simulation models are user-defined. No existing generic simulation packages can accomodate all the details that the modeller wants to specify in the model. This is the tradeoff between using a general purpose simulation package and building a tailor-made simulation program specified to the problem. A program generator, used in conjunction with a set of transparent library routines, can be used to compensate for this dilemma. For the higher-end users who require complex specification rules of the model, the system can be used to construct a first basic model. Any other details that are not covered by the specification system can be added to the generated program. For the lower-end users who just want to build a simple model, the generated code will tell the story about 'how things actually work inside the system' and thereby increase the simulation modelling knowledge of the user.

5.2.6. Visual Simulation

Visual simulation is an essential part of a simulation system. Although one might argue that no one will actually sit in front of a computer screen and watch the visual simulation run once the final model is obtained, the system should provide the user with the option of visual running.

In this context, we distinguish two types of visual simulation run - a state simulation run and an animation run. The former refers to a state change display in a pictorial form, for example, where the run screen shows the start or completion of an activity. Animation refers to the actual physical movement of each entity that is present during the simulation run. Although an animation run shows a clearer picture of the dynamic movements within the system, it is time consuming and should only be used for verification of the model.

Visual simulation is a very useful tool to aid the customer understand the system if explained properly. A short running period with a reasonably low speed is a convincing tool to show that the model is working correctly according to the final user-defined specification. Moreover, with the presence of the background picture behind the movement of entities, visual simulation looks more real and alive. Any error in the definition of model logic can thus be easily detected in such an environment.

5.2.7. Simulation Run : Output display

Once the computer model is verified, assuming all parties of interest agree to this final version of the computer model, the next stage is to collect experimental data so that data analysis can be performed. At this point of the simulation process, the user of the simulation model is concentrating only on the output results (for example, how resource usage varies during the simulation run). Very often the user prefers to see a graphical representation of the data distribution by means of a line graph, bar chart, scatter plot or histogram, than to see numerical output. Interactive graphical output during a simulation run might be an attractive tool to the modeller so that he can see how the data accumulates during the running process. Visual simulation cannot satisfy this type of data requirement efficiently since it is impossible for the human mind to capture all the details during a simulation run visually.

In our design of a complete graphics driven simulation environment, the user should be able to select the output he wants to see during a simulation run by means of an output screen. This output screen option of the system also enables the user to check the logic of a particular part of the system by looking at the corresponding data that is generated during the running process. Due to the restriction of the size of the computer screen, it seems almost impossible for the user to display all the output he would like to view on one screen. Hence, the system should enable the user to design different

screens so that he can switch to the desired screen during a simulation run in order to see what is happening in another part of the system.

5.2.8. Simulation Run : Text display

A simulation run with text display, indicating which activities start and end at each time advance, is another useful tool in the verification of the computer model. If the entities of each entity type are labelled with an index, the user can check which entity of the entity type is actually taking part in the activity during a simulation run.

When the user is only interested in the final output of the simulation run, the option of switching off all the displays on the screen during the running process can be exercised. This will increase the speed of running proportionally, especially with a long running period. The user can then collect the report that is generated after the run.

5.2.9. Report Generation

Simulation reports are the output of the whole simulation process, and will be handed over to the management decision making team of the organisation for which the simulation model was built. The results presented in the report will help the decision maker in the decision making process and should therefore be presented in a clear and accurate format. The ideal system allows the user to design the format of the report so that it will contain the appropriate level of detail according to the requirement of the management team.

5.2.10. Built-in Macro Language

A built-in macro language is often used for incorporating user-defined

routines or procedures that are not covered by the specification system. For example, the user might want to write a procedure for calculating the duration time of an activity, or he might want to add in a form of output that is not automatically produced by the system. Macro language allows the user to write this type of coding inside the system. The system will then act as an interpreter so that this level of detail is included in the simulation run. This will not require the user to modify the generated coding outside the system in a high-level language such as Pascal or C. The macro language should be simple to use, preferrably in a structured English format, so that the user who has a limited amount of programming knowledge can use it with little difficulty.

5.3. THE MACGRASE APPLICATION

MacGraSE (Macintosh Graphical Simulation Environment) is a graphical simulation system developed on the Apple Macintosh, incorporating most of the design structures described above. The application can be opened by double-clicking on its application icon or selecting the Open command from the File menu on the Macintosh desktop. The system uses the three phase modelling structure, discussed by Crookes et. al. (87), Paul (89a), Pidd (84) and Pidd (88). The interface design of the MacGraSE application is given in section 5.3.1, and a description of the main menus is given in section 5.3.2. The formulation mechanism is described in section 5.3.3. An example walkthrough of the application can be found in Appendix F.

5.3.1. The Interface

Once the MacGraSE application is opened, three windows will appear on the screen (figure 5.2). The top left window is the mode box window which consists of six palette buttons (figure 5.2a) that are used for modelling. The bottom left window is the tool box window which contains all the tools palette buttons (figure 5.2b) that are used

for drawing the background picture. The window labelled 'Untitled' is the application window or main window where a model can be constructed. Figure 5.3 summarises the appearance of different objects on the screen.

Figure 5.2. The MacGraSE Application

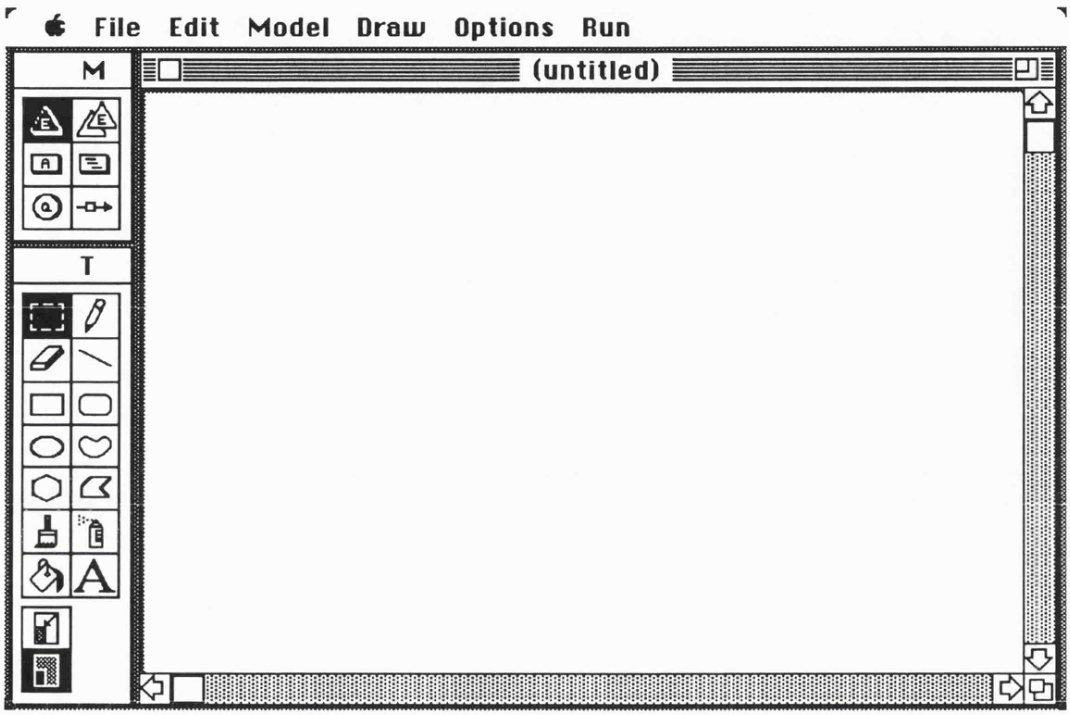


Figure 5.2a. The Palette Buttons in the Mode Box Window

Figure 5.2b. The Palette Buttons in the Tool Box Window

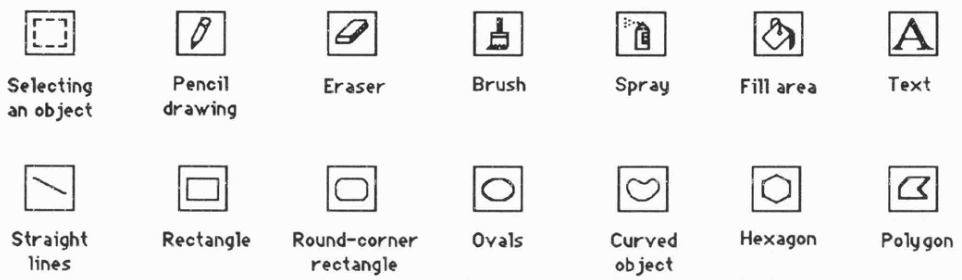
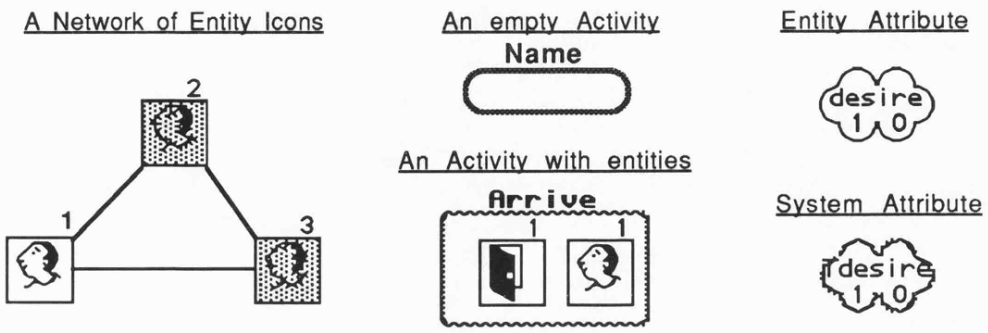


Figure 5.3. Objects inside the MacGraSE application



There are six palette choices in the mode box window. Once the 'Select entity' button is selected, MacGraSE will redraw the screen by showing all the entity icons that are present in the model. Each entity icon is surrounded by a square frame (25 x 25 pixels).

The 'Duplicate entity' button is used for duplicating entity icons of an entity type. The icons list on the screen is used to indicate the life path of the entity type inside the system.

The application will redraw the screen by showing all the activities that have been defined in the model if the 'Select activity' button is selected. The user can then move or resize the activity rectangle anywhere within the main window. He can also specify information for the activity by double-clicking on the activity rectangle and entering the relevant information in the activity dialog box.

The 'Select attribute' button is used for modelling the attributes of the system. Once this button is selected, the application will redraw the screen, including all the attribute evaluations that are linked to each activity. If it is an attribute that belongs to an entity type, the user can link the attribute object to an entity icon on the screen. If it is a system attribute, then the user can link the attribute object to an activity object on the

screen. Duplicate attribute objects of an attribute type can also be created so as to indicate the evaluation flow among and within activities in the model.

The 'Select queue' button is used for displaying the queues that are automatically generated by the application. The user can then move the position of the queues and enter the relevant information by double-clicking on the selected queue.

The 'Conditional path' button is used for adding conditional paths within an entity life cycle. Once the button is selected, the application will redraw the model picture with the generated queues. The user can create conditional paths by moving the mouse to an entity icon, click to select it, and then by dragging the mouse to the destination queue and releasing the mouse button. A line linking the selected queue and activity will be drawn, and the condition for the path can be entered by clicking at the small box on the line.

5.3.2. The Main Menu

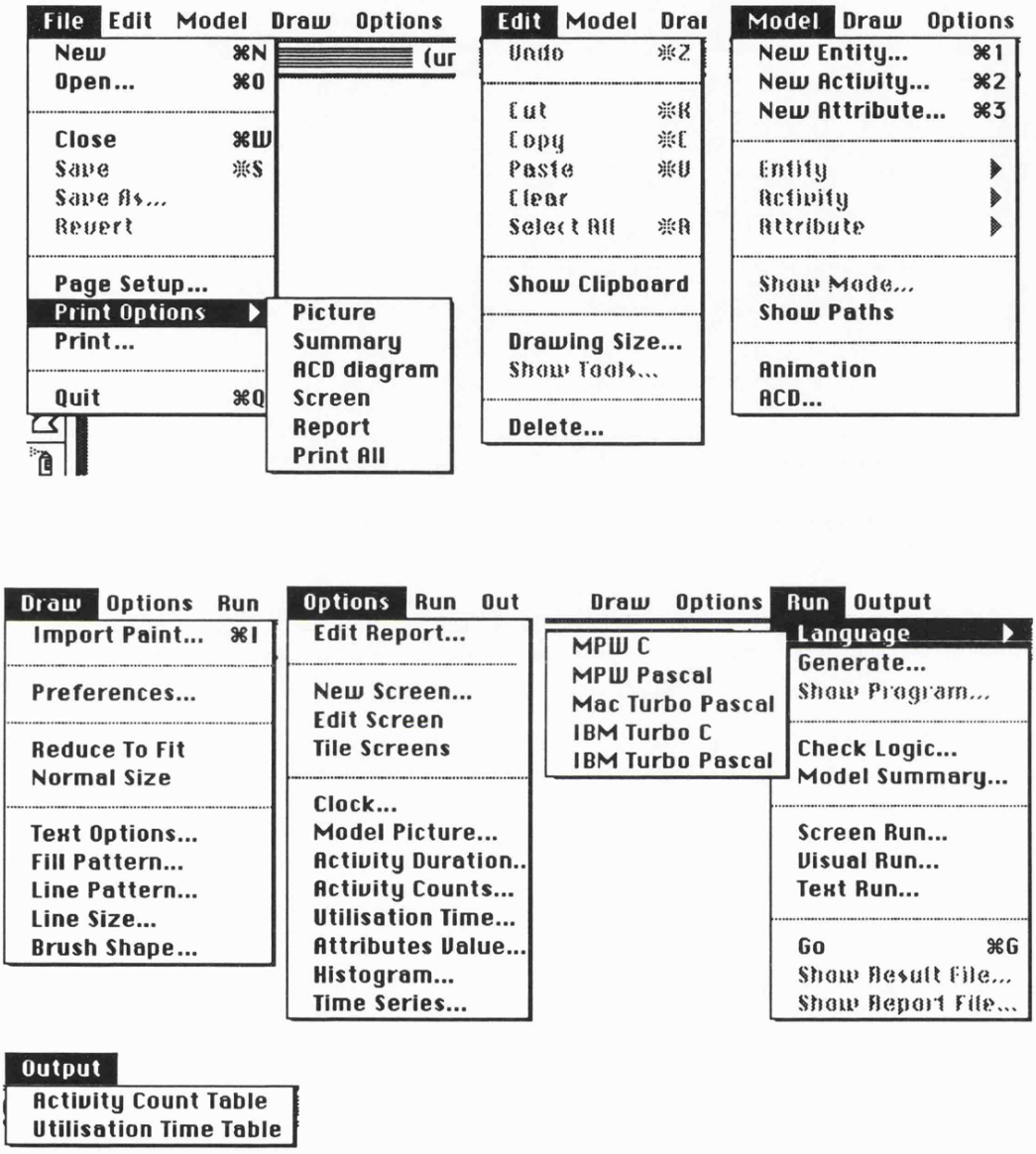
There are seven menus in the main menu bar of the application, excluding the standard apple desk accessories menu (). A diagram showing the appearance of the menus inside the MacGraSE application is shown in figure 5.4.

Like a conventional Macintosh application, MacGraSE contains the typical File and Edit menus.

The File menu handles all the filing procedures within the application, including creating new files and editing existing files, printing and quitting the application.

The Edit menu contains the cut, copy and paste command to aid the editing of the background picture and textual data input. The user can also select the drawing size of the model and delete the selected components of the model.

Figure 5.4. The Main Menu of MacGraSE



The Model menu contains all the commands for building up a simulation model. New components - entity, activity or attribute, can continuously be added to the model by using the 'New Entity', 'New Activity' and the 'New Attribute' menu items respectively. These components will be appended to the appropriate submenu (Entity, Activity and Attribute) within the Model menu. The user can view and edit information for each individual component by selecting the item in the submenu. Other options include setting the 'Show Path' mode on, so that the paths (in terms of straight lines) of

the entity type that the user is constructing are seen on screen; generation of an ACD which allows the user to manipulate data via an activity cycle diagram; and the animation command, which allows an animation run where the user can see the actual movement of individual entities that move inside the system during a simulation run.

The Draw menu is used to aid the drawing of the background picture. The user can set the pen size, pen pattern, fill pattern or text options by using the appropriate commands inside the menu. It includes an 'Import Paint' command in which the user can import a picture that is drawn in other Macintosh drawing applications in 'Pict' format, instead of using the drawing facilities inside MacGraSE.

The Options menu has two main functions. The first function is report editing, so that the user can edit the report format to produce a desired report after a simulation run. The second function is screen editing in which the user can create multiple output screens for a simulation run. The objects that can be put in a screen are mainly the simulation clock, utilisation time table, activity count table, histograms, time series, numerical statistics, attribute values and status of an entity type.

The Run menu is used for performing simulation runs on the model. The user can use the 'Generate' command to generate a three-phase simulation program in the selected language choosen in the 'Language' command. The program can be viewed within the application by using the 'Show Program' command. The user can also check the model logic and see a textual description of the current state of the model by the 'Check Logic' and 'Model Summary' command respectively. There are three options for a simulation run - Screen, Visual and Text. When the parameters inside each option are entered, the user can select the Go command to process the simulation run. In each case, the user can select the options of saving the result file and the report file of a simulation run. The result file can be viewed after a simulation run by using the Show Result command, whereas the report file can be viewed using the Show Report command in the Options menu.

The Output menu is only appended to the menu bar when a simulation run is executed by the user. Any specified histograms, time series or graphs are added to this menu, so that the user can select the output to be reviewed after a simulation run. The default setting of this menu includes the utilisation time table and the activity count table.

5.3.3. The Formulation Mechanism

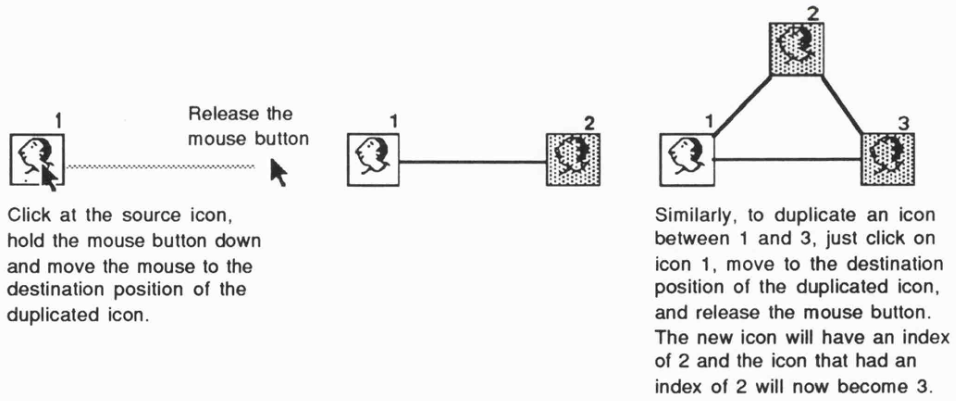
The proposed simulation system models three types of objects on the screen - entity, activity, and attribute. An entity can either be active, i.e. taking part in an activity, or inactive, i.e. waiting for an activity to start. An activity usually involves the cooperation of different classes of entity. The term 'queue' is used to refer to the idle state of the entity, i.e. a state in which the entity waits for something to happen. Hence, a queue involves no cooperation between different classes of entity.

Each entity type has its own life cycle pattern which is made up of a sequence of activities in which it participates in the system. An entity life cycle must be closed unless the entity is of temporary type, in which case the entity is created when it enters, and destroyed when it leaves during a simulation run. We also distinguish between permanent and facility entities. The former has its own individual different waiting point before an activity that it is involved in starts; whereas the latter has a common resting place before any of its associated activities starts. Some paths of an entity type's life cycle may be conditional, i.e. there is a certain condition which controls the feasibility of the path. A condition is usually the value of a certain 'attribute' that is possessed by the corresponding entity type. An entity type can have more than one attribute and its attributes can be evaluated at any point along its life cycle.

Entity Life Cycle

A new entity type can be created by using the 'New Entity' command in the Model menu. An entity type is represented by an icon within a square frame (25 x 25 pixels). Each entity type has one and only one main icon and multiple 'ghost' icons which are images of the movement of the entity within the system. Ghost icons are indexed by the number that is just above the top-right hand corner of the icon frame. A ghost icon of an entity type can be duplicated from the main icon, or another preceeding ghost icon of the same entity type. The entity life cycle can be constructed by duplicating icons of the entity type and then placing each icon into an activity in the system. Figure 5.5 shows the duplication of icons for an entity type.

Figure 5.5. Duplication of Entity Icons

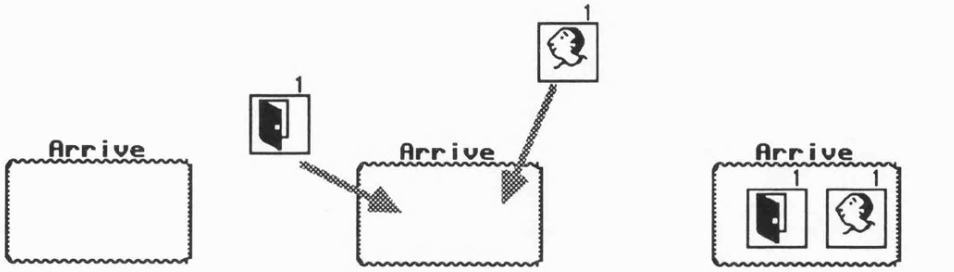


Specifying an Activity

Each icon on the screen should be placed within an activity and should not be allowed to wander around in the system. A new activity can be created by selecting the 'New Activity' command from the Model menu. An activity is represented by a round-cornered rectangle on the screen. Entity icons that lie within the boundary of the rectangle represent the different entity types that are involved in the corresponding

activity. There are two ways of specifying an entity that takes part in an activity. The first method is by moving the entity icon into the activity rectangle, and the second method is to pick up an entity icon by moving the activity rectangle over an entity icon. Figure 5.6 shows how to specify entities that are involved in an activity.

Figure 5.6. Specifying an Activity



An empty activity.

Click at the door icon, hold the mouse button until it is inside the rectangle of activity arrive. Release the mouse button. Similarly for the customer icon.

Activity 'Arrive' involves the co-operation of the entity door and customer.

Editing Queues

Queues are automatically produced when a ghost entity icon is duplicated from the source icon. This is because an entity icon on the screen is always assumed to be always placed inside an activity. Thus a queue is associated with every entity icon on the screen which indicates where the entity icon comes from before entering the activity that it is involved in. For both temporary and permanent entities, one queue is generated for every duplicate icon and the last entity icon of an entity type is always linked to the queue associated with the first original icon of the same type. This is to ensure that the life cycle is always closed. For facility entities, only one queue is generated for an entity type.

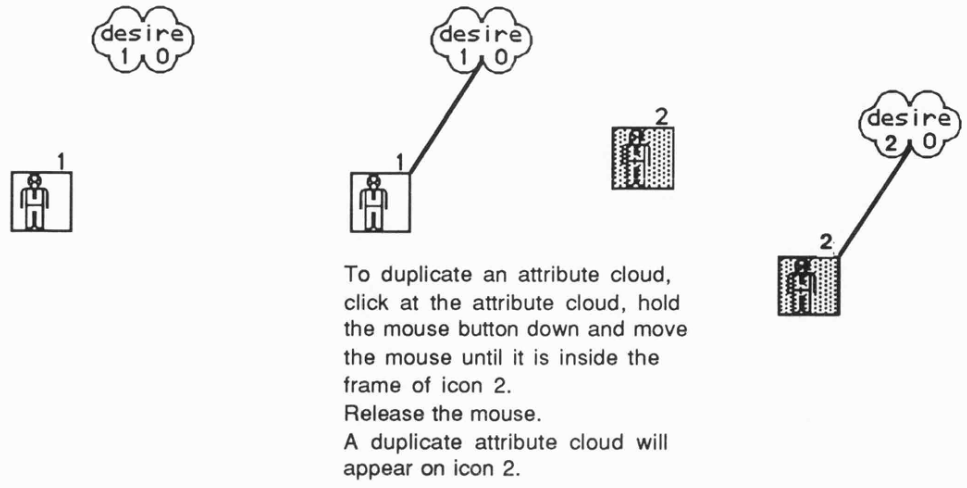
The generated queues are initially invisible. However, the user is able to see the queues in the diagram (represented by circles) by using the 'Select queue' button in the mode box window. Information about the queue can be entered if the user double-

clicks at the selected queue. He can enter the number of the entities that are present in the queue, or create any histograms or time series that are required in a simulation run.

Attribute Assignments

A new attribute can be created by selecting the 'New Attribute' command in the Model menu. An attribute is represented by a cloud shaped object on the screen. For attributes which are evaluated at more than one activity within the system, an image of the attribute can be duplicated so that the user can enter a different formula. Figure 5.7 shows how an entity attribute object can be duplicated in the model. There are two indices inside an attribute cloud. The left-hand corner index indicates the rank of the cloud among all its associate duplicates, whereas the right-hand corner index indicates the order of attribute evaluations within the activity in which the cloud is linked. In figure 5.7 it can be seen that the left-hand corner index of the duplicate attribute cloud carries an index of '2', whereas the original attribute cloud carries an index of '1'. The right-corner indices of both clouds are '0' since their associated entity icons are not yet placed in an activity.

Figure 5.7. Duplicating an Entity Attribute object



Attributes can be classified into two types : system and entity attributes.

System attributes are attributes that belong to the system and can be evaluated at any event that is happening in the system. Entity attributes are ones that belong to an entity type and can only be evaluated at the event that the entity type is involved in. A system attribute is represented by a cloud object drawn with a thicker pen, whereas an entity attribute is represented by a cloud object drawn with a thinner pen.

5.4. SUMMARY

In this chapter, we have discussed the design principles that lie behind our ideal graphical simulation system. The key feature of our system is to allow the user to specify a logical pictorial description of the model, and to be able to use this picture directly for a visual simulation run. The ideal system should have a program generator, so that any level of details that are not covered by the specification interface can be added to the generated code. A report generator is an important component of the system, so that user-defined simulation reports can be generated from a simulation run. A built-in macro language is desirable since this will allow the user to add user-defined routines within the application itself. This will increase the flexibility of the system.

We have also introduced part of the application MacGraSE (Macintosh Graphical Simulation Environment) which lies within the scope of our design guidelines. The interface design and the formulation mechanism allow the user to construct and build a simulation model in a very simple and quick way. The client is able to discuss the specification of the simulation model with the analyst and, at the same time, construct an initial basic model showing the logic of the system. The system allows the user to see a visual imitation picture of the real world system on the computer screen, and to formulate the model logic within such an environment.

In the next chapter, we discuss MacGraSE in greater depth - the data structures and the data input mechanism within the application.

CHAPTER 6

IMPLEMENTATION OF THE GRAPHICAL SIMULATION MODELLING ENVIRONMENT

This chapter looks at the internal structure of the MacGraSE application. The data structures within the application are given in section 6.1. Section 6.2 discusses the data input interface of the application. The program generator module is discussed in section 6.3. The modules that handle the running of a simulation model, and that handle the simulation outputs, are described in sections 6.4 and 6.5 respectively. Section 6.6 discusses the experience gained from this research development. Section 6.7 gives a summary of this chapter. A summary of all the MacGraSE units which were written in MPW (Macintosh Programming Workshop) Pascal can be found in Appendix G.

6.1. MACGRASE DATA STRUCTURES

The data structures within the MacGraSE application are shown in figure 6.1. The type definitions of entity, activity, and attribute are further discussed in sections 6.1.1, 6.1.2, and 6.1.3 respectively.

Figure 6.1. Data Structures of MacGraSE

```
quehistrec  =  record
                    nam              :  str10;
                    base,width       :  integer;
                    tflag            :  longint;
                    total            :  longint;
                    sosq             :  real;
                    count            :  longint;
                    id               :  integer;
                    data             :  array[0..16]  of  integer;
               end;
quehistptr  =  ^quehistrec;
quehisthdl  =  ^quehistptr;
```

```
tserieshdl   =  ^tseriesptr;
tseriesptr   =  ^tseriesrec;
tseriesrec   =  record
                    nam                 :  str10;
                    int                 :  integer;
                    count               :  integer;
                    plot                :  integer;
                    tflag               :  longint;
                    id                  :  integer;
                    data                :  array[0..449]  of  integer;
                end;

modenthdl    =  ^modentptr;

queobjrec    =  record
                    loc                 :  point;
                    nam                 :  str10;
                    num                 :  str5;
                    ql,qt               :  quehisthdl;
                    ts                  :  tserieshdl;
                    fromact             :  integer;
                    acdloc              :  point;
                    qnum                :  integer;
                    modent              :  modenthdl;
                end;
queobjptr    =  ^queobjrec;
queobjhdl    =  ^queobjptr;

entity       =  ^entityptr;

entobjhdl    =  ^entobjptr;
entobjptr    =  ^entobjrec;
entobjrec    =  record
                    id                  :  integer;
                    ent                 :  entity;
                    loc                 :  point;
                    que                 :  queobjhdl;
                    act                 :  integer;
                    min,max             :  char;
                    next,nact           :  entobjhdl;
                end;

entityptr    =  ^entityrec;
entityrec    =  record
                    id                  :  integer;
                    nam                 :  str10;
                    num                 :  str5;
                    etype               :  integer;
                    icon                :  char;
                    display             :  integer;
                    mobile              :  boolean;
                    pat                 :  integer;
                    utime               :  real;
                    oent                :  entobjhdl;
                    next                :  entity;
                end;

activity     =  ^activityptr;
activityptr  =  ^activityrec;
activityrec  =  record
                    id                  :  integer;
                    nam                 :  str10;
                    dur                 :  str30;
```

```
                dist,a,b,c,d,e    : integer;
                ref               : str30;
                rec               : rect;
                acdloc            : point;
                anum              : integer;
                count             : integer;
                inv               : entobjhdl;
                next              : activity;
            end;

attcalhdl   = ^attcalptr;
attcalptr   = ^attcalrec;
attcalrec   = record
                arg               : integer;
                con               : str30;
                cal               : str30;
            end;

attobjhdl   = ^attobjptr;
attobjptr   = ^attobjrec;
attobjrec   = record
                id,aid            : integer;
                rec               : rect;
                atcai             : attcalhdl;
                ent               : entobjhdl;
                act               : activity;
                next              : attobjhdl;
            end;

atthistrec  = record
                nam               : str10;
                base,width        : integer;
                totai             : longint;
                sosq              : real;
                count             : longint;
                id                : integer;
                data              : array[0..16] of integer;
            end;
atthistptr  = ^atthistrec;
atthisthdl  = ^atthistptr;

attribute   = ^attributeptr;
attributeptr= ^attributerec;
attributerec= record
                id                : integer;
                nam               : str10;
                ent               : integer;
                glo               : boolean;
                hist              : atthisthdl;
                oatt              : attobjhdl;
                next              : attribute;
            end;

conobjhdl   = ^conobjptr;
conobjptr   = ^conobjrec;
conobjrec   = record
                ent1,ent2         : entobjhdl;
                con               : str30;
                next              : conobjhdl;
            end;

modatthdl   = ^modattptr;
modattptr   = ^modattrec;
```

```
modattrec   = record
                att                 : attribute;
                val                 : longint;
                next                : modatthdl;
              end;

modentptr    = ^modentrec;
modentrec   = record
                id                  : integer;
                st                  : longint;
                matt                : modatthdl;
                next                : modenthdl;
              end;
```

6.1.1. Entity

Entity is defined as a handle to the record *entityrec* which contains all the information about an entity type. *nam* and *num* are the entity name and number of entities respectively. Since all the dialogue boxes accept data in the form of strings, *num* is regarded as a string variable. It will be converted to an integer variable when necessary. The entity type can be specified using the *etype* parameter where '1' stands for permanent entity, '2' stands for temporary entity, and '3' stands for facility entity. The icon that represents an entity type is stored as a character in the record referenced by *icon*. The mobility of the entity is set by using the parameter *mobile*. *utime* is a variable which is used to record the utilisation time of an entity type during a simulation run. The next entity record in the entity link list is referred to by *next*.

The duplication of entity icons in our formulation mechanism is recorded by using the parameter *oent* in the entity record. This contains the list of entity icons on screen that belongs to the entity type. *oent* is a handle to the record type *entobjrec*. *entobjrec* contains information about each entity icon on the screen, including the index of the icon (*id*), the entity to which it belongs (*ent*), the location on the screen (*loc*), the activity in which it is placed (*act*), the minimum (*min*) and the maximum (*max*) number of entities that are required to start the activity, and the next icon in the entity icon list (*next*). The handle *nact* is a handle to the next entity icon of a different entity type which is involved in the activity and is only used in a simulation run.

In the case of permanent and temporary entities, the automation of queues in our formulation mechanism is achieved by creating a handle *que* for the record type *queobjhdl* for every duplicate entity icon. Since only one queue handle is created for a facility entity, the parameter *que* of any duplicate entity icons points to the same handle. A queue record contains information about the name of the queue (*nam*), the number of entities in the queue (*num*), the queue location (*loc*), the queue location in its associate activity cycle diagram (*acdloc*), the index of the activity preceding the queue (*fromact*), a handle to the queue length histogram (*ql*), a handle to the queueing time histogram (*qt*) and a handle to the time series (*ts*). The parameter *qnum* is used to record the number of entities in the queue during a simulation run. *modent* is a list of model entities which is attached to the queue and is used during a simulation run. Each entity in the simulation model, except temporary entities, is labelled before a simulation run. The *modentrec* record contains information about the index of the entity and the time when the entity enters the queue (*st*) during a run. *next* is a handle to the next model entity in the list. *matt* is a handle to the list of entity attributes that is being evaluated during the simulation run and is used to store the value of the attribute during a run.

A conditional path in the life cycle of an entity type is recorded by using the handle *conobjhdl*. The information in the record includes the addresses of the handles of the entity icon records in which the path comes from and that for the record in which the path goes to. The condition is recorded by using the parameter *con*. *next* is the handle that points to the next conditional path record in the list.

6.1.2. Activity

Activity is defined as a handle to the record type *activityrec* which contains all the information about an activity. *nam* and *id* are the name and index of the activity respectively. The textual form of the duration formula is stored in the parameter *dur*.

dist, a, b, c, d and *e* are used to identify the distribution function and its associate parameters for the activity. *ref* is used to store the comment that the user enters in the activity dialogue box. The activity rectangle on the screen is referenced by *rec*. The location of the activity rectangle in its associated activity cycle diagram is recorded by *acdloc*. *anum* is used to record the number of attribute assignments that are present in the activity. The parameter *count* is used to record the number of times that the activity successfully starts in a simulation run. *inv* is a list of entity icons records *entobjhdl* which are involved in the activity. *next* is a handle to the next activity in the activity link list.

6.1.3. Attribute

Attribute is a handle to the record type *attributerec* which holds all the information about an attribute defined in the model. *nam* and *id* are the name and index of the attribute respectively. The entity to which the attribute belongs is referenced by *ent*. The boolean variable *glo* is set to true if the attribute is a system attribute. *hist* is a handle to an attribute histogram, if any. The next attribute in the attribute link list is referenced by *next*.

The duplication of attribute objects on screen is recorded by using the parameter *oatt* which is a handle to the record type *attobjrec*. Each *attobjrec* record contains information about the index of the cloud among all of its associated attribute objects (*id*), the index of the attribute assignment within an activity (*aid*), the rectangle of the cloud (*rec*), the entity icon to which the attribute cloud is linked (*ent*), the activity in which the attribute cloud is evaluated (*act*), and the next attribute cloud in the list (*next*). Each attribute cloud is associated with one evaluation record referenced by *atcal*. The parameter *atcal* is a handle to the record which contains information about the argument of the attribute being evaluated (*arg*), the condition of the assignment (*con*), and the evaluation formula (*cal*).

6.2. DATA INPUT INTERFACE

This section discusses the data input interface of the MacGraSE application. The drawing facilities are described in section 6.2.1. The parameter details of the model components are entered in the application via dialogue boxes. The structure of the entity, activity, queue and attribute information dialogue boxes are given in sections 6.2.2 to 6.2.5 respectively. Section 6.2.6 discusses the generation of an activity cycle diagram.

6.2.1. Background Drawing Facilities

MacGraSE supports the general drawing environment of a Macintosh application. The user can select the drawing size of the model by using the Drawing Size command in the Edit menu. The drawing size dialogue box is shown in figure 6.2. The palette buttons in the tool box window allows the user to draw any items on the screen. A summary of the functions of the palette buttons is given in figure 6.3.

Figure 6.2. The Drawing Size Dialogue Box

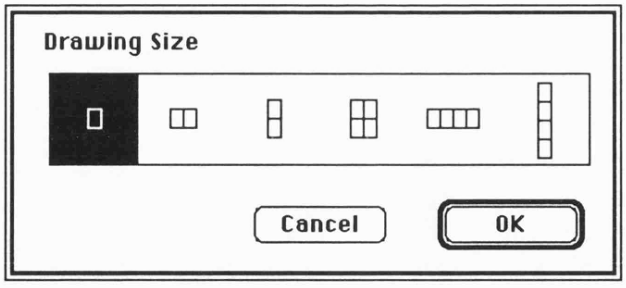
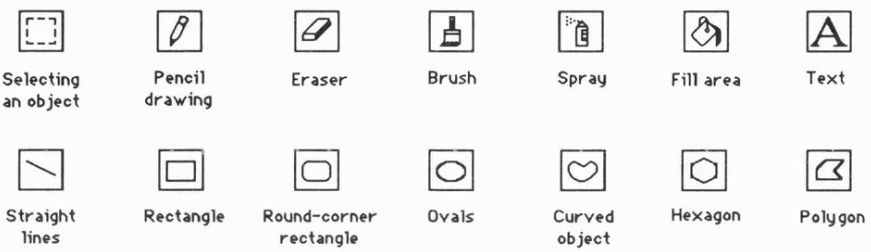


Figure 6.3. A Summary of the Palette Buttons inside the Tool Box Window



Apart from being able to draw the background picture inside the application, the user can also import a picture from other Macintosh drawing applications by using the Import Paint command in the Draw menu. The user can set the pen size (figure 6.4), pen pattern (figure 6.5), fill pattern (figure 6.6) and brush shape (figure 6.7) by using the appropriate commands in the draw menu. The dialogue box used for setting the appearance of the text on screen is shown in figure 6.8.

Figure 6.4. The Pen Size Dialogue Box

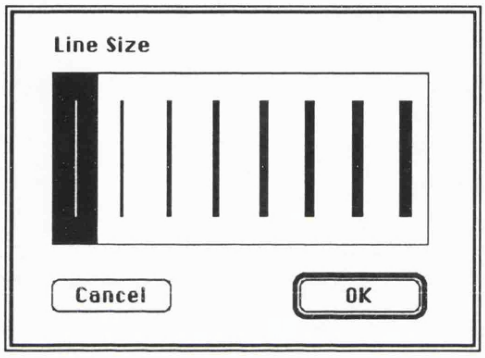


Figure 6.5. The Pen Pattern Dialogue Box

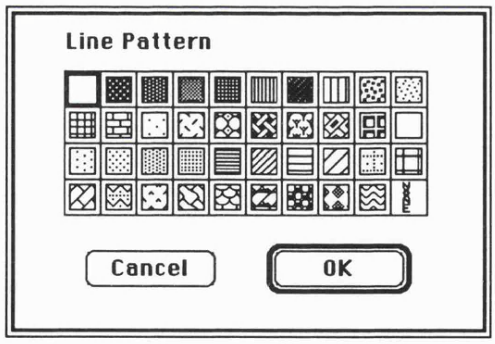


Figure 6.6. The Fill Pattern Dialogue Box

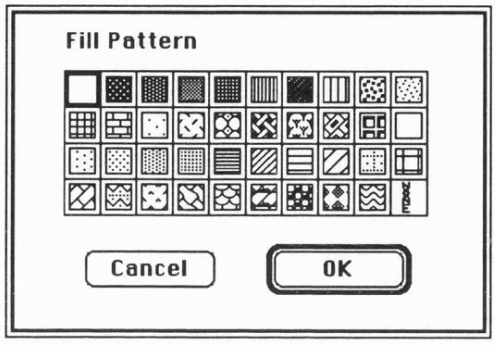


Figure 6.7. The Brush Shape Dialogue Box

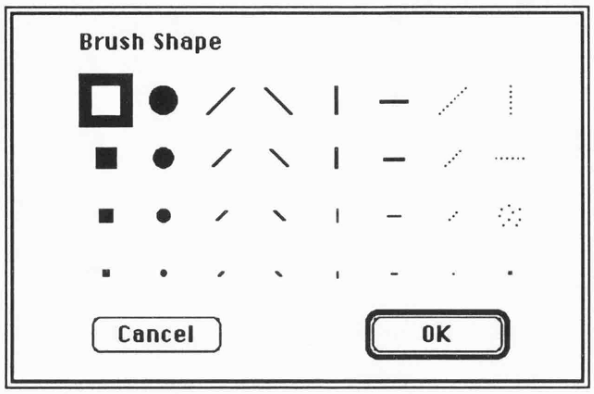
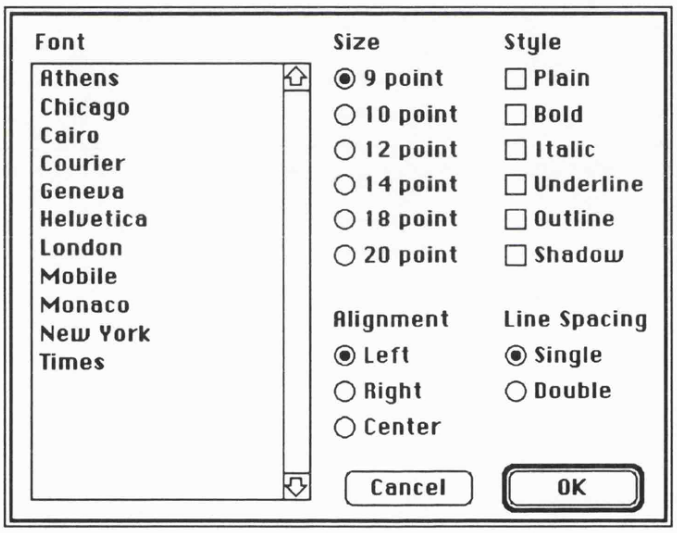


Figure 6.8. The Text Edit Dialogue Box

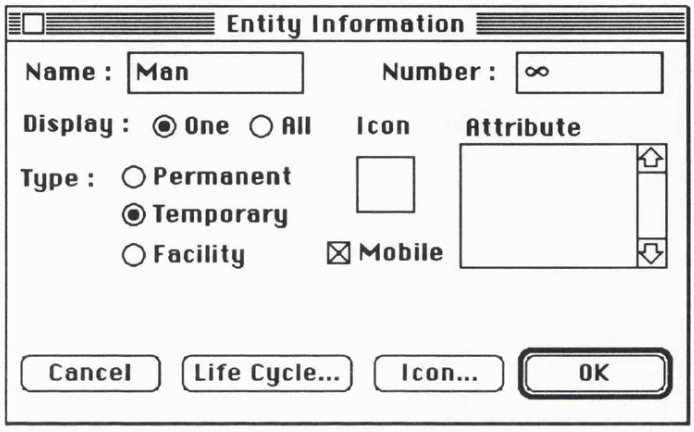


6.2.2. Entity Information

A new entity type can be created by using the New Entity command in the Model menu. The entity dialogue box is shown in figure 6.9. The user can enter the name of the entity type, the number of entities present in the system and the entity classification (permanent, temporary or facility). He can also choose to display all the entities of the selected entity type or just use one icon to represent all its associate members. The entity type can be either mobile or non-mobile. A mobile entity type

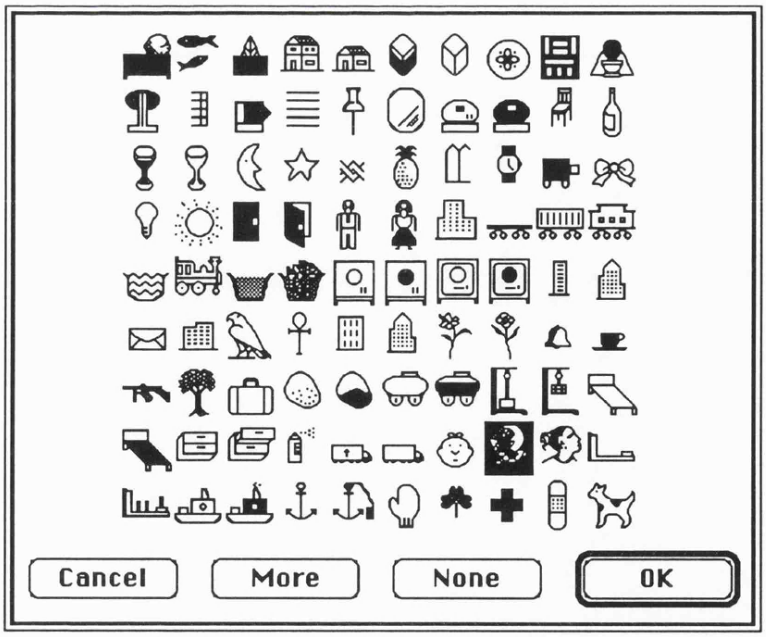
moves from one place to another in a simulation run whereas a non-mobile entity remains stationary.

Figure 6.9. The Entity Dialogue Box



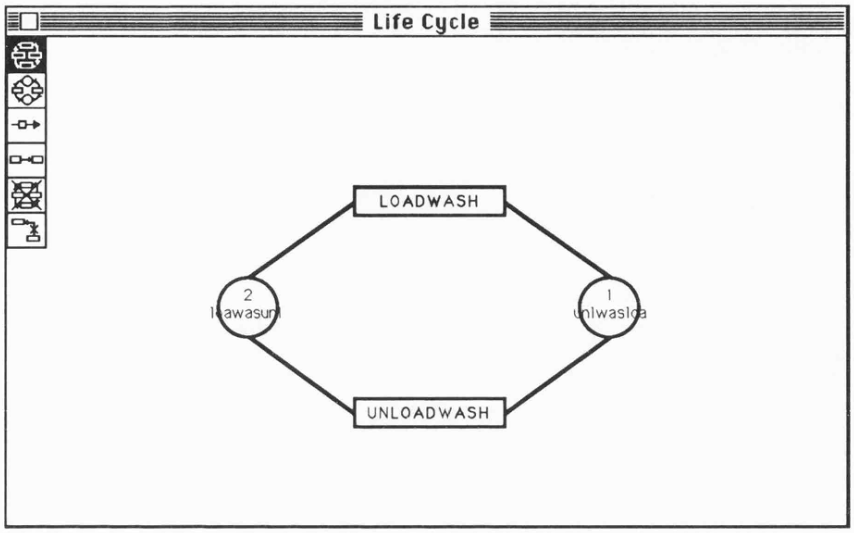
The 'Icon' button is used to select an icon for the entity type. Figure 6.10 shows the icon dialogue box where the user can select an entity icon by clicking at the appropriate position.

Figure 6.10. The Icon Dialogue Box



A flow diagram of the life cycle of an entity type will be shown in the life cycle window (figure 6.11) if the user clicks at the 'Life Cycle' button.

Figure 6.11. The Life Cycle Window



6.2.3. Activity Information

A new activity can be defined by selecting the New Activity command from the Model menu. The activity dialogue box is shown in figure 6.12. The user can specify the name of the activity and enter its duration formula via the duration dialogue box (figure 6.13).

Figure 6.12. The Activity Dialogue Box

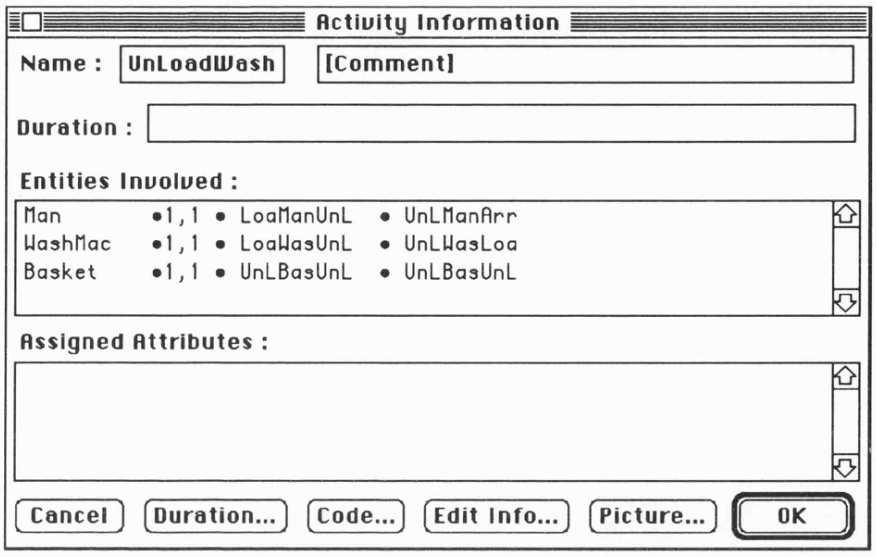
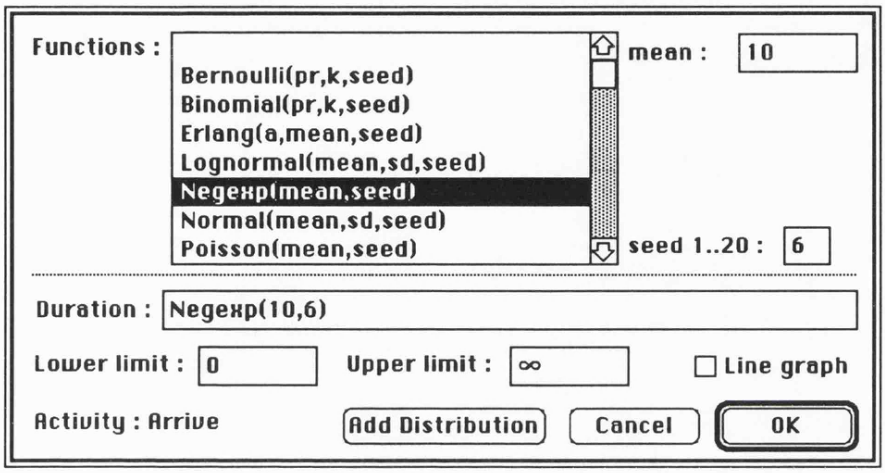


Figure 6.13. The Duration Dialogue Box



The entities involved table in the activity dialogue box shows the list of entities that are involved in the activity defined by the user and the attributes evaluation table shows the list of attributes that are evaluated at the activity.

The 'Picture' button is used to invoke the activity picture dialogue box (figure 6.14) in which a picture that represents the happening of the activity can be drawn. This picture can then be used for a simulation run. The 'Edit Info' button allows the user to see a structured flow diagram representation of the selected activity in the activity information window (figure 6.15).

Figure 6.14. The Activity Picture Dialogue Box

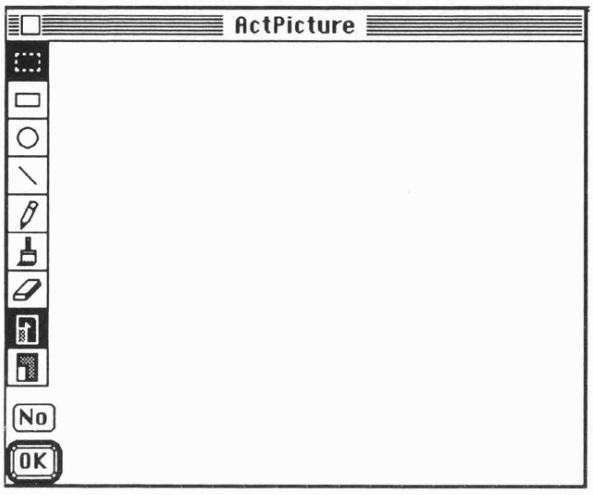
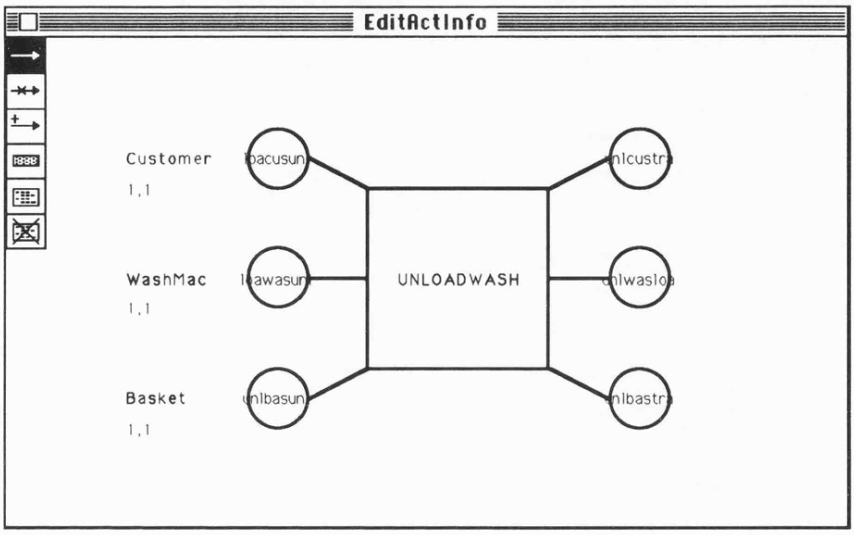


Figure 6.15. The Activity Information Dialogue Box



6.2.4. Queue Information

Queues are automatically produced in the MacGraSE specification system. The algorithm used is to generate a queue record whenever a ghost image of an entity icon is duplicated for permanent and temporary entities, and one common queue record for each facility entity type. Since an entity icon on the screen is assumed to be associated with an activity, the queue that is generated represents where the entity comes from before the activity begins. For permanent and temporary entities, a closed loop structure is always ensured by linking the last duplicated entity icon to the queue of the first original icon of the entity type. For facility entities, a duplicated entity icon always goes back to the common queue of the entity type.

The queues within the model can be reviewed by selecting the 'Select queue' palette button from the mode box window. The queue dialogue box shown in figure 6.16 can be invoked by clicking at the desired queue. The name of the queue is automatically created by a combination of the first three characters of each of the preceding activity, the name of the entity and the succeeding activity. The user can

specify a queue length histogram, a queuing time histogram and a time series for the queue. The dialogue box also shows the activity from which the entity comes and that to which the entity goes.

Figure 6.16. The Queue Dialogue Box

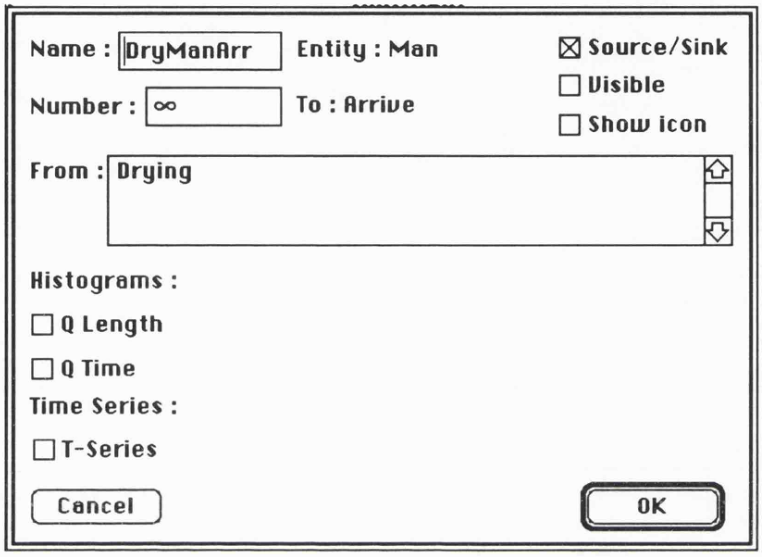
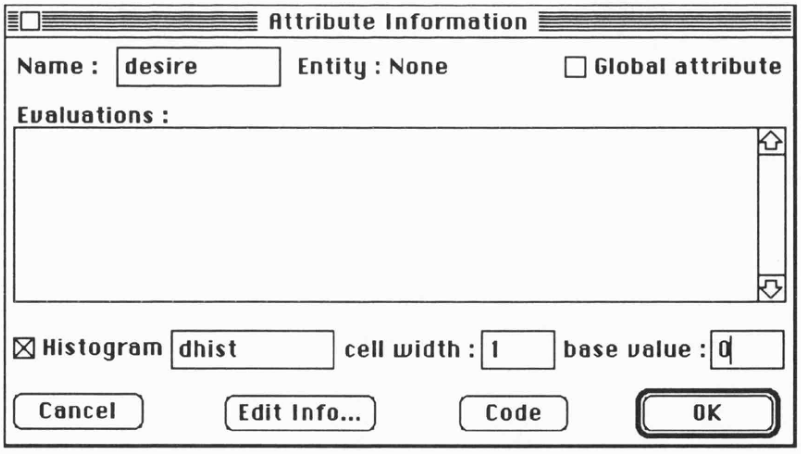


Figure 6.17. The Attribute Dialogue Box



6.2.5. Attribute Information

A new attribute can be created by using the New Attribute command from the Model menu. The attribute dialogue box is shown in figure 6.17. The user can specify

the name of the attribute and decide whether it is an entity attribute or a system attribute. A histogram can also be defined by the user. The evaluation table shows the list of calculations for the attribute that are defined in the model. The 'Edit Info' button allows the user to see a structured logical diagram of the status of the attribute in the attribute information window (figure 6.18). The evaluation formula of an attribute object can be entered via the attribute evaluation dialogue box (figure 6.19). This dialogue box is invoked when the user double-clicks at the selected attribute cloud on the screen.

Figure 6.18. The Attribute Information Window

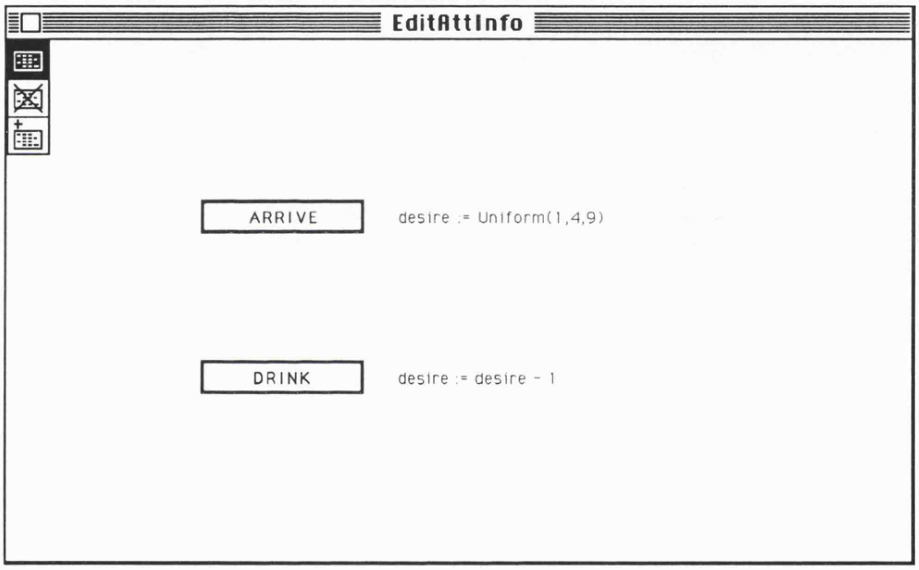
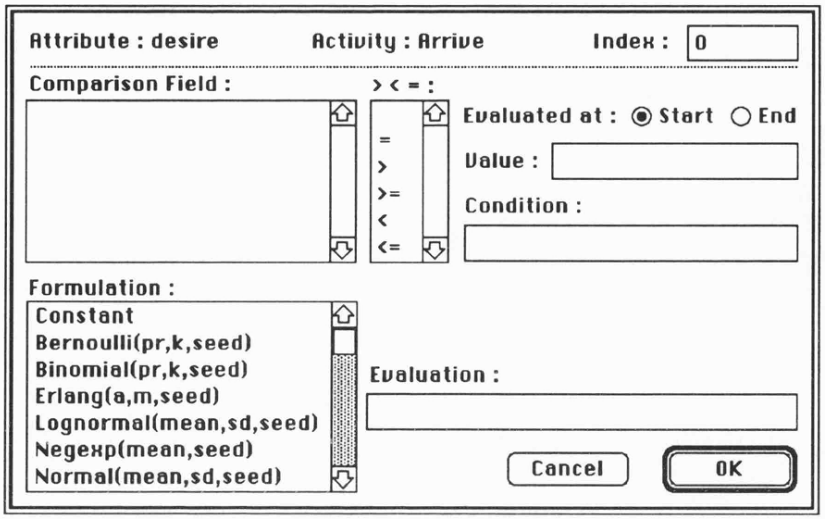


Figure 6.19. The Attribute Evaluation Dialogue Box



6.2.6. Activity Cycle Diagram Generation

MacGraSE supports the generation of an ACD from the pictorial description of the model. This allows the user to look at the model in a more structured manner. The parameters can be entered via the ACD. Since the initial positions of the activities and queues are set using their relevant positions within this pictorial description of the model, the first generated ACD might look cluttered on the screen (figure 6.20a). However, a clear diagram can be obtained by changing the positions of the objects (figure 6.20b).

Figure 6.20a. A First Generated Activity Cycle Diagram

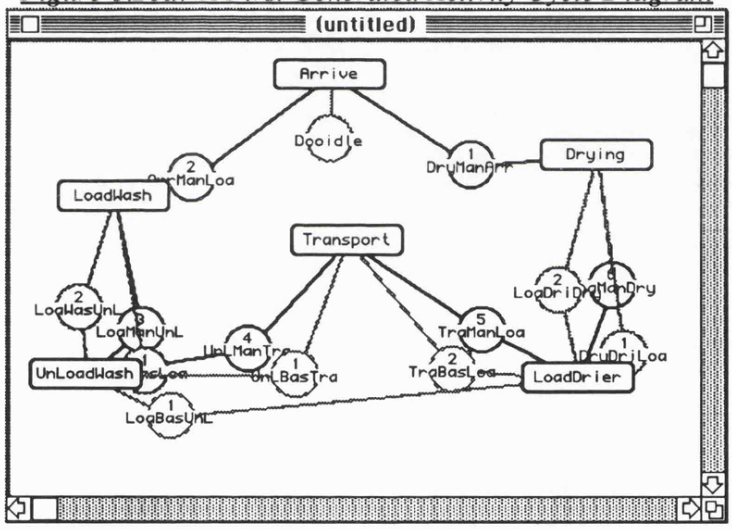
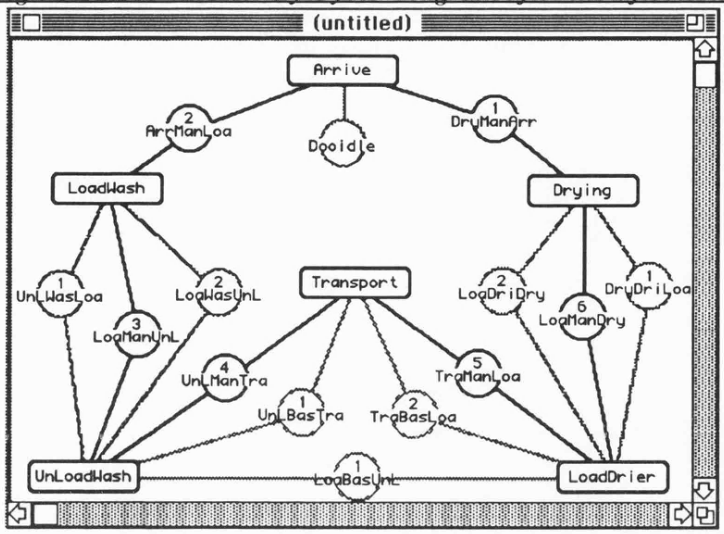


Figure 6.20b. An Activity Cycle Diagram after modifications



6.3. PROGRAM GENERATION

MacGraSE has a built-in program generator within the application. A selection of languages is found in the Language submenu of the Run menu. The user can select a language and use the Generate command to generate a three-phase simulation program. If the Generate command is selected, the system will prompt the user to give a name for the generated program (figure 6.21). A program will then be generated if the user clicks the 'OK' button. The user can see the generated code by selecting the 'Show Program' command in the Run menu. The program generator for Turbo Pascal was developed during this research and an example of the generated code is shown in figure 6.22.

Figure 6.21. The Generate Dialogue Box

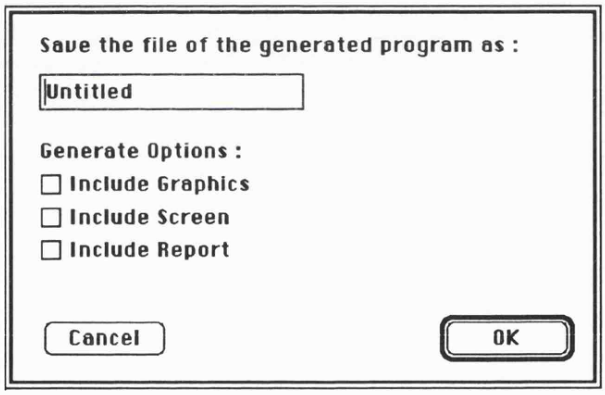
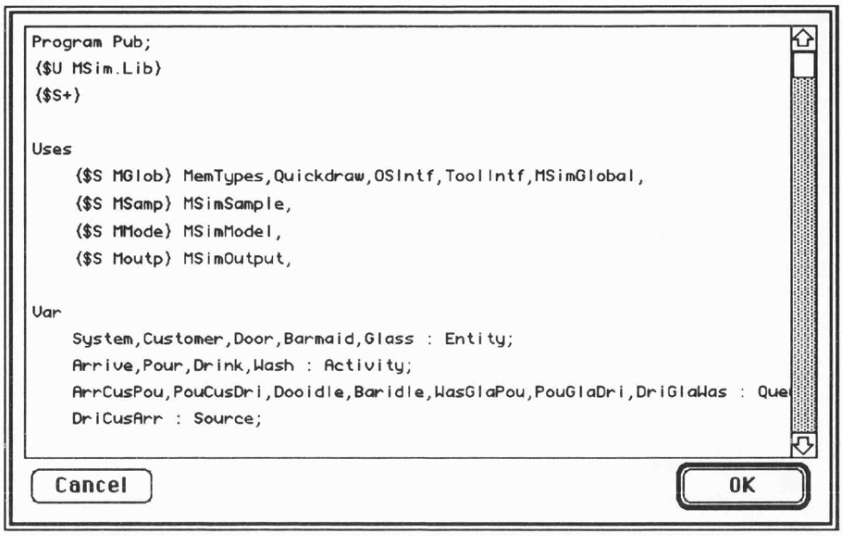


Figure 6.22. The Program Code generated by MacGraSE in Turbo Pascal



6.4. THE RUN MODULE

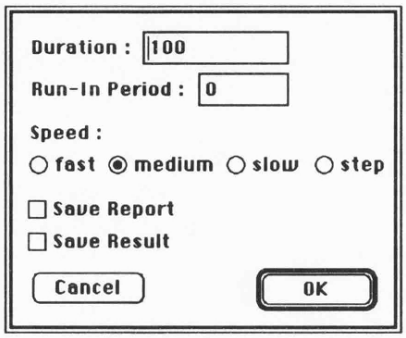
MacGraSE supports three types of simulation run - visual run, text run and screen run, discussed in sections 6.4.1, 6.4.2, and 6.4.3 respectively. The algorithm behind the simulation run in the three modes is the same. Time is initially set to zero. The conditions for starting an activity are checked throughout the entire activity list in the model. Any activity that can start is added to the event queue defined in the run module. The time that the activity ends and a list of the model entities that are involved are recorded. Time is then set to the next earliest time that an activity ends in the event queue. The associated resources that are scheduled to end at this time are released to their corresponding queues. A scanning of the activities is repeated and the event queue is updated. Time is then reset. This procedure repeats until the time of the simulation clock is equal to the duration of the simulation run.

6.4.1. Visual Run

A visual run displays the dynamics of a simulation model in terms of its pictorial description. During a visual run, the start of an activity is represented by a redrawing the activity rectangle containing its associated entity icons. The value of the activity count, i.e. the number of times that the activity has successfully started during the run, is also shown. Overlapping rectangles are used whenever there is more than one activity of the same kind happening at an instant of time during the run. The ending of an activity is shown by moving the entity icons to the corresponding position of the queues. The entity icon is drawn inside a circular frame to indicate that it is in the process of entering a queue.

Figure 6.23 shows the visual run dialogue box. The user can specify the duration, the run-in period and the speed of the simulation run. A report file and a result file can also be specified.

Figure 6.23. The Visual Run Dialogue Box

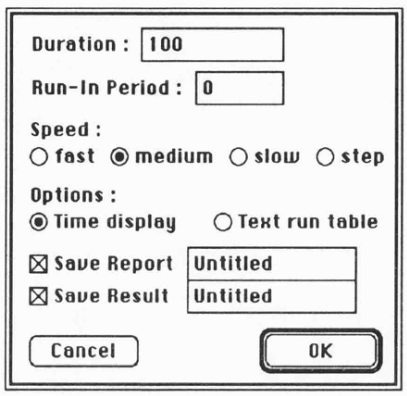


Another type of visual simulation that is supported by MacGraSE is model animation, which shows the movements and information concerning individual active entities during a run. This option is mainly intended use for model verification. A long duration run is not recommended.

6.4.2. Text Run

Figure 6.24 shows the text run dialogue box. The user can specify the duration, the run-in period and the speed of the simulation run. The user can also choose to display a text run table, which shows a textual description of what happens at every time advance in a table, or to display only the simulation clock. A report file and a result file can be specified in the dialogue box.

Figure 6.24. The Text Run Dialogue Box

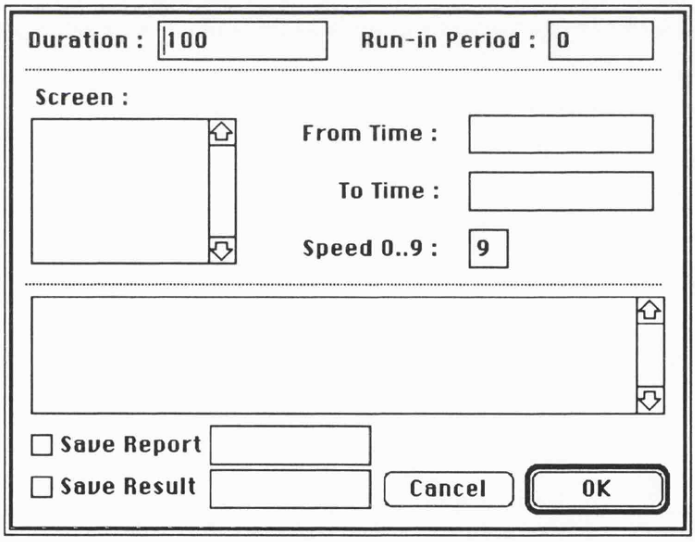


6.4.3. Screen Run

A screen run allows the user to run the simulation model by using the predefined output screens. An output screen is mainly made up of charts, graphs, histograms and time series. The user can design an output screen by arranging the position of the selected components within the screen window.

Figure 6.25 shows the screen run dialogue box. The user can specify the duration, the run-in period and the speed of the simulation run. The switching of screens can be preprogrammed by setting the time interval that a selected screen will appear during a simulation run. A report file and a result file can be specified.

Figure 6.25. The Screen Run Dialogue Box



6.5. THE OUTPUT MODULE

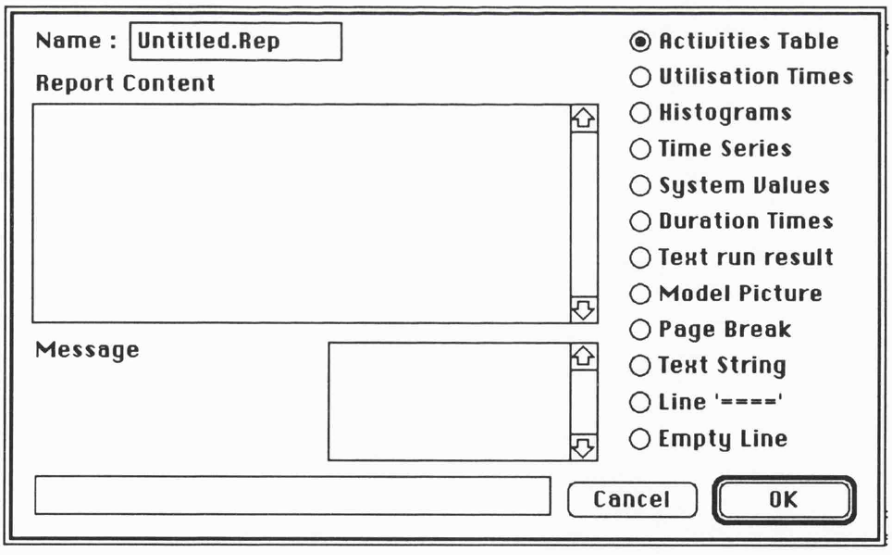
The report generation facility of the MacGraSE application is discussed in section 6.5.1. Apart from the simulation result file, which contains textual description of the events that occured during a simulation run, there are five types of output data that can be specified in the model - the entity utilisation time chart, the activity count

table, the system values table, histograms and time series (discussed in sections 6.5.2 to 6.5.6 respectively). The first two items are automatically recorded during a simulation run, others are specified by the actions of the user.

6.5.1. The Report Generator

The format of the simulation report can be edited by using the report dialogue box as shown in figure 6.26. The user can select the data recording items (activity count table, utilisation time chart, histograms, time series, system value table, activity duration time graph, text run results, and the model picture) that are specified in the model and reorder the report sequence. Other editing facilities include insertion of a page break, text string, straight line and empty line between items.

Figure 6.26. The Report Dialogue Box

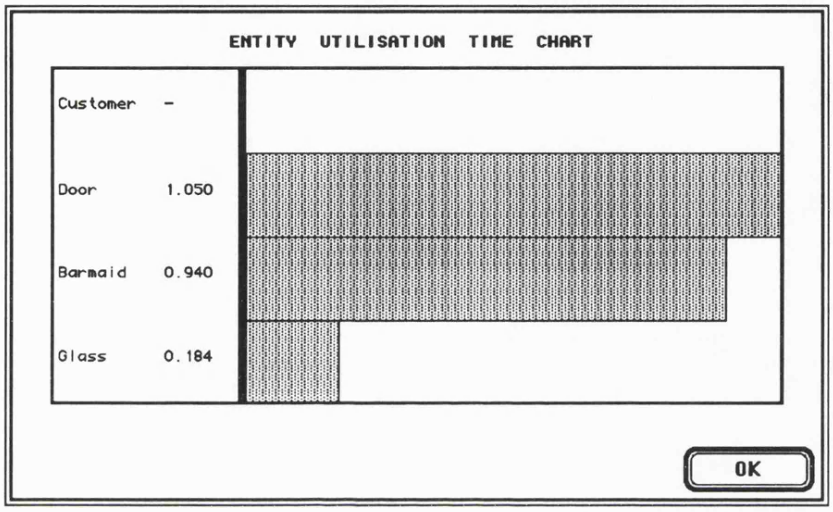


6.5.2. The Entity Utilisation Time Chart

The utilisation time chart records the proportion of the time that the permanent entities in the system are engaged in activities. Temporary entities are not recorded in

this way since they are only in the system for a limited duration. Figure 6.27 shows an example of a utilisation time chart.

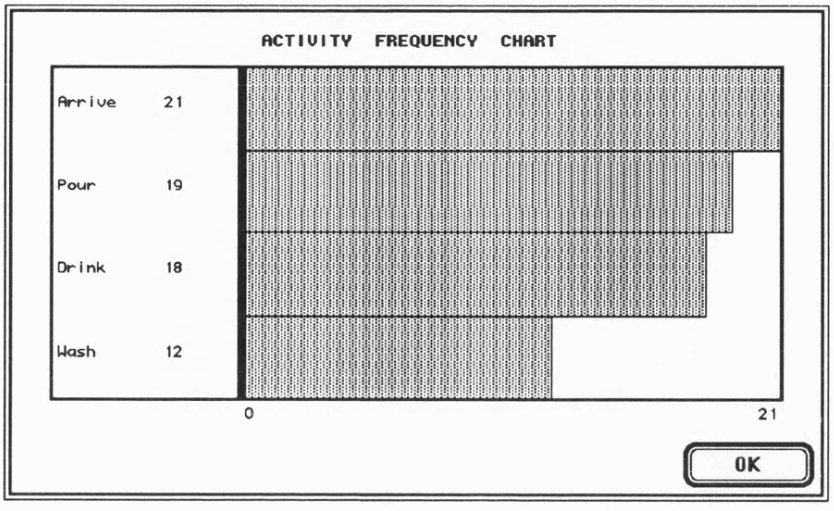
Figure 6.27. An Entity Utilisation Time Chart



6.5.3. The Activity Count Chart

The activity count chart records the number of times each activity was successfully started during the simulation run. Figure 6.28 shows an example of an activity count chart.

Figure 6.28. An Activity Count Chart



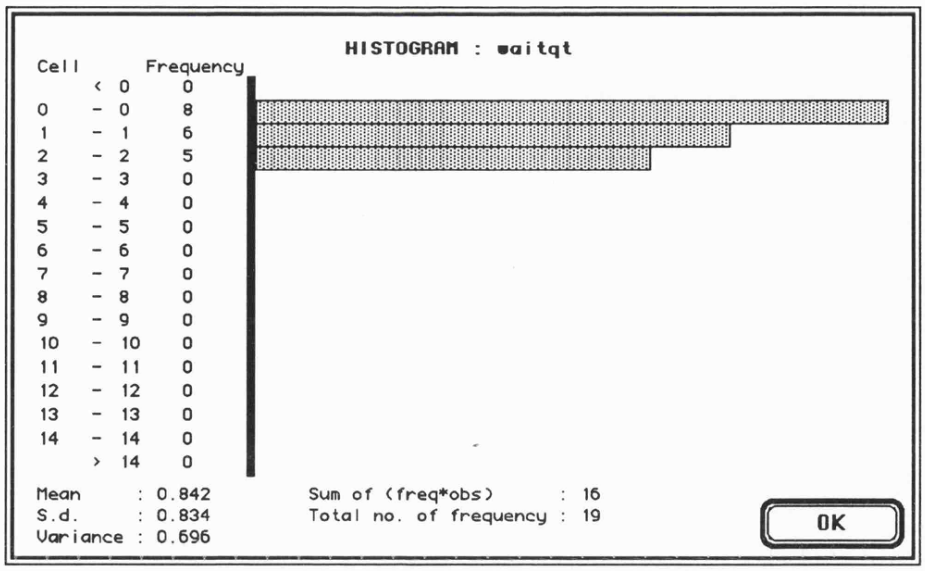
6.5.4. The System Values Table

The system values table gives the last value of all the attributes of the system. This will be of particular use if the attribute defined by the user was designed to calculate some global variable of interest.

6.5.5. Histograms

There are two types of histograms which the user can specify - queue histograms and attribute histograms. A queue histogram can be sub-divided into a queue length histogram, which records the average length of the queue, and a queuing time histogram, which records the length of time that each entity of that entity type spends in the queue. Some basic statistical results include the mean, the standard deviation, the variance, the total number of frequencies, and the sum of each frequency multiplied by the number of observations. An example of a histogram produced by MacGraSE is given in figure 6.29.

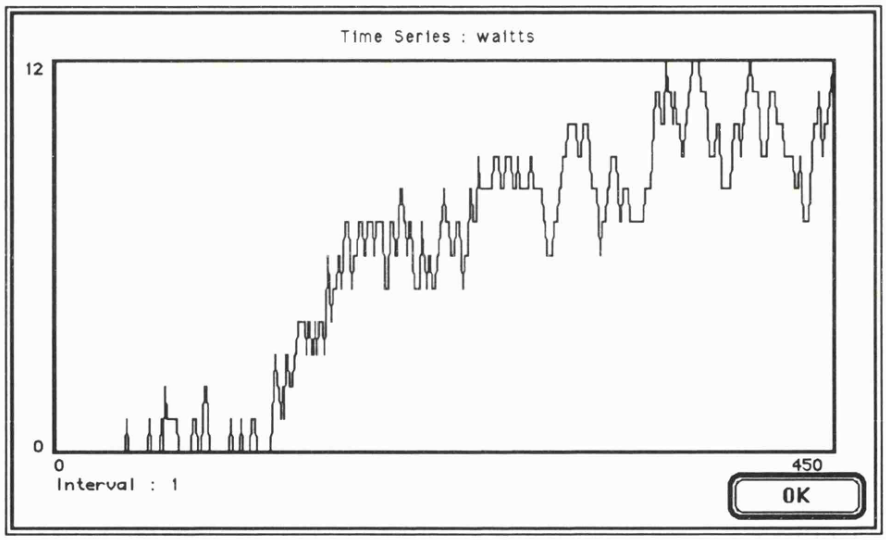
Figure 6.29. A Histogram



6.5.6. Time Series Chart

A time series chart records the queue length in a queue with respect to time during a simulation run. The drawing of a time series is automated and the user can specify the time series chart to be drawn in either one of three modes - a bar, a line or a scatter plot. An example of a line time series chart produced by MacGraSE is given in figure 6.30.

Figure 6.30. A Time Series Chart



6.6. EXPERIENCE GAINED FROM MACGRASE

The advantages and limitations of MacGraSE are discussed in section 6.6.1 and 6.6.2 respectively.

6.6.1. Advantages of MacGraSE

The main advantage of using an application such as MacGraSE is its user-friendly visual interactive interface. Macintosh programs are all event-driven, i.e. the

application is programmed to respond to the actions of the computer user. With this interface, the user has complete control over the application during the construction of the simulation model. Moreover, the formulation mechanism allows the user to reconstruct the model continuously throughout the model building process. The tight indexing of entity icons is so apparent that the user can easily modify an existing path or create an additional path anywhere within its life cycle. For simulation models that are heavily attribute-based, the user can easily model movements of attribute evaluation in the system. The main benefit of MacGraSE is to aid a user to construct the model logic in a stimulating environment.

MacGraSE is easy to use. The iconic menus provided by the system allow the user can to familiar with the application very quickly. All that is required from the user is an ability to draw, not necessarily as skilful as a painter, but to draw some resemblance to the real world system.

6.6.2. Limitations of MacGraSE

Although MacGraSE is a powerful simulation system, it suffers the same deficiency that most data-driven packages have - one can only specify the logic which the simulation system is programmed to accept. Any other logic that a user wants to specify, and which is not included in the system, requires the user to add code to the generated program.

MacGraSE is Macintosh-specific. It is written by using the routines inside the ROM of the Macintosh and cannot therefore be translated onto other machines. On the other hand, if the application is developed on other non-graphics orientated machines, the final system might not be as impressive, and the drawing routines and the speed of drawing are always a problem . Another disadvantage of the application is that it is memory-dependent. The user has to make sure there is enough RAM on the machine to

run the application, especially if a long simulation run is required.

It is inevitable that not all the real world systems can be drawn. For a large and complex system, different levels of diagrams might be required. MacGraSE does not allow the user to create a pictorial description within another pictorial description but it is one of the area that will be further researched into.

6.7. SUMMARY

This chapter looks at the technical aspects of the MacGraSE application. The program is still undergoing further improvements and refinements so as to increase the code efficiency and thereby reduce the running time of the application.

The users of MacGraSE include both inexperienced simulation modellers and intelligent modellers. It also provides the right environment for the client and the analyst to formulate the problem in a collaborate manner. The client is not required to understand any diagramming techniques since what she sees on the computer screen is a self-explanatory picture of the real world system, with indexed entity images indicating the movement of different entity types within the system.

MacGraSE can be used for building prototypes of simulation models. For complicated simulation models, prototyping can help to narrow the gap between the client and the analyst. MacGraSE is a complete simulation environment since it allows the user to perform experiments on the model and generate reports from the simulation run. It is also a very good tool for simulation beginners to learn about simulation techniques.

CHAPTER 7

CONCLUSIONS & FURTHER RESEARCH

This is the final chapter of the thesis. Section 7.1 summarises the thesis and section 7.2 draws conclusions from the research. Section 7.3 draws some conclusions concerning the extensive experience gained using the Apple Macintosh microcomputer during this research. Section 7.4 discusses research that can be pursued in the near future on the basis of research reported in this thesis.

7.1. SUMMARY

This thesis has investigated the potential of computer graphics in providing a graphics driven specification system that contains sufficient structure and content to form the simulation model itself. Chapter 1 set the scene on the research, describing the nature of discrete event simulation modelling, the diagramming method of activity cycle diagrams which underpinned this research, the three phase simulation model structure, and a discussion about visual simulation modelling. These approaches were combined in this research endeavour, having been put into perspective with respect to other published research work in Chapter 2.

The basic research method adopted was to build systems that exemplified the state of thinking at the time. The purpose of this method was to enable ideas to be developed, discarded and enhanced, and for new ideas to emerge. Chapter 3 discussed the first development, MacACD, which gave experience in the construction of such systems on an Apple Macintosh. Apart from providing the basis for proposals concerning the enhancement of ACD rules of construction, MacACD demonstrated the

limitations of an ACD interface and the need for a more intricate connectivity between textual and diagrammatic modes of model specification.

Chapter 4 described the development of HyperSim, a simulation system developed using Hypercard. This system has all the power of interconnectivity demonstrated as a need by MacACD, but has severe limitations both in terms of security of system development, and an inability to provide a running model directly due to lack of speed. However, the power of an icon based interconnected textual and diagrammatic based system was demonstrated, and led to the development of the final system described in this thesis : MacGraSE.

Chapter 5 sets out the basic requirements of a graphics driven specification and modelling system, based on the research experiences described in the previous chapters. Whilst textual and ACD interfaces are major interconnected parts of MacGraSE, the main input device is a picture representing the problem, including a background display. This system allows for dynamic icon based visual model running, as well as code generation for complete model embellishments, interactive report writing, and representational graphics outputs. Chapter 6 describes the methods by which the design of MacGraSE was implemented on the Apple Macintosh microcomputer.

7.1. CONCLUSIONS

The qualities of graphics and the ease of using graphics in an application will continuously advance with the current rapid growth in technology. Graphics has become an essential component in every type of application software in recent years. High quality and well-presented graphics facilitates human understanding and helps to break the communication gap between parties of different interests.

The popularity of visual simulation modelling has rapidly increased recently. It is commonly an integral part of a simulation system. However, the pros and cons of using visual simulation modelling are debatable. Visual simulation should not be misused or overused. For large and complex models, an animation picture running on the screen might cause misunderstandings unless the monitor is large and clear. It might mean absolutely nothing to the user unless he understands what the picture represents. Paul (89b) emphasised that : "Although visual modelling is a powerful complement to an analyst's problem solving capabilities, it has new problems to overcome as well. The problems of visual simulation includes the fact that vision is interpreted by the brain which does not remember all the visual detail. Moreover, the customer is required to understand the simulation in order to understand what the visual simulation represents. Also, visual simulation is time consuming and it is impossible to test all the model interactions visually for a complex model. The most important potential benefit of visual simulation is the increasing ability to help a decision maker by working together in a collaborative effort. Graphics is the way forward. Humans think 'visually'. Text is a poor approximation, and is even more unreliable."

The practical side of this research started with the production of a graphical specification system in which the user can define a simulation model by drawing an activity cycle diagram on the screen. The parameters are input via dialogue boxes that are evoked by actions of the user. The application produces a text file from the specification model which can be passed down to a VAX computer and read by the program generator AUTOSIM (Chew 86) to produce a three-phase simulation program. It was a successful system and proved the usefulness of such a diagramming specification system and the potential of the Apple Macintosh in producing graphical simulation applications.

At this point in the research, we felt the need to look into other diagramming techniques and see if there is a better technique than the activity cycle diagram method.

We found that although the ACD method is not able to accommodate all the details of the model, especially in the assignment of attributes, it is the most clear and precise way of displaying the model logic in a discrete-event simulation model. Other diagramming techniques cannot provide such a simple but logical view of the model of interest. We further proposed that the conventional activity cycle diagram method can be further enhanced in a computer-aided environment.

The nature of Hypertext gave insight into how to produce a flexible specification system. Research progressed into the Hypercard environment on the Apple Macintosh. With its magnificent web-structure between different stacks, we produced a flexible textual/graphical specification system with a built-in program generator. This research highlights the usefulness of such a flexible specification system and how the ideas can be carried forward to produce a complete graphical simulation environment.

The last part of the research developed a graphical simulation environment which allows the user to define the model logic by drawing a picture on the screen, entering the details of the parameters by some textual description, and then directly running this picture on the screen. The user can interrupt at any point during the simulation run, alter the parameters and continue the run. The user can also design the output screen where the results of the simulation run can be displayed.

Graphics is an invaluable tool to both the modeller and the client of the system in simulation modelling. The universal use of diagramming techniques as a means of developing an informal simulation model are because diagrams are beneficial both for clear thinking and human communication. Although different diagramming techniques have their own conventions and definitions, the system that diagrams represent and which is drawn using different diagramming methods, is the same. These diagrams are all drawn from the unseen imaginative picture of the real world system that is inside the human mind. The breakthrough of the research in this thesis is the possibility of

bringing the user image directly onto the computer screen and to allow the user to reconstruct the model logic within his own thinking environment.

In an industrial environment, the objective of a simulation user is to improve the quality of decision-making. High-level languages, program generators and generic modelling software have only the partial goal of aiding model coding, and so they naturally omit important features. They do not adequately direct experimental procedures, they do not invoke the appropriate statistical tools, and they do not provide output in several styles appropriate to a broad user hierarchy. Such tools shorten the coding cycle, they do not guide the user or reduce the experimental task. We should always take a longer-term view than the practitioner or consultant. The CASM research team at the LSE is aiming to provide aids for problem formulation, program generation and output analysis.

7.3. THE APPLE MACINTOSH

The Apple Macintosh provides a visual user interface based on menus, icons, windows, and a mouse as the input device. A graphics-based visual system does require some sort of pointing device, and the mouse works as well as, or better than, most. Among the different models of Macintoshes, only the Macintosh II supports colour graphics. The more popular models (Plus and SE) only have monochrome graphics. This is often seen as the biggest disadvantage of the Macintosh when a colour simulation system is preferred. The disadvantage of having a small size 8" screen can be overcome by connecting the main unit to a 'Megascreen' (A4 or A3 size screen) via the SCSI port. For simulation modelling, it is ideal to have a large screen and a colour system so as to display the full picture of the dynamics of the simulation run. Despite these deficiencies, the Macintosh is extremely flexible in producing high-quality graphics and sophisticated representational graphical output.

The Macintosh uses the Macintosh Operating System (MOS). The MOS takes up only a fraction of the ROM. There is also a User Interface Toolbox (UIT) which consists of hundreds of callable routines that are used to implement the standard Macintosh application interface. Since graphic operations tend to be memory intensive, most developers on the Macintosh will not have enough memory to perform their own sophisticated graphics. However, they can always get access to the libraries of procedures and functions found in the toolbox and in the operating system itself.

The main advantage in using graphics on a Macintosh is its speed in drawing. QuickDraw, the magician artist in the ROM of the Macintosh, allows you to draw complicated graphics at a very high speed. Another advantage concerns its ability to store resources separate from the application code.

Macintosh supports the use of icons and bit-mapped graphics. 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 visual simulation to be run on the screen are more appropriately developed in this type of environment. The use of bit-mapped graphics allows the 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.

7.4. FURTHER RESEARCH

There is a lot of work that can be done to improve the MacGraSE application. The multiple windowing interface of the Macintosh has made a layering pictorial description possible. For large and complex system, the user should ideally be able to

integrate the entire system into subsystems so that each has a pictorial description of its own. A large scale model can then be built in such an environment.

Colour capabilities should be included as one of the options available in the application. Visual simulation can be more attractively displayed. The generated ACD can then be drawn in different colours so that each colour represents the life cycle of one entity type. Although only ACDs can be generated from MacGraSE, the pictorial description can be used as a basis for generating other logical structured diagrams.

The recent version of MacGraSE does not have a built-in macro language. The use of macros in an application will increase its specification flexibility since the user can write user-defined procedures to meet his own specification requirement. The language must be simple to use, with easy-to-understand syntax so that users with limited programming experience will find it easy to comprehend. The main benefit of macros is to allow the user to add his own routines to the model and to be able to run the modified model within the system. In this case, the user does not need to modify the generated code outside the system.

The output analysis facility is another area in which further improvements can be made. More statistical analysis, for example regression or analysis of variance, should be available in the application. Statistical tests can then be performed within the simulation environment without the need to export data into other statistical packages.

The interaction facility of current visual interactive models is passive (Hurrion 89). The user must decide when to interact, what action to take, and when to accept the validity of the results. The expertise of the user is lost and not retained by the simulation system. The ideal interactive facility should be 'two-way'. The user may interrupt the execution of a model, or an advisory and monitoring function within the model can interrupt its own execution and suggest to the user alternative model parameters or experiments. The possibility of giving visual interactive models a

learning and intelligence aspect is due to recent advances in artificial intelligence (further discussed in Filtman & Hurrion 87, Doukidis & Paul 85, 86, and Paul 89c, 89d).

The next generation of visual interactive models will probably include an expert element, i.e. the model itself will take an active role in the search for a solution. The addition of artificial intelligence methods to simulation is the next possible quantum advance for simulation.

APPENDIX A

THREE-PHASE SIMULATION

PROGRAM STRUCTURES IN PASCAL

This appendix is divided into four sections. Section A.1 shows a skeleton three-phase structure simulation program. Listings of simulation programs for the example models in this thesis are given in sections A.2, A.3 and A.4 respectively.

A.1. The Skeleton Program

A Macintosh Turbo Pascal three-phase simulation program requires the use of the precompiled library MacSim.Lib (see Appendix B) in the same directory as that of the program. The listing of the skeleton program is shown as follows :

```
PROGRAM  anyname;

{$U  MacSim.Lib}
{$R+}
{$S+}

Uses
   {$S  MGlob}  MemTypes,Quickdraw,OSIntf,ToolIntf,PackIntf,
                MSimGlobal,
   {$S  MSamp}  MSimSample,
   {$S  MMode}  MSimModel,
   {$S  MOutp}  MSimOutput;

VAR
   SYSTEM_ENT  :  Entity;

procedure  Build_Model;
begin
end;

procedure  C_phase;forward;

procedure  Create_Recording;
begin
end;

procedure  Cs_in_progress;
begin
end;

procedure  Startup;
begin
   InitVars;
   set_SYSTEM_ENT(SYSTEM_ENT);
   Input_FillQue('anyname.QUE');
   Create_Recording;
   Cs_in_progress;
   C_phase;
end;
```

```
procedure  Report;
begin
end;

procedure  C_activity;
begin
end;

procedure  B1_end_activity_entity;
begin
end;

procedure  B2_end_activity_entity;
begin
end;

procedure  EndRunIn;
begin
   Create_Recording;
end;

function  A_phase:boolean;
begin
   increase_time;
   A_phase  :=  (TIM  <=  Duration);
end;

procedure  B_phase;
begin
   while  (Current_time(TIM))  do
   begin
      while  (Get_next_bevent)  do
      begin
         case  Bevent_num  of
            1  :  B1_end_activity_entity;
            2  :  B2_end_activity_entity;
            127  :  EndRunIn;
         end;
      end;
   end;
end;

procedure  C_phase;
begin
   C_activity;
end;

procedure  Execute;
begin
   Open_File('anyname.RST');
   Startup;
   if  (Run_In_Period  <>  0)  then
      Init_Restart(127,Run_In_Period);
   while  (A_phase)  do
   begin
      B_phase;
      C_phase;
   end;
   Report;
   Close_File('anyname.RST');
end;
```

```
BEGIN                    { main program }
   Initialise_model(false);
   Build_model;
   repeat
      Input_Duration;
      if ( Duration > 0 ) then Execute;
   until ( Duration<=0 );
END.
```

A.2. The Pub Program

```
Program PUB;

{$U MacSim.Lib}
{$R+}
{$S+}

Uses
   {$S MGlob}  MemTypes,Quickdraw,OSIntf,ToolIntf,PackIntf,
               MSimGlobal,
   {$S MSamp}  MSimSample,
   {$S MMode}  MSimModel,
   {$S MOutp}  MSimOutput;

var
   CUSTOMER,BARMAID,GLASS,DOOR,SYSTEM_ENT  :  Entity;
   WAIT,CLOSED,READY,IDLE,CLEAN,FULL,DIRTY  :  Queue;
   ARRIVE,POUR,DRINK,WASHUP  :  Activity;
   OUTSIDE  :  Source;

procedure Build_Model;
begin
   MakeEnt(SYSTEM_ENT,'SYSTEM_ENT');
   MakeAtt(SYSTEM_ENT,'TDESIRE');
   MakeEnt(CUSTOMER,'CUSTOMER');
   MakeAtt(CUSTOMER,'DESIRE');
   MakeEnt(BARMAID,'BARMAID');
   MakeEnt(GLASS,'GLASS');
   MakeEnt(DOOR,'DOOR');
   MakeAttHist(CUSTOMER,'DESIRE','DHIST',1,0);
   MakeSou(CUSTOMER,OUTSIDE,'OUTSIDE');
   MakeQue(CUSTOMER,WAIT,'WAIT');
   MakeQLenHist(WAIT,'WAITTIME',1,0);
   MakeQTimHist(WAIT,'WAITLENGTH',1,0);
   MakeTSeries(WAIT,'WAITTS',2);
   MakeQue(DOOR,CLOSED,'CLOSED');
   MakeQue(CUSTOMER,READY,'READY');
   MakeQue(BARMAID,IDLE,'IDLE');
   MakeQue(GLASS,CLEAN,'CLEAN');
   MakeQue(GLASS,FULL,'FULL');
   MakeQue(GLASS,DIRTY,'DIRTY');
   MakeAct(ARRIVE,'ARRIVE');
   MakeAct(POUR,'POUR');
   MakeAct(DRINK,'DRINK');
   MakeAct(WASHUP,'WASHUP');
end;
```

```
procedure  C_Phase;forward;

procedure  Create_Recording;
begin
   Init_Utimes;
   Init_Nacts;
   Init_Hist;
   Init_TSeries;
end;

procedure  Cs_In_Progress;
begin
end;

procedure  StartUp;
begin
   InitVars;
   Set_SYSTEM_ENT(SYSTEM_ENT);
   Input_FillQue('PUB.QUE');
   Create_Recording;
   Cs_In_Progress;
   C_Phase;
end;

procedure  Report;
begin
   Open_Report_File('PUB.REP');
   Display_Act_Ntimes;
   Display_Ent_Utimes;
   Display_Sys_Values;
   Display_Histograms;
   Display_Time_Series;
   Close_Report_File('PUB.REP');
end;

procedure  C_ARRIVE;
var   Count:integer;
      Ment:mod_ent;
      Name:str10;
      Value:real;
      Desire1:real;
begin
   while ( Qsize(CLOSED) >= 1 )
   do
   begin
      ActTime:=NEGEXP(10,3);
      for count:=1 to 1 do
      begin

         { Evaluation of DESIRE }
         Ment  :=  NuminSource(OUTSIDE,count);
         Name  :=  'DESIRE';
         Value  :=  TRUNC(1+4*RND(5));
         Assign_Attribute(Ment,Name,Value);
         { End of Evaluation of DESIRE }
         LogAttData(NumInSource(OUTSIDE,count),Name,'DHIST');
      end;

         { Evaluation of TDESIRE }
         Ment  :=  SYSTEM_MOD_ENT;
         Name  :=  'TDESIRE';
         Desire1  :=
   Evaluate_Attribute(NumInSource(OUTSIDE,1),'DESIRE');
```

```
            Value  :=  Evaluate_Attribute(SYSTEM_MOD_ENT,Name);
            Value  := Value + Desire1;
            Assign_Attribute(Ment,Name,Value);
            { End of Evaluation of TDESIRE }

         Start_ScheduleB(ARRIVE);
         ScheduleB(1,TakeFromFront(CLOSED));
         ScheduleB(2,TakeFromSource(OUTSIDE));
         End_ScheduleB;
      end;
   end;

   procedure  B1_End_ARRIVE_DOOR;
   begin
      AddToBack(Current_ent,CLOSED);
   end;

   procedure  B2_End_ARRIVE_CUSTOMER;
   begin
      LogQueData(WAIT,'WAITLENGTH');
      AddToBack(Current_ent,WAIT);
   end;

   procedure  C_POUR;
   begin
      while ( Qsize(CLEAN) >= 1 )
         and ( Qsize(WAIT) >= 1 )
         and ( Qsize(IDLE) >= 1 )
      do
      begin
         ActTime:=NORMAL(6,1,7);
         Start_ScheduleB(POUR);
         ScheduleB(3,TakeFromFront(CLEAN));
         LogQueData(WAIT,'WAITLENGTH');
         ScheduleB(4,TakeFromFront(WAIT));
         ScheduleB(5,TakeFromFront(IDLE));
         End_ScheduleB;
      end;
   end;

   procedure  B3_End_POUR_GLASS;
   begin
      AddToBack(Current_ent,FULL);
   end;

   procedure  B4_End_POUR_CUSTOMER;
   begin
      AddToBack(Current_ent,READY);
   end;

   procedure  B5_End_POUR_BARMAID;
   begin
      AddToBack(Current_ent,IDLE);
   end;

   procedure  C_DRINK;
   var    Ment:mod_ent;
          Name:str10;
          Value:real;
          Desire1:real;
   begin
      while ( Qsize(FULL) >= 1 )
         and ( Qsize(READY) >= 1 )
```

```
   do
   begin
      ActTime:=UNIFORM(5,10,17);

      { Evaluation of DESIRE }
       Ment := NumInQue(READY,1);
       Name := 'DESIRE';
         Desire1 := Evaluate_Attribute(NumInQue(READY,1),Name);
       Value := Desire1 - 1;
         Assign_Attribute(Ment,Name,Value);
      { End of Evaluation of DESIRE }

      Start_ScheduleB(DRINK);
      ScheduleB(6,TakeFromFront(FULL));
      ScheduleB(7,TakeFromFront(READY));
      End_ScheduleB;
   end;
end;

procedure  B6_End_DRINK_GLASS;
begin
   AddToBack(Current_ent,DIRTY);
end;

pracedure  B7_End_DRINK_CUSTOMER;
var   Condition:boolean;
      Desire:real;
begin
   Desire  :=  Evaluate_attribute(Current_Ent,'DESIRE');
   Condition  :=  Desire>0;
   if  condition  then
   begin
      LogQueData(WAIT,'WAITLENGTH');
      AddToBack(Current_ent,WAIT);
   end  else
   begin
      Sink(Current_ent);
   end;
end;

procedure  C_WASHUP;
var   Count:integer;
begin
   while  (  Qsize(DIRTY)  >=  1  )
      and  (  Qsize(IDLE)  >=  1  )
   do
   begin
      ActTime:=5;
      Start_ScheduleB(WASHUP);
      for  count:=1  to  MINOF(3,QSIZE(DIRTY))  do
      begin
         ScheduleB(8,TakeFromFront(DIRTY));
      end;
      ScheduleB(9,TakeFromFront(IDLE));
      End_ScheduleB;
   end;
end;

procedure  B8_End_WASHUP_GLASS;
begin
   AddToBack(Current_ent,CLEAN);
end;
```

```
procedure  B9_End_WASHUP_BARMAID;
begin
   AddToBack(Current_ent,IDLE);
end;

procedure  EndRunIn;
begin
   Create_Recording;
end;

function  A_Phase:boolean;
begin
   increase_Time;
   A_Phase  :=  (TIM <= Duration);
end;

procedure  B_Phase;
begin
   while  (  Current_Time(TIM)  )  do
   begin
      while  (Get_Next_Bevent)  do
      begin
         case  Bevent_Num  of
            1  :  B1_End_ARRIVE_DOOR;
            2  :  B2_End_ARRIVE_CUSTOMER;
            3  :  B3_End_POUR_GLASS;
            4  :  B4_End_POUR_CUSTOMER;
            5  :  B5_End_POUR_BARMAID;
            6  :  B6_End_DRINK_GLASS;
            7  :  B7_End_DRINK_CUSTOMER;
            8  :  B8_End_WASHUP_GLASS;
            9  :  B9_End_WASHUP_BARMAID;
         127  :  EndRunIn;
         end;
      end;
   end;
end;

procedure  C_Phase;
begin
   C_ARRIVE;
    C_POUR;
    C_DRINK;
    C_WASHUP;
end;

procedure  Execute;
begin
   Open_File('PUB.RST');
   StartUp;
   if  (  Run_In_Period <> 0  )  then
      Init_Restart(127,Run_In_Period);
   while  (A_Phase)  do
   begin
      B_Phase;
      C_Phase;
   end;
   Report;
   Close_File('PUB.RST');
end;

begin           (* main program *)
   Initialise_Model(false);
```

```
   Build_Model;
   repeat
      Input_Duration;
      if ( Duration > 0 ) then Execute;
    until ( Duration<=0 );
end.
```

A.3. The Launderette Program

```
Program  LAUNDERETTE;

{$U  MacSim.Lib}
{$R+}
{$S+}

Uses
   {$S MGlob}   MemTypes,Quickdraw,OSIntf,ToolIntf,PackIntf,
                MSimGlobal,
   {$S MSamp}   MSimSample,
   {$S MMode}   MSimModel,
   {$S MOutp}   MSimOutput;

var
   CUSTOMER,WASH_MAC,BASKET,DRIER,ENTRANCE,SYSTEM_ENT  :  Entity;
   WASHQ,EIDLE,UNLDQ,WIDLE,WULDQ,TRPTQ,BIDLE,
   BTRPTQ,DRYQ,BDRYQ,DQ,DIDLE,DDQ  :  Queue;
   ARRIVE,LOADWASH,UNLOADWASH,TRANSPORT,LOADDRIER,
   DRY:  Activity;
   OUTSIDE  :  Source;

procedure  Build_Model;
begin
   MakeEnt(SYSTEM_ENT,'SYSTEM_ENT');
   MakeEnt(CUSTOMER,'CUSTOMER');
   MakeEnt(WASH_MAC,'WASH_MAC');
   MakeEnt(BASKET,'BASKET');
   MakeEnt(DRIER,'DRIER');
   MakeEnt(ENTRANCE,'ENTRANCE');
   MakeSou(CUSTOMER,OUTSIDE,'OUTSIDE');
   MakeQue(CUSTOMER,WASHQ,'WASHQ');
   MakeQue(ENTRANCE,EIDLE,'EIDLE');
   MakeQue(CUSTOMER,UNLOADQ,'UNLOADQ');
   MakeQue(WASH_MAC,WIDLE,'WIDLE');
   MakeTSeries(WIDLE,'WIDLETS',2);
   MakeQue(WASH_MAC,WUNLOADQ,'WUNLOADQ');
   MakeQue(CUSTOMER,TRANSPORTQ,'TRANSPORTQ');
   MakeQue(BASKET,BIDLE,'BIDLE');
   MakeQLenHist(BIDLE,'BIBLELEN',1,0);
   MakeQue(BASKET,BTRANSPORT,'BTRANSPORT');
   MakeQue(CUSTOMER,DRYQ,'DRYQ');
   MakeQue(BASKET,BDRYQ,'BDRYQ');
   MakeQue(CUSTOMER,DQ,'DQ');
   MakeQue(DRIER,DIDLE,'DIDLE');
   MakeQue(DRIER,DDQ,'DDQ');
   MakeAct(ARRIVE,'ARRIVE');
   MakeAct(LOADWASH,'LOADWASH');
   MakeAct(UNLOADWASH,'UNLOADWASH');
```

```
   MakeAct(TRANSPORT,'TRANSPORT');
   MakeAct(LOADDRIER,'LOADDRIER');
   MakeAct(DRY,'DRY');
end;

procedure  C_Phase;forward;

procedure  Create_Recording;
begin
   Init_Utimes;
   Init_Nacts;
   Init_Hist;
   Init_TSeries;
end;

procedure  Cs_In_Progress;
begin
end;

procedure  StartUp;
begin
   InitVars;
   Set_SYSTEM_ENT(SYSTEM_ENT);
   Input_FillQue('LAUNDERETTE.QUE');
   Create_Recording;
   Cs_In_Progress;
   C_Phase;
end;

procedure  Report;
begin
   Open_Report_File('LAUNDERETTE.REP');
   Display_Act_Ntimes;
   Display_Ent_Utimes;
   Display_Sys_Values;
   Display_Histograms;
   Display_Time_Series;
   Close_Report_File('LAUNDERETTE.REP');
end;

procedure  C_ARRIVE;
begin
   while ( Qsize(EIDLE) >= 1 )
   do
   begin
      ActTime:=NEGEXP(10,7);
      Start_ScheduleB(ARRIVE);
      ScheduleB(1,TakeFromSource(OUTSIDE));
      ScheduleB(2,TakeFromFront(EIDLE));
      End_ScheduleB;
   end;
end;

procedure  B1_End_ARRIVE_CUSTOMER;
begin
   AddToBack(Current_ent,WASHQ);
end;

procedure  B2_End_ARRIVE_ENTRANCE;
begin
   AddToBack(Current_ent,EIDLE);
end;
```

```
procedure  C_LOADWASH;
begin
   while ( Qsize(WASHQ) >= 1 )
      and ( Qsize(WIDLE) >= 1 )
   do
   begin
      ActTime:=25;
      Start_ScheduleB(LOADWASH);
      ScheduleB(3,TakeFromFront(WASHQ));
      ScheduleB(4,TakeFromFront(WIDLE));
      End_ScheduleB;
   end;
end;

procedure  B3_End_LOADWASH_CUSTOMER;
begin
   AddToBack(Current_ent,UNLOADQ);
end;

procedure  B4_End_LOADWASH_WASH_MAC;
begin
   AddToBack(Current_ent,WUNLOADQ);
end;

procedure  C_UNLOADWASH;
begin
   while ( Qsize(UNLOADQ) >= 1 )
      and ( Qsize(WUNLOADQ) >= 1 )
      and ( Qsize(BIDLE) >= 1 )
   do
   begin
      ActTime:=UNIFORM(1,4,8);
      Start_ScheduleB(UNLOADWASH);
      ScheduleB(5,TakeFromFront(UNLOADQ));
      ScheduleB(6,TakeFromFront(WUNLOADQ));
      ScheduleB(7,TakeFromFront(BIDLE));
      End_ScheduleB;
   end;
end;

procedure  B5_End_UNLOADWASH_CUSTOMER;
begin
   AddToBack(Current_ent,TRANSPORTQ);
end;

procedure  B6_End_UNLOADWASH_WASH_MAC;
begin
   AddToBack(Current_ent,WIDLE);
end;

procedure  B7_End_UNLOADWASH_BASKET;
begin
   AddToBack(Current_ent,BTRANSPORT);
end;

procedure  C_TRANSPORT;
begin
   while ( Qsize(TRANSPORTQ) >= 1 )
      and ( Qsize(BTRANSPORT) >= 1 )
   do
   begin
      ActTime:=UNIFORM(1,4,9);
      Start_ScheduleB(TRANSPORT);
```

```
      ScheduleB(8,TakeFromFront(TRANSPORTQ));
      ScheduleB(9,TakeFromFront(BTRANSPORT));
      End_ScheduleB;
   end;
end;

procedure  B8_End_TRANSPORT_CUSTOMER;
begin
   AddToBack(Current_ent,DRYQ);
end;

procedure  B9_End_TRANSPORT_BASKET;
begin
   AddToBack(Current_ent,BDRYQ);
end;

procedure  C_LOADDRIER;
begin
   while ( Qsize(DRYQ) >= 1 )
      and ( Qsize(BDRYQ) >= 1 )
      and ( Qsize(DIDLE) >= 1 )
   do
   begin
      ActTime:=2;
      Start_ScheduleB(LOADDRIER);
      ScheduleB(10,TakeFromFront(DRYQ));
      ScheduleB(11,TakeFromFront(BDRYQ));
      ScheduleB(12,TakeFromFront(DIDLE));
      End_ScheduleB;
   end;
end;

procedure  B10_End_LOADDRIER_CUSTOMER;
begin
   AddToBack(Current_ent,DQ);
end;

procedure  B11_End_LOADDRIER_BASKET;
begin
   AddToBack(Current_ent,BIDLE);
end;

procedure  B12_End_LOADDRIER_DRIER;
begin
   AddToBack(Current_ent,DDQ);
end;

procedure  C_DRY;
begin
   while ( Qsize(DQ) >= 1 )
      and ( Qsize(DDQ) >= 1 )
   do
   begin
      ActTime:=NORMAL(6,1,5);
      Start_ScheduleB(DRY);
      ScheduleB(13,TakeFromFront(DQ));
      ScheduleB(14,TakeFromFront(DDQ));
      End_ScheduleB;
   end;
end;

procedure  B13_End_DRY_CUSTOMER;
begin
```

```
   Sink(Current_ent);
end;

procedure  B14_End_DRY_DRIER;
begin
   AddToBack(Current_ent,DIDLE);
end;

procedure  EndRunIn;
begin
   Create_Recording;
end;

function  A_Phase:boolean;
begin
   Increase_Time;
   A_Phase  :=  (TIM  <=  Duration);
end;

procedure  B_Phase;
begin
   while  (  Current_Time(TIM)  )  do
   begin
      while  (Get_Next_Bevent)  do
      begin
         case  Bevent_Num  of
            1   :  B1_End_ARRIVE_CUSTOMER;
            2   :  B2_End_ARRIVE_ENTRANCE;
            3   :  B3_End_LOADWASH_CUSTOMER;
            4   :  B4_End_LOADWASH_WASH_MAC;
            5   :  B5_End_UNLOADWASH_CUSTOMER;
            6   :  B6_End_UNLOADWASH_WASH_MAC;
            7   :  B7_End_UNLOADWASH_BASKET;
            8   :  B8_End_TRANSPORT_CUSTOMER;
            9   :  B9_End_TRANSPORT_BASKET;
            10  :  B10_End_LOADDRIER_CUSTOMER;
            11  :  B11_End_LOADDRIER_BASKET;
            12  :  B12_End_LOADDRIER_DRIER;
            13  :  B13_End_DRY_CUSTOMER;
            14  :  B14_End_DRY_DRIER;
            127 :  EndRunIn;
         end;
      end;
   end;
end;

procedure  C_Phase;
begin
   C_ARRIVE;
   C_LOADWASH;
   C_UNLOADWASH;
   C_TRANSPORT;
   C_LOADDRIER;
   C_DRY;
end;

procedure  Execute;
begin
   Open_File('LAUNDERETTE.RST');
   StartUp;
   if  (  Run_In_Period  <>  0  )  then
      Init_Restart(127,Run_In_Period);
   while  (A_Phase)  do
```

```
    begin
       B_Phase;
       C_Phase;
    end;
    Report;
    Close_File('LAUNDERETTE.RST');
end;

begin                { main program }
    Initialise_Model(false);
     Build_Model;
    repeat
       Input_Duration;
       if ( Duration > 0 ) then Execute;
    until ( Duration<=0 );
end.
```

A.4. The Steelworks Program

```
Program  STEELWORKS;

{$U  MacSim.Lib}
{$R+}
{$S+}

Uses
   {$S MGlob}  MemTypes,Quickdraw,OSIntf,ToolIntf,PackIntf,
               MSimGlobal,
   {$S MSamp}  MSimSample,
   {$S MMode}  MSimModel,
   {$S MOutp}  MSimOutput;

var
   BLASTF,TORPEDO,PIT,CRANE,STEELF,SYSTEM_ENT  :  Entity;
   A,B,TBLOWQ,C,PITQ,CRANEQ,CREADY,QD,G,QE,LOADQ,QF  :  Queue;
   MELT,BLOW,GOING,FILL,LOADIN,REFINE,TRAVEL,RETURN  :  Activity;

procedure  Build_Model;
begin
   MakeEnt(SYSTEM_ENT,'SYSTEM_ENT');
   MakeAtt(SYSTEM_ENT,'WASTE');
   MakeAtt(SYSTEM_ENT,'TRANSFER');
   MakeEnt(BLASTF,'BLASTF');
   MakeAtt(BLASTF,'BCAST');
   MakeEnt(TORPEDO,'TORPEDO');
   MakeAtt(TORPEDO,'TCAST');
   MakeEnt(PIT,'PIT');
   MakeEnt(CRANE,'CRANE');
   MakeAtt(CRANE,'CCAST');
   MakeEnt(STEELF,'STEELF');
   MakeQue(BLASTF,A,'A');
   MakeQue(BLASTF,B,'B');
   MakeQue(TORPEDO,TBLOWQ,'TBLOWQ');
   MakeQLenHist(TBLOWQ,'TBLOWQLEN',1,0);
   MakeQTimHist(TBLOWQ,'TBLOWQTIM',5,0);
   MakeTSeries(TBLOWQ,'TBLOWQTSER',1);
   MakeQue(TORPEDO,C,'C');
```

```
    MakeQue(TORPEDO,PITQ,'PITQ');
    MakeQue(CRANE,CRANEQ,'CRANEQ');
    MakeQue(CRANE,CREADY,'CREADY');
    MakeQue(TORPEDO,QD,'QD');
    MakeQue(PIT,G,'G');
    MakeQue(CRANE,QE,'QE');
    MakeQue(STEELF,LOADQ,'LOADQ');
    MakeQue(STEELF,QF,'QF');
    MakeAct(MELT,'MELT');
    MakeAct(BLOW,'BLOW');
    MakeAct(GOING,'GOING');
    MakeAct(FILL,'FILL');
    MakeAct(LOADIN,'LOADIN');
    MakeAct(REFINE,'REFINE');
    MakeAct(TRAVEL,'TRAVEL');
    MakeAct(RETURN,'RETURN');
end;

procedure  C_Phase;forward;

procedure  Create_Recording;
begin
    Init_Utimes;
    Init_Nacts;
    Init_Hist;
    Init_TSeries;
end;

procedure  Cs_In_Progress;
begin
end;

procedure  StartUp;
begin
    InitVars;
    Set_SYSTEM_ENT(SYSTEM_ENT);
    Input_FillQue('STEELWORKS.QUE');
    Create_Recording;
    Cs_In_Progress;
    C_Phase;
end;

procedure  Report;
begin
    Open_Report_File('STEELWORKS.REP');
    Display_Act_Ntimes;
    Display_Ent_Utimes;
    Display_Sys_Values;
    Display_Histograms;
    Display_Time_Series;
    Close_Report_File('STEELWORKS.REP');
end;

procedure  C_MELT;
var    Ment:mod_ent;
       Name:str10;
       Value:real;
begin
    while ( Qsize(A) >= 1 )
    do
    begin
       ActTime:=NORMAL(110,15,1);
       { Evaluation of attribute 'BCAST' }
```

```
        Ment := NumInQue(A,1);
        Name := 'BCAST';
        Value := NORMAL(360,50,2);
         Assign_Attribute(Ment,Name,Value);
      { End of Evaluation of attribute 'BCAST' }
      Start_ScheduleB(MELT);
      ScheduleB(1,TakeFromFront(A));
      End_ScheduleB;
   end;
end;

procedure  B1_End_MELT_BLASTF;
begin
   AddToBack(Current_ent,B);
end;

procedure  C_BLOW;
var   Count:integer;
      Condition:boolean;
      Ment:mod_ent;
      Name:str10;
      Value:real;
      Bcast1:real;
begin
   while ( Qsize(TBLOWQ) >= 0 )
      and ( Qsize(B) >= 1 )
   do
   begin
      ActTime:=10;

      Condition := QSIZE(TBLOWQ)>=2;
      if Condition then
      begin

         { Evaluation of attribute TCAST1 }
         Ment := NumInQue(TBLOWQ,1);
         Name := 'TCAST';
         Value := 300;
         Assign_Attribute(Ment,Name,Value);
         { End of Evaluation of attribute TCAST1 }

         { Evaluation of attribute TCAST2 }
         Ment := NumInQue(TBLOWQ,2);
         Name := 'TCAST';
         Bcast1 := Evaluate_Attribute(NumInQue(B,1),'BCAST');
         Value := Bcast1 - 300;
         Assign_Attribute(Ment,Name,Value);
         { End of Evaluation of attribute TCAST2 }

      end;

      Condition := QSIZE(TBLOWQ)=1;
      If Condition then
      begin

         { Evaluation of attribute TCAST1 }
         Ment := NumInQue(TBLOWQ,1);
         Name := 'TCAST';
         Value := 300;
         Assign_Attribute(Ment,Name,Value);
         { End of Evaluation of attribute TCAST1 }

         { Evaluation of attribute WASTE }
```

```
            Ment  := SYSTEM_MOD_ENT;
            Name  := 'WASTE';
            Bcast1  :=  Evaluate_Attribute(NumInQue(B,1),'BCAST');
            Value  :=  Evaluate_Attribute(SYSTEM_MOD_ENT,'WASTE');
            Value := Value + Bcast1 - 300;
            Assign_Attribute(Ment,Name,Value);
            { End of Evaluation of attribute WASTE }

         end;

         Condition  :=  QSIZE(TBLOWQ)=0;
         if Condition then
         begin

            { Evaluation of attribute WASTE }
            Ment  := SYSTEM_MOD_ENT;
            Name  := 'WASTE';
            Bcast1  :=  Evaluate_Attribute(NumInQue(B,1),'BCAST');
            Value  :=  Evaluate_Attribute(SYSTEM_MOD_ENT,'WASTE');
            Value := Value + Bcast1;
            Assign_Attribute(Ment,Name,Value);
            { End of Evaluation of attribute WASTE }

         end;

         Start_ScheduleB(BLOW);
         for count:=1 to MINOF(2,QSIZE(TBLOWQ)) do
         begin
            LogQueData(TBLOWQ,'TBLOWQLEN');
            ScheduleB(2,TakeFromFront(TBLOWQ));
         end;
         ScheduleB(3,TakeFromFront(B));
         End_ScheduleB;
      end;
   end;

   procedure  B2_End_BLOW_TORPEDO;
   begin
      AddToBack(Current_ent,C);
   end;

   procedure  B3_End_BLOW_BLASTF;
   begin
      AddToBack(Current_ent,A);
   end;

   procedure  C_GOING;
   var   Count:integer;
   begin
      while ( Qsize(C) >= 1 )
      do
      begin
         ActTime:=POISSON(10,3);
         Start_ScheduleB(GOING);
         for count:=1 to MINOF(2,QSIZE(C)) do
         begin
            ScheduleB(4,TakeFromFront(C));
         end;
         End_ScheduleB;
      end;
   end;

   procedure  B4_End_GOING_TORPEDO;
```

```
begin
   AddToBack(Current_ent,PITQ);
end;

procedure  C_FILL;
var    Ment:mod_ent;
       Name:str10;
       Value:real;
       Tcast1:real;
       Ccast1:real;
       Transfer:real;
begin
   while ( Qsize(CRANEQ) >= 1 )
      and ( Qsize(PITQ) >= 1 )
      and ( Qsize(G) >= 1 )
   do
   begin
      ActTime:=10;

      { Evaluation of TRANSFER }
        Ment := SYSTEM_MOD_ENT;
      Name  :=  'TRANSFER';
      Tcast1  :=  Evaluate_Attribute(NuminQue(PITQ,1),'TCAST');
      Ccast1  :=  100-
                  Evaluate_Attribute(NumInQue(CRANEQ,1),'CCAST');
        Value  :=  MINOF(Tcast1,Ccast1);
         Assign_Attribute(Ment,Name,Value);
      { End of Evaluation of TRANSFER }

      { Evaluation of TCAST }
      Ment  :=  NumInQue(PITQ,1);
      Name  :=  'TCAST';
      Tcast1  :=  Evaluate_Attribute(NumInQue(PITQ,1),'TCAST');
      Transfer  :=
                  Evaluate_Attribute(SYSTEM_MOD_ENT,'TRANSFER');
      Value  :=  Tcast1  -  Transfer;
      Assign_Attribute(Ment,Name,Value);
      { End of Evaluation of TCAST }

      { Evaluation of CCAST }
      Ment  :=  NumInQue(CRANEQ,1);
      Name  :=  'CCAST';
      Ccast1  :=  Evaluate_Attribute(NumInQue(CRANEQ,1),'CCAST');
      Transfer  :=
                  Evaluate_Attribute(SYSTEM_MOD_ENT,'TRANSFER');
      Value  :=  Ccast1  +  Transfer;
      Assign_Attribute(Ment,Name,Value);
      { End of Evaluation of CCAST }

      Start_ScheduleB(FILL);
      ScheduleB(5,TakeFromFront(CRANEQ));
      ScheduleB(6,TakeFromFront(PITQ));
      ScheduleB(7,TakeFromFront(G));
      End_ScheduleB;
   end;
end;

procedure  B5_End_FILL_CRANE;
var    Condition:boolean;
       Ccast:real;
begin
   Ccast  :=  Evaluate_attribute(Current_Ent,'CCAST');
   Condition  :=  Ccast<100;
```

```
    if Condition then
    begin
       AddToFront(Current_ent,CRANEQ);
    end else
    begin
       AddToBack(Current_ent,CREADY);
    end;
end;

procedure  B6_End_FILL_TORPEDO;
var    Condition:boolean;
       Tcast:real;
begin
    TCast  :=  Evaluate_attribute(Current_Ent,'TCAST');
    Condition  :=  Tcast>0;
    If Condition then
    begin
       AddToFront(Current_ent,PiTQ);
    end else
    begin
       AddToBack(Current_ent,QD);
    end;
end;

procedure  B7_End_FILL_PIT;
begin
    AddToBack(Current_ent,G);
end;

procedure  C_LOADIN;
var    Ment:mod_ent;
       Name:str10;
       Value:real;
begin
    while ( Qsize(CREADY)  >=  1  )
       and ( Qsize(LOADQ)  >=  1  )
    do
    begin
       ActTime:=10;

       { Evaluation of Ccast }
       Ment  :=  NumInQue(CREADY,1);
       Name  :=  'CCAST';
       Value  :=  0;
       Assign_Attribute(Ment,Name,Value);
       { End of Evaluation of Ccast }

       Start_ScheduleB(LOADIN);
       ScheduleB(8,TakeFromFront(CREADY));
       ScheduleB(9,TakeFromFront(LOADQ));
       End_ScheduleB;
    end;
end;

procedure  B8_End_LOADIN_CRANE;
begin
    AddToBack(Current_ent,QE);
end;

procedure  B9_End_LOADIN_STEELF;
begin
    AddToBack(Current_ent,QF);
end;
```

```
procedure  C_REFINE;
begin
   while ( Qsize(QF) >= 1 )
   do
   begin
      ActTime:=50+NEGEXP(10,4);
      Start_ScheduleB(REFINE);
      ScheduleB(10,TakeFromFront(QF));
      End_ScheduleB;
   end;
end;

procedure  B10_End_REFINE_STEELF;
begin
   AddToBack(Current_ent,LOADQ);
end;

procedure  C_TRAVEL;
begin
   while ( Qsize(QE) >= 1 )
   do
   begin
      ActTime:=2;
      Start_ScheduleB(TRAVEL);
      ScheduleB(11,TakeFromFront(QE));
      End_ScheduleB;
   end;
end;

procedure  B11_End_TRAVEL_CRANE;
begin
   AddToBack(Current_ent,CRANEQ);
end;

procedure  C_RETURN;
begin
   while ( Qsize(QD) >= 1 )
   do
   begin
      ActTime:=4;
      Start_ScheduleB(RETURN);
      ScheduleB(12,TakeFromFront(QD));
      End_ScheduleB;
   end;
end;

procedure  B12_End_RETURN_TORPEDO;
begin
   LogQueData(TBLOWQ,'TBLOWQLEN');
   AddToBack(Current_ent,TBLOWQ);
end;

procedure  EndRunIn;
begin
   Create_Recording;
end;

function  A_Phase:boolean;
begin
   Increase_Time;
   A_Phase  :=  (TIM <= Duration);
end;
```

```
procedure  B_Phase;
begin
   while  ( Current_Time(TIM)  )  do
   begin
      while  (Get_Next_Bevent)  do
      begin
         case  Bevent_Num  of
            1  :  B1_End_MELT_BLASTF;
            2  :  B2_End_BLOW_TORPEDO;
            3  :  B3_End_BLOW_BLASTF;
            4  :  B4_End_GOING_TORPEDO;
            5  :  B5_End_FILL_CRANE;
            6  :  B6_End_FILL_TORPEDO;
            7  :  B7_End_FILL_PIT;
            8  :  B8_End_LOADIN_CRANE;
            9  :  B9_End_LOADIN_STEELF;
            10  :  B10_End_REFINE_STEELF;
            11  :  B11_End_TRAVEL_CRANE;
            12  :  B12_End_RETURN_TORPEDO;
            127  :  EndRunIn;
         end;
      end;
   end;
end;

procedure  C_Phase;
begin
   C_MELT;
   C_BLOW;
   C_GOING;
   C_FILL;
   C_LOADIN;
   C_REFINE;
   C_TRAVEL;
   C_RETURN;
end;

procedure  Execute;
begin
   Open_File('STEELWORKS.RST');
   StartUp;
   if  ( Run_In_Period <> 0  )  then
      Init_Restart(127,Run_In_Period);
   while  (A_Phase)  do
   begin
      B_Phase;
      C_Phase;
   end;
   Report;
   Close_File('STEELWORKS.RST');
end;

begin                { main program }
   Initialise_Model(false);
   Build_Model;
   repeat
      Input_Duration;
      if  ( Duration > 0  )  then  Execute;
    until  ( Duration<=0  );
end.
```

APPENDIX B

MACSIM.LIB : THE MACINTOSH SIMULATION LIBRARY

B.1. THE MSIMGLOBAL UNIT

B.1.1. Type Definition

B.1.2. Global Variables

B.2. THE MSIMSAMPLE UNIT

B.2.1. Sampling Routines

B.2.2. Distribution Functions

B.2.3. Arithematic Functions

B.3. THE MSIMMODEL UNIT

B.3.1. Initialisation Routines

B.3.2. Model Setup Routines

B.3.3. Recording and Assigning Routines

B.3.4. C Event Routines

B.3.5. Increasing Time Routines

B.3.6. B Event Routines

B.3.7. Interface Routines

B.4. THE MSIMOUTPUT UNIT

B.4.1. Displaying Routines

B.4.2. Filing Routines

This appendix describes the syntax and the functions of *MacSim.Lib* - the set of simulation library routines which was written in Turbo Pascal on the Apple Macintosh. MacSim.Lib was written in four units - *MSimGlobal*, *MSimSample*, *MSimModel*, and *MSimOutput*. These units are discussed in sections B.1, B.2, B.3 and B.4 respectively.

B.1. The MSimGlobal Unit

This is the declaration unit of the program. The type definition and the global variables in the simulation library are shown in figure B.1.

Figure B.1. The MSimGlobal Unit

```
Unit  MSimGlobal(1);

{$0  MacSim.Lib}

interface

Uses  MemTypes,QuickDraw,OSIntf,ToolIntf,PackIntf;

Type
      str10        =  string[10];

      histogram    =  ^histptr;
      histptr      =  ^histrec;
      histrec      =  record
                          name          :  str10;
                          htype         :  integer;
                          base,
                          width,
                          count,
                          tflag         :  longint;
                          total,
                          sosq          :  real;
                          data           :  array[0..16] of longint;
                    end;

      attribute    =  ^attptr;
      attptr       =  ^attrec;
      attrec       =  record
                          name          :  str10;
                          hist          :  histogram;
                          next          :  attribute;
                    end;

      entity       =  ^entptr;
```

```
entptr      = ^entrec;
entrec      = record
                  name          : str10;
                  num           : integer;
                  att           : attribute;
                  utime         : real;
                  next          : entity;
              end;

tseries     = ^tseriesptr;
tseriesptr  = ^tseriesrec;
tseriesrec  = record
                  name          : str10;
                  interval      : longint;
                  data          : array[0..449] of integer;
                  count         : integer;
                  plot          : integer;
                  tflag         : longint;
              end;

modent      = ^modentptr;
modentptr   = ^modentrec;

queue       = ^queptr;
queptr      = ^querec;
querec      = record
                  name          : str10;
                  ent           : entity;
                  ment          : modent;
                  count         : integer;
                  qlhist        : histogram;
                  qthist        : histogram;
                  ts            : tseries;
                  next          : queue;
              end;

source      = ^souptr;
souptr      = ^sourec;
sourec      = record
                  name          : str10;
                  ent           : entity;
                  buffer        : modent;
                  count         : integer;
                  buffercount   : integer;
                  next          : source;
              end;

activity    = ^actptr;
actptr      = ^actrec;
actrec      = record
                  name          : str10;
                  count         : integer;
                  next          : activity;
              end;

modatt      = ^modattptr;
modattptr   = ^modattrec;
modattrec   = record
                  att           : attribute;
                  value         : real;
                  next          : modatt;
              end;
```

```
        modentrec    = record
                            ent          : entity;
                            index        : integer;
                            matt         : modatt;
                            bnum         : integer;
                            clock        : longint;
                            next         : modent;
                       end;
Var
        TIM,
        Duration,
        Run_in_period,
        ActTime              : longint;

        Mainent              : entity;

        Mainact              : activity;

        Mainque              : queue;

        Mainsou              : source;

        Current_ent,
        SYSTEM_modent        : modent;

        Bevent_num           : integer;

        Fileset              : boolean;

        F,RF                 : text;

        NofRun               : integer;
```

B.1.1. Type Definition

Entity is a handle to the record type *entrec* which contains information about an entity type that is specified in the simulation program. The field *name* is the name of the entity type. *num* is the number of entities of an entity type that are present in the system. *att* is a handle to the list of attributes that belongs to the entity type. The field *utime* is used for accumulating the units of time that the members of an entity type have been involved in activities during a simulation run. *next* is a handle to the next entity in the entity link list.

Attribute is a handle to the record type *attrec* which contains information about an attribute which belongs to an entity type. Each attribute is assigned to an entity type in the model. The field *name* is the name of the attribute. *hist* is a handle to the

histogram that belongs to the attribute. If the attribute does not have a histogram, *hist* is set to 'nil'. *next* is the next attribute in the attribute list.

Activity is a handle to the record type *actrec* which contains information about an activity that is specified in the program. The field *name* is the name of the activity. *Count* is a counter variable to indicate how many times the activity has taken place during a simulation run. *next* is a handle to the next activity in the activity link list.

Queue is a handle to the record type *querec* which contains information about a queue that is specified in the program. The field *name* is the name of the queue. The entity handle *ent* is a handle to the entity to which the queue belongs. *ment* is the list of model entities (modents) that are present in the queue during a simulation run. *count* is used for recording the size of the queue during a simulation run. *qlhist* and *qthist* are histogram handles of type queue length and queueing time respectively. *ts* is a handle to the time series of the queue, if any. *next* is a handle to the next queue in the queue link list.

Source is a handle to the record type *sourec* which contains information about a source/sink queue that is specified in the program. The field *name* is the name of the source/sink queue. *ent* is a handle to the entity to which the queue belongs. *buffer* is the model entity that is present in the queue during a simulation run. *count* is used for recording the size of the queue during a simulation run. *buffercount* is a counter to record the number of temporary entities that has been created from the source buffer. *next* is a handle to the next source/sink queue in the list.

Histogram is a handle to the record type *histrec* which contains information about a histogram, belonging either to a queue or an attribute. The field *name* is the name of the histogram. *htype* can either be of value 1, 2 or 3. The value '1' indicates that the histogram is of type queue length whereas '2' indicates that it is of type queueing time. A value of '3', indicates that the histogram is an attribute histogram. The field *base* and *width* are the base value and the cell width of the histogram

respectively. *data* is the array of data that holds the frequency count of the appropriate cell during a simulation run. *count* is the number of times recording has taken place during a run. *tflag* is used by a queueing time histogram to store the last time that the histogram was recorded before a new reading takes place. *total* is the summation of all the individual cell value multipled by its appropriate frequency count. *sosq* is the sum of squares of cell values multipled by its frequency count.

Tseries is a handle to the record type *tseriesrec* which contains information about a time series specified for a queue. The field *name* is the name of the time series. *Interval* is the interval of the time axis of the time series, i.e. the gap between two consecutive recordings. *count* is the number of times recording has taken place during a simulation run. *tflag* is used for storing the last time that the recording has taken place before a new reading is taken. *plot* is used for specifying the drawing mode of the time series. A value of '1', '2', or '3' indicates that the output time series takes the form of a bar chart, line plot or scatter plot respecitively. *data* is the array of data that holds the reading of queue length during a simulation run.

Modent is a handle to the record type *modentrec* which contains information about an entity that is being scheduled during a simulation run. The field *ent* is a handle pointing to the entity to which this copy of modent belongs. *modatt* is the list of model attributes which the model entity possesses and is used during a simulation run. *index* is the rank of the model entity among all its associate members of the same entity type. Each model entity carries a *clock* which is used for recording the time that the entity enters a queue. *next* is a handle to the next modent in the list.

Modatt is a handle to the record type *modattrec* which contains the value of an attribute belonging to a model entity during a simulation run. The field *att* is a handle pointing to the attribute to which this copy of modatt belongs. *value* is the variable that stores the current value of the modatt during a simulation run. *next* is a handle to the next modatt in the list.

B.1.2. Global Variables

TIM is the time of the simulation clock during a simulation run.

Duration is the duration of the simulation run defined by the user.

Run_in_period is the run-in period (i.e. the time elapsed before the actual recording of data takes place) of the simulation run defined by the user.

ActTime is the activity duration of the activity of interest during a simulation run.

Mainent is the global variable that holds the information about all the different entity types that exist in the program.

Mainact is the global variable that holds the information about all the activities specified in the program.

Mainque is the global variable that holds the information about all the queues that are defined in the program.

Mainsou is the global variable that holds the information about all the source/sink queues that are defined in the program.

Current_ent is a handle to the current model entity that is being scheduled in the event loop during a simulation run.

System_modent is a handle to the system model entity that is being scheduled in the event loop during a simulation run.

Bevent_num is an index to each of the B event procedures that are present in the simulation program.

Fileset is a boolean value used for checking the existence of a specified file used

by the program.

F and *RF* are the global variables for the result file and the report file specified in the program respectively.

NofRun is the counter for the number of successive runs that are being executed by the user.

B.2. The MSimSample Unit

This unit contains all the sampling routines that can be called by the simulation program. The declaration part of the unit is shown in figure B.2. Any user-defined arithematic or statistical routines that are used for determining the activity duration of an activity or for assigning values to an attribute should be amended in this unit.

Figure B.2. The MSimSample Unit

```
Unit  MSimSample(2);

{$U  MacSim.Lib}
{$O  MacSim.Lib}

Interface

uses  MemTypes,QuickDraw,OSIntf,ToolIntf,PackIntf,
      MSimGlobal;

const dsize              = 20;    (* sampling *)
      streamnum          = 20;

type  Stre               = 1..streamnum;
      distdata           = array [1..dsize] of real;
      distribution       = ^distrec;
      distrec            = record
                               x,y          : distdata;
                               stream       : Stre;
                           end;

var   seedx,origseedx,
      seedy,origseedy,
      seedz,origseedz          : array[Stre] of integer;

{===============   SAMPLING ROUTINES   ================}
Procedure   InitSam;
Procedure   MakeDist(var  dist:distribution;  s:Stre;
```

```
                              xx:distdata;yy:distdata);
Function     Rnd(s:Stre):Real;
Function     Sample(dist:distribution):Real;

{===============   DISTRIBUTION FUNCTIONS   ===============}
Function     Bernoulli(pr:real;  s:stre):integer;
Function     Binomial(pr:real;  k:integer;  s:stre):integer;
Function     Erlang(ei:integer;  m:real;  s:stre):integer;
Function     LogNormal(m,sd:real;s:stre):integer;
Function     Negexp(m:Real;s:Stre):integer;
Function     Normal(m,sd:real;s:stre):integer;
Function     Poisson(m:Real;s:stre):integer;
Function     Uniform(a,b:integer;s:stre)  :  integer;
Function     Welbull(a,b:real;s:stre):integer;

{================   ARITHEMATIC FUNCTIONS   ================}
Function     Minof(a,b:Real):integer;
Function     Maxof(a,b:Real):integer;
```

B.2.1. Sampling Routines

InitSam is a procedure to initialise the variables that are used for generating a random number in the *Rnd* procedure.

MakeDist is a procedure for creating a distribution by specifying the stream number in the parameter *s*, the distdata in the parameter *xx* and *yy* respectively. The distribution is returned in the variable *dist*.

Rnd is a procedure used for generating a random number between 0 and 1 by using the specified stream number *s*.

Sample is a function that returns a real number from the given distribution referenced by *dist*.

B.2.2. Distribution Functions

Bernoulli returns an integer value of either 0 or 1 from a bernoulli distribution specified by the given parameter *pr* and the stream number *s*.

Binomial returns an integer value from a binomial distribution specified by the given parameter *pr* and *k*, and the stream number *s*.

Erlang returns an integer value from an erlang distribution specified by the given parameter *ei*, with mean *m* and stream number *s*.

LogNormal returns an integer value from a lognormal distribution with mean *m*, standard deviation *sd* and stream number *s*.

Negexp returns an integer value from a negative exponential distribution with mean *m* and stream number *s*.

Normal returns an integer value from a normal distribution with mean *m*, standard deviation *sd*, and stream number *s*.

Poisson returns an integer value from a Poisson distribution with mean *m* and stream number *s*.

Uniform returns an integer value that is uniformly distributed between parameter *a* and *b*, with stream number specified in *s*.

Weibull returns an integer value from a weibull distribution specified by parameter *a* and *b*, with stream number *s*.

B.2.3. Arithematic Functions

Minof is a function which returns the minimal of two values specified in parameter *a* and *b*.

Maxof is a function which returns the maximal of two values specified in parameter *a* and *b*.

B.3. The MSimModel Unit

This unit handles all the modelling routines in the simulation program. The declaration part of the unit is shown in figure B.3.

Figure B.3. The MSimModel Unit

```
unit  MSimModel(3);

{$U  MacSim.Lib}
{$O  MacSim.Lib}

Interface

uses  MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
      MSimGlobal, MSimSample;

{==============     INITIALISATION ROUTINES    =============}
procedure    Init_Utimes;
procedure    Init_Nacts;
procedure    Init_Hist;
procedure    Init_Tseries;
Procedure    InitVars;

{==============   MODEL SETUP ROUTINES   ================}
procedure    MakeEnt(var  nent:entity;  nam:str10);
procedure    MakeAtt(var  nent:entity;  nam:str10);
procedure    MakeAct(var  ac:activity;anam:str10);
procedure    MakeSou(nent:entity;  var  sou:source;  snam:str10);
procedure    MakeQue(nent:entity;  var  que:queue;  qnam:str10);
procedure    Set_SYSTEM_ENT(sent  :  entity);
procedure    MakeAttHist(var  nent:entity;  anam,hnam:str10;
                                        wid,bas:longint);
procedure    MakeQLenHist(var  que:queue;  hnam:str10;
                                       wid,bas:longint);
procedure    MakeQTimHist(var  que:queue;  hnam:str10;
                                       wid,bas:longint);
procedure    MakeTseries(var  que:queue;  tnam:str10;  d:integer);

{=============   RECORDING & ASSIGNING ROUTINES =============}
function     Qsize(que:queue):integer;
procedure    LogQueData(que:queue;  hnam:str10);
procedure    LogAttData(ment:modent;  anam:str10;  hnam:str10);
function     Evaluate_attribute(cent:modent;  anam:str10):real;
procedure    Assign_attribute(cent:modent;  anam:str10;
                                        val:real);

{==============   C EVENTS ROUTINES ==============}
function     FirstInQue(que:queue):modent;
function     LastInQue(que:queue):modent;
function     Takefromfront(que:queue):modent;
function     Takefromback(que:queue):modent;
function     TakefromSource(sou:source):modent;
procedure    Start_ScheduleB(sact:activity);
procedure    ScheduleB(b:integer;ment:modent);
procedure    End_scheduleB;
```

```
procedure    DescheduleB(act:activity);

{==============   INCREASING TIME ROUTINES  ==============}
function     Current_time(t:longint):boolean;
function     Get_next_bevent:boolean;
procedure    Increase_time;
procedure    Init_Restart(bnum,runin:integer);

{==============   B EVENTS ROUTINES   ==============}
procedure    Addtoback(cent:modent;  var  que:queue);
procedure    Addtofront(cent:modent;  var  que:queue);
function     NumInQue(que:queue;num:integer):modent;
function     NumInSource(sou:source;num:integer):modent;
procedure    Sink(cent:modent);

{===============   INTERFACE   ===============}
procedure    Select_Mode;
procedure    Initialise_model(on:boolean);
Procedure    Input_FillQue(fname:str255);
procedure    Input_duration;
```

B.3.1. Initialisation Routines

Init_Utimes is a procedure used to initialise the *utime* field in each entity type record in the entity link list to '0.0' before a simulation run.

Init_Nacts is a procedure used to initialise the *count* field in each activity record in the activity link list to '0' before a simulation run.

Init_Hist is a procedure used to initialise the fields used for recording in each histogram record before a simulation run.

Init_Tseries is a procedure used to initialise the fields used for recording in each time series record before a simulation run.

InitVars is a procedure used to initialise the timing tree and empty all the queues before a simulation run.

B.3.2. Model Setup Routines

MakeEnt is used to create a dynamic allocation for the entity record *nent* with name *nam*. This variable is then linked to the *Mainent* link list.

MakeAtt is used to create a dynamic allocation for the attribute record belong to the entity variable *nent* with name *nam*. This attribute is then linked to the *att* field within the entity record *nent*.

MakeAct is used to create a dynamic allocation for the activity record *ac* with name *anam*. This variable is then linked to the *Mainact* link list.

MakeSou is used to create a dynamic allocation for the source/sink queue record *sou* with name *snam*, belonging to the entity referenced by *nent*. This variable is then linked to the *Mainsou* link list.

MakeQue is used to create a dynamic allocation for the queue record *que* with name *qnam*, belonging to entity referenced by *nent*. This variable is then linked to the *Mainque* link list.

Set_System_Ent is used to set up the system entity specified by *sent*.

Make_Att_Hist is used to create a dynamic allocation for the histogram record with name *hnam*, cell width *wid* and base value *bas*, belong to the attribute referenced by name *anam* of the entity *nent*.

Make_QLenHist is used to create a dynamic allocation for a queue length histogram record with name *hnam*, cell width *wid* and base value *bas*, belonging to the queue *que*.

Make_QTimHist is used to create a dynamic allocation for a queueing time histogram record with name *hnam*, cell width *wid* and base value *bas*, belonging to the queue *que*.

Make_Tseries is used to create a dynamic allocation for the time series record with name *tnam*, drawing mode *d*, belonging to the queue *que*.

B.3.3. Recording and Assigning Routines

QSize is a function that returns the number of entities that are in the queue *que*.

LogQueData is a procedure used for recording the length of the queue *que* for the histogram with name *hnam*.

LogAttData is a procedure used for recording the value of the attribute with name *anam* which belong to the model entity *ment*, for the histogram with name *hnam*.

Evaluate_attribute is a function that returns a real number from the *value* field of the modatt record specified by the name of the attribute *anam* and the model entity *cent* to which the modatt belongs.

Assign_attrobute is a function that assigns a value *val* to the *value* field of a modatt record specified by the name of the attribute *anam* and the model entity *cent* to which the modatt belongs.

B.3.4. C Event Routines

FirstInQue returns the first modent that is present in the queue *que*.

LastInQue returns the last modent that is present in the queue *que*.

TakefromFront returns the first member in the queue *que* and removes it from the front of the queue.

TakefromBack returns the last member in the queue *que* and removes it from the

back of the queue.

TakefromSource creates a model entity from the source/sink queue *sou* and removes it from the source/sink queue.

Start_ScheduleB prepares the activity *sact* to be scheduled in the simulation run.

ScheduleB schedules the B event referenced by *b* with the model entity *ment.*

End_ScheduleB marks the end of the scheduling process of the activity.

DescheduleB deschedules the activity *act* by removing the activity from the timing tree during a simulation run.

B.3.5. Increasing Time Routines

Current_time is a boolean function that checks the current time *Tim* of the simulation clock in a simulation run. It returns 'true' if the current time is less than the duration of the run and 'false' otherwise.

Get_Next_Bevent is a boolean function that checks the next B events that are due to occur in a simulation run. It returns 'true' if there is a B event available in the event queue and 'false' otherwise. The value of the next B event is returned in *Bevent_num.*

Increase_time is a procedure used to move the simulation clock to when something next happens in the system, according to the timing tree.

Init_Restart is used at the end of the run-in period during a simulation run to prepare the system for actual recording to begin.

B.3.6. B Event Routines

AddtoBack adds the model entity *cent* to the back of the queue *que*.

AddtoFront adds the model entity *cent* to the front of the queue *que*.

NumInQue returns the *num*th model entity in the queue *que*.

NumInSou returns the *num*th model entity in the source/sink queue *sou*.

Sink adds the model entity *cent* to its associated source/sink queue and change its identity for later use.

B.3.7. Interface Routines

Select_Mode is a procedure which allows the user to select the run options, to enter the simulation environment and to quit the program. There are three run options available : text run, visual run or no-display run. The user can also turn the result file option on if it is required to look at the events that happen at every time advance.

Initialise_Model is a procedure for initialising all the variables in the system. The parameter *on* should be set to 'true' if visual run is available in the simulation program. The default value of *on* is 'false'.

Input_FillQue is a procedure used for reading the '.QUE' file, i.e. the file which stores the location and number of entities of each entity type that are present in the system. The model entities are also prepared for scheduling within this procedure.

Input_duration is a procedure used to accept the duration, run-in-period and the running speed specified by the user before a simulation run.

B.4. The MSimOutput Unit

This unit contains the displaying results routines after a simulation run and the filing routines in the system. The declaration part of the unit is shown in figure B.4. Any additional displaying or filing routines should be amended in this unit.

Figure B.4. The MSimOutput Unit

```
Unit  MSimOutput(4);

{$U  MacSim.Lib}
{$O  MacSim.Lib}

Interface

uses  MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,
      MSimGlobal, MSimSample, MSimModel;

{=============== DISPLAYING ROUTINES ===============}
procedure    Display_act_ntimes;
procedure    Display_ent_utimes;
procedure    Display_sys_values;
procedure    Display_hist(h:histogram);
procedure    Display_histograms;
procedure    Display_tseries(q:str10;t:tseries);
procedure    Display_time_series;

{=============== FILING ROUTINES ===============}
procedure    Open_File(fname:str255);
procedure    Close_File(fname:str255);
procedure    Open_Report_File(fname:str255);
procedure    Close_Report_File(fname:str255);
```

B.4.1. Displaying Routines

Displaying_act_ntimes is a procedure used for displaying the activity count table which indicates the number of times each activity has successfully started during a simulation run.

Displaying_ent_utimes is a procedure used for displaying the entity utilisation time chart which indicates the proportion of time that each individual entity type has been involved in activities during a simulation run.

Displaying_hist is a procedure which handles the drawing of a histogram specified by the parameter *h* and produces some basic statistics including the mean, standard deviation and variance of the histogram.

Displaying_histograms is a procedure which displays all the histograms (one after another) that are specified in the simulation program.

Displaying_tseries is a procedure which draws the time series referenced by the parameter *t* and *q* - the name of the queue to which the time series belongs.

Displaying_time_series is a procedure which displays all the time series (one after another) that are specified in the simulation program.

B.4.2. Filing Routines

Open_File rewrites a file referenced by the global variable *F*, specified by the file name *fname* so that simulation results can be written to this file.

Close_File closes the result file *F*, specified by the file name *fname*.

Open_Report_File rewrites a file referenced by the global variable *RF*, specified by the file name *fname*. The resulting output after a simulation run is written to this file.

Close_Report_File closes the result file *RF*, specified by the file name *fname*.

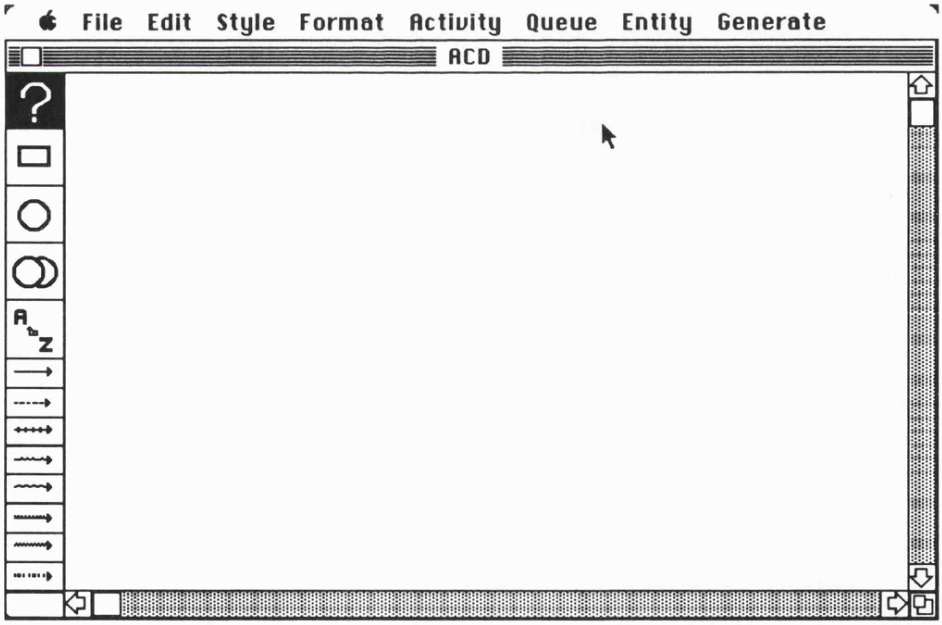
APPENDIX C

AN EXAMPLE WALKTHROUGH WITH MACACD

This appendix shows how to use the application MacACD with an illustrative example of the Pub model (see section 1.2.1). Section C.1, C.2 and C.3 shows how an entity, an activity and a queue can be created respectively. The mechanism for drawing the life paths of an entity type is given in section C.4. Section C.5 shows how to edit objects on screen. The generation of a data file that can be fed into AutoSim is discussed in section C.6. Section C.7 is a menu reference for the MacACD application.

MacACD consists of 9 pull-down menus and 13 iconic menus. Figure C.1 shows the interface of MacACD. The pull-down menu is across the top of the screen and the iconic menu is on the left hand side of the screen. The pull-down menu handles all the file management and data recording procedures whilst the iconic menu mainly deals with the creation of graphics on the screen.

Figure C.1. The MacACD Application

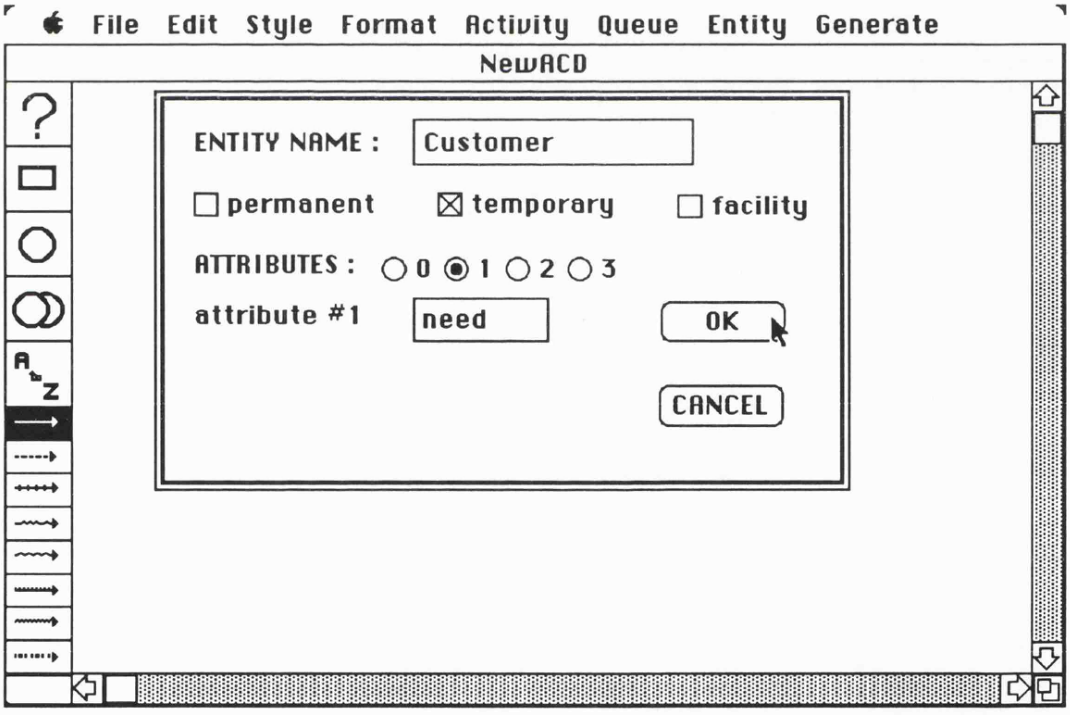


C.1. Creating Entities

The first step in creating an ACD in MacACD is to assign the arrow mode boxes in the iconic menu to different entity types in the system. There are 8 arrow mode boxes altogether, each of which can be assigned to an entity type.

An entity type can be created by firstly clicking in one of the arrow mode boxes in the iconic menu and then entering the information in the entity dialogue box. Figure C.2 shows the dialogue box when a new entity type CUSTOMER is created in the Pub example :

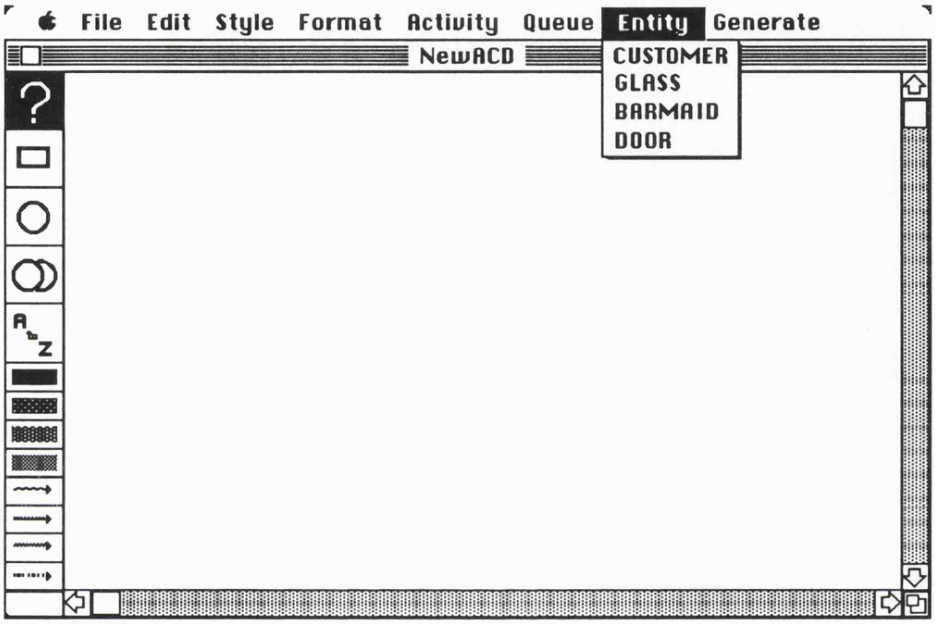
Figure C.2. Creating an Entity



The 'OK' button in the dialogue box fixes the entity to the arrow mode box of entry, and so the arrow mode box is filled with the line pattern that belongs to the entity. The 'CANCEL' button cancels the entry. This pattern is used for drawing the life cycle of that entity type.

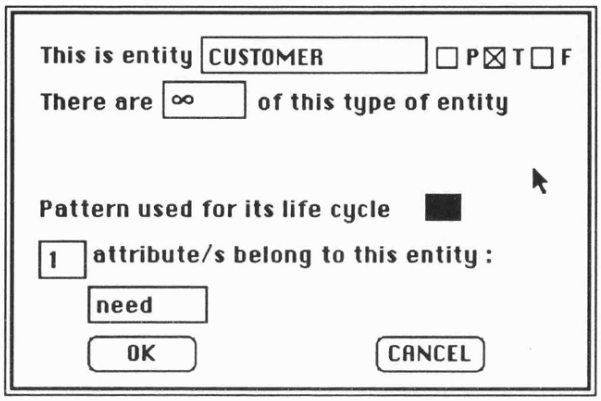
Any newly created entity is appended to the Entity menu. The following diagram shows the appearance of the Entity Menu after all the entities have been created in the Pub example :

Figure C.3. The Entity Menu



Information about an entity type can be viewed and edited by selecting its name from the Entity menu. When an entity is selected, the dialogue box of the entity type will appear, allowing the information to be edited. Figure C.4 shows the dialogue box for the entity type CUSTOMER when Customer is selected from the Entity menu :

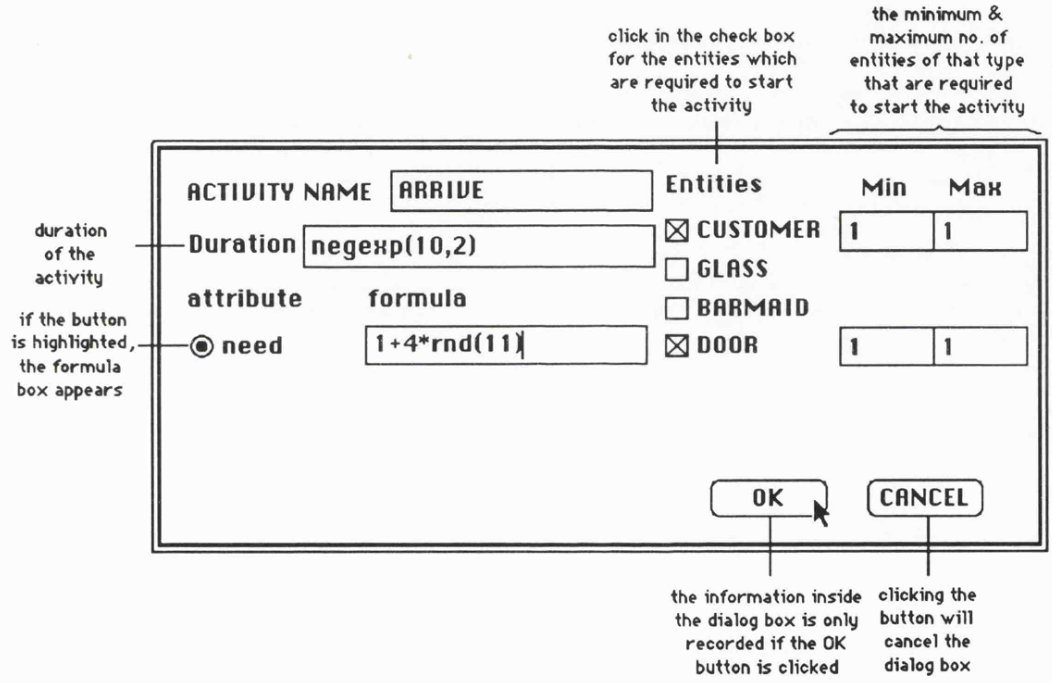
Figure C.4. The Entity Dialogue Box



C.2. Creating Activities

An activity can be created by using the rectangle mode box in the iconic menu. After a rectangle is drawn, the current iconic menu will return to the Find icon (?). The information about an activity can then be entered by double-clicking on the rectangle. Figure C.5 shows the activity dialogue box.

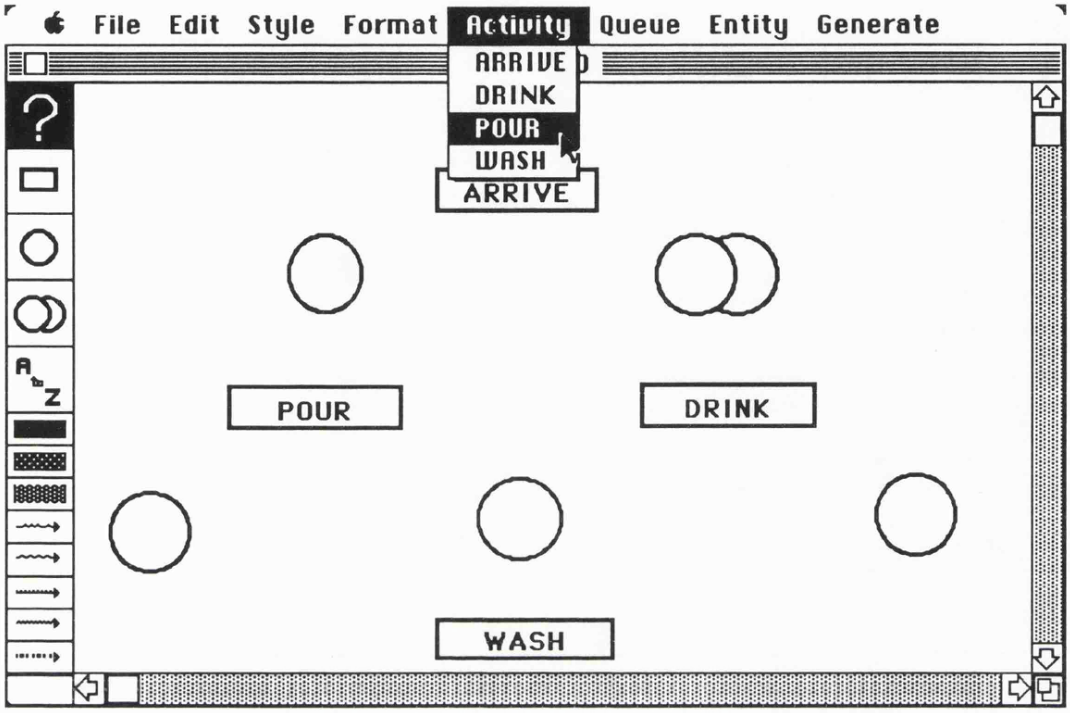
Figure C.5. Entering an Activity



Suppose all the entity types in the Pub example have already been created. The user can click at its associated check box to select the entities that are involved in this activity. The name, the activity duration and any attribute assignment can be entered.

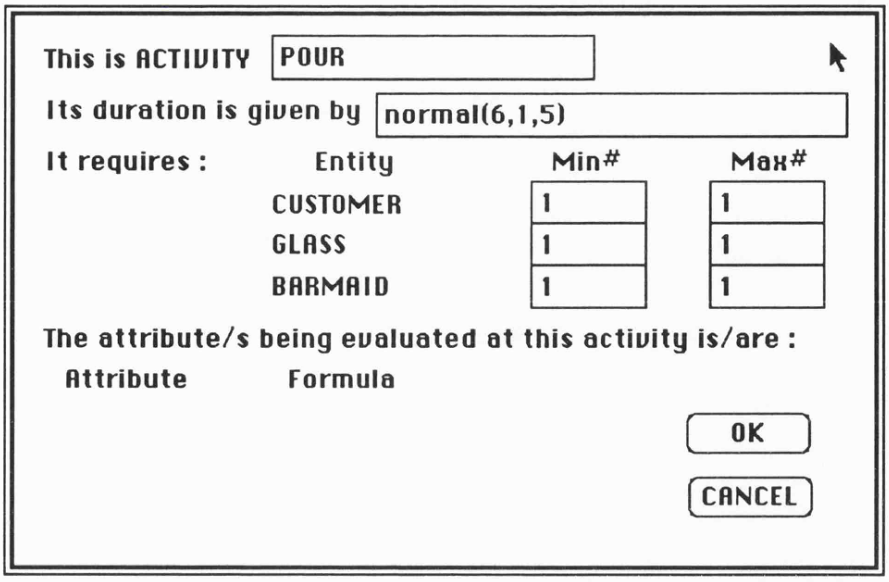
An activity will be added to the system only if the 'OK' button is clicked. The activity will not be added to the system if the user clicks the 'CANCEL' button. If the 'OK' button is clicked, the information for the activity is stored and the name of the activity will appear inside its associated rectangle on the underlying screen. Any newly created activity will be appended to the Activity menu. Figure C.6 shows the Activity menu when all the activities has been created in the pub example.

Figure C.6. The Activity Menu



Activity information can be viewed and edited by selecting its name from the Activity menu. Figure C.7 shows the dialogue box when the activity POUR is selected from the Activity menu :

Figure C.7. The Activity Dialogue Box



C.3. Creating Queues

A queue can be created by using the circle mode box. After a circle is drawn, the iconic menu will return to the Find icon (?). Queue information can be edited by double-clicking on the circle. Similarly, a source/sink queue can be created by using the overlapping circles mode box. Figures C.8 and C.9 shows the queue dialogue boxes when the queue WAIT and the source/sink queue OUTSIDE are created respectively.

Figure C.8. Creating a Queue

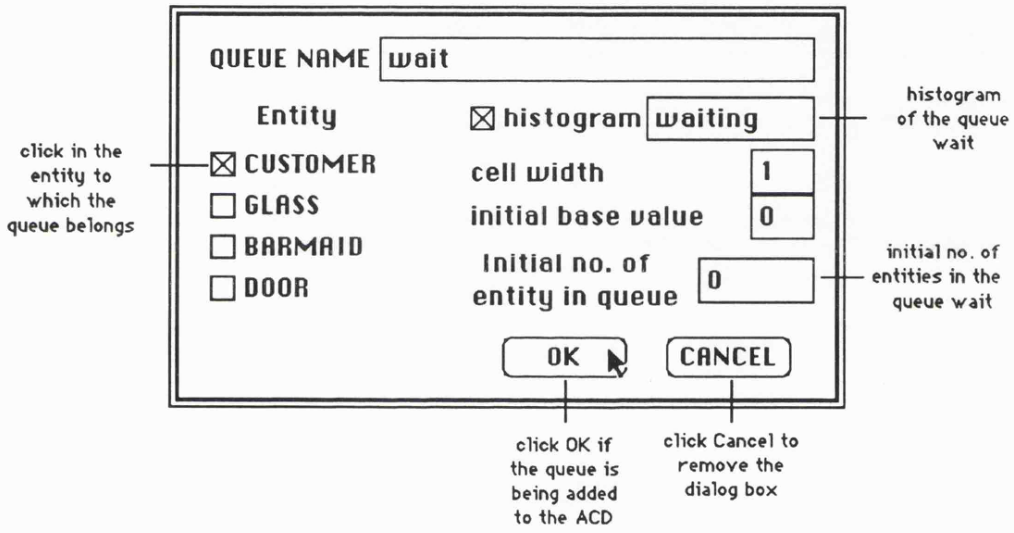
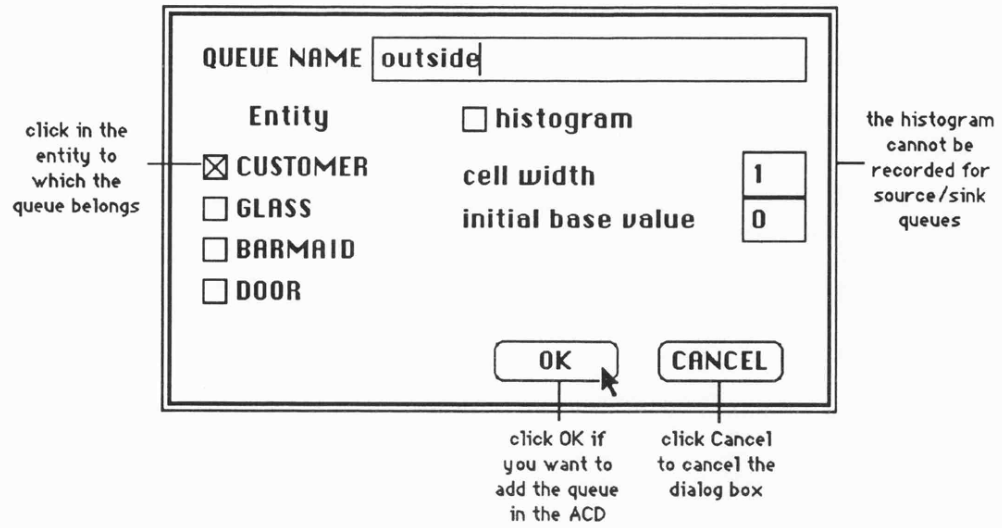
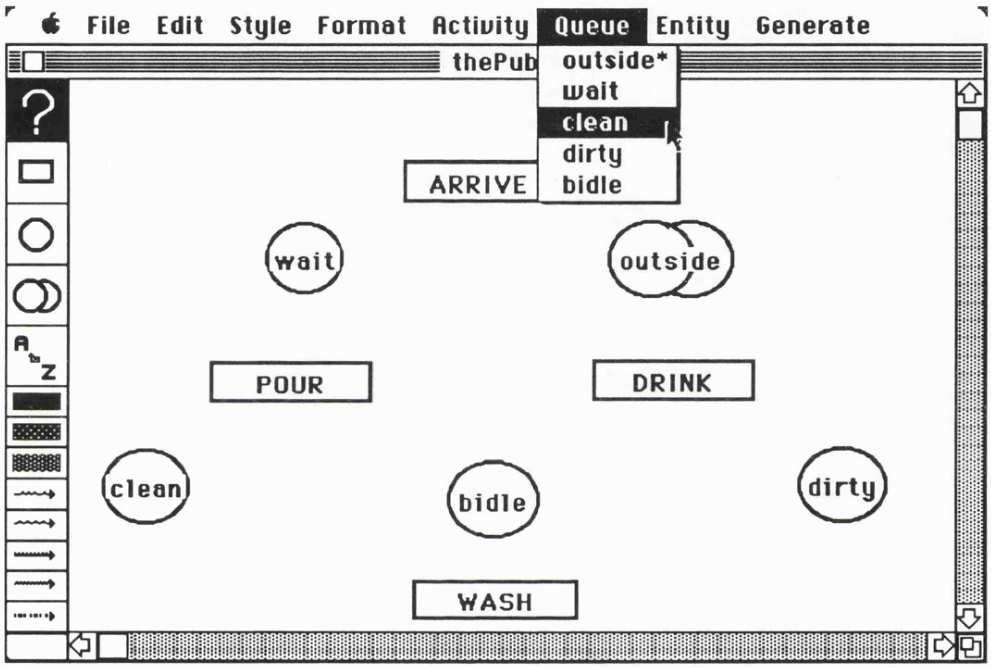


Figure C.9. Creating a Source/Sink Queue



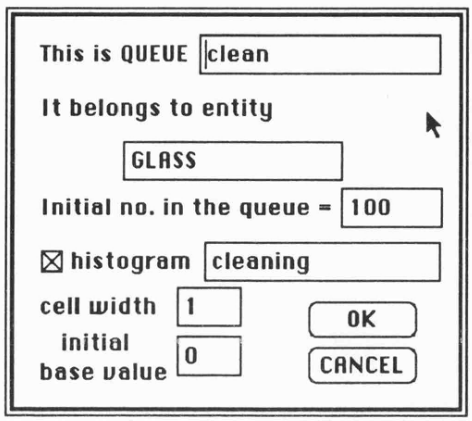
If the 'OK' button is clicked, the queue is added to the Queue menu. If it is a source /sink queue, there will be an asterisk '*' besides the name of the queue. Any newly created queue is appended to the Queue menu. Figure C.10 shows the Queue menu when all the queues are created for the Pub example.

Figure C.10. The Queue Menu



Queue information can be viewed and edited by selecting it from the menu. Fig.C.11 shows the queue dialogue box for the queue CLEAN.

Figure C.11. The Queue Dialogue Box



C.4. Drawing Life Paths for an Entity Type

The life paths for an entity type can be drawn by using the corresponding line pattern mode box in the iconic menu (i.e. the assigned arrow mode box). To draw the life cycles of the temporary and permanent entities, choose a starting queue, then hold the mouse button down and move the cursor from the starting queue to the desired activity. A line connecting the pair of shapes on the screen will then appear with an arrow indicating the direction of movement of the mouse. Figure C.12 shows how to create a path between a queue and an activity.

Figure C.12 Creating a Path

The linking procedure can then be continued until the starting queue is equal to the final queue where the cycle ends. When two activities are connected, MacACD will create a dummy queue between the two activities. When the same activity is connected to more than one queue consecutively, (i.e. the entity can move to either one of the queues after the activity is finished), MacACD will ask for the conditions required to go to the different queues.

MacACD cannot automatically produce a dummy queue for a facility entity - an entity which facilitates several activities, but only has one queue. The user has to create an idle queue for a facility entity. To draw the life cycle of a facility entity, move the cursor from the idle queue to the activity that requires the entity. A pair of arcs will automatically branch out from, and into, the activity so that there is a cycle for the

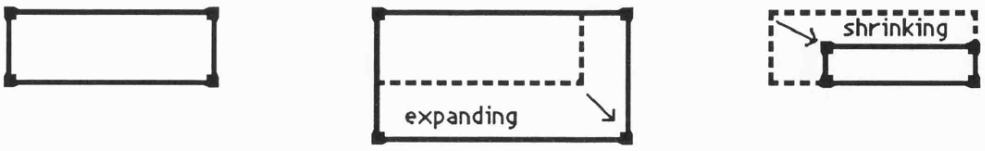
entity. If the entity has more than one activity, the procedure is repeated. Figure C.13 shows how to create a path for a facility entity.

Figure C.13. Creating a Path for Facility Entity

C.5. Editing Objects on the screen

The size and position of the objects on the screen can be varied by means of the select mode box, which takes the shape of a question mark in the iconic menu. When clicking on an object, four small boundary rectangles will appear on the four corners of the object. By clicking inside the object and then dragging the mouse to a new location, the object will move. By clicking on one of the four small boundary rectangles of an object and dragging the mouse to a new location, the object will either shrink or expand, according to the direction of the movement of the mouse. Figure C.14 shows the sizing of a rectangle on the screen.

Figure C.14. Editing an Object on Screen



Apart from changing the size of the object on screen, the user can also invoke an activity or queue information dialogue box by double-clicking at the desired object on the screen.

C.6. Generating a Text file for AutoSim

A text file, which can be fed into AutoSim, can be created by using the *Go* command in the Generate menu. A generated file for the Pub model is shown in figure C.15.

Figure C.15. The Text File Pub.Dat Generated in the Pub Example

Pub.Dat	Pub.Dat (cont.)	Pub.Dat (cont.)	Pub.Dat (cont.)
CUSTOMER	qclean	negexp(10,2)	waiting
""""""""""""	aPOUR	normal(6,1,5)	1
GLASS	qbidle	5+4*rnd(7)	0
100	aDRINK	5	y
BARMAID	qdirty	ARRIVE	clean
5	aWASH	DRINK	cleaning
""""""""""""	qclean	""""""""""""	1
qoutside	y	1+4*rnd(1)	0
aARRIVE	qbidle	need-1	y
qwait	aWASH	n	bidle
aPOUR	qbidle	y	barhist
qbidle	aPOUR	need	1
aDRINK	qbidle	ARRIVE	0
qwait	y	needhist	n
cneed>0	n	1	n
qoutside	n	0	""""""""""""
""""""""""""	n	n	100
outside	n	y	5
y	n	wait	0

C.7. The Pull-down Menus in MacACD

There are altogether nine pull-down menus in MacACD.

C.7.1. The Apple Menu

The Apple Menu contains desk accessories that are present in the Macintosh system file.

C.7.2. The File Menu

The File menu consists of the following commands - *New* is used for creating a a new model; *Open* opens any existing MacACD file which has been saved on the disk; *Close* closes the current active window; *Save* saves the activity cycle diagram in the current window and all the user-defined information to a file; *Save As* allows the user to save the current file under another name; *PageSetup* allows the user to select the type setting of the paper; *Print* prints the ACD;*Quit* quits the application and returns the user to the Finder.

C.7.3. The Edit Menu

The commands in the Edit menu can be used to assist in the drawing of objects on the screen. *Undo* retrieves the picture just before the last change was made. *Cut* deletes a selected object and copies it to the Clipboard file. *Copy* copies the selected object to the Clipboard file. *Paste* copies the object from the Clipboard file onto the window.

C.7.4. The Style Menu

Text can be added to the screen by using the 'A to Z' mode box in the icon menu. The text can be modified by using the Style menu. *Point* allows the user to choose the desire point size (9, 10, 12, 14) for the selected text. *Plain* writes the selected text in plain style. *Bold* writes the selected text in **bold** style. *Italic* writes the selected text in *italic* style. *Underline* underlines the selected text. *Outline* outlines the selected text. *Shadow* shadows the selected text.

C.7.5. The Format Menu

The Format menu consists of the following commands - *Reduce* reduces the diagram to fit the screen; *Normal* returns the normal size of the diagram; *AutoDraw* allows the user to draw a rectangle of predefined size automatically; *Manual* allows the user to draw a rectangle of any size freely; *Chopper* is used for deleting the life cycle of an entity.

C.7.6. The Activity Menu

Whenever an activity is added to the model, the activity name will be appended to this menu. The user can then select any activity from the menu and look at the summary information of the selected activity in the Activity dialogue box (figure C.7).

C.7.7. The Queue Menu

Whenever a queue is added to the model, the activity name will be appended to this menu. The user can then select any queue from the menu and look at the summary information of the selected queue in the Queue dialogue box (figure C.11).

C.7.8. The Entity Menu

Whenever an entity is added to the model, the entity name will be appended to this menu. The user can then select any entity from the menu and look at the summary information of the selected entity in the Entity dialogue box (figure C.4).

C.7.9. The Generate Menu

Go is used to generate a text file from the ACD. The system will ask for the name of the file in which the data is to be saved. The default name of the text file is <filename.DAT>. *ShowTFile* allows the generated code to be shown in another window.

APPENDIX D

THE USE OF DIAGRAMMING TECHNIQUES

This appendix shows a summary table of the areas in which some of the current diagramming techniques are applied.

	DATA	ACTIVITIES	
Entity-relationship diagram	**Strategic overview of corporate data**	**Strategic overview of corporate functions**	*Decomposition diagram* *Action diagram* *Warnier-Orr diagram*
Data structure diagram	**Detailed logical data model**	**Logical relationship among processes**	*Decomposition diagram* *Action diagram* *Warnier-Orr diagram* *Dependency diagram* *Data flow diagram* *HIPO diagram* *HOS chart* *State-transition diagram*
Data structure diagram *Jackson diagram* *Warnier-Orr diagram*	**Program level view of data**	**Overall program structure**	*Action diagram* *Data navigation diagram* *Warnier-Orr diagram* *HIPO diagram* *Structure chart* *HOS chart* *Jackson diagram*
Data structure diagram *Jackson diagram*	**Program usage of data**	**Detailed program logic**	*Action diagram* *Data navigation diagram* *Flowchart* *Nassi-Shneiderman chart* *HOS chart* *Decision tree & table*

APPENDIX E

HOW TO USE HYPERSIM

This appendix is a brief guide to HyperSim, with an illustrative example using the Launderette model (section 1.2.1). Section E.1 shows how a new model can be created. The creation of a new entity type and the selection of entity icons are discussed in sections E.2 and E.3 respectively. Section E.4 shows how to create an activity for the model. The mechanisms used for specifying the life paths of entities is given in section E.5. Section E.6 shows how to create and assign attributes in HyperSim. The generation of a three-phase simulation program is discussed in section E.7.

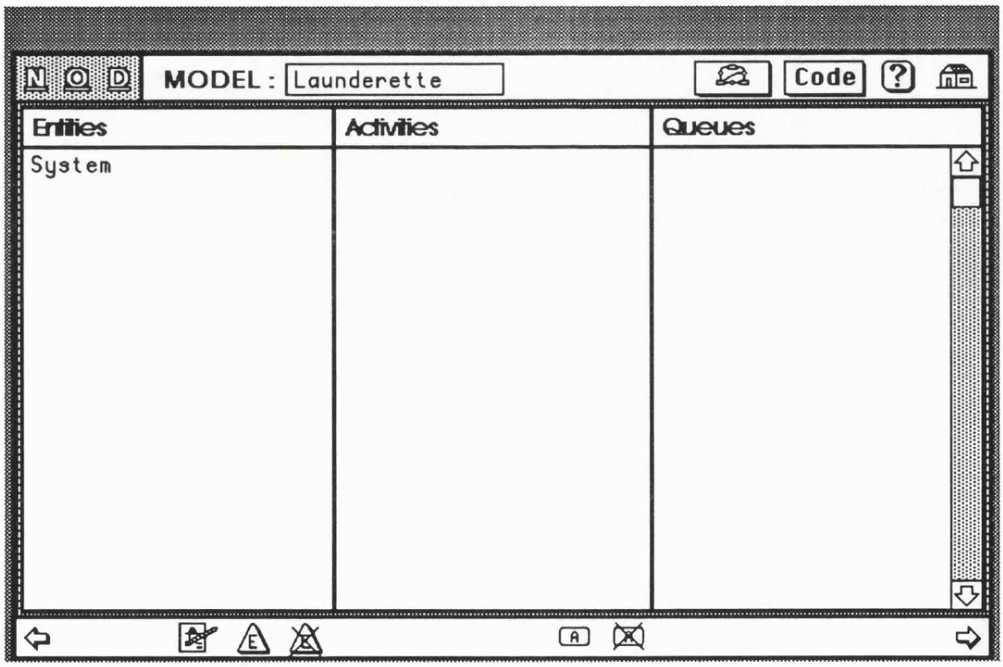
E.1. Creating a New Model

HyperSim is made up of nine stacks. The first stack that the user should go to is the Model stack where a new model can be created or an existing model can be selected. The Model stack can be opened by either double-clicking the icon on the Macintosh desktop or by clicking the 'Start Modelling' button in the Reference stack.

Assume that the Model stack has been opened. A new model can be created by clicking at the 'ℕ' button on the left-hand corner of the card. The system will prompt the user for a model name. If the user types in the model name and clicks 'OK', 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 select a model by clicking at the name of the model in the model list table. The user can also scan through the model cards by using the left and right arrow buttons located at the bottom sides of the card. The '𝔻' button is used for deleting an existing model. The user can select the model that is to be deleted from the model list table. All the cards that belongs to the selected model inside all the HyperSim stacks will then be deleted after further confirmation from the user.

For example, to create a new model called 'Launderette', click the 'ℕ' button, type 'Launderette' and click 'OK'. An ACD card and a system entity card are then automatically generated. The word 'System' will appear in the Entity column of the summary table. Figure E.1 shows the Model card when the Launderette model is created. The user can either create entities by clicking at the 'New Entity' button, create activities by clicking at the 'New Activity' button, create system attributes for the 'System' entity in the Entity stack, or create icons in the Icon stack.

Figure E.1. Creating the Launderette Model in the Model Stack



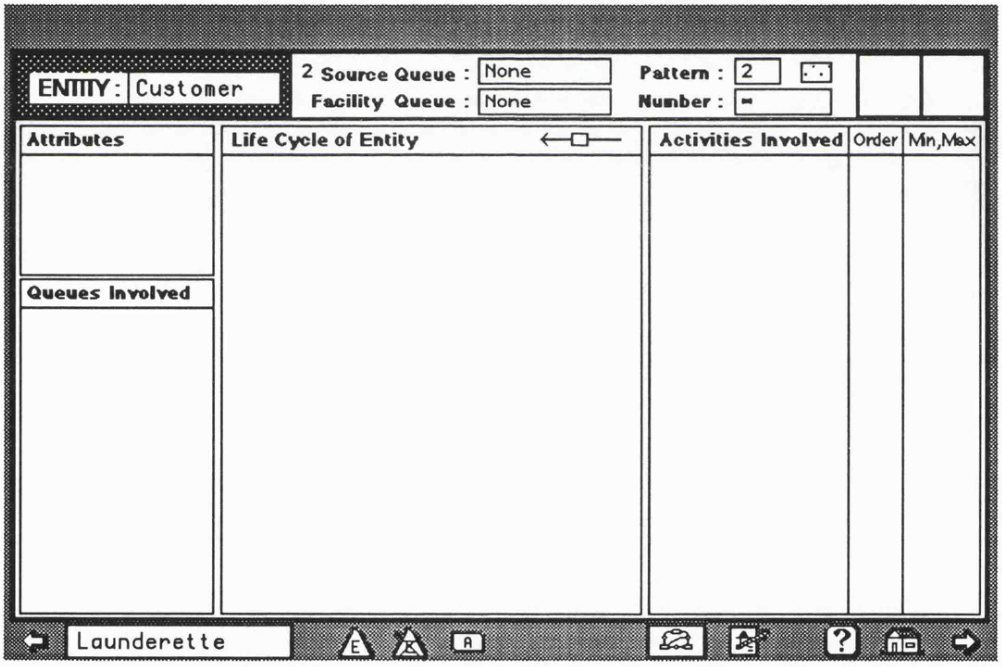
E.2. Creating Entities

An entity type can be created anywhere in the system if there is a 'New Entity' button in the stack where the user is working. To create a new entity type, select the 'New Entity' button, type the name of the new entity type in the dialogue box and click 'OK'. A new card in the Entity Stack will be created where the user can enter any relevant information about the newly defined entity type. In the Entity stack, the user

can either create attributes (see section E.6), select an icon (see section E.3), or create a source/sink or facility queue for an entity type in the Entity stack.

For example, a new entity type 'Customer' can be created by clicking the 'New Entity' button, type in the name 'Customer' and click 'OK'. A new card called 'Customer' is then created in the Entity stack. Figure E.2 shows the Entity card when the entity type 'Customer' is created. The user can look at the entity list table by clicking at the 'Entity' button in an entity card. He can go to the appropriate entity card by clicking at the corresponding entity name in the entity list table.

Figure E.2. Creating an Entity type 'Customer' in the Launderette Model



Let us look at the creation of a source/sink queue for an entity type. Since 'Customer' is a temporary entity in the Launderette model, we can create a source/sink queue for this entity type. Click at the 'Source' field in the entity card of the 'Customer', type 'outside' in the dialogue box and click 'OK'. A new card in the Queue stack called 'outside' will be created. The word 'outside' will appear in the 'Queues Involved' table in the 'Customer' entity card. To go to a queue card from the Entity stack, just click at its corresponding name in the 'Queues Involved' table. Figure

E.3a and E.3b shows the queue card 'outside' and the entity card 'Customer' after a source/sink queue has been created.

Figure E.3a. The Queue card for source/sink queue 'outside' for entity 'Customer'

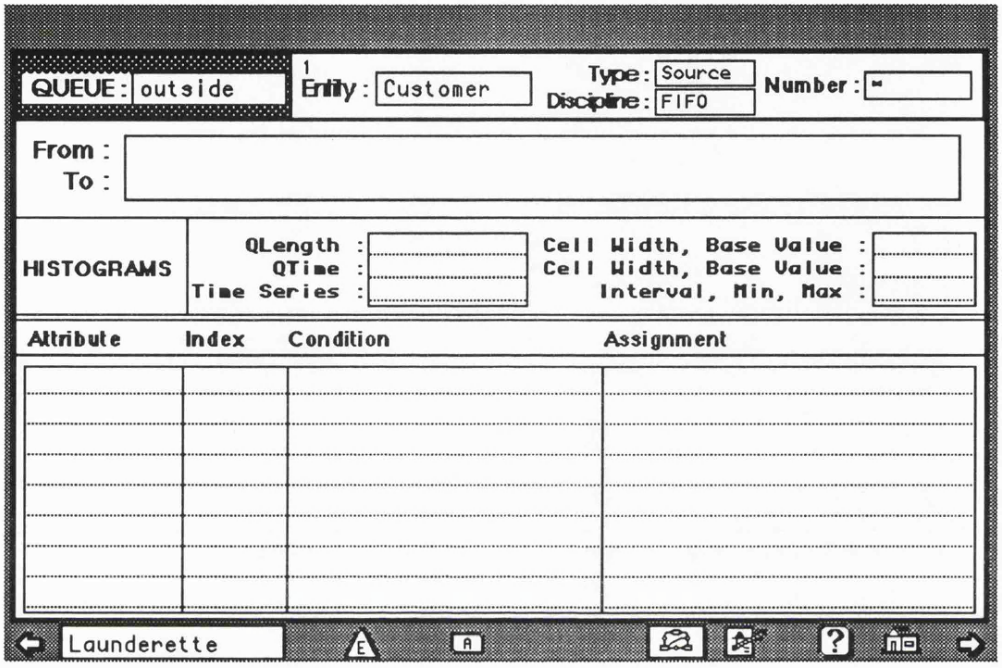
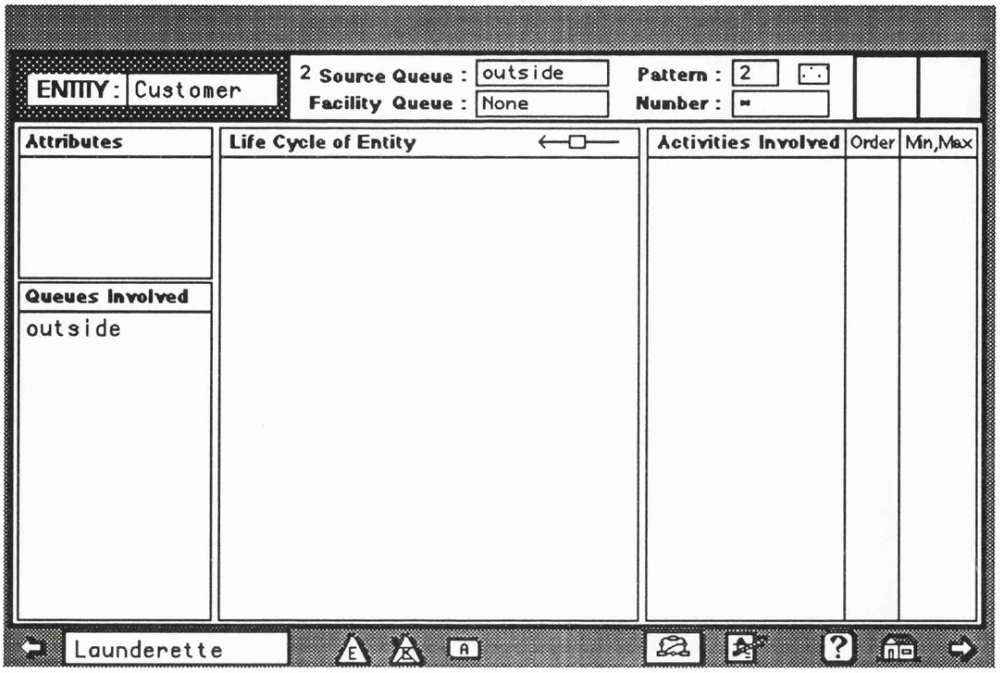


Figure E.3b. The Entity card 'Customer' after a source/sink queue is created



Similarly, a facility queue can be created for a facility entity, for example, the Door, by clicking at the 'Facility queue' field in the entity card of the 'Door'.

E.3. Selecting an Icon for an Entity type

By clicking the 'GotoIcon' button at the bottom of the card, HyperSim will take the user to the Icon stack. The Icon stack is shown in figure E.4a. The user can either create a new icon or edit an existing icon in any one of the 135 rectangles in an icon card. To select an icon, make sure the name of the entity in the 'Entity' field is the one you want the selected icon to be transported to. Click at the 'Select Icon' button and click anywhere inside the rectangle where the selected icon is enclosed by. The user will then be taken back to the corresponding entity card in the Entity stack with the selected icon pasted onto the top-right rectangle in the card. Figure E.4b shows how the entity card for 'Customer' after an icon has been selected from the Icon stack.

Figure E.4a. Selecting an Icon for Entity 'Customer' in the Launderette model

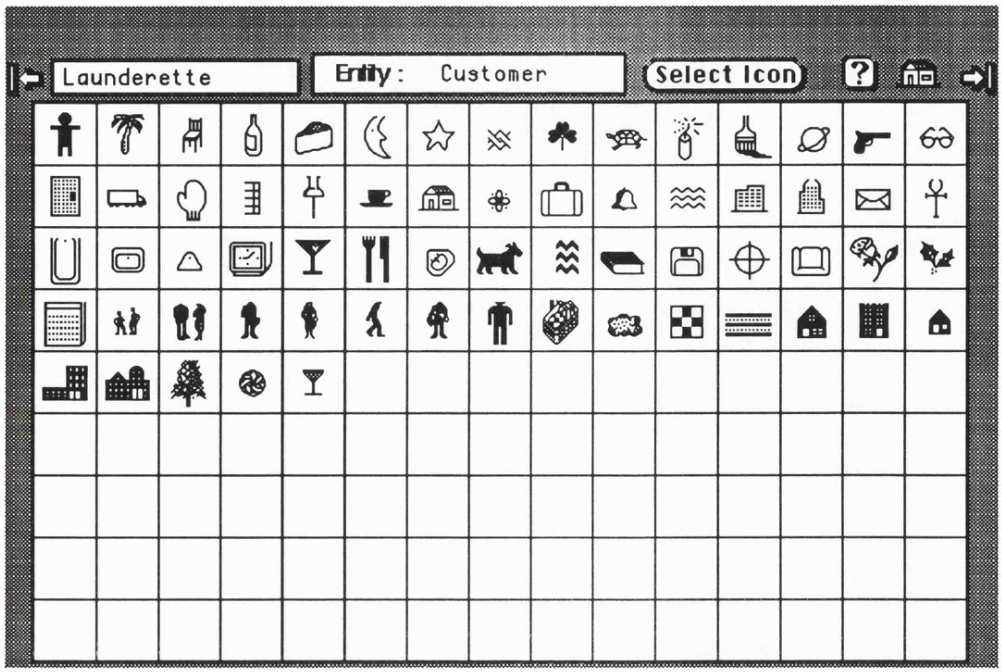
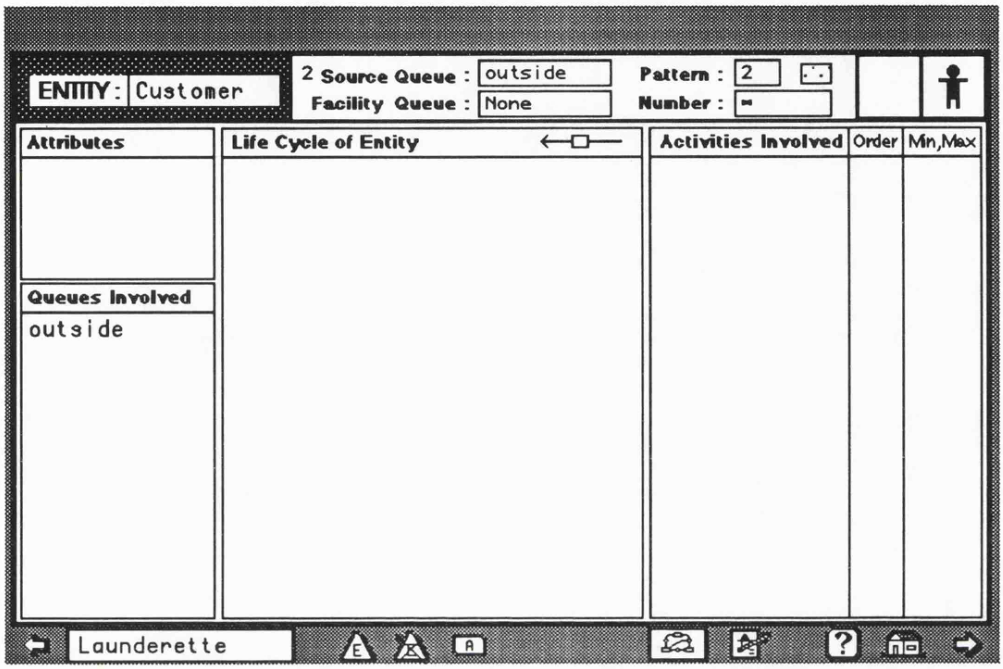


Figure E.4b. The Entity Card 'Customer' after an Icon has been selected



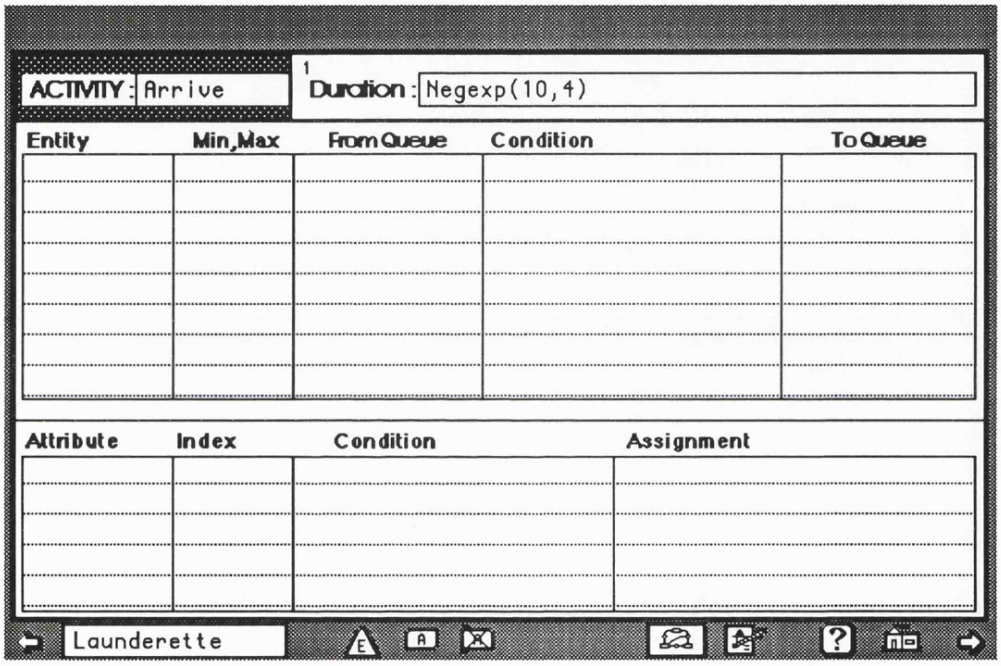
E.4. Creating Activities

An activity can be created anywhere in the system if there is a 'NewActivity' button in the stack in which the user is working with. To create a new activity, click the 'NewActivity' button, type the name of the new activity in the dialogue box and click 'OK'. A new card in the Activity Stack will be created where the user can enter any relevant information about the new activity. 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 complete ACD before entering information for an activity.

For example, activity 'Arrive' can be created by clicking the 'NewActivity' button, type 'Arrive' and click 'OK'. An activity card called 'Arrive' will be created. Figure E.5 shows how to create an activity 'Arrive' for the Launderette model.

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

Figure E.5. Creating activity 'Arrive' in the Launderette model



E.5. 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 of specifying whereas in the second method, the user has to enter the name of the queues manually. The author of HyperSim is currently working on 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 assumed that the following entity types for the Launderette model have been created - System, Customer, Door, Washing Machine, Drier and Basket. We also assume that activities - Arrive, LoadWash, UnLoadWash, Transport, LoadDrier and Dry have been created. However, it should be pointed out that HyperSim allows the user to add new entity types or new activities after part of or all of the entities' life cycles is drawn.

E.5.1. Using the Entity Stack

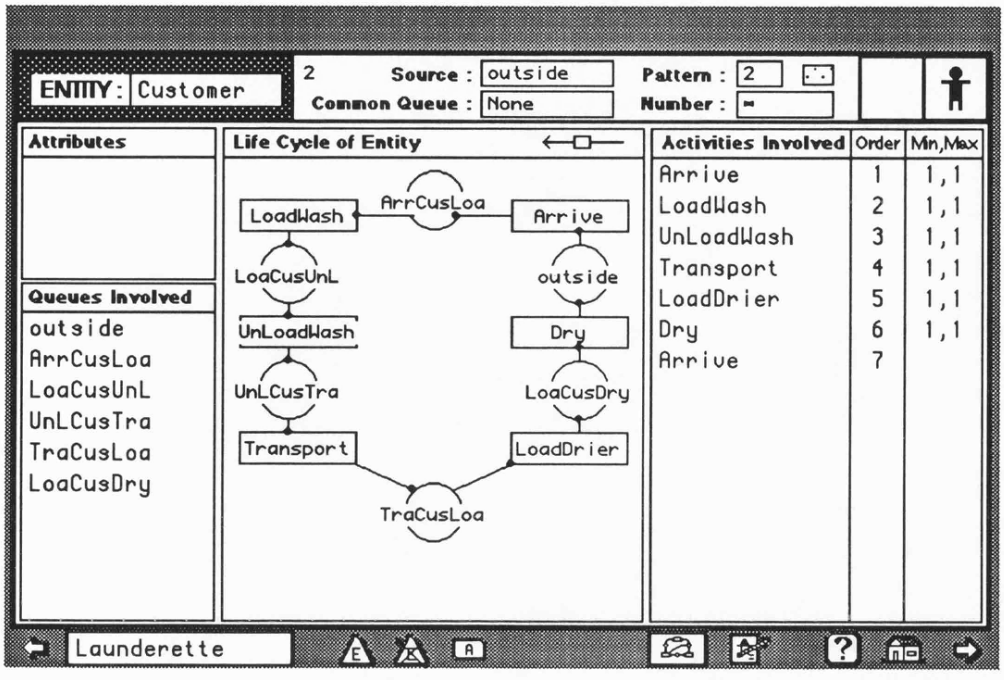
This is the simplest method of all, but is only appropriate for simple simulation models where the life cycle of individual entity types can easily be identified. This method does not allow the user to see an overall view of the model during the construction of the model logic.

For example, let us define the life cycle of entity type 'Customer'. From the activity cycle diagram of the Launderette (see figure 1.5) the order of activities for a customer is - 'Arrive', 'LoadWash', 'UnLoadWash', 'Transport', 'LoadDrier' and 'Dry'. To enter this cycle is simple. 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 in 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 'LoadWash' can be added to the 'Activities Involved' table. When activity 'LoadWash' is added, a queue 'ArrCusLoa' connecting 'Arrive' and 'LoadWash' is automatically created for 'Customer'. (The default name of the

queue is set by combining the first three characters of the first activity name, the first three characters of the entity name, and the first three characters of the second activity name.) This queue is added to the 'Queues Involved' table in the entity card of the customer. Finally when activity 'UnLoadWash' is added, a queue 'LoaCusUnl' connects 'LoadWash' and 'UnLoadWash'. Similarly, add the rest of the activities in the life cycle. To indicate that the cycle is completed, select 'Arrive' again.

On completion of the life cycle of an entity, click at the 'Life Cycle of Entity' button. A life cycle diagram of the 'Customer' is drawn. A conditional path for an entity type can be specified by selecting the 'Conditional Arrow' button (an arrow with a box in the centre), click anywhere inside the activity rectangle which the conditional path starts from and then click anywhere inside the queue where the path goes to. The user will then be asked to enter the condition for the entity to go from the original activity 'Drink' to the destination queue. After the user types in the condition and clicks 'OK', the conditional path will be drawn in the life cycle diagram for the entity. Figure E.6 shows the completed entity card for entity type 'Customer'.

Figure E.6. The Entity card '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 are involved in and when finished, click the 'Life Cycle of Entity' button. However, the user has to define a facility queue before the life cycle is defined.

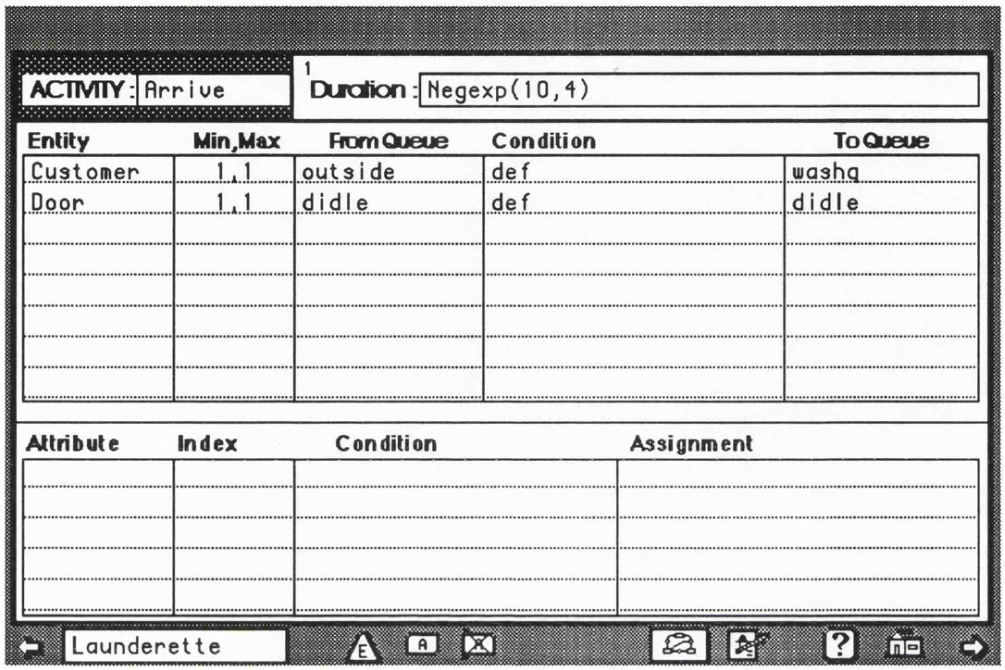
Information of this specification of life cycles of entities is updated in the 'Entities Involved' table in the Activity stack. The user can re-specify conditions in the 'Condition' field of a path in the corresponding row.

E.5.2. 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, let us define the activity 'Arrive' which takes place when there is a customer in the source/sink queue 'outside' and a door in the queue 'didle'. By clicking at the word 'Entities Involved', the user can select the entity type 'Customer' from the entity list table. He will then be asked to 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'. In the 'queue in and queue out dialogue box', 'outside' appears as the queue in (since 'Arrive' is the first activity that a customer is involved in) and 'newqueue' appears as the queue out. The user must type in a name to replace 'newqueue'. If the user types 'wait', then a queue card called 'wait' will be generated. The user can continue to select the facility entity type 'Door' for activity 'Arrive'. There is no need to specify queue in and queue out for 'Door' since both queues are assigned to 'didle' by default. Figure E.7 shows the completed activity card for activity 'Arrive'.

Figure E.7. The Activity card 'Arrive'



In order to help the user keep track of the queues associated with an activity for an entity type, the default queue in for an activity is set to be the queue out from the previous activity which the entity is involved in. For example, if activity 'LoadWash' is defined next and entity 'Customer' is selected, then the default queue in would be 'washq'.

E.5.3. Using the ACD Stack

This method allows the user to build up the model logic with an overall view. 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). Selecting an object in the drawing area can be done by clicking anywhere inside the object. The ACD stack contains an iconic menu which aids the user in drawing an ACD.

Displaying Activities

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. For example, to move activity 'UnLoadDry' further to the right can be done by first clicking the vertical 'Activity' button, select the activity 'UnLoadDry' and finally by clicking at the final position where you want the activity to be placed.

Entering Information for Activities and Queues

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 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 selecting the desired queue.

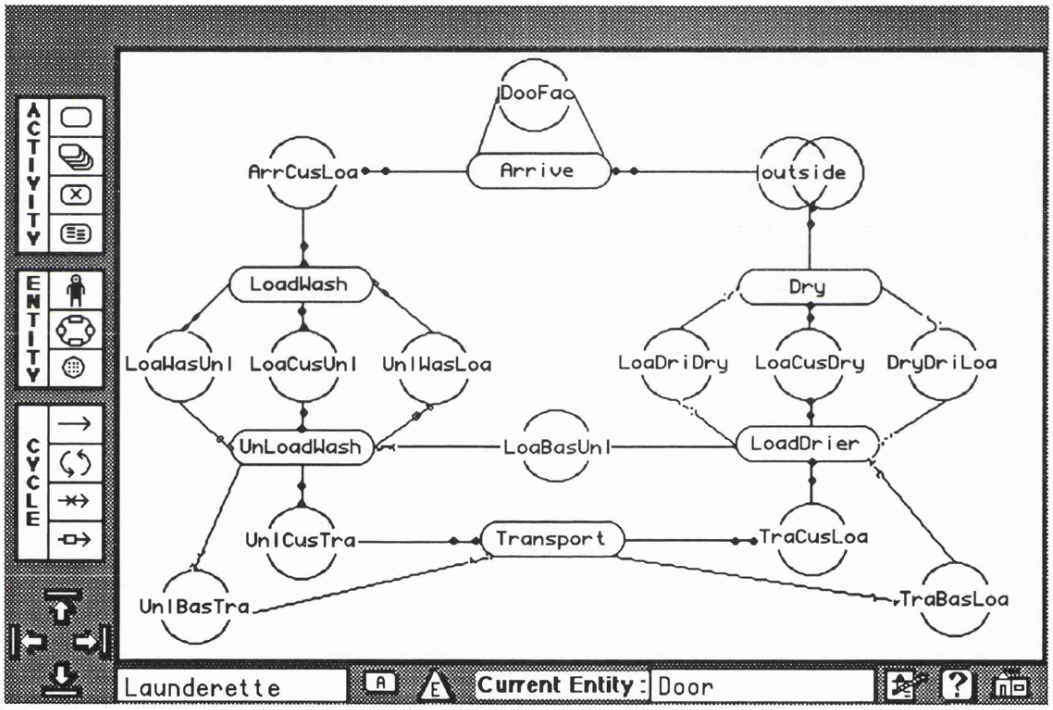
Drawing Life Paths for an Entity type

Before drawing life paths for an entity, an entity type must be selected by first clicking at the 'Select Entity' button and then selecting the appropriate entity type from the entity list table.

Drawing a life cycle for a facility entity is simple. For example, to define the life cycle for the entity type 'Door', select the 'Facility Path' button. First click anywhere in the drawing area where you want the facility queue for 'Door' - 'didle' to be placed. The queue 'didle' will then be drawn automatically. If however the user has forgotten

to define a facility queue for 'Door' and has selected the 'Facility Path' button, he will be prompted for the name of the facility queue. If he types 'didle' and clicks 'OK', a queue card called 'didle' will be created. The user can then select activity 'Arrive'. Two paths - one going from 'didle' to 'Arrive' and one from 'Arrive' to 'didle' are then drawn. If there is more than one activity involved for the facility entity, just select the 'Facility Path' button again and click at another activity in which the entity is involved in. The paths will be automatically drawn.

Figure E.8. The ACD card for the Launderette Model



Drawing life cycles for temporary and permanent entities is the same except that the source/sink queue for a temporary entity takes the shape of an overlapping circle. For example, to draw the life cycle of the entity type 'Customer', first select the 'Path' button and click anywhere in the drawing area where you want the queue 'outside' to be placed. The queue will then be automatically drawn. The user can then select activity 'Arrive'. A path is drawn from 'outside' to 'Arrive. The life cycle can be drawn continually by selecting activity 'LoadWash'. A queue 'AriCusLoa' which links 'Arrive' and 'LoadWash' for 'Customer' is automatically created. Similarly, the queue

'LoaCusUnl' is created when 'UnLoadWash' is selected next. Similarly, the rest of the activities can be selected by the same manner. Finally, select activity 'Arrive' so as to complete the cycle. The cycle does not have to be completed all in one go. The user can draw paths at different times during the construction of the model logic. Conditional paths can be added by selecting the 'Conditional Path' button which works in the same way as described in the first method of defining the model logic by using the Entity stack. Figure E.8 shows the complete activity cycle diagram for the Launderette model.

E.6. Creating and Assigning Attributes

An attribute can only be created inside the Entity stack. To create a new attribute, click the word 'Attribute' at the top of the attribute table in an entity card, type in the name and click 'OK'. An attribute card will be created in the Attribute stack. Figure E.9a and E.9b shows the attribute card 'cusatt' for the entity type 'Customer' and the entity card 'Customer' after the attribute has been specified respectively.

Figure E.9a. The Attribute Card 'Cusatt' for entity 'Customer'

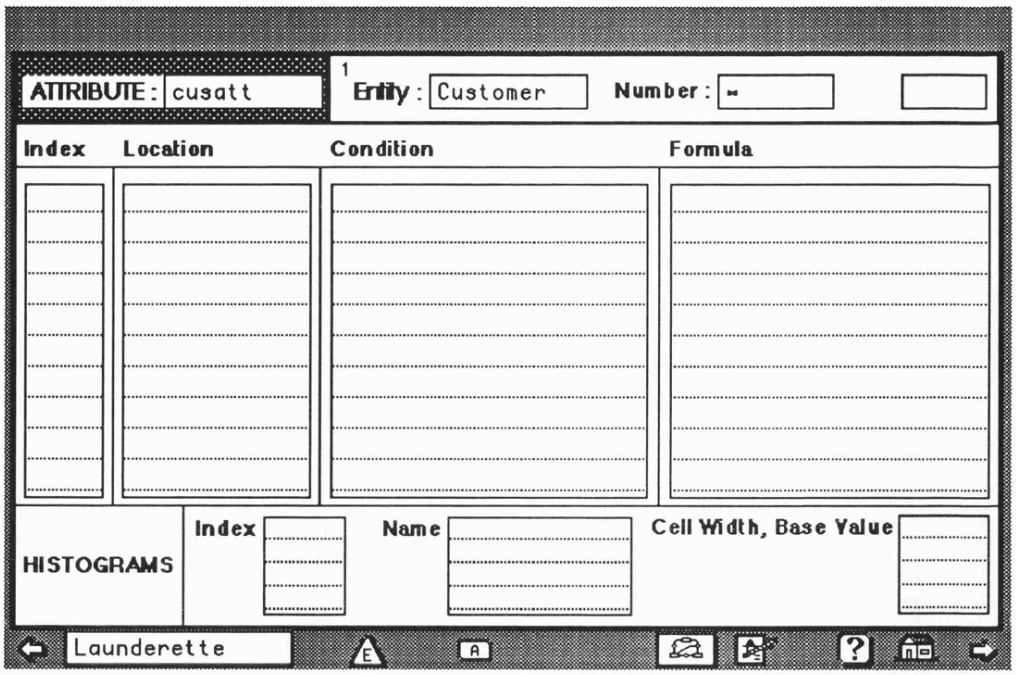


Figure E.9b. The Entity Card 'Customer' after an attribute has been created

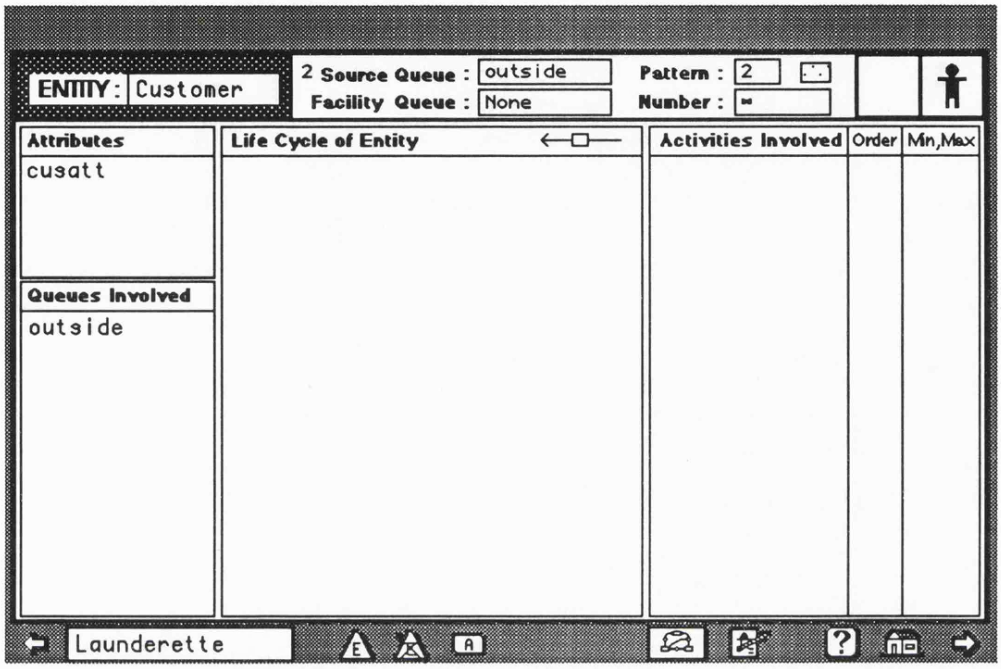
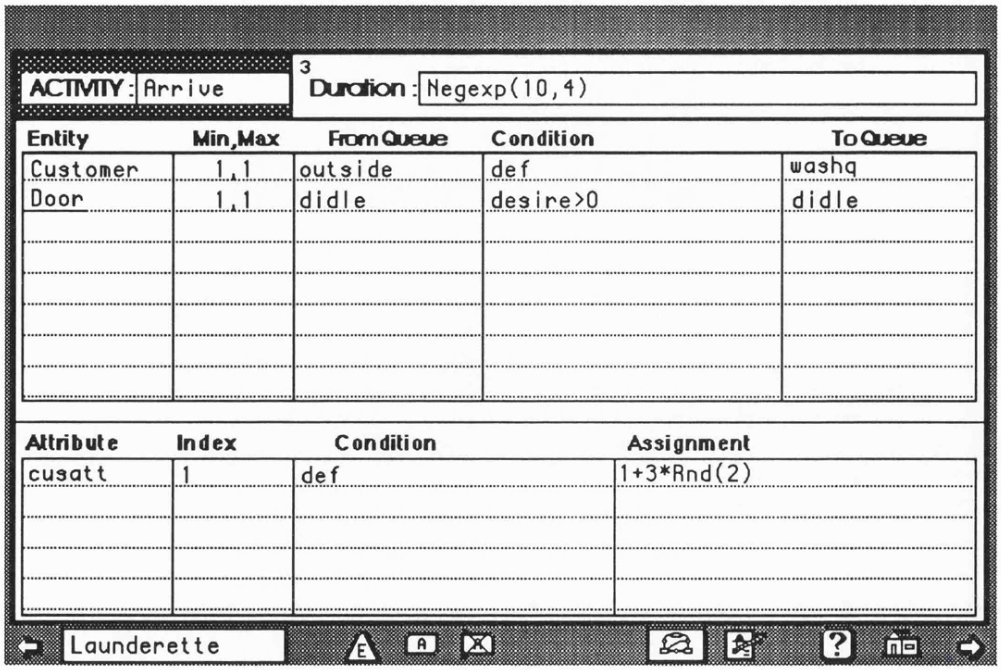


Figure E.10. Assignment of Attribute 'cusatt' in Activity 'Arrive'



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 be prompted for the value of the argument of the attribute. After typing in the value and click 'OK', HyperSim will set the condition of the assignment to 'def', i.e. default, and the user can type in the formula of the assignment.

For example, if the attribute 'cusatt' is evaluated in activity 'Arrive', click once at the word 'Attribute', enter '1' for the argument value and click 'OK'. Type in '1+4*Rnd(2)' in the assignment field. Figure E.10 shows how to assign attribute 'desire' in activity 'Arrive'. Similarly, attributes can be assigned in the same way in a queue.

E.7. Generating a Simulation Program

After the specification is completed, the user can return to the model card 'Launderette' and select the 'Code' button. The system will take the user to the Code stack where a simulation program can be generated. For example, a program for the Launderette model can be generated by clicking the 'Generate' button on the card. Lines of code will appear in the scrolling field on the card. Figure E.11 shows the appearance of the Launderette code card in the Code stack. Modifications of code can be done by editing the program text in the card. The program can also be exported as a text file by using the 'Export' button. The user can go directly into Turbo Pascal by clicking at the 'Turbo' button. However, he should make sure that there is a copy of Turbo Pascal available on the hard disk of the machine.

To run the simulation program, a copy of the Turbo Pascal application and the simulation library -MacSim.Lib should be available on the Macintosh.

Figure E.11. The Code Stack of the Launderette Model


```
Program Launderette;
{$U MacSim.Lib}
Uses
   {$S MGlob}  MemTypes,Quickdraw,OSIntf,ToolIntf,PackIntf,MSimGlobal,
   {$S MSamp}  MSimSample,
   {$S MMode}  MSimModel,
   {$S MOutp}  MSimOutput;

Var
  System,Customer,WashMac,Basket,Drier,Door : entity;
  Arrive,LoadWash,UnLoadWash,Transport,LoadDrier,Dry : activity;
  ArrCusLoa,LoaCusUnL,UnLCusTra,TraCusLoa,LoaCusDry,DooFac : queue;
  outside : source;

Procedure BuildModel;
begin
  MakeEnt(SystemEnt,'SystemEnt');
  MakeEnt(Customer,'Customer');
  MakeEnt(WashMac,'WashMac');
  MakeEnt(Basket,'Basket');
  MakeEnt(Drier,'Drier');
  MakeEnt(Door,'Door');
  MakeAct(Arrive,'Arrive');
  MakeAct(LoadWash,'LoadWash');
```

APPENDIX F

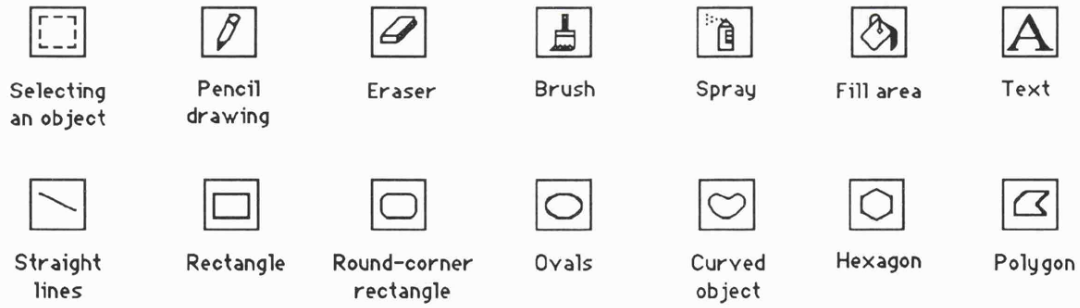
USING MACGRASE

This appendix shows how to use the Macintosh application MacGraSE with an illustrative example using the steelworks model. Section F.1 shows how a background picture can be drawn in the application. The procedures in creating an entity type, an activity, and an attribute are given in sections F.2, F.3, and F.4 respectively. Section F.5 shows how to specify information for a generated queue. The generation of an activity cycle diagram and a simulation program is discussed in sections F.6 and F.7 respectively. Section F.8 gives a brief description in how to run the simulation model.

F.1. Drawing a Background Picture

The drawing of a background picture is managed by using the palette buttons inside the tool box window. Figure F.1 summarises the functions of the buttons.

Figure F.1. The Palette Buttons in the Tool Box Window



For example, to draw a rectangle on the screen to indicate the production area of the blast furnace, we first set the condition of the drawing pen. This can be done by selecting the line size in the line size dialogue box (figure F.2), then selecting the pen pattern in the line pattern dialogue box (figure F.3), and finally selecting the fill pattern in the fill pattern dialogue box (figure F.4). To draw a rectangle on the screen, select the 'Rectangle' button in the tool box window. Click at the desired position of the top-left corner of the rectangle, then hold the mouse button down and drag to the desired position of the bottom-right corner of the rectangle. Release the mouse button. A

rectangle then appears on screen using the current drawing pen. Figure F.5 shows this drawing process. Other objects can be drawn in a similar manner. A complete background diagram for the steelworks model is shown in figure F.6.

Figure F.2. The Line Size Dialogue Box

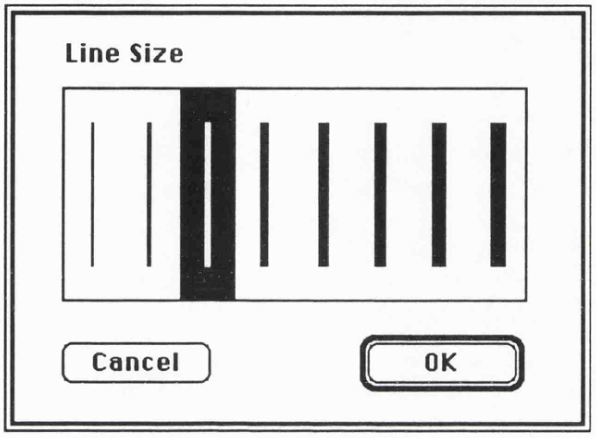


Figure F.3. The Line Pattern Dialogue Box

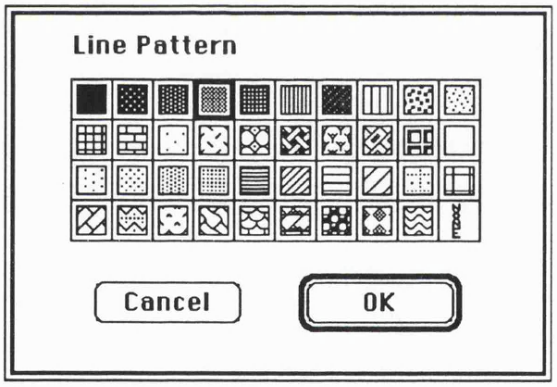


Figure F.4. The Fill Pattern Dialogue Box

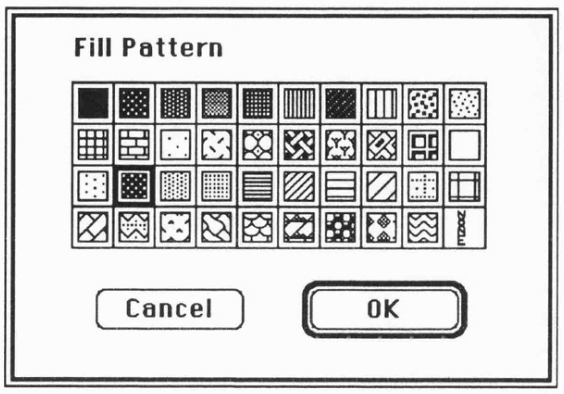


Figure F.5. Drawing a Rectangle on the screen

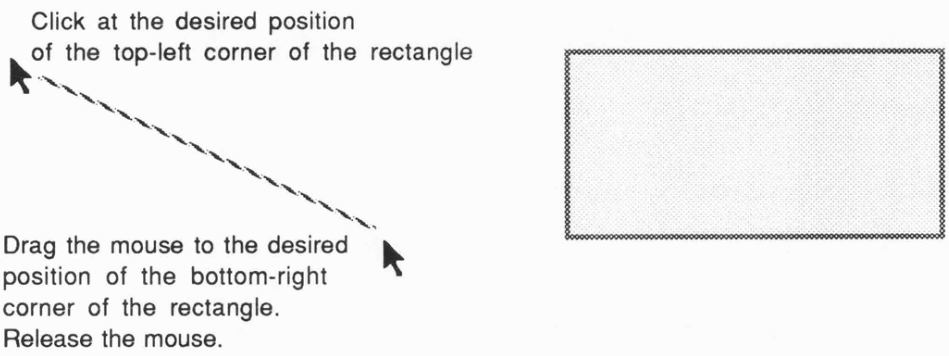
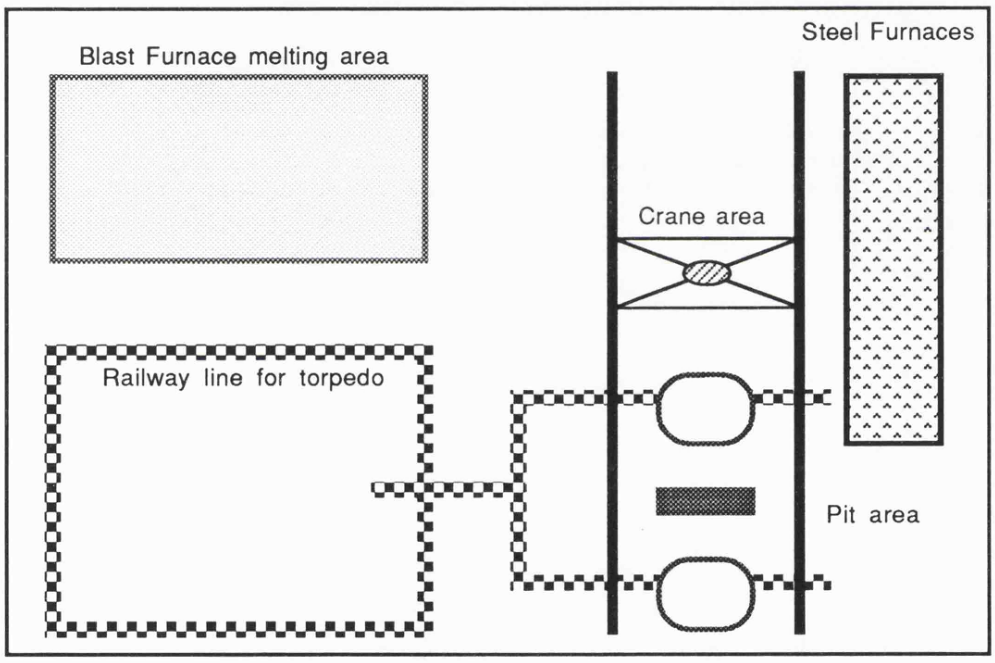


Figure F.6. A Background Diagram of the Steelworks Model



F.2. Creating an Entity Type

To create a new entity type, select the New Entity command from the Model menu. The entity dialogue box is shown in figure F.7. For example, to create the entity type blast furnace in the steelworks model, enter 'BLASTF' in the name field and '2' in the number field. Since blast furnace is a permanent entity in the model, the radio

button 'permanent' should be selected. Once the 'OK' button is clicked, the data for the entity type 'BLASTF' is added to the model. The name 'BLASTF' is appended to the Entity submenu in the Model menu. Figure F.8 shows the appearance of the Entity submenu after all the steelworks entity types have been added to the model. The information for each entity type can be reviewed and edited by selecting the appropriate entity in the Entity submenu.

Figure F.7. Creating the Entity Type 'BLASTF'

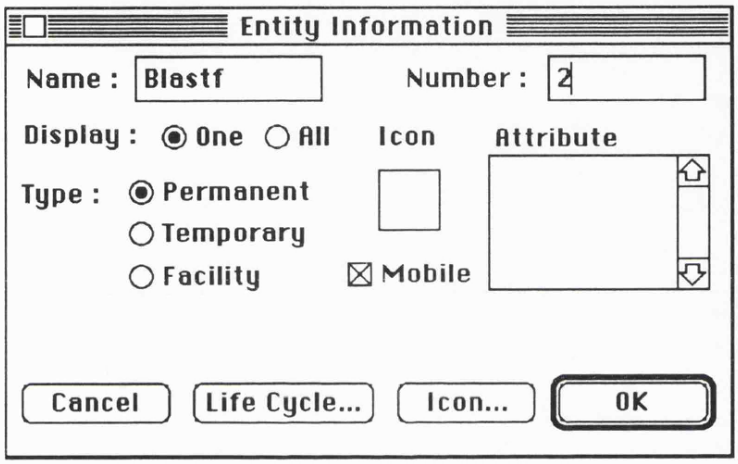
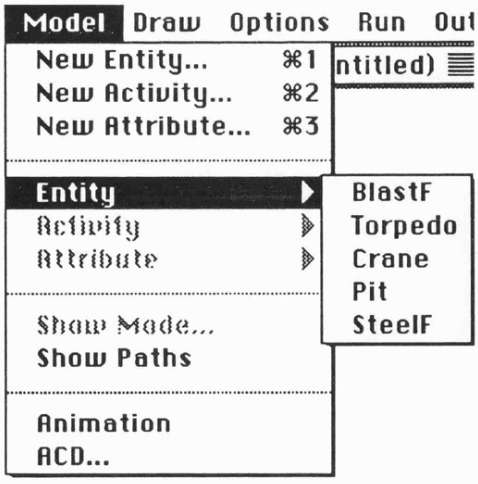


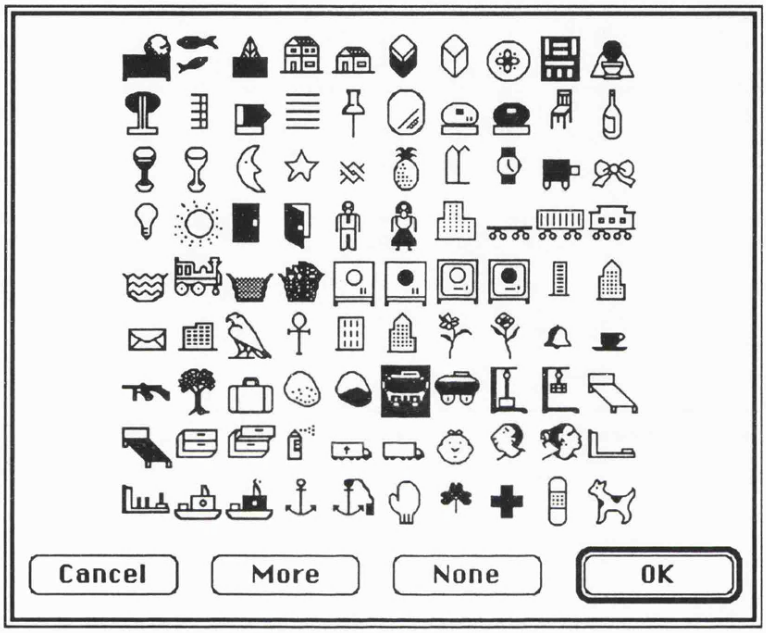
Figure F.8. The Entity Menu of the Steelworks Model



The first icon of an entity type is always automatically created on screen when the entity type is created. If an icon is not selected for the entity type, an empty square frame with an index of '1' will appear at the top-left hand corner of the main window.

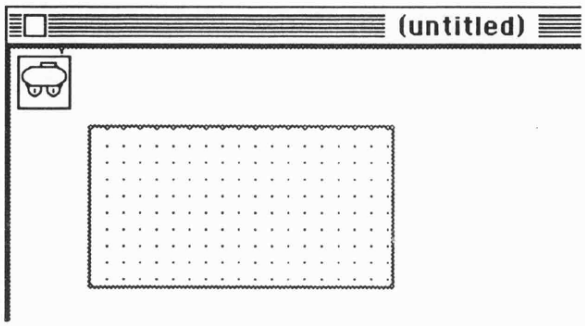
To select an icon that represents the entity type, click at the 'Icon' button inside the entity dialogue box. Figure F.9 shows the selection of an icon for the entity type 'TORPEDO' in the steelworks model. The user can select an icon by clicking at the appropriate choice displayed in the icon palette.

Figure F.9. Selecting an Icon for the entity type 'TORPEDO'



Once the 'OK' button inside the icon dialogue box is clicked, the entity icon will appear at the top-left hand corner of the main window (figure F.10) for the newly created entity type. An icon can be changed in the same manner, and any associated icons on the screen belonging to the entity type will be redrawn using the newly selected icon.

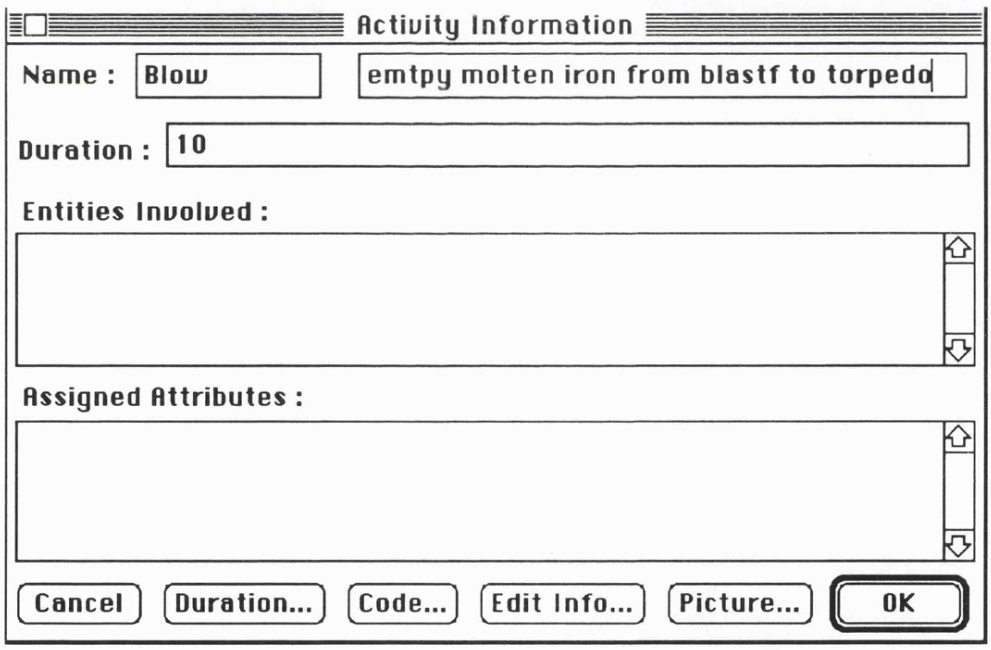
Figure F.10. The Main Window - after an icon is selected for 'TORPEDO'



F.3. Creating an Activity

To create an activity in the model, select the New Activity command from the Model menu. The activity dialogue box is shown in figure F.11. For example, to create an activity 'BLOW' which represents the process of emptying the molten iron produced in the blast furnace into the torpedoes in the model, enter 'BLOW' in the name field and any comments in the comment field. The activity will be added to the model if the user clicks the 'OK' button in the dialogue box. The name of the activity is then added to the Activity submenu in the Model menu so that the user can review and edit the information by selecting the appropriate activity in the menu. Figure F.12 shows the appearance of the Activity menu when all the activities are defined in the model.

Figure F.11. Creating the Activity 'BLOW'



If the duration of an activity is a constant, the value can be added directly to the duration field. The duration formula can also be entered via the 'Duration' button in the activity dialogue box. The duration dialogue box is shown in figure F.13. For example, to specify the duration formula for activity 'MELT', which is normally distributed with a mean of 110 and standard deviation of 15, the user clicks at the cell labelled

'Normal(mean, sd, seed)' in the distribution table. The dialogue box will prompt the user for the corresponding parameters. The duration formula is recorded when the user clicks the 'OK' button in the dialogue box.

Figure F.12. The Activity Menu of the Steelworks Model

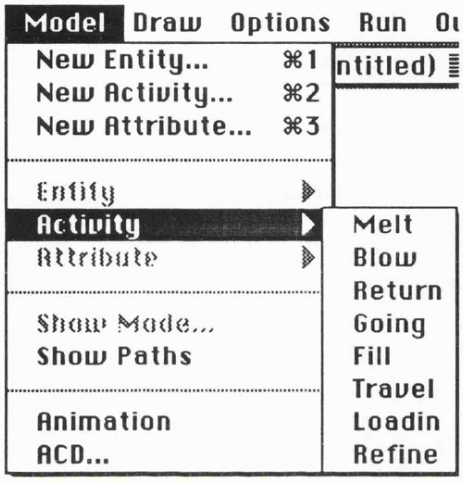
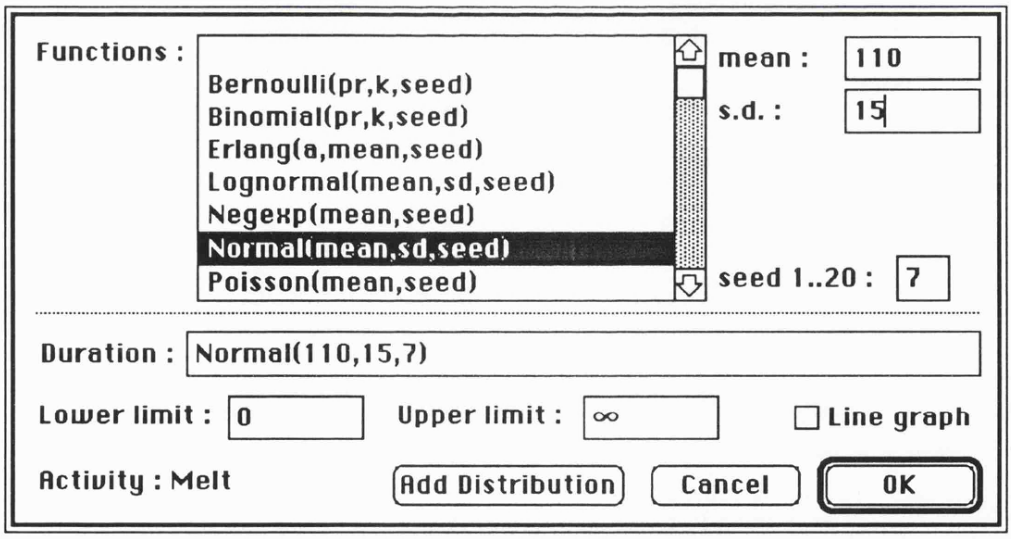
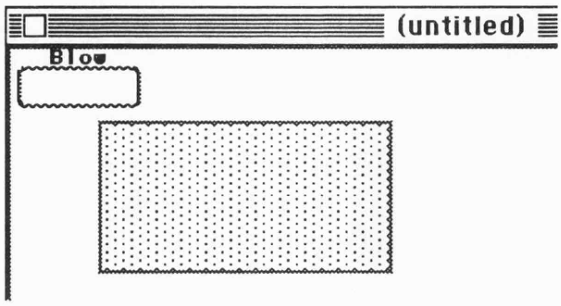


Figure F.13. Entering the Duration formula for activity 'MELT'



After an activity has been successfully added to the model, a round-cornered rectangle labelled with the name of the activity above the rectangle will appear at the top-left hand corner of the main window. Figure F.14 shows the main window after the activity 'BLOW' has been added to the model.

Figure F.14. The Main Window - after activity 'BLOW' is created



F.4. Creating an Attribute

A new attribute can be created by selecting the New Attribute command in the Model menu. For example, to create an attribute 'BCAST' (i.e. the amount of molten iron that the blast furnace produced in activity 'MELT'), enter 'BCAST' in the name field and click the 'OK' button. A histogram can be specified in the attribute dialogue box (figure F.15) by clicking at the histogram check box. The attribute 'BCAST' is added to the model and the name of the attribute is appended to the Attribute submenu in the Model menu. Figure F.16 shows the Attribute menu when attributes 'BCAST' and 'CCAST' are added. The user can review and edit attribute information by selecting the appropriate item in the Attribute submenu.

Figure F.15. Creating an Attribute 'BCAST'

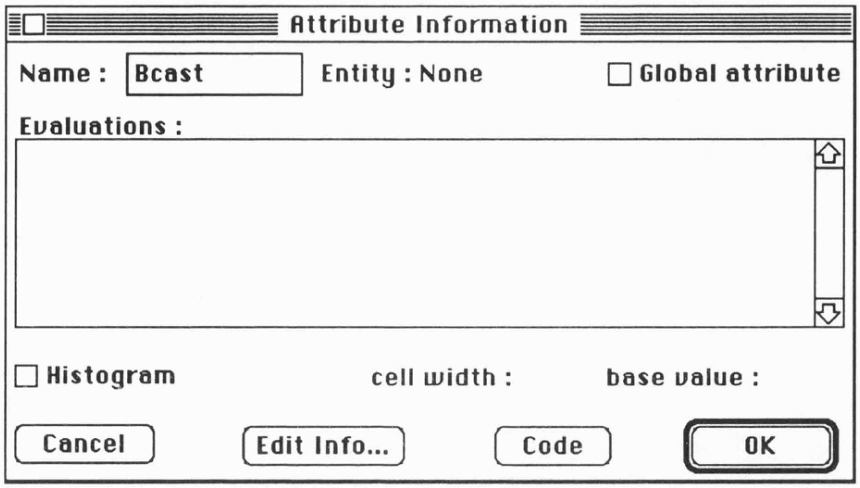
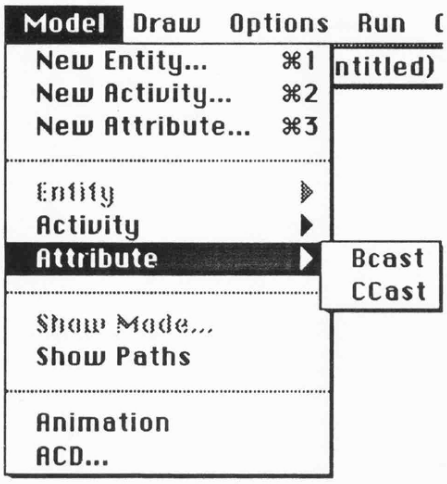
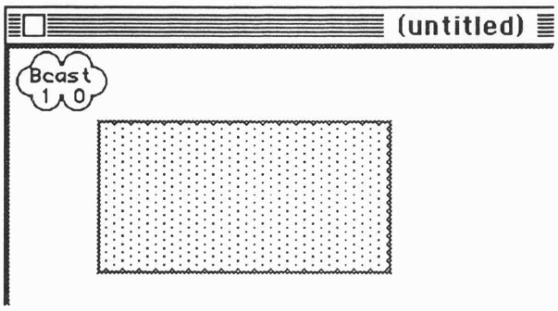


Figure F.16. The Attribute Menu of the Steelworks Model



After an attribute is created, its first attribute cloud appears in the top-left hand corner of the main window as shown in figure F.17. A newly created entity attribute is not linked to an entity in the model. The user has to identify the relationship by selecting the attribute cloud, and dragging the mouse button to the appropriate entity icon on the screen. When the user releases the mouse, a line will be drawn between the attribute cloud and the entity icon. Similarly, a link can be established between a system attribute and an activity in the same manner.

Figure F.17. The Main Window - after the attribute 'BCAST' is created



F.5. Formulating the Model Logic

The formulating mechanism of the system is manipulated by using the six

palette buttons inside the mode box window of the application. Figure F.18 shows the functions of the buttons used in modelling.

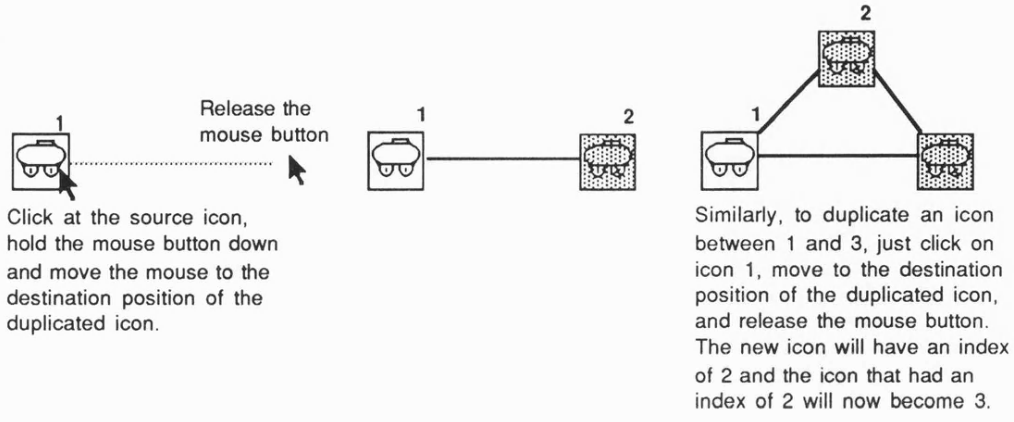
Figure F.18. The Palette Buttons inside the Mode Box Window

F.5.1. Duplicating Entity Icons

The life cycle of an entity type is depicted by using chains of entity icons created in the main window. These entity icons are indexed. The user can either create a ghost icon from an existing icon within the entity icon chain, or delete an existing ghost icon from the chain. Each entity icon on screen should be placed inside an activity.

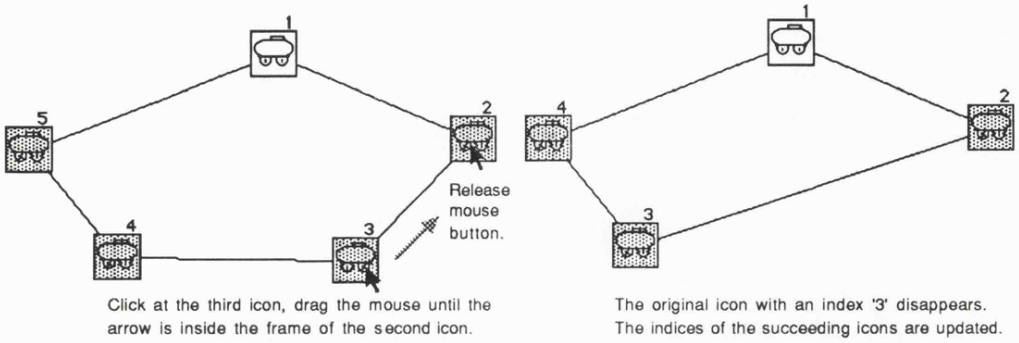
The 'Select entity' button in the mode box window is used to move the position of the entity icons on screen. Once the button is clicked, the application will redraw the screen using square frames and indices for each individual entity icon. The user can select an entity icon by clicking anywhere within its frame and moving the icon to the desired position. To duplicate an entity icon, the 'Duplicate entity' button is used. Figure F.19 shows how to duplicate icons for the entity type 'TORPEDO'.

Figure F.19. Duplicating entity icons for the entity type 'TORPEDO'



A ghost entity icon can be deleted by moving the entity icon to be deleted towards its preceeding member in the chain (figure F.20). For example, to delete the third torpedo icon in its entity chain, first click at the entity icon and then drag the mouse towards the position of the second torpedo icon. The third torpedo icon will then be deleted and the indices of all its succeeding members will be updated.

Figure F.20. Deleting an entity icon from its entity chain



F.5.2. Specifying an Activity

There are two ways of specifying the entities that are involved in an activity : either by moving the entity icons into the activity rectangle (figure F.21a) or by moving the activity rectangle over the icons (figure F.21b).

Figure F.21a. Specifying activity 'FILL' in the Steelworks Model

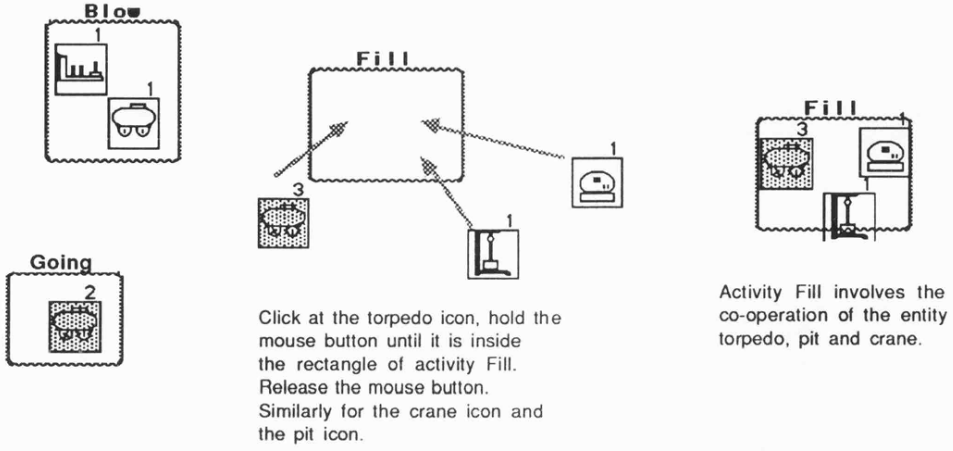
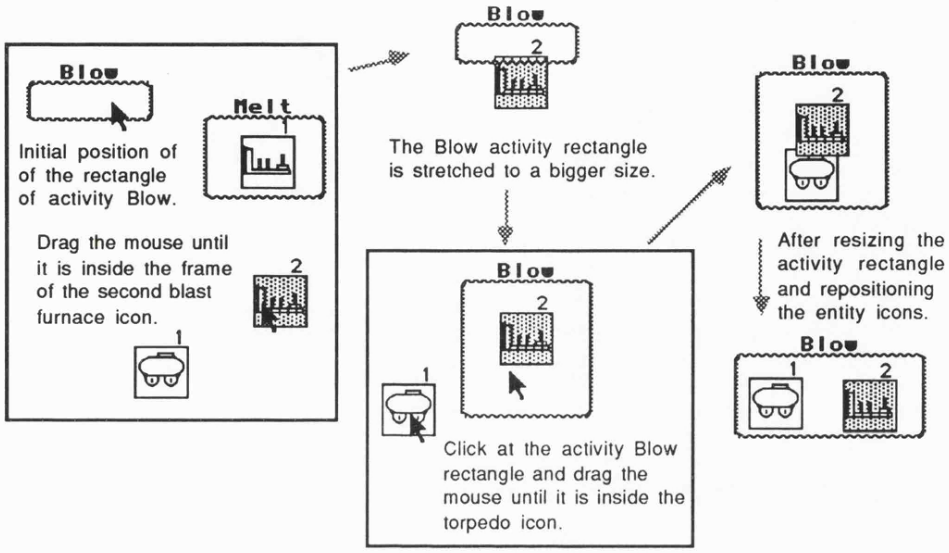


Figure F.21b. Specifying activity 'BLOW' in the Steelworks Model



The status of the model picture is constantly updated after each user's action so that information about an activity or an entity can be instantly reviewed. For example, figure F.22a and F.22b show the Dialogue box of activity Blow before and after the torpedo entity is added to the activity.

Figure F.22a. The Blow Activity Dialogue Box - before torpedo is added

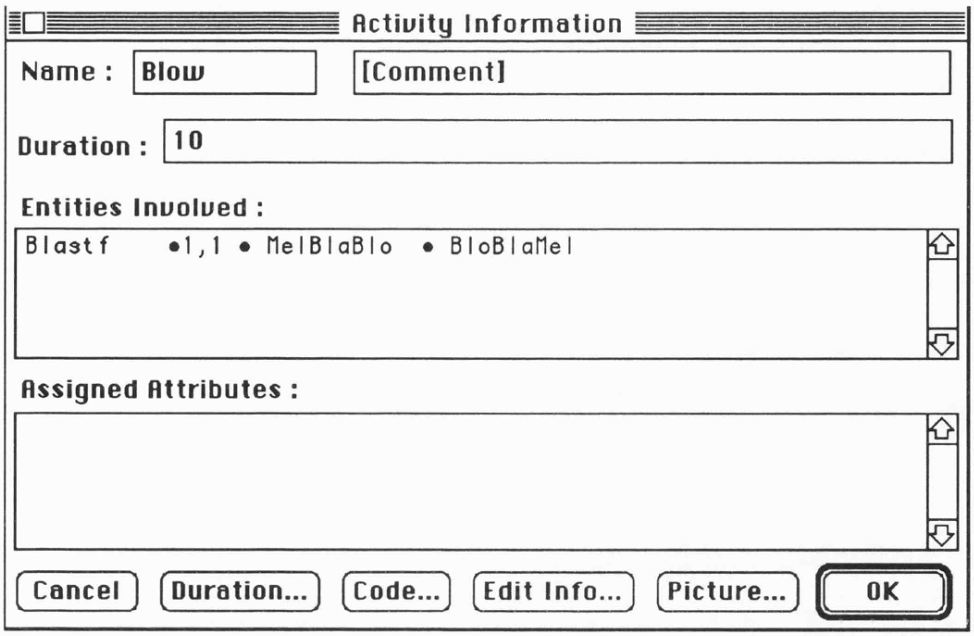
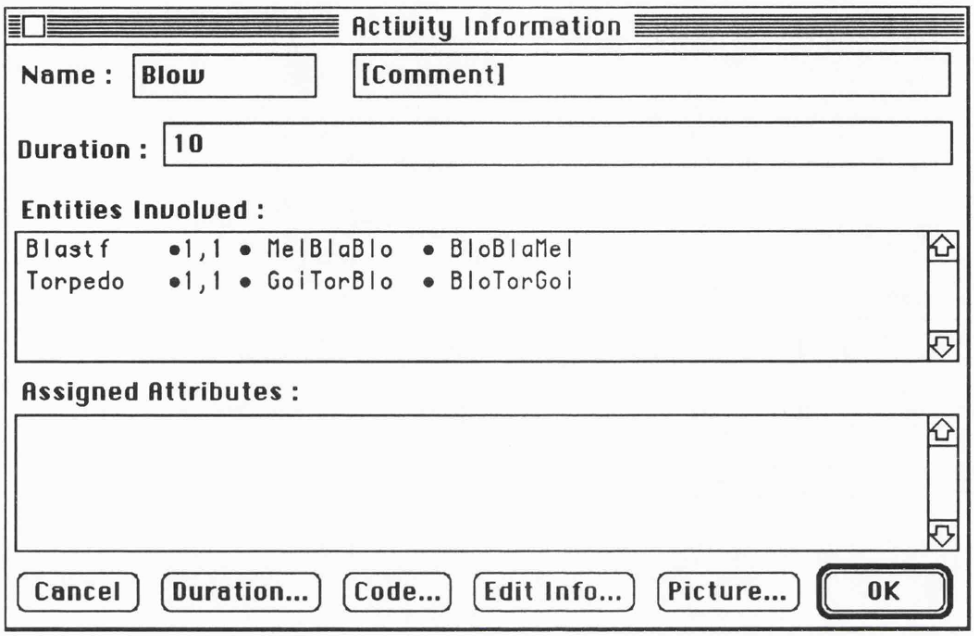


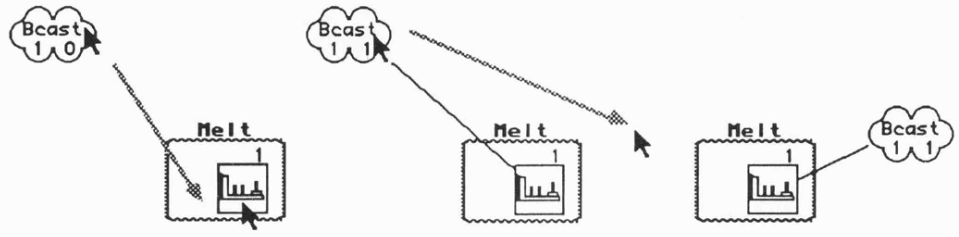
Figure F.22b. The Blow Activity Dialogue Box - after torpedo is added



F.5.3. Attribute Assignments

A newly created entity attribute should be linked to an entity and a system attribute should be linked to an activity. Figure F.23 shows how to link the attribute 'BCAST' to the entity blast furnace. Once the attribute is linked to an entity or an activity, the user can enter the evaluation formula by double-clicking at the attribute cloud on the screen. The attribute evaluation Dialogue box is shown in figure F.24.

Figure F.23. Linking Attribute 'BCAST' to Entity 'BLASTF'

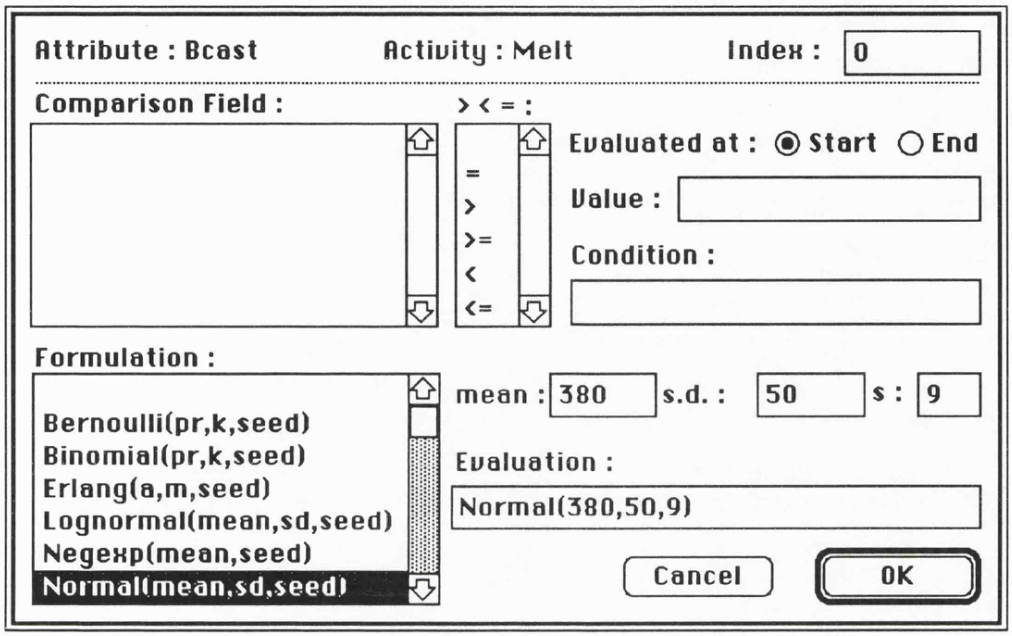


The right-hand corner index of the cloud is '0' as it is not placed in an activity. Click at the Bcast attribute cloud, drag the mouse until it is inside the frame of the blast furnace icon. Release the mouse button.

A line is drawn between the bcast attribute cloud and the blast furnace icon. Click at the cloud and move it to the right hand side of the Melt activity rectangle.

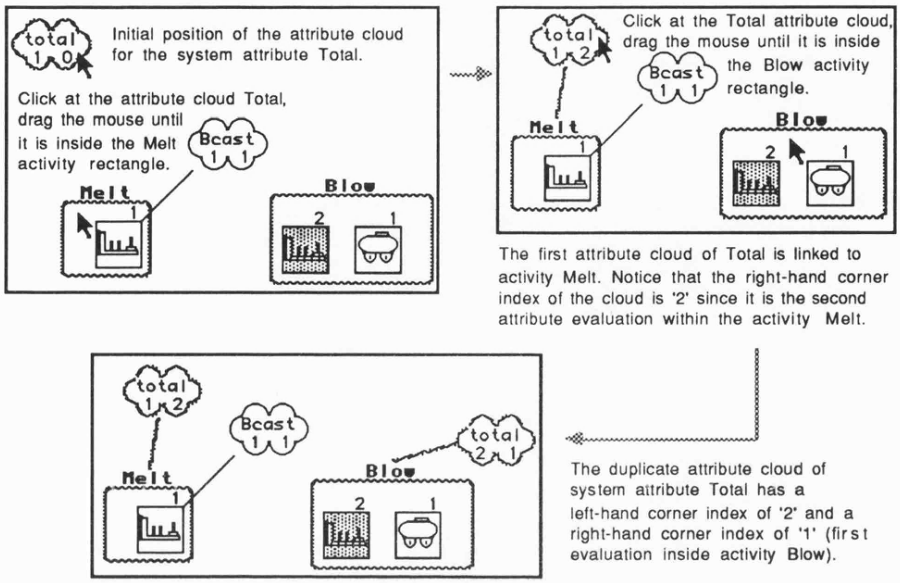
N.B. The right-hand corner index of the Bcast attribute cloud is '1' since it is the first attribute evaluation inside the activity Melt.

Figure F.24. The Attribute Evaluation Dialogue Box for Attribute 'BCAST'



If there is more than one evaluation of the attribute either among the other activities, or within the same activity in the model, cloud images can be duplicated for the attribute type. For example, figure F.25 shows how an attribute cloud of a system attribute 'TOTAL', which is evaluated at both activities 'MELT' and 'BLOW', can be duplicated.

Figure F.25. Duplicating an Attribute Cloud

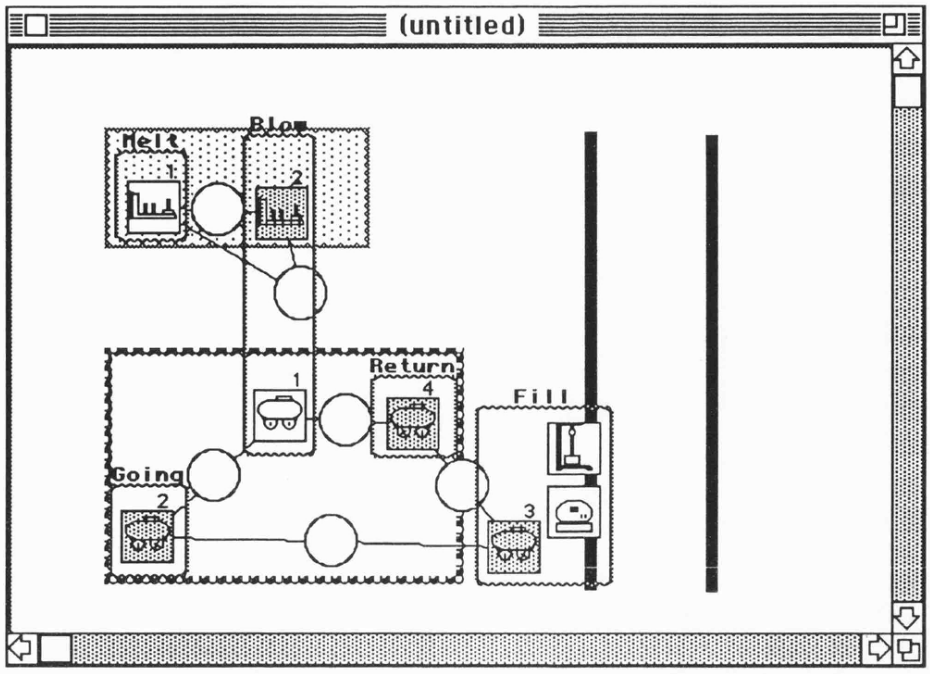


Like the duplication of entity icons mechanism, a cloud object in an attribute cloud chain can be deleted in the same manner. For example, if the third cloud member is to be deleted, click at this cloud, drag the mouse until it is inside the second cloud of the attribute chain, and then release the mouse. The original cloud with a left-hand corner index of '3' will disappear and the indices of its succeeding members in the attribute cloud chain will be updated.

F.5.4 Specifying Queue Information

All the queues are generated automatically by the MacGraSE application. They are initially invisible to the user. To see the picture with queues, click at the 'Select queue' button in the mode box window. The application will then redraw the pictorial model with queues between entity icons. Figure F.26 shows the picture model when the 'Select queue' button inside the mode box window is selected after the life cycle of the torpedo is completed.

Figure F.26. A Model Picture with Queues



To edit information for a queue, just double-click at its appropriate location in the model picture. The queue Dialogue box is shown in figure F.27. The user can change the default name of the queue in the name field, enter the number of entities in the queue in the number field, and create histograms and time series for the queue by clicking the appropriate check box inside the queue Dialogue. Any histograms or time series specified are appended to the Output menu.

Figure F.28 shows the appearance of the Output menu after a queue length histogram, a queuing time histogram, and a time series histogram have been created for queue 'RETTORBLO'. This Output menu allows the user to select the results to be reviewed after completing a simulation run.

Figure F.27. The Queue Dialogue Box

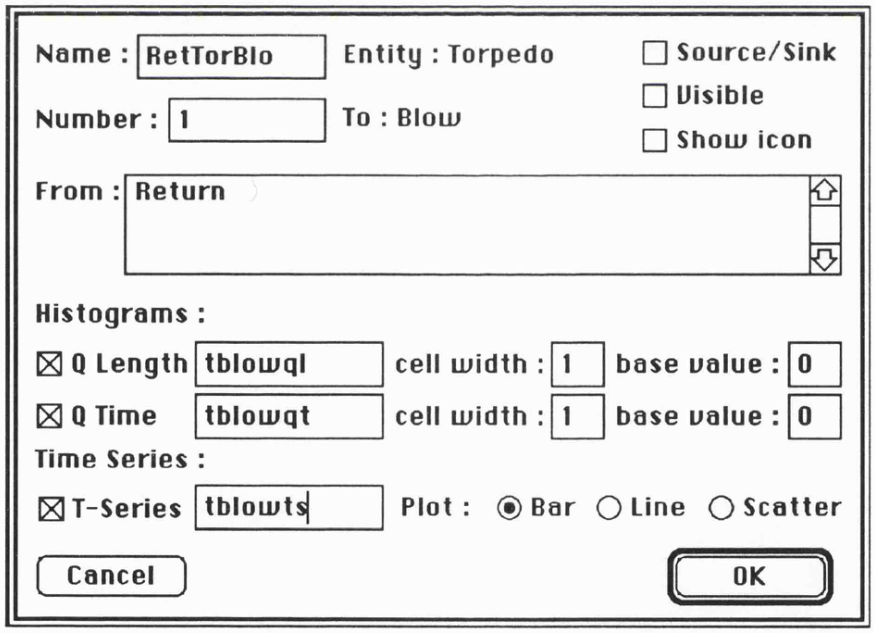
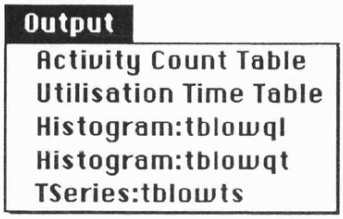


Figure F.28. The Output Menu



F.5.5. Adding a Conditional Path

A conditional path for an entity life cycle can be created by using the 'Conditional path' button inside the mode box window. Once the button is selected, the system will redraw the model picture with queues between entity icons. To create a conditional path, click at the entity icon from which the path originate and then drag the mouse into the destination queue. The system will redraw the life cycle with the new conditional path (a line with a small square in the middle). To enter the condition for the path, click at the small square to invoke the condition dialogue box (figure F.30).

Figure F.29. Creating a Conditional Path

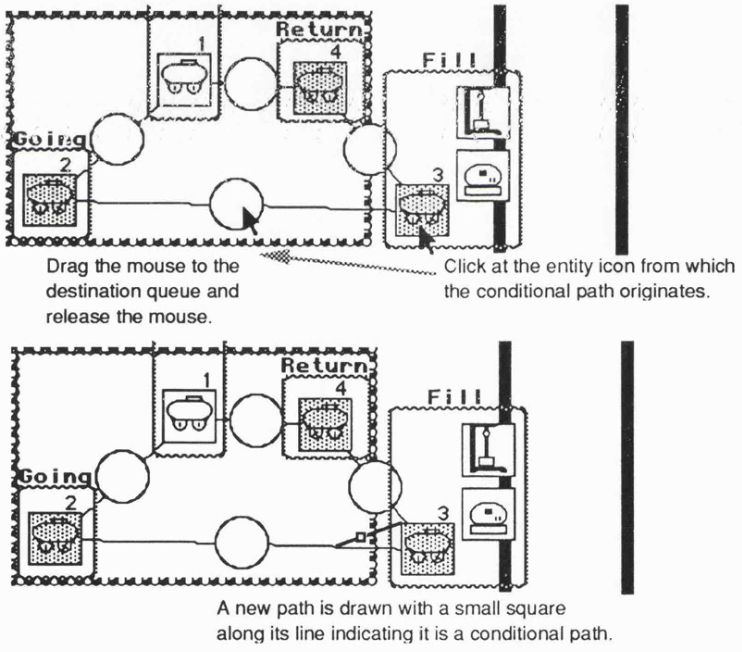
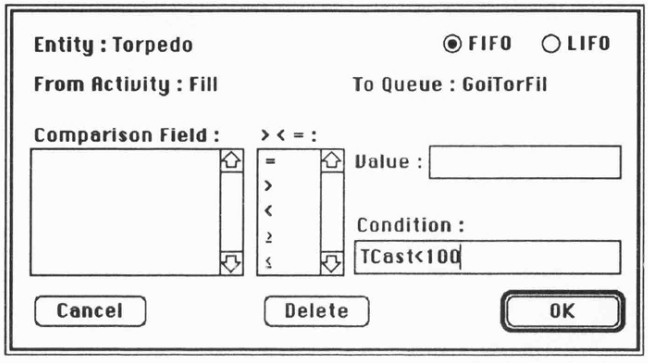


Figure F.30. The Condition Dialogue Box



F.6. Generating an Activity Cycle Diagram

An activity cycle diagram of the model can be generated on screen by selecting the 'ACD' command in the Model menu. Since the initial positions of the activities and queues are set by using the corresponding locations in the model picture, overlapping of objects often occurs for the first generated ACD. However, the objects can be easily rearranged and the new locations are recorded. Any additional components to the model will not affect the user-defined positions of the original existing objects. These added components will be drawn by using their default positions and the user can reposition them if preferred. Figures F.31a and F.31b show the original generated ACD and the revised ACD after rearranging the objects for the steelworks model.

Apart from being a flow diagram that represents the logic of the model, the user can also enter the information of activities and queues within the ACD. To enter the information of an activity, double-click at the appropriate activity rectangle in the ACD. The activity Dialogue box is then invoked and the user can edit the information of the selected activity. Similarly, the information of a queue can be edited in the same way.

Figure F.31a. The Original Generated ACD of Steelworks

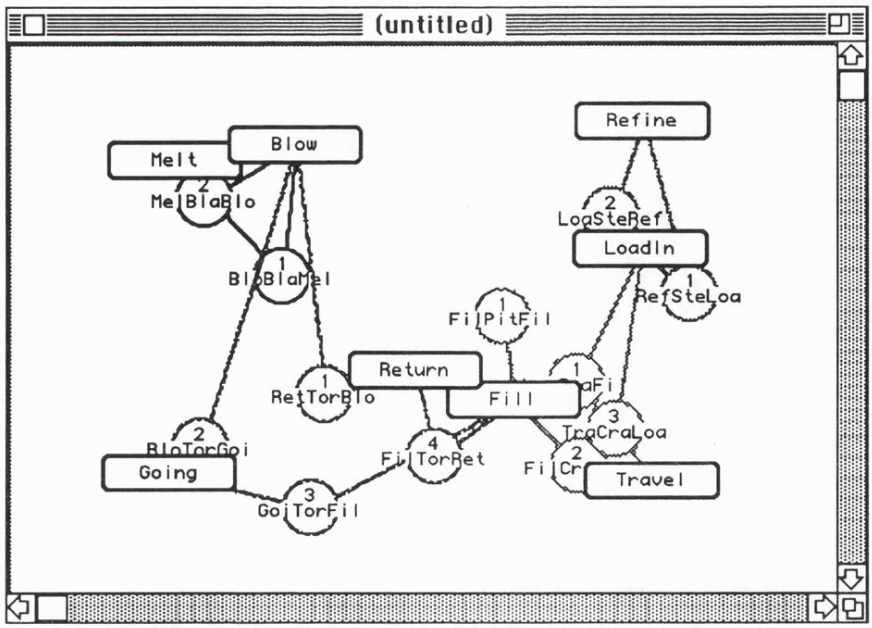
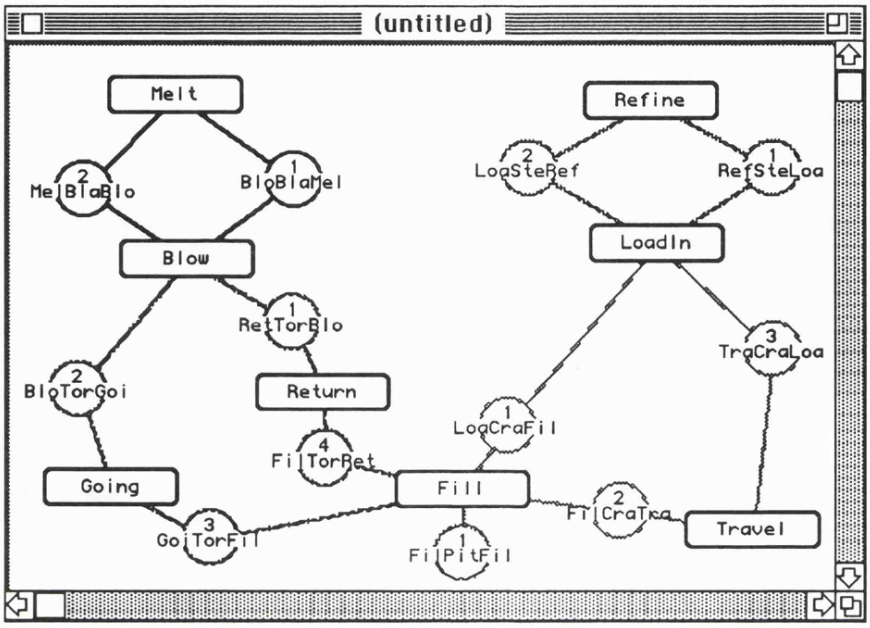


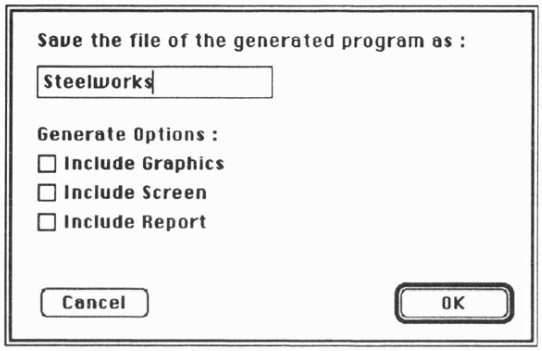
Figure F.31b. The Revised ACD of Steelworks



F.7. Generating a Simulation Program

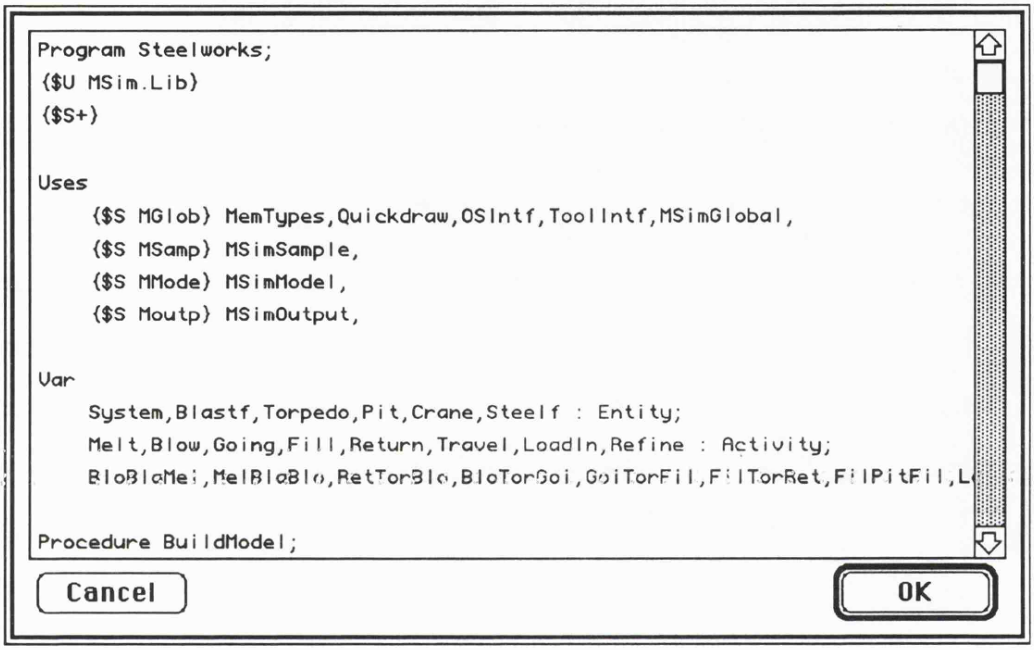
Turbo Pascal is the default language. The user can select the desired language in the Language submenu of the Run menu. To generate a simulation program, select the Generate command in the Run menu. The application will prompt the user for the name of the generated files and allow the user to select some options for program generation via the generate Dialogue box (figure F.32). The program will not be generated until the 'OK' button is clicked.

Figure F.32. The Generate Dialogue Box



To see the generated program within the application, select the Show Program command from the Run menu. The program is then displayed in the Dialogue box as shown in figure F.33.

Figure F.33. The Show Program Dialogue Box



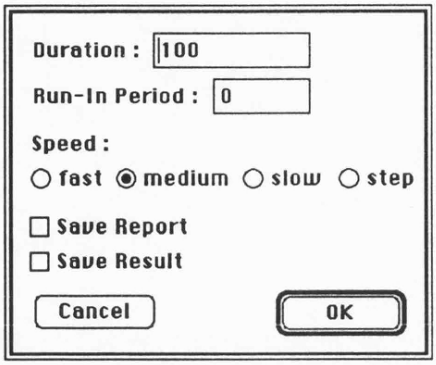
F.8. Running the Simulation Model

The MacGraSE application supports three types of simulation run - a visual run, a text run, and a screen run. In each case, the duration, the run-in period and the speed can be specified.

F.8.1. Visual Run

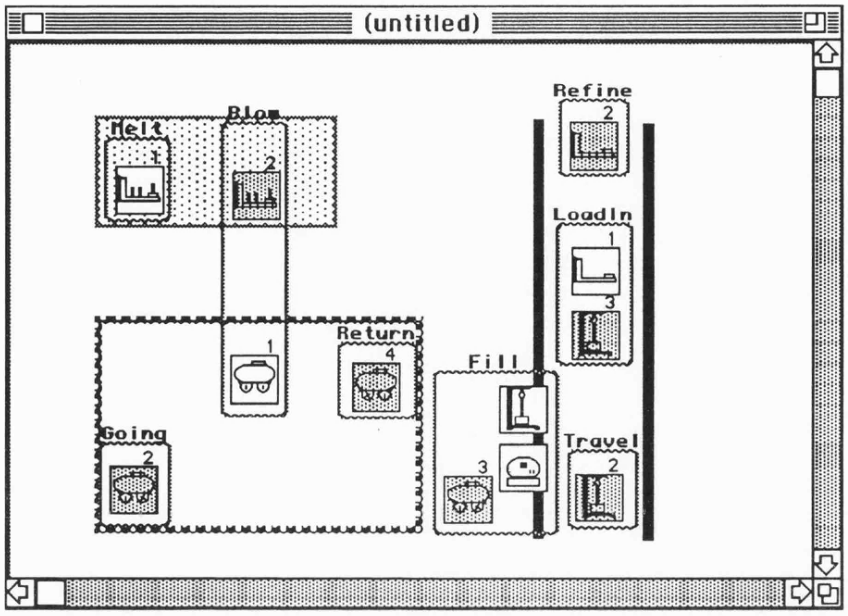
A visual run is executed by selecting the Visual Run command from the Run menu. The visual run Dialogue box is shown in figure F.34. The visual run can be proceeded by selecting the Go command from the Run menu.

Figure F.34. The Visual Run Dialogue Box



The model picture of the steelworks model is shown in figure F.35. The dynamics of the model during a simulation run is shown by displaying the start and completion of activities at each time advance in the model picture. The start of an activity is shown by drawing the activity rectangle with the icons of the entity types that are involved in the activity. The number of times that the activity has started since the beginning of the simulation run is also shown. The completion of an activity for an entity is shown by drawing the entity icon within a circular frame at the location of the queue where the entity enters after the activity. Lines are also drawn during a visual run to indicate where the entity comes from and where it goes to.

Figure F.35. The Visual Picture



F.8.2. Text Run

A text run is executed by selecting the Text Run command from the Run menu. The text run Dialogue box is shown in figure F.36. The user can select the option of displaying either a text run table, or a simulation clock during the simulation run. The text run table is shown in figure F.37.

Figure F.36. The Text Run Dialogue Box

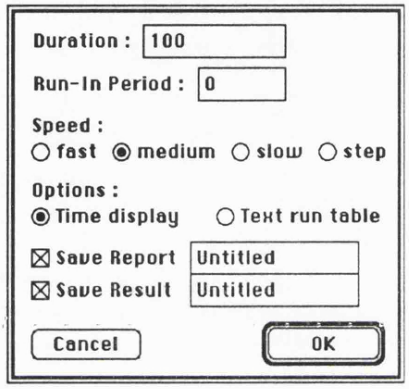


Figure F.37. The Text Run Table

Time : 0

End of Activities			Start of Activities		
Activity	Count	Entity	Activity	Count	Entity
			Melt	1	Blastf 1

The text run table is divided into two main columns - the End of Activities column and the Start of Activities column. Each column is then subdivided into three subsections to display the name of the activity, the index indicating the rank of the activity, and the name and index of the entity that is involved in the activity.

F.8.3. Screen Run

A screen run for a predefined screen in the model is executed by selecting the Screen Run command from the Run menu. The screen run Dialogue box is shown in figure F.38. After setting the options in the Dialogue box, the user can proceed the simulation run by selecting the Go command from the Run menu. Figure F.39 displays an example of the appearance of a screen which contains a time series of a queue during a simulation run.

Figure F.38. The Screen Run Dialogue Box

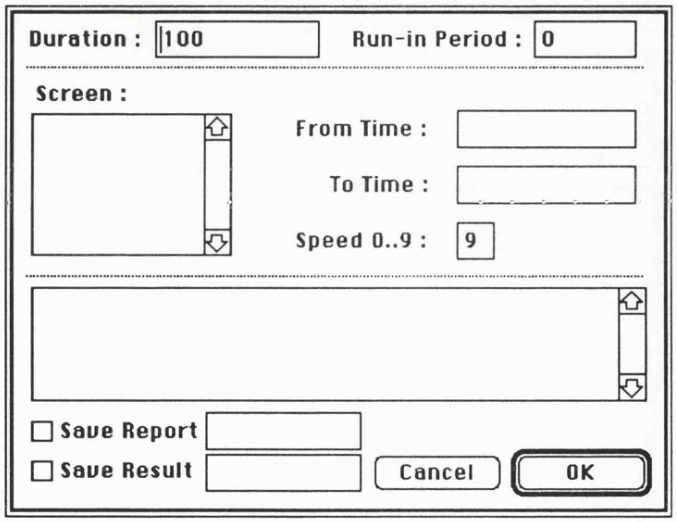
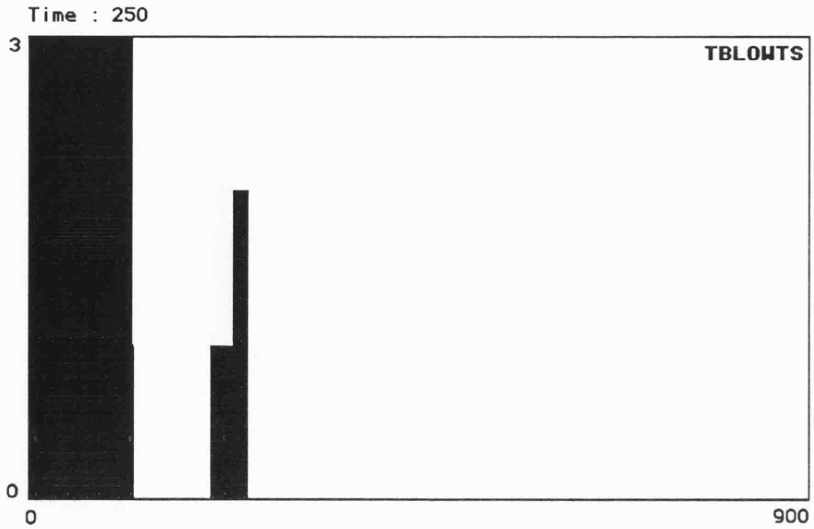


Figure F.39. An Output Screen



F.9. Displaying Results

The output results can be reviewed by selecting the menu items inside the Output menu. Figures F.40 to F.43 show the display of some of the results after a simulation run of 900 time units for the steelworks model.

Figure F.40. The Activity Count Chart

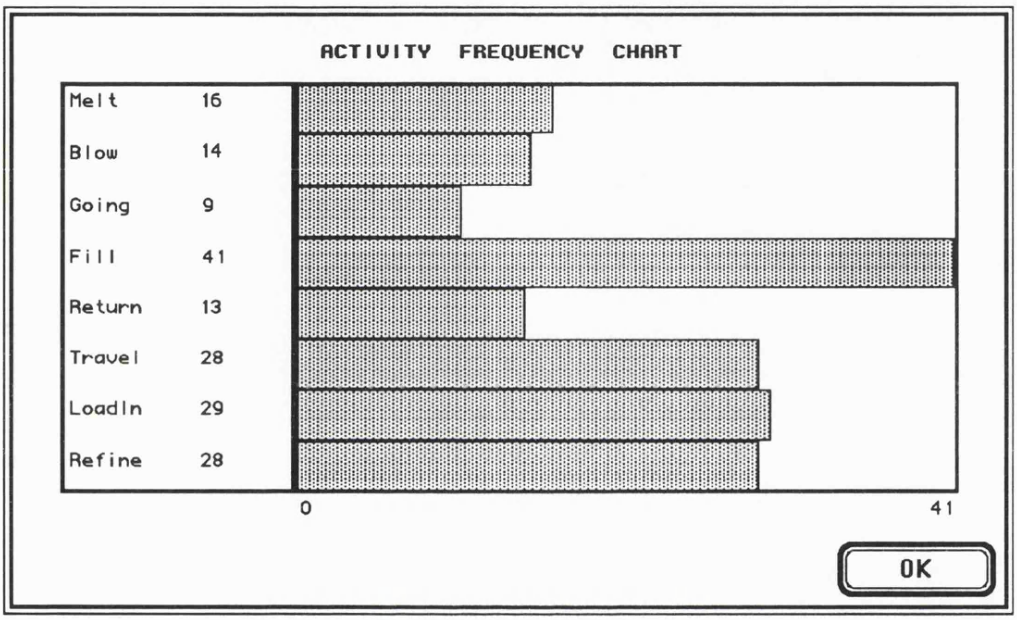


Figure F.41. The Entity Utilisation Chart

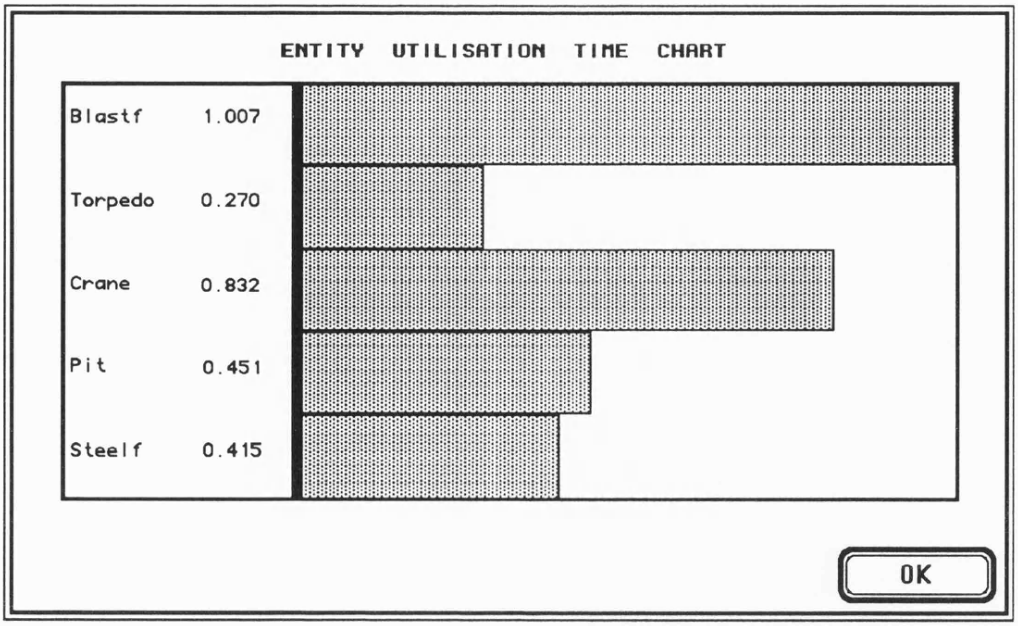


Figure F.42. The 'RETTORBLO' Queue Length Histogram : 'TBLOWQL'

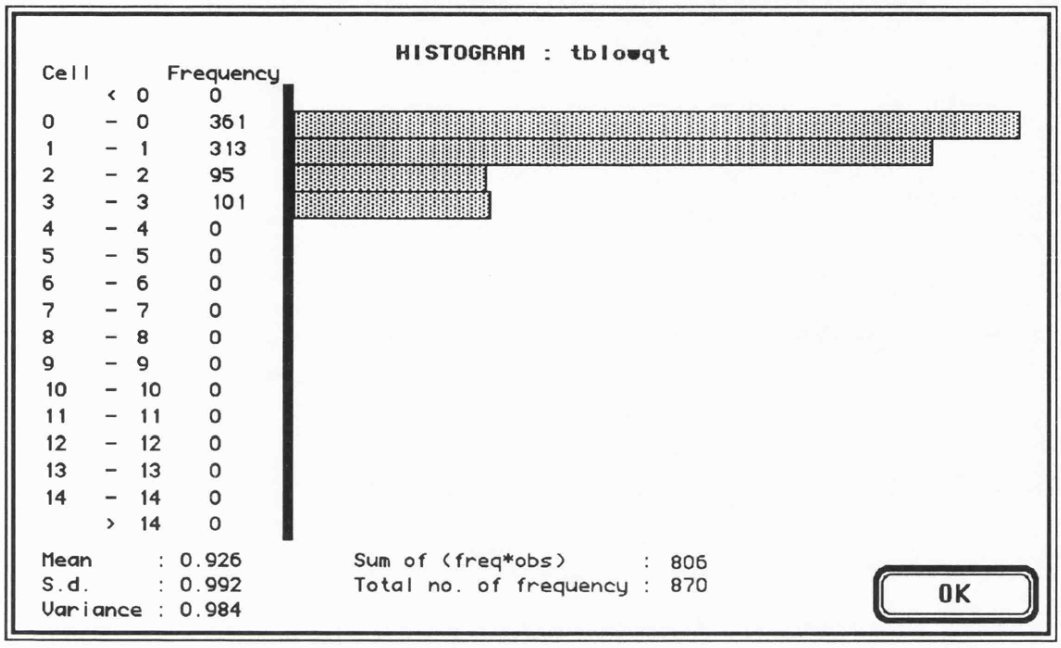
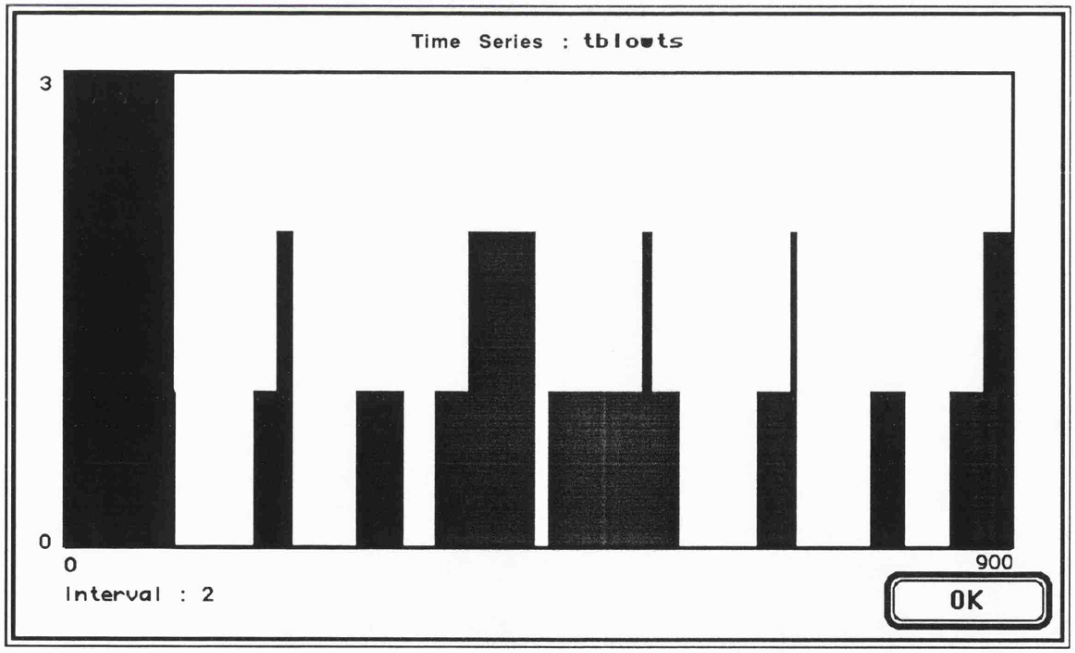


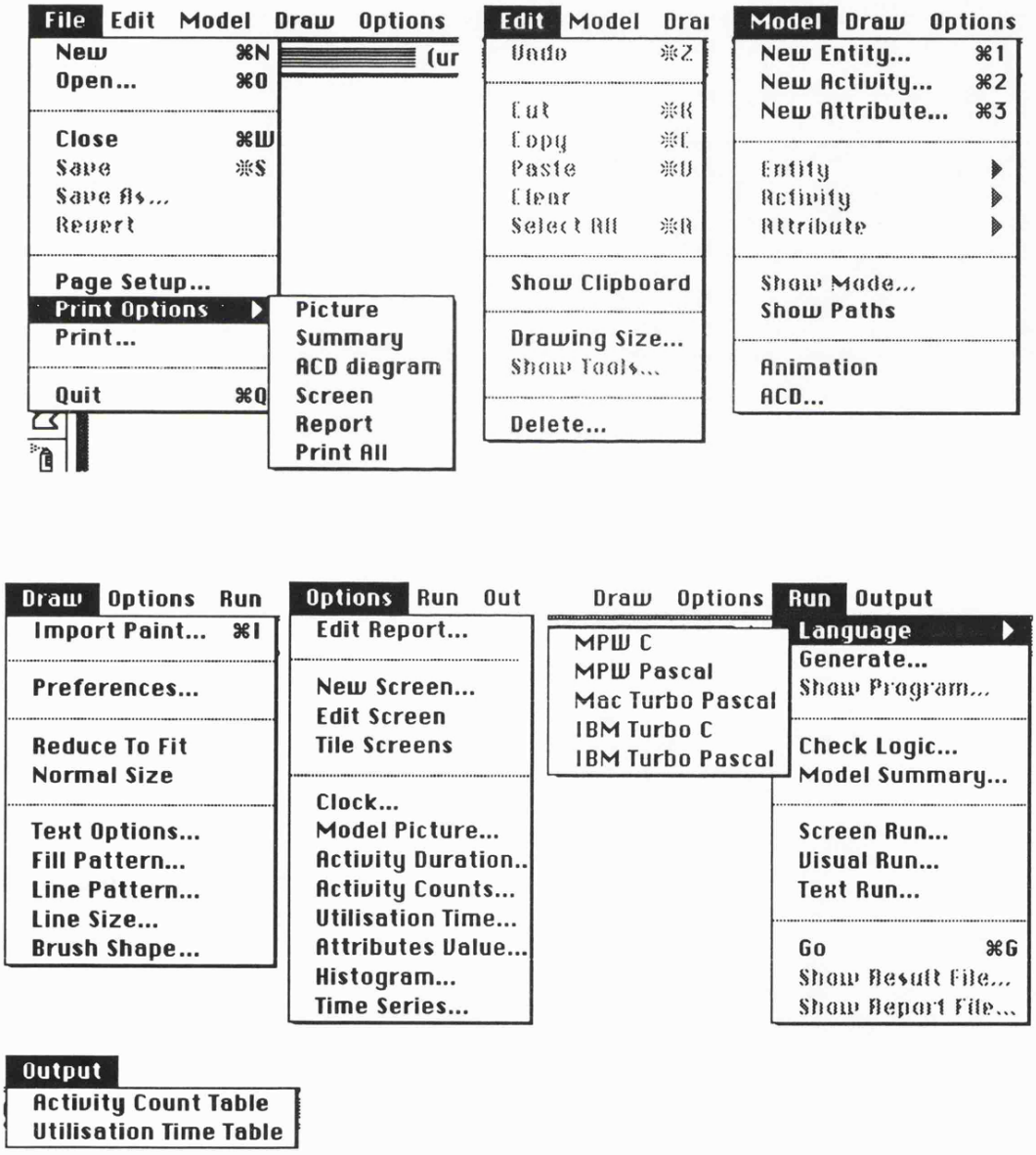
Figure F.43. The 'RETTORBLO' Time Series : 'TBLOWTS'



F.10. The Menu Reference

There are seven menus in the MacGraSE application, excluding the standard apple desk accessories menu (). Some of the menu items have hierarchical menus. Figure F.44 shows the appearance of the main menu.

Figure F.44. The Main Menu of MacGraSE



The File menu handles all the filing procedures within the application. *New* is used for creating a new MacGraSE file. *Open* is used for opening an existing file.

Close is used for closing the application window. *Save* and *Save As* is used for saving the working file. *Revert* allows the user to revert to the last copy of the file before any changes is made. *Page SetUp* is used for setting the options of the paper size in printing. *Print Options* is a hierarchical menu which allows the user to select the option of different representations of the model to be printed. *Print* is used for printing the working file. *Quit* is used to quit the application.

The Edit menu contains the Cut, Copy and Paste commands to aid the editing of the background picture and textual data input. *Drawing Size* allows the user to select the drawing size of the model. *Delete* is used for deleting the selected components of the model. The tool box window can be hidden and shown on the screen by using the *Show Tools* command.

The Model menu contains all the commands for building up a simulation model. New components - entity, activity or attribute, can continuously be added to the model by using the *New Entity*, *New Activity* and the *New Attribute* menu items respectively. These components will be appended to the appropriate submenu (*Entity*, *Activity* and *Attribute*) within the Model menu. The user can view and edit information for each individual component by selecting the item in the submenu. *Show Path* allows the user to display the path (in terms of straight lines) of the entity type that is being constructed on screen. *ACD* is used for generating an activity cycle diagram. The user can also manipulate data via the generated ACD. *Animation* is used to execute an animation run so that the user can see the actual movement of individual entities that move inside the system during a simulation run.

The Draw menu is used to aid the drawing of the background picture. *Preferences* is used for setting the appearance of the ruler and grid inside the main window. *Line Size*, *Line Pattern*, and *Fill Pattern* are used to select the pen size, pen pattern, and fill pattern of the drawing pen. The appearance of the text is set by using the *Text Options* command. *Import Paint* allows the user to import a picture that is

drawn in other Macintosh drawing applications.

The Options menu has two main functions. The first function is report editing, so that the user can edit the report format to produce a desired report after a simulation run. This is achieved by using the *Edit Report* command. The second function is screen editing in which the user can create multiple output screens for a simulation run. *New Screen* is used to create a new output screen in the model. *Edit Screen* is used in specifying the objects inside a selected screen. The objects that can be put in a screen are mainly the simulation clock, utilisation time table, activity count table, histograms, time series, numerical statistics, attribute values and status of an entity type. *Tile Screens* allows the user to display all the designed screen in a tiling format on the computer desktop.

The Run menu is used for performing simulation runs on the model. *Language* is a hierarchical menu which allows the user to select the language for the generated program. *Generate* is used for generating a three-phase simulation program. *Show Program* displays the generated program in a Dialogue box. *Check Logic* is used for checking the logic of the model. A textual description of the current state of the model can be reviewed by using the *Model Summary* command. There are three options for a simulation run - *Screen Run*, *Visual Run*, and *Text Run*. When the parameters inside each option are entered, the user can select the *Go* command to process the simulation run. In each case, the user can select the options of saving the result file and the report file of a simulation run. *Show Result* and *Show Report* display the result file and the report file in a Dialogue box respectively.

The Output menu is only appended to the menu bar when a simulation run is executed by the user. Any specified histograms, time series or graphs are added to this menu, so that the user can select the output to be reviewed after a simulation run. The default setting of this menu includes the Utilisation Time Table and the *Activity Count Table*.

APPENDIX G

INSIDE MACGRASE

This appendix shows the structure of the Macintosh application MacGraSE program. MacGraSE was written in 60 units, used in conjunction with the AppMaker library which handles most of the interface part of the application The program was written by using the Macintosh Programming Workshop Pascal Compiler. The make file of the MacGraSE application is shown in figure G.1.

Figure G.1. The MacGraSE Make file for MPW Pascal

```
#     File:          MacGraSE.make
#     Target:        MacGraSE

LinkFiles = ∂
            ResourceDefs.p.o  ∂
            Globals.p.o  ∂
            ShowMode.p.o  ∂
            ShowTools.p.o  ∂
            MainWindow.p.o  ∂
            MainMenu.p.o  ∂
            Filing.p.o  ∂
            Edit.p.o  ∂
            Language.p.o  ∂
            Condition.p.o  ∂
            NewEntity.p.o  ∂
            MEntity.p.o  ∂
            NewActivity.p.o  ∂
            MActivity.p.o  ∂
            MAttribute.p.o  ∂
            AttCalculation.p.o  ∂
            NewAttribute.p.o  ∂
            Model.p.o  ∂
            Draw.p.o  ∂
            Options.p.o  ∂
            Run.p.o  ∂
            PrintOptions.p.o  ∂
            Output.p.o  ∂
            EditScreen.p.o  ∂
            BrushShape.p.o  ∂
            TextOptions.p.o  ∂
            FillPattern.p.o  ∂
            LinePattern.p.o  ∂
            LineSize.p.o  ∂
            Generate.p.o  ∂
            DrawingSize.p.o  ∂
            Preferences.p.o  ∂
            ScreenRun.p.o  ∂
            EditReport.p.o  ∂
            MSummary.p.o  ∂
            VisualRun.p.o  ∂
            TextRun.p.o  ∂
            NewScreen.p.o  ∂
            Delete.p.o  ∂
            Show.p.o  ∂
            ImportPaint.p.o  ∂
            EntIcon.p.o  ∂
```

```
                Distributions.p.o  ∂
                Histogram.p.o  ∂
                TimeSeries.p.o  ∂
                QueInfo.p.o  ∂
                ACDRect.p.o  ∂
                ACDLine.p.o  ∂
                ACDQue.p.o  ∂
                Priority.p.o  ∂
                ActDuration.p.o  ∂
                Code.p.o  ∂
                LifeCycle.p.o  ∂
                ActPicture.p.o  ∂
                EditActInfo.p.o  ∂
                EditAttInfo.p.o  ∂
                CheckLogic.p.o  ∂
                ShowReport.p.o  ∂
                Animation.p.o  ∂
                Dispatcher.p.o  ∂
                "{MPW}AMLibraryP:"AMLib.o

'MacGraSE'  ƒƒ  {LinkFiles}
      Link  -w  -t  APPL  -c  XXXX  ∂
                {LinkFiles}  ∂
                "{Libraries}"Interface.o  ∂
                "{Libraries}"Runtime.o  ∂
                "{PLibraries}"PasLib.o  ∂
                "{PLibraries}"SANELib.o  ∂
#               -l  -lf  >  'MacGraSE.map'  ∂
                -o  'MacGraSE'
```

G.1. The Interface Units

Globals This unit contains all the global declarations of the program, including all the simulation modelling data structures and variables.

ResourceDefs This unit contains named constants for all named resources and all menus and menu items and contains menu handles for all of the menu resources.

MainWindow This is the unit that controls the appearance of the main window on screen in response to the user's actions. It contains procedures for drawing the background pictures and routines for modelling the objects on screen. Procedures for scrolling, updating, activating and sizing of the main window can also be found.

MainMenu This module contains code to initialise the menus, to choose from the menu, and to update the menu - to enable/disable menu items, for example.

Dispatcher The dispatcher routine is called by other units for window or modeless dialog-related activities. It determines which window or dialog is involved and branches to the appropriate module.

G.2. The File Menu Units

Filing The Filing module contains code to handle the File menu items. It handles New, Open, Close, Save, Save As, Revert, PageSetup, Print, and Quit. It also contains procedures for the actual reading and writing of a file.

PrintOptions This is a hierarchical menu module which handles the selection of menu items from the Print Options menu. The Print Options menu allows the user to choose which representation of the model to be printed on paper.

G.3. The Edit Menu Units

Edit The Edit module contains code to handle the Edit menu items.

DrawingSize This is a modal dialog box module which handles the selection of the size of the drawing area for the model being constructed.

ShowTools This is a modeless dialog box module which handles the selection of the palette choices inside the tool box window. The tool box window is used to facilitate the drawing of the background picture of the model.

Delete This is a modal dialog box module which handles the deletion of simulation data structures that are predefined by the user.

G.4. The Model Menu Units

Model The Model module contains code to handle the Model menu items. The Model menu is used for the modelling of the simulation problem.

NewEntity This is a modeless dialog box module which handles the user's interaction with the entity dialog box when either a new entity type is being created or information about an existing entity type is being reviewed.

EntIcon This is a modal dialog box which handles the selection of an entity icon for an entity type.

LifeCycle This unit handles the drawing of an individual entity life cycle when the 'Life Cycle' button inside the entity dialog box is clicked.

Condition This is a modal dialog box module which is used for accepting the condition on a conditional path which is created by the user in the model.

NewActivity This is a modeless dialog box module which handles the user's interaction with the activity dialog box when either a new activity is being created or information about an existing activity is being reviewed.

ActDuration This is a modal dialog box module which contains routines for helping the user enter the duration formula for an activity when the 'Duration' button inside the activity dialog box is clicked.

ActPicture This is a modeless dialog box module which contains drawing routines for creating an action picture for an activity when the 'Picture' button inside the activity dialog box is clicked.

EditActInfo This unit handles the drawing of a flow diagram which shows the current status of an activity in the model when the 'Edit Info' button inside the activity dialog box is clicked.

Code This is a modal dialog box module which handles the generation of simulation code for an activity when the 'Code' button inside the activity dialog box is clicked.

Priority This is a modal dialog box module which contains routines for switching the priorities of activities defined in the model.

NewAttribute This is a modeless dialog box module which handles the user's interaction with the attribute dialog box when either a new attribute is being created or information about an existing attribute is being reviewed.

AttCalculation This is a modal dialog box module which contains routines for helping the user enter the evaluation formula for an attribute object on screen.

EditAttInfo This unit handles the drawing of a flow diagram which shows the current status of an attribute in the model when the 'Edit Info' button inside the attribute dialog box is clicked.

QueInfo This is a modal dialog box module which contains information about a queue that is generated by the application. It also contains procedures for defining histograms and time series for the queue.

MEntity This is a hierarchical menu module which handles the selection of an entity type that is defined in the model. This invokes the entity dialog box and the data for the selected entity type will be fed into the appropriate entries inside the dialog box.

MActivity This is a hierarchical menu module which handles the selection of an activity that is defined in the model. This invokes the activity dialog box and the data for the selected activity will be fed into the appropriate entries inside the dialog box.

MAttribute This is a hierarchical menu module which handles the selection of an attribute that is defined in the model. This invokes the attribute dialog box and the data for the selected attribute will be fed into the appropriate entries inside the dialog box.

ShowMode This is a modeless dialog box module which handles the selection of the palette choices inside the mode box window. The mode box window is used to facilitate the construction of the simulation model.

Animation This unit contains routines used for model animation.

ACD This unit contains routines used for the generation of an activity cycle diagram representation of the model.

ACDRect This is a modal dialog box which contains routines used to control the appearance of activity rectangles in an activity cycle diagram.

ACDLine This is a modal dialog box which contains routines used to control the appearance of life paths in an activity cycle diagram.

ACDQue This is a modal dialog box which contains routines used to control the appearance of queue circles in an activity cycle diagram.

G.5. The Draw Menu Units

Draw The Draw module contains code to handle the Draw menu items. The Draw menu is used to facilitate the drawing of the background picture.

ImportPaint This unit contains procedures used for importing other pictures that are drawn in other Macintosh applications.

Preferences This is a modal dialog box module that allows the user to select

the options (ruler and grid) he prefers during drawing of the background picture.

TextOptions This is a modal dialog box module which contains routines for controlling the appearance of text on screen.

FillPattern This is a modal dialog box module which contains routines for selecting the pattern that is used to fill an object on screen.

LinePattern This is a modal dialog box module which contains routines for selecting the pen pattern that is used to draw an object on screen.

LineSize This is a modal dialog box module which contains routines for selecting the size of the pen that is used to draw an object on screen.

BrushShape This is a modal dialog box module which contain routines for selecting the brush shape that is used to paint on screen.

G.6. The Options Menu Units

Options The Options module contains code to handle the Options menu items.

EditReport This is a modal dialog box which is used for setting the format of the simulation report.

NewScreen This unit is used for generating a new output screen used in running a simulation.

EditScreen This unit contains routines used for selecting and editing a predefined output screen in the model.

Histogram This is a modal dialog box module which is used for selecting a predefined histogram that is to be placed onto an output screen.

TimeSeries This is a modal dialog box module which is used for selecting a predefined time series that is to be placed onto an output screen.

G.7. The Run Menu Units

Run The Run module contains code to handle the Run menu items. The Run menu contains routines that control the running of the simulation model.

Language This is a hierarchical menu module which contains routines for selecting the language to be used when generating a three-phase simulation program.

Generate This is a modal dialog box which contains code allowing the user to specify the name of the generated file and to generate the simulation program.

CheckLogic This unit checks the logic of the model defined by the user and reports any errors that are detected.

MSummary This is a modal dialog box module which is used to present the simulation model in a textual format.

Distributions This unit contains all the functions that are used for evaluating the duration of an activity and the value of an attribute during a simulation run.

ScreenRun This is a modal dialog box module which sets the screen run mode on and allows the user to specify the duration, the run-in period, and the speed of the simulation run. The user can also specify a result file and a report file for the run.

VisualRun This is a modal dialog box module which sets the visual run mode on and allows the user to specify the duration, the run-in period, and the speed of the simulation run. The user can also specify a result file and a report file for the run.

TextRun This is a modal dialog box module which sets the text run mode

on and allows the user to specify the duration, the run-in period, and the speed of the simulation run. The user can also specify a result file and a report file for the run.

Show This unit allows the user to review the result file or the report file in a modal dialog box after the completion of a simulation run.

G.8. The Output Menu Units

Output The Output module contains code to handle the Output menu items. The Output menu contains all the user-specified data recording items, for examples, histograms and time series.

ShowReport The Show module invokes a modal dialog box in which the selected data recording item is displayed after the completion of a simulation run.

REFERENCES

Adelsberger, H.H. and F. Broeckx. 1987
Discrete Event Simulation and Operations Research.
The Society for Computer Simulation, San Diego, USA.

AppMaker. 1989
AppMaker The Application Generator.
Bowers Development Corporation, USA.

Apple Computer, Inc. 1985
68000 Development System User's Manual.
Addison Wesley.

Apple Computer, Inc. 1985-88
Inside Macintosh Volumn I - V.
Addison Wesley.

Apple Computer, Inc. 1987-89
Macintosh Programmer's Workshop Reference.
APDA, London.

Apple Computer, Inc. 1987-89
Macintosh Programmer's Workshop C Reference.
APDA, London.

Apple Computer, Inc. 1987-89
Macintosh Programmer's Workshop Pascal Reference.
APDA, London.

Apple Computer, Inc. 1988
HyperCard's Script Language Guide.
Addison Wesley.

Apple Computer, Inc. 1987
HyperCard's User Guide.
Addison Wesley.

Au, G. 1987
System Specification in Simulation using Graphics on the Apple Macintosh.
Unpublished M.Sc. Project Report, L.S.E.

Au, G. and R.J. Paul. 1989
" Graphical Simulation Model Specification based on Activity Cycle Diagrams. "
Accepted by Computers and Industrial Engineering.

Au, G. and R.J. Paul. 1989
" Flexible Simulation Model Specification Using a HyperCard Implementation of Activity Cycle Diagrams. "
CASM Research Report. Dept. of Stats, L.S.E.

Au, G. and R.J. Paul. 1990
" A Complete Graphical Discrete Event Simulation Modelling Environment. "
Paper presented at the Young OR Conference 1990, University of Warwick.

Balci, O. 1986
" Credibility Assessment of Simulation Results. "
Proc. 1986 Winter Simulation Conference : 38-43.

Balmer, D.W. 1985a
" Validation using Simulation. "
Paper presented at the 7th European Congress of Operational Research for EURO IV, Bologna (16-19 June).

Balmer, D.W. 1985b
" An Intelligent Environment for Simulation. "
Paper presented at the 7th European Congress of Operational Research for EURO IV, Bologna (16-19 June).

Balmer, D.W. 1986
" Statistical Analysis of Simulation Output. "
CASM Research Report, Dept. of Statistics, L.S.E.

Balmer, D.W. 1987a
" Software Support for Hierarchical Modelling. "

Paper presented at the Conference on Methodology and Validation
Orlando, USA (6-9 April).

Balmer, D.W. 1987b
" Polishing the Analysis of the Statistical Output of Comparative Simulation Experiments. "
Simulation 49 : 123-126.

Balmer, D.W. 1987c
" Hierarchical Modelling for Discrete Event Simulation. "
In the Proceedings of the 1987 United Kingdom Simulation Conference, Bangor.

Balmer, D.W. 1987d
" Modelling Styles and their Support in the CASM Environment. "
In the Proceedings of the 1987 Winter Simulation Conference.

Balmer, D.W. and R.J. Paul. 1986
" CASM - The Right Environment for Simulation. "
Journal of the Operational Research Society 37 : 443-452.

Barnette, C.C. 1986
" Simulation in Pascal with Micro-PASSIM. "
Proc. 1986 Winter Simulation Conference : 151-155, Washington DC, New Jersey.

Bell, P.C. 1985
" Visual Interactive Modelling in Operational Research : Successes and Opportunities. "
Journal of the Operational Research Society 36 : 975-982.

Bell, P.C. 1986
" Visual Interactive Modelling in 1986. "
In Recent Developments in Operational Research (V. Belton & R.M. O'Keefe Eds.)
Pergammon Press, Oxford, 1986

Bell, P.C. and R.M. O'Keefe. 1987
" Visual Interactive Simulation - History, Recent Developments, and Major Issues."
Simulation 49 : 109-116.

Birtwhistle, G.M. 1979
DEMOS - Discrete Event Modelling on SIMULA.
Macmillan, London.

Birtwhistle, G.M. 1985
AI, Graphics and Simulation.
The Society for Computer Simulation, San Diego, USA.

Borland International.
Turbo Pascal Macintosh.
Borland, USA.

Bryant, R.W. 1981
SIMPAS User Manual.
Technical Report, Computer Science Dept., University of Wisconsin-Madison.

Bush, Vannevar. 1945
" As we may think. "
Atlantic Monthly : 176, 101-108.

CACI. 1983
SIMSCRIPT II.5 User Manual.
CACL Ltd.

CACI. Ltd.
SIMFACTORY Simulation Package.

CAP Scientific Ltd.
SLAM II Simulation Package.

Claris UK Ltd.
ClarisCAD (CAD software).

Claris UK Ltd.
MacDraw II (graphics application).

Claris UK Ltd.
MacPaint 2.0 (graphics application).

Clementson, A.T. 1985
ECSL - Extended Control and Simulation Users Manual.
Cle. Com Ltd., Birmingham, England.

Chew, S.T. 1986
Program Generators For Discrete Event Digital Simulation Modeling.
Unpublished Ph.D. Thesis, University of London, England.

Crookes, J.G. 1987
" Generators, Generic Models and Methodology. "
Journal of the Operational Research Society 38 : 765-768.

Crookes, J.G., D.W. Balmer, S.T. Chew and R.J. Paul. 1986
" A Three Phase Simulation Systems written in Pascal. "
Journal of the Operational Research Society 37 : 603-618.

Davies, R.M. and R.M. O'Keele. 1989
Simulation Modelling with Pascal.
Prentice Hall, London.

Desktop Engineering Systems
ArchiCAD (CAD software).

Domingo, L.T. and R.J. Paul. 1990
" An Introduction to Simulation Specification Using Formal Methods. "
Paper presented at the Young OR Conference 1990, University of Warwick.

Doukidis, G.I. 1985
Discrete Event Simulation Model Formulation using Natural Language
Understanding Systems.
Unpublished Ph.D. Thesis, University of London, England.

Doukidis, G.I. 1987
" An Anthology on the Homogy of Simulation with Artificial Intelligence. "
Journal of the Operational Research Society 37 : 701-712.

Doukidis, G.I. and R.J. Paul. 1985

" Research into Expert Systems to Aid Simulation Model Formulation. "
Journal of the Operational Research Society 36 : 319-325.

Doukidis, G.I. and R.J. Paul. 1986
"Experiences in Automating the Formulation of Discrete Event Simulation Models."
in AI Applied to Simulation (E.J.H. Kerckhoffs, G.C. Vansteenkiste & B.P. Zeigler, Eds.), Simulation Series Vol. 18, no. 1 (February) : 79-90.
The Society for Comupter Simulation, San Diego, USA.

Doukidis, G.I. and R.J. Paul. 1987a
" ASPES : A Skeletal Pascal Expert System. "
in Expert Systems and Artificial Intelligence in Decision Support Systems (H.G. Sol et. al., Eds.), D. Reidel, Dordrecht, Holland. pp. 227-246.

Doukidis, G.I. and R.J. Paul. 1987b
" Artificial Intelligence Aids in Discrete Event Digital Simulation Modelling. "
IEE Proceedings, Vol. 134, Pt. D, no. 4 (July) : 278-286.

Doukidis, G.I. and R.J. Paul. 1988
" SIPDES : A Simulation Program Debugger using an Expert System. "
Accepted by Expert Systems with Applications, Vol.2.

El Sheikh, A.A.R. 1987
Simulation Modelling using a Relational Database Package.
Unpublished Ph.D. Thesis, University of London, England.

El Sheikh, A.A.R. and R.J. Paul. 1988a
" Discrete Event Simulation Modelling using a Relational Database Package. "
CASM Research Report, Dept. of Statistics, L.S.E. In preparation.

El Sheikh, A.A.R. and R.J. Paul. 1988b
" INGRESSIM : Simulation Modelling using the Relational Database Package INGRES. "
CASM Research Report, Dept. of Statistics, L.S.E. In preparation.

El Sheikh, A.A.R., R.J. Paul, A.S. Harding and D.W. Balmer. 1987
" A Microcomputer Based Simulation Study of a Port. "
Journal of the Operational Research Society 37 : 673-681.

Electronic Arts.
Studio/8 1.1 (graphics application).

Epsim Ltd.
EPSIM Simulation Package.
Epsim Ltd., Bristol, UK.

Fiddy, E., Bright, J.G. and Hurrion, R.D. 1981
" See-why : Interactive Simulation on the Screen. "
Proc. Institute of Mechanical Engineers, C293/81 : 167-172.

Flitman, A.M. and Hurrion R.D. 1987
" Linking Discrete-Event Simulation Models with Expert Systems. "
Journal of the Operational Research Society 38 : 723-733.

Goodman, D.H., D.W. Balmer and G.I. Doukidis. 1987
" Interfacing Expert Systems and Simulation for Job-Shop Production Scheduling."
in proceedings of the Third International Expert Systems Conference (June)
Learned Information Ltd., Oxford, England.

High Order Software Inc.
Use-it software.

Holder, R.D. and R.P. Gittins 1989
" The Effects of Warship and Replenishment Ship Attrition on War Arsenal Requirements. "
Journal of the Operational Research Society 40 : 155-166.

Hurrion, R.D. and Secker, R.J.R. 1978
" Visual Interactive Simulation, and aid to decision making. "
Omega, 6(5) : 419-426.

Hurrion, R.D. 1989
" Graphics and Interaction. "
in Computer Modelling for Discrete Simulation (M. Pidd, Ed.), Wiley, London.

Index Technology
Excelerator software.

Insight International Ltd.
GENETIK Simulation Package.

Istel Ltd.
WITNESS Simulation Package.
Istel Ltd., Worcs., UK.

Knaster, S. 1986
How to Write Macintosh Software.
Hayden Books.

Krantz, D. & J. Stanley. 1986
68000 Assembly Language Techniques for Building Programs.
Addison Wesley.

Ledgards, H. and A. Singer. 1986
Pascal for the Macintosh.
Addison Wesley.

Mak, J. and R.J. Paul. 1990
" Combining Discrete-Event Simulation and Systems Dynamics. "
Paper presented at the Young OR Conference 1990, Unversity of Warwick.

Manufacturing Management Ltd.
PROPHET Simulation Package.

Martin, J. & C. McClure 1985
Diagramming Techniques for Analysts and Programmers.

Mashhour, A. 1989
" Automated Simulation Program Generation using a Relational Database
Simulation System. "
Unpublished Ph.D. Thesis, University of London, England.

Mathewson, S.C. 1977
" A Programming Language for SIMON Simulation in Fortran. "
Imperial College, London.

Mathewson, S.C. 1985
" Simulation Program Generators : Code and Animation on a PC. "
Journal of Operational Research Society 36 : 538-539.

Mathewson, S.C. 1987
" Draft/Draw/SSIM - an Integrated Network based Toolkit for Simulation. "
U.K. Simulation Conference, Bangor.

Mathewson, S.C. 1989a
" The Implementation of Simulation Languages. "
in Computer Modelling for Discrete Simulation (M. Pidd, Ed.), Wiley, London.

Mathewson, S.C. 1989b
" Simulation Support Environments. "
in Computer Modelling for Discrete Simulation (M. Pidd, Ed.), Wiley, London.

McLeod, J. 1988
" Simulation : The Formulative Years. "
The Society for Computer Simulation International Journal.

Nelson, T. 1972
" As we will think. "
In Proceedings of Online 1972, Uxbridge : Brunel University.

O'Keefe, R.M. 1986
" Simulation with Pascal-Sim. "
Winter Simulation Conference 1986.

O'Keefe, R.M. and J.W. Roach. 1987
" Artificial Intelligence Approaches to Simulation. "
Journal of Operational Research Society 38 : 713-722.

P-E Inbucon Ltd.
HOCUS Simulation Package.
P-E Inbucon Ltd., Surrey, UK.

Paul, R.J. 1987
" A.I. and Stochastic Process Simulation. "

in Interactions in Artificial Intelligence and Statistical Methods, (B. Phelps Ed.)
Gower Technical Press : 85-98.

Paul, R.J. 1988a
" Simulation Model Formulation using an Intelligent Graphical Approach. "
CASM Research Report, Dept. of Statistics, L.S.E.

Paul R.J. 1988b
" The Three-Phase Discrete Event Modelling Approach. "
CASM Research Report, Dept. of Statistics, L.S.E.

Paul, R.J. 1988c
" Simulation Modelling : The CASM Project. "
Accepted by Journal of the Brazilian Operations Research Society.

Paul, R.J. 1989a
" Recent Developments in Simulation Modelling. "
Accepted by Journal of the Operational Research Society.

Paul, R.J. 1989b
" 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.

Paul, R.J. 1989c
" Artificial Intelligence and Simulation Modelling. "
In Computer Modelling for Discrete Simulation (M. Pidd, Ed.), Wiley, London.

Paul, R.J. 1989d
" Combining Artificial Intelligence and Simulation. "
In Computer Modelling for Discrete Simulation (M. Pidd, Ed.), Wiley, London.

Paul, R.J. and D.W. Balmer. 1989
Simulation Modelling.
In press. Chartwell-Bratt Student-Text Series

Paul, R.J. and S.T. Chew. 1987

" Simulation Modelling using an Interactive Simulation Program Generator. "
Journal of Operational Research Society 38 : 735-752.

Paul, R.J. and G.I. Doukidis. 1986
" Further Developments in the Use of Artificial Intelligence Techniques which Formulate Simulation Problems. "
Journal of the Operational Research Society 37 : 787-810.

Paul, R.J. and E. Saliby. 1988
" Adaptable Simulation Support Environments : a case study. "
CASM Research Report, Dept. of Statistics, L.S.E. In preparation.

Persona-TMC.
SuperPaint 2.0 (graphics application).

Pidd, M. 1984
" Computer Simulation for Operational Research in 1984. "
In Developments in Operational Research (R.W. Eglese and G.K. Rand, Eds.)
Pergamon Press, Oxford.

Pidd, M. 1988
Computer Simulation in Management Science. 2nd Edition.
Wiley, London.

Pidd, M. 1989
Computer Modelling for Discrete Simulation.
Wiley, London.

Pressman, R.S. 1987
Software Engineering : A Practitioner's Approach.
McGraw Hill, London.

Principal Distribution Ltd.
PixelPaint 2.0 (graphics application).

Pritsker, A.A.B. 1979
Introduction to Simulation and SLAM.
John Wiley & Sons, New York.

Pritsker, A.A.B. and P. Kiviat. 1969
Simulation with GASP II : A Fortran-based Simulation Language.
John Wiley & Sons, New York.

Saliby, E. and R.J. Paul. 1988
" How to Implement Descriptive Sampling in a Simulation Software System."
CASM Research Report, Dept. of Statistics, L.S.E. In preparation.

Seila, A.F. 1988
" SIMTOOLS : A Software Tool Kit for Discrete Computer Simulation in Pascal. "
Simulation 50 : 93-99.

Shafer, D. 1988
HyperTalk Programming.
Hayden Books.

Shriber T. 1974
Simulation Using GPSS.
Wiley, London.

Simsoft. 1986
PCModel Manual.
Simsoft, San José, California, USA.

Spinelli de Carvahlo, R. and J.G. Crookes. 1976
" Cellular Simulation. "
Journal of Operational Research Society 29 : 31-40.

Systems Designers Scientific.
SYSMOD Simulation Package.

Szymankiewicz, J., J. McDonald and K. Turner. 1988.
Solving Business Problems by Simulation. 2nd Edition.
McGraw Hill, London

Talyor, R.P. and Hurrion, R.D. 1988
" An Expert Advisor for Simulation Experimental Design and Analysis. "

Proc. Multi-Conference on Artificial Intelligence and Simulation, (T. Henson ed.)
SCS International, San Deigo, California.

The MacSerious Company.
Visual Interactive Programming (programming application).
The MacSerious Company.

Williams, T.M., R.P. Gittins & D.M. Burke. 1989
" Replenishment at Sea. "
Journal of Operational Research Society 40 : 881-887.