c/

ANISIM: AN ANIMATED INTERACTIVE SIMULATION MONITORING SYSTEM

by

WARD WALKER, JR.

B.A.(Honors), Washington State University, 1970

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the Department

of

COMPUTER SCIENCE

We accept this thesis as conforming to the

required standard

THE UNIVERSITY OF BRITISH COLUMBIA

April 1974

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver 8, Canada

Date May 13, 1974

ABSTRACT

An interactive system is described which allows for the graphic construction, simulation, and simultaneous animation of an arbitrary network of queues. A method is proposed and implemented for representing the events of a discrete simulation by a continuous animation on a graphics terminal. Techniques are presented for the display of parallel animation "sequences," and a non-trival mapping of simulation time into animation time is described which preserves the relative order and time relationships between events. The program implemented combines this animation facility with other simulation monitoring and control features. The usefulness of this type of approach is discussed with respect to computer-aided design applications, educational tools, and research tools. An interactive dialogue which makes use of the lightpen and a menu of commands is implemented for the construction and modification of the queuing network. Certain relevent aspects of man-machine interaction are discussed. Also, some prospects are considered for applying the animation techniques developed in this implementation to other discrete event processes.

TABLE OF CONTENTS

INTRODUCTION 1

0.1 Animating Simulations 1

0.2 Scope 2

0.3 Queuing Networks 4

0.4 Simulation States And Events 7

I. INTERACTIVE GRAPHICS FOR COMPUTER-AIDED MODELLING 11

1.1 Types Of Systems 11

1.2 ANISIM 13

1.3 Important Principles 14

1.4 Model Representation 15

II. ANIMATION TECHNIQUES 18

2.1 Computer Animation 18

2.2 Representing Events 19

2.3 Animation Time Frame 22

2.3.1 Types Of Sequences 22

2.3.2 Internal Cycles 28

2.3.3 Editing The Event List 30

2.4 The Display Process 34

2.4.1 The Display Buffer 34

2.4.2 Compiling Sequences Into Display Programs 35

2.4.3 Double Buffering 40

2.5 The Overall Visual Effect 43

III. INTERACTIVE FEATURES 47

3.1 Simulation Monitoring And Control 47

3.1.1 System Control 47
3.1.2 Simulation Monitoring 50
3.2 Model Design And Modification 57
3.2.1 Network Construction 57
3.2.2 Model Verification 65
IV UTILITY OF ANIMATION 70
4.1 Simulation Tool 70
4.2 Educational Tool 73
4.3 Research Tool 77
V CONCLUSIONS, PROSPECTS, AND EXTENSIONS 81
5.1 Analysis 81
5.2 Limitations 82
5.3 Extensions 85
5.4 Prospects For Further Work 88
BIBLIOGRAPHY 92
APPENDIX A -- PROGRAM DESIGN 94
APPENDIX B -- DATA STRUCTURE 104
APPENDIX C -- USER'S GUIDE 106

LIST OF FIGURES

1. A Simple Queuing Network 6
2. Modelling Abstractions 10
3. Animation Sequences For Display Of A Departure Event ...21
4. Simple Time Expansion 23
5. Mapping Events Into Sequences 23
6. The Internal Cycle 28
7. Display Program To Alter The State Of A Queue 37
8. Display Program Timer Words 39
9. Buffer Co-Ordination 42
10. Commands Available 49
11. Labelled Entities 52
12. Display Of Blocked Items 56
13. Menu For Network Design 59
14. Specifying Queues 62
15. Buffer Sketching 63
16. Dialogue For Specifying Service Times 66
17. Queuing Theory Comparison 76
18. Simple Deadlock 78
19. A Partially Deadlocked Network 79
20. Transaction Grouping 84
21. System Configuration 94
22. The NODE Record 104
23. Data Structure For Queues 106
24. The EVENT Record 107

ACKNOWLEDGEMENT

I would like to acknowledge Dhirendra Chheda and Miguel Alemparte for their significant part in the programming and formulation of the original system. I would also like to thank Dr. Doug Seeley for his help and guidance, and Drs. Jim Varah and Dean Uyeno for their suggestions. I would especially like to express my appreciation to Karen Hartley for her moral support and assistance in finishing this thesis.

INTRODUCTION

0.1 Animating Simulations

Discrete event simulation has become an important tool in the analysis and design of complex systems. Conclusions about the performance of a system can be drawn from the statistics produced by simulating a model of that system with various parameters and specifications. When analyzing the output of such a simulation, the modeller must first of all be aware of the inherent assumptions of the model, as well as the values of the parameters. Secondly, he may need to know more about certain characteristics of the model that may be masked by statistical averages or extremes. Determining just how to alter a model in order to improve its performance can be a difficult or even counterintuitive process. If only the modeller could "step inside" his model and "watch" it perform as the real system might perform.

This thesis describes the implementation of a system that brings the modeller to a closer understanding of his model by providing a graphical animation of its simulation. The animation facility is the main feature of a complete interactive package which allows an on-line graphical definition of the model and extensive monitoring of its simulation. Considerable emphasis has been placed on providing for a reasonably smooth and meaningful dialogue between the user and the system.

One particular use of this system, that of studying in general the behavior known as "deadlock", will be discussed, as well as applications to computer aided design and to computer animation as an educational tool. A general technique is developed for the mapping of simulation "events" into a set of co-ordinated animation "sequences" for display. Also, some implications are drawn concerning the use of similar animation techniques for monitoring other, perhaps real-time, processes. Baecker [1] sums up the utility of computer animation in visualizing dynamic phenomena of mathematics, science, and engineering:

> "The computer has proved particularly useful because of its ability to construct precise, mathematically determined images, because of its ability to simulate hypothetical worlds, because of its ability to expand or contract space and time, and because of its ability to portray complex spatial phenomena, particularly those in three dimensions."

0.2 Scope

When considering the objective of animating simulations, perhaps the ultimate goal would be to develop a system that could animate any arbitrary simulation program. This was not attempted, and is probably not even possible. Baecker and his students [1] are developing a variety of systematic techniques for representing computer processes with dynamic images. Their emphasis, so far, has been on animating computer programs, rather than processes which are described by simulation

programs. One conclusion they have already reached, is that it is impossible to build a system which could produce a good animation of any program in any language running on a specific machine. Instead, they want to build a variety of powerful special-purpose tools, each suited to the animation of a particular class of programs. An animation of a simulation must manipulate graphical symbols in such a way that they resemble the objects and processes being modelled. Even a very intelligent program could not create meaningful symbols and motions by scanning the code of a simulation program. After all, the model itself is only an abstraction of reality. The animation must create a visual image of a real situation; a task which requires information often not available from, or important to, the model. In fact, in the case of a discrete event simulation (described in Section 0.4), the animation must appropriately "fill in" the time between simulated events in order to provide a continuous display.

A lesser goal, therefore, might be to animate programs written in a specific simulation language, such as Simscript [21] or GPSS [9]. It may be possible to provide a limited set of pre-defined, or easily defined, graphic primitives (both symbols and motions), and a set of function calls to be inserted into the simulation where required to produce the desired animation effects. This method would be difficult to implement and would place an overwhelming burden on the user to create a well-defined program with all of the necessary information

available for the animation routines.

Thus, the system which has been implemented, hereafter referred to as ANISIM, does not attempt to animate an existing simulation, but contains its own special simulation program designed to handle a subclass of models known as queuing network models. This still involves a large variety of possible models, but the system is now able to help the user formulate a well-defined model at the graphics terminal while it creates the data structures necessary for the simulation and animation programs. It will be shown later, how the techniques used in ANISIM could be applied to certain other models or classes of models. The concept of a built-in simulation means that the user works with a command language rather than a programming language. The trade-off is between using a simple and convenient command language and having the ability to tailor the power of the simulation to handle specific needs.

0.3 Queuing Networks

Before proceeding to further discussion of ANISIM, it will be useful to describe what is meant by queuing networks and by simulation states and events.

A queuing network model consists basically of items travelling along logical paths of a network of "queues" and "servers". A queue represents a waiting line of items (or "transactions") trying to get into one of the servers of the

"queue/server system". A server represents a delay of the item occupying it before the item can move on to the next queue/server system or exit from the network. In classical queuing theory, the "movement" of an item between any two nodes in the network is assumed to occur instantaneously. Items enter the network from a "source" and leave the network by going to a "sink". Figure 1 shows a queue/server system with ten items, six of them waiting to be served and four being served (the small squares). The symbols in Figure 1 are those used by ANISIM and do not represent any queuing theory conventions. Certain fairly simple queuing networks can be "solved" analytically--that is, average queue lengths and waiting times can be found, mathematically, for the queuing system when it is in steady state [11]. Networks which can be solved in this manner are largely restricted to those with infinite queues and special constraints on arrival and service time distributions. Also, no information is available regarding the performance of the system during more transient or time-dependent states. When analytical solutions are inadequate or unavailable, simulation must be used to analyze the system. Like any other model, a queuing network is an abstraction of reality, although perhaps a more formal one than some. Thus any conclusions drawn about the behaviour of the queuing model can only be considered as approximations of the behavior of the real system being modelled.

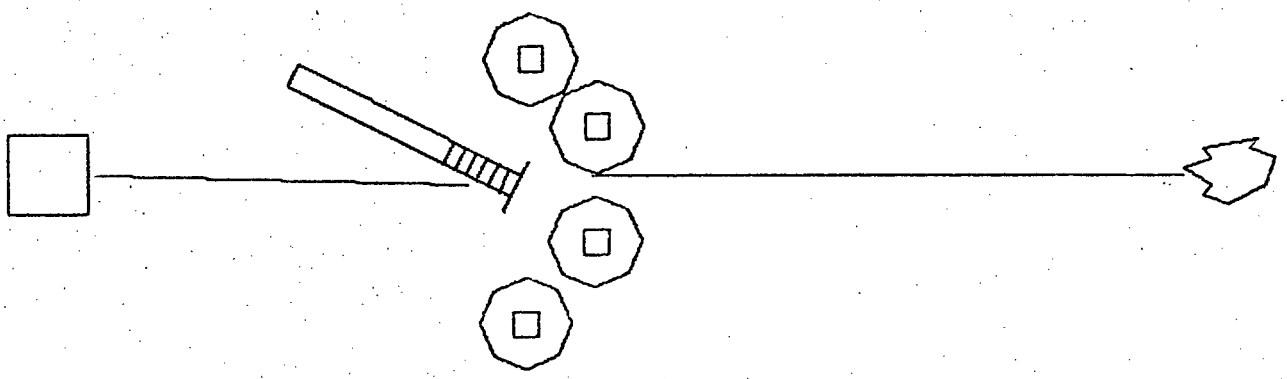


FIGURE 1: A Simple Queuing Network
From Left to Right: Source, Queue, Servers, and Sink.

0.4 Simulation States and Events

The type of simulation involved here is known as a discrete event simulation. This means that the model can be characterized by its "state" at any particular time and by a set of "events", or state changes, which occur at discrete time instants. Events can be exogenous (generated outside the system) or endogenous (generated within the system as a result of a previous event or state). There are a limited number of general types or "classes" of events that can occur, and each event is an instance of an event class.

Consider the following example of a discrete event model for a one-car ferry across a river. An observer on a nearby hill, sees an "arrival" as a car appearing at the last bend of the road and moving to the end of the lineup. He sees a "service" as the loading of the first car into the ferry plus the actual crossing, the unloading, and perhaps the return trip of the ferry. Finally he sees a "departure" as the car moving on the other bank until it disappears. A typical discrete event model of this situation is the so-called "single server first-in first-out queue." The state of the system is merely the number of cars in the lineup plus the one being served by the ferry. An "arrival" is the instantaneous exogenous event that adds one to the state and a "departure" is the instantaneous endogenous event that subtracts one from the state. The actual crossing or "service" is the abstract concept of wait or delay between two

departures. Completely irrelevant in this particular model abstraction are the actions defined by the movement of the cars or the ferry (or the fact that they are cars at all).

Of course, one could model a variety of situations using only these basic concepts. The modelling power may be increased by defining additional primitives while still maintaining the structure of a queuing network. For example, at an early stage of development it was decided that ANISIM should allow a "buffer" entity that imposes a finite common storage limit on two or more queues. (e.g. If the ferry terminal had two one-car ferries, each with their own waiting line, but the two lines had to share the same limited parking area.) If the number of features added to the system to increase modelling power were allowed to become very large, (approaching, say, the capabilities of GPSS [9]), then two things might happen. First, the programming system, which is large to begin with, may become too costly to use and awkward to maintain (as many large programming systems tend to be). More importantly, the graphical primitives would in all probability fail to keep up with the much wider variety of modelling abstractions, each of which require visual aids to restore some reality to the model being monitored.

One of the main objectives of ANISIM is to make it possible for a human at the display terminal to monitor a simulation and to do it as easily and accurately as possible. However, for

even a moderately complex system where the state of the system cannot be accurately represented by a simple number as in the ferry example, but by a vector with many components, monitoring the simulation is not a trivial task. What to display and how to display it becomes the main problem. Solutions range from the display of the state vector itself every time it changes (i.e. Display numbers or other symbols), to a sophisticated animation where sequences of moving elements are added as visual aids. ANISIM uses the latter, under the assumption that the necessary abstractions which were, indeed, so useful for analytical and computational purposes, hinder the process of monitoring the system (see figure 2).

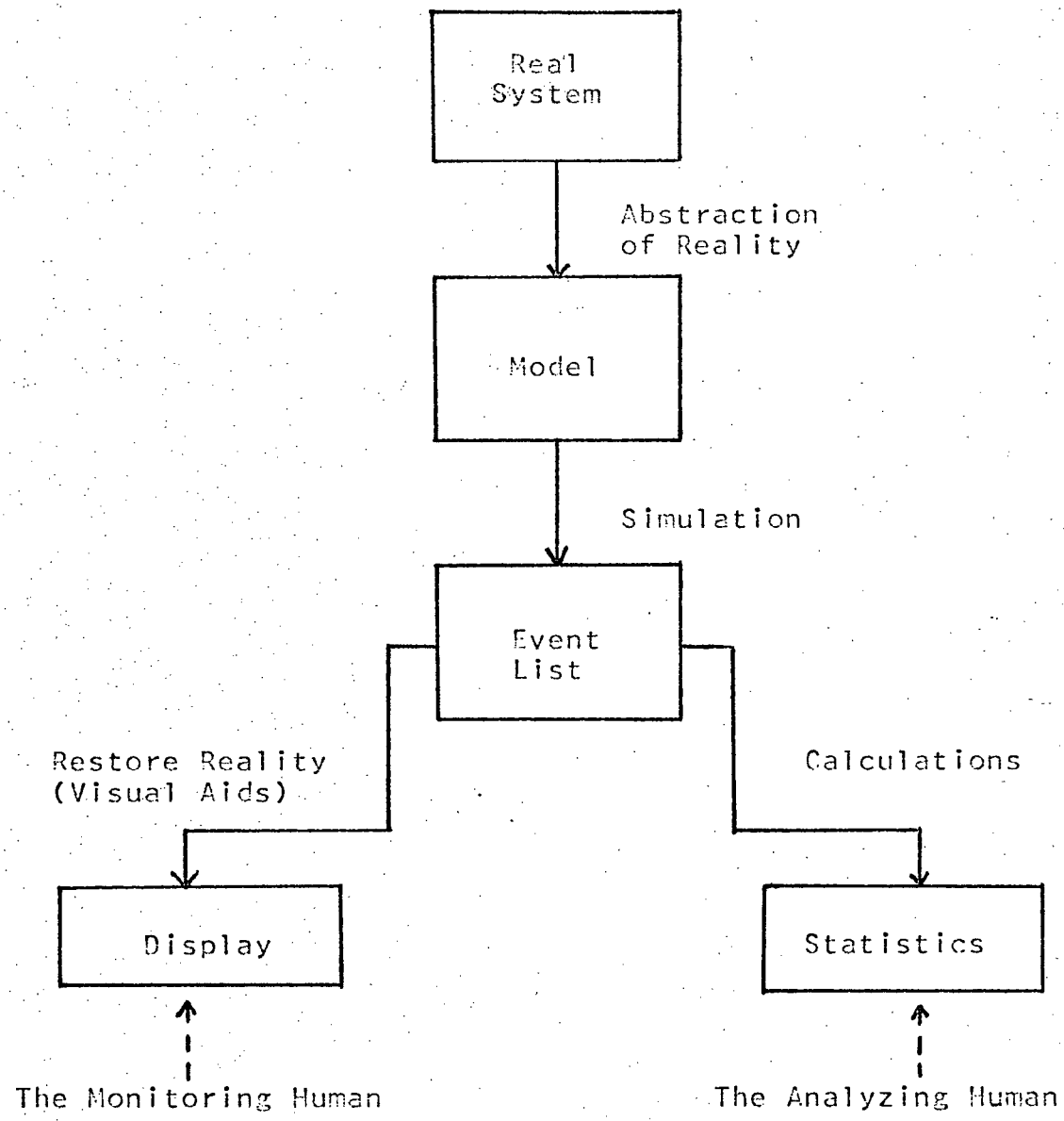


FIGURE 2: Modelling Abstractions

I. INTERACTIVE GRAPHICS FOR COMPUTER-AIDED MODELLING

1.1 Types of Systems

A number of graphics systems have been developed in the last few years that provide a capability to model a process or spatial configuration in order to learn more about it and, hopefully, improve its design. (Several of these are mentioned below. Further discussion and references may be found in Prince [18], Smith [23], and Newman and Sproull [16].) In many cases, the information available from such a model can be considerably enhanced by allowing the viewer to interact with the system that generates or displays the model, even if this interaction is simply, say, the rotation about an axis of a 3-D model of a molecule [17]. Some systems use the graphics terminal as a sophisticated sketchpad, while others rely on the terminal for input to, and output from, a non-graphics processing program. In almost all systems, a sufficient data structure must be created by the program to allow, at least, the saving and later restoring of models generated. Also, in even the simplest system, some thought must be given to the manner in which the user and the program communicate with each other (see Chapter III).

Consider the following two examples: a) an interactive program to design the spatial layout of integrated circuits [19], and b) and architecturally oriented program for generating

perspective drawings of three-dimensional polyhedra [14]. In each case, the user can quickly sketch preliminary "models", at a convenient scale, of something he may later construct or design in more detail. The processing program is confined mostly to graphical techniques which help the designer create a well-defined model and then observe as much as possible about the model that he has just created.

Another type of computer-aided modelling allows the user to see, and thus further comprehend, the results produced by a non-graphics processing program. For example, a number of interactive systems have been developed to aid in curve-fitting and other mathematical approximation techniques [15,23,12]. The user specifies input data, parameters and options, any of which he may wish to alter after viewing graphical representations of the output (or of intermediate steps). Such systems usually must also make available more detailed printed information for later study. ANISIM is partially this type of system, as it allows the user to quickly analyze the effects of simulation parameters by displaying the results of the simulation. The interactive features which facilitate this capability are also discussed in Chapter III.

On the other hand, there are cases when an interactive graphics capability provides for a more natural input medium to a non-graphics program in terms of speed, convenience, or possibly reliability [24]. For example, Forrester [8], has

found it necessary to describe complex dynamic simulation models graphically, in order to follow the inter-relationships between the variables, and then code the simulation in a programming language. Chheda [4] has developed an interactive system for creating this graphical representation of a dynamic model on a graphics terminal in such a way that the program statements for the actual simulation are automatically generated. Of course, this type of system may require the user to enter a large amount of detailed information at the graphics terminal. The quality of the dialogue between the system and the user therefore, becomes of paramount importance to the success of this approach to model design.

1.2 ANISIM

ANISIM is also, in part, this latter type of system. In Chapter III we see how the program guides the model builder through the necessary steps required to build a well-defined queuing network, while constructing the necessary data base for the simulation (and animation). The advantages of graphical input, in the case of network design, also include the immediate visual feedback that the modeller can utilize in order to help verify that the intended model is being properly created.

Thus we see that ANISIM combines the advantages of graphical input to a processing program with those of a graphical display of the results. In the case of simulation

however, it is sometimes desirable to display not just the representation of a "final state", but a detailed animation of the model as the simulation progresses. In other words, the processing of the simulation and the display of its events must occur almost simultaneously in order to enable the model designer to actually interact with the simulation (the "almost" is explained in Chapter II). A user of ANISIM, once he has created a network, may decide to monitor an animation of its simulation for a while, interrupt the simulation when he is not satisfied with the model's performance, and possibly display some statistics for further appraisal. He may then wish to edit one or more parameters or even the model structure, and resume monitoring the simulation (after reseting the statistics and clock, if necessary).

1.3 Important Principles

There are then, three main principles that seem to be present in most interactive graphics modelling systems: a) the system provides a means of testing model design that is faster, easier, more reliable, or otherwise more convenient than other possible methods, b) the system allows the modeller to proceed, with a relatively short turn-around time, through the cycle of design, study, and re-design in his attempt to optimize the model (or otherwise test variations), and c) the system makes use of the graphics terminal as an additional I/O medium to augment the information that may be presented by other media.

In fact, in some cases, the graphics terminal is the only reasonable device on which the above two principles can be observed (such as in the architecture program mentioned earlier). Furthermore, the interactive aspect of the system allows the user to selectively provide, display, or watch only that information which he feels to be most pertinent.

1.4 Model Representation

One problem associated with monitoring the animation of a queuing network model in order to aid in the design of a real system lies in the formulation of that system in the queuing network terms. The animation certainly aids in the understanding of the queuing network, but in some cases, the model is formulated in such a way that it does not visually resemble the real system that much. The formulation may be a very accurate approximation to the system, and the statistics produced may provide important information about the system, but the animation is of more value towards system design if the modeller can mentally translate what he sees into what it means in the real system. A simple example of this phenomenon is that of an object which travels between two service facilities (say, a ship between two ports). The transit time is important to the model and cannot easily be combined with the delay of the first service facility (e.g. the loading times at the first port are assumed to be exponentially distributed and the travel times are uniformly distributed). The queuing network model of this

situation thus requires one server for each service facility and a third server to approximate the delay associated with the route. Of course the animation then shows the object en route as a box inside a server symbol. (We shall see in the next chapter how the animation shows the box moving between servers merely to display, smoothly, state changes which are assumed to be instantaneous in the model.) In a fairly complex system, several occurrences of this problem may multiply the complexity of the network, further reducing the usefulness of the overall animation (although the modeller may still find specific portions of the animation instructive to watch).

There are two ways of combating this representation problem. A user of ANISIM, with some practice, soon learns to position the symbols of the network so that the animation most closely resembles, in a logical sense, the processes being modelled. With a little more practice he becomes more adept at thinking of things in terms of strict queuing network representations. Alternatively, ANISIM could be expanded to include optional features (such as transit times) that would allow a wider range of models to resemble the real systems. To a certain extent this can and should be done. The drawback is that the model definition phase (Chapter III) would become increasingly tedious for the user and it would be more difficult for the program to help insure that the model created was well-defined. Chapter V contains further discussion of potential extensions to ANISIM.

Additional comments on the utility of animated monitoring as a simulation tool are in Chapter IV.

II. ANIMATION TECHNIQUES

2.1 Computer Animation

Combining computer animation with simulation as a design or education tool is not new. This seems particularly true in fields such as physics, chemistry, electrical engineering, and medicine, where laboratory experiments are being replaced by graphic simulation systems. An example is a system which allows a medical student to observe the effects of different stimuli on a diagram of a living, moving organ [13]. However, most such systems involve continuous simulation rather than discrete event simulation. The model consists of a set of mathematical relationships which determine the value of each variable at every time unit as the simulation proceeds. Thus, at each time increment, the position, size, length, or whatever, of each entity in the animation is recomputed and the display is updated. There is no problem with parallel processes. On the other hand, an event-oriented model changes state by discrete steps at irregular time intervals. It is usually necessary to invent short, meaningful animation sequences to portray the state change of each component of the model. Most of this chapter describes how ANISIM accomplishes this task while maintaining the time relationships of the simulation.

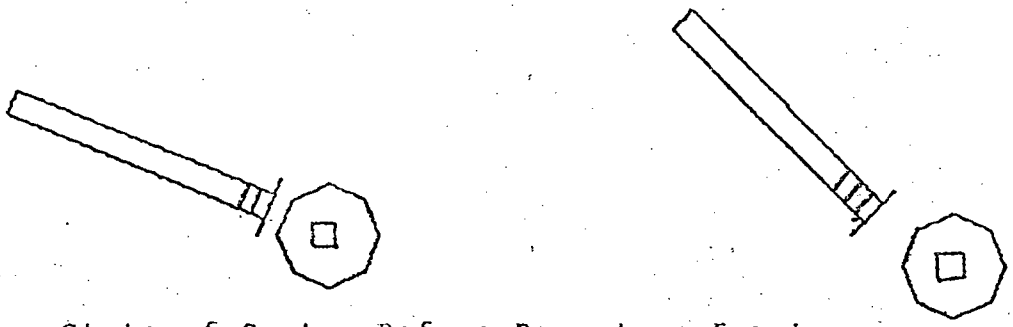
One example of an animated continuous simulation system that otherwise contains some striking parallels to ANISIM is the

DYNIS program at the University of Waterloo [20]. DYNIS simulates and displays the response of three-dimensional mechanical systems (composed of masses, springs, dampers, force drivers and position drivers). Like ANISIM, interactive control of the simulation is provided and a standard set of representative symbols is used. Both programs have a "first stage" which leads the user through a series of systematic decisions by the use of "menus" offering certain possible choices. The use of a closed set of simulation entities (and, therefore, animation primitives), in each case, allows this first stage to create a well-defined model without carrying out an overly tedious dialogue. Like ANISIM, the symbols used are abstractions representing a wide variety of possible real-world objects. On the surface, then, these systems perform in much the same way when used in an interactive, system design environment. The significant difference LIES IN THE INTERFACE BETWEEN THE SIMULATION AND THE ANIMATION.

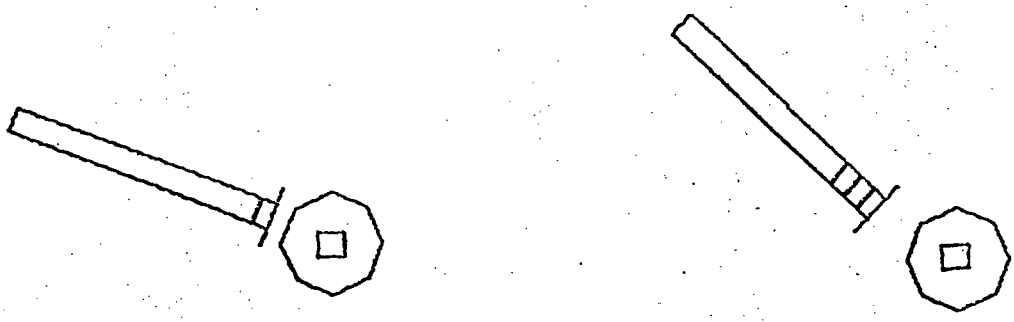
2.2 Representing Events

As mentioned earlier, simulation events, or state changes, are assumed to happen instantaneously at discrete points in time. However, such a change of state is normally a modelling abstraction which corresponds to a real-world process that has some relatively short duration. In the ferry terminal example of Chapter I, an arrival to the queue corresponds to a car driving up to the parking area. In order to graphically monitor

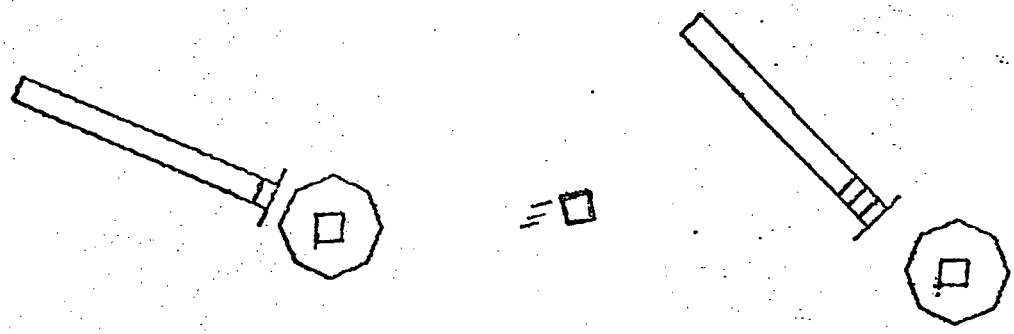
a ferry terminal simulation, every time an arrival event occurred, one should see a symbol of a car moving to a symbol of the waiting area. Likewise, three common visual aids for events in ANISIM are 1) an item moving from a source to a queue (arrival event), 2) an item moving from a server to a new queue (departure event), and 3) an item moving from a server to a sink (departure-from-system event). These, and other visual aids for events will be referred to henceforth as "animation sequences", although they are single components of the overall network animation. (We shall see in Section 2.4.2 how each animation sequence is compiled into a small display program.) Some sequences have no duration, such as the sequence that displays one more or one less item in a queue. Also, not all events require a sequence, such as a blocked departure which must be rescheduled again (i.e. the next queue or buffer is still full). Finally, a single event may trigger off more than one sequence. This is true of a departure event, which requires a sequence to update the state of the losing queue/server system, a moving sequence (with a duration), and a sequence to update the state of the gaining queue/server system (see Figure 3). If either queue is in a "buffer", then a fourth of fifth sequence may be required to update the bar graphs that show how full the buffers are. Additional sequences currently in ANISIM use arrows and blinking to identify blocked items and blocking queues or buffers.



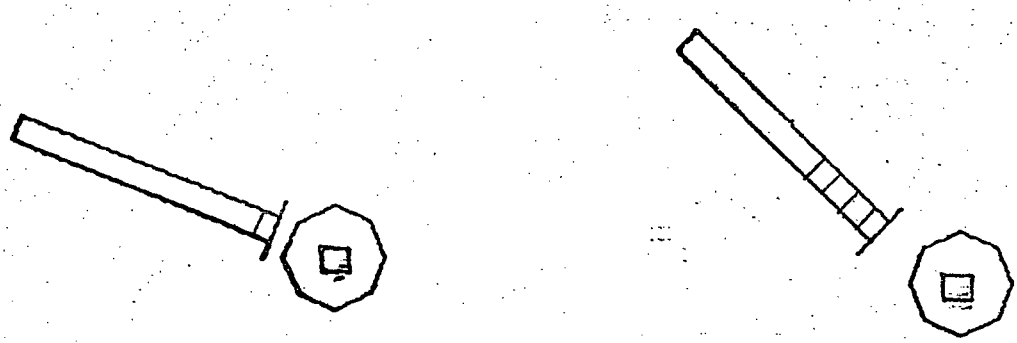
a. State of System Before Departure Event.



b. Animation Time = T: Instantaneous Sequence Updates Queue/Server System; Moving Sequence Begins.



c. Animation Time = T+50: Moving Sequence in Progress.



d. Animation Time = T+100: Moving Sequence Ends; Instantaneous Sequence Updates Second Queue/Server System.

FIGURE 3: Animation Sequences for Display of a Departure Event

2.3 Animation Time Frame

A requirement of animating a discrete simulation is to preserve the relative precedence of events and to preserve the relative time between events. Thus the question arises as to how and when to display sequences which we choose to give a positive duration. The mapping of simulation time into animation time requires some formalization.

2.3.1 Types of Sequences

One, perhaps trivial, way of adding sequences would be to expand the simulated time scale by the duration of the sequence every time an event occurs (see Figure 4).

This method, however, has one serious drawback which is that it stops or "binds" the ongoing simulation time every time an event occurs. Also, the display is completely sequential, leaving no provision for parallel sequences. For example, take the case where two cars approach the line up for the ferry. A small distance separates them, and they are going at the same speed. All our simulation model knows is that car "a" arrives at the queue at time "t(a)", and that car "b" arrives at time "t(b)", where the difference between the two times is a relatively small positive number. However, since the two "arrivals" are two simulation events in series, the two animation sequences will be in series. Car "b" will not begin moving towards the queue until after car "a" has arrived and

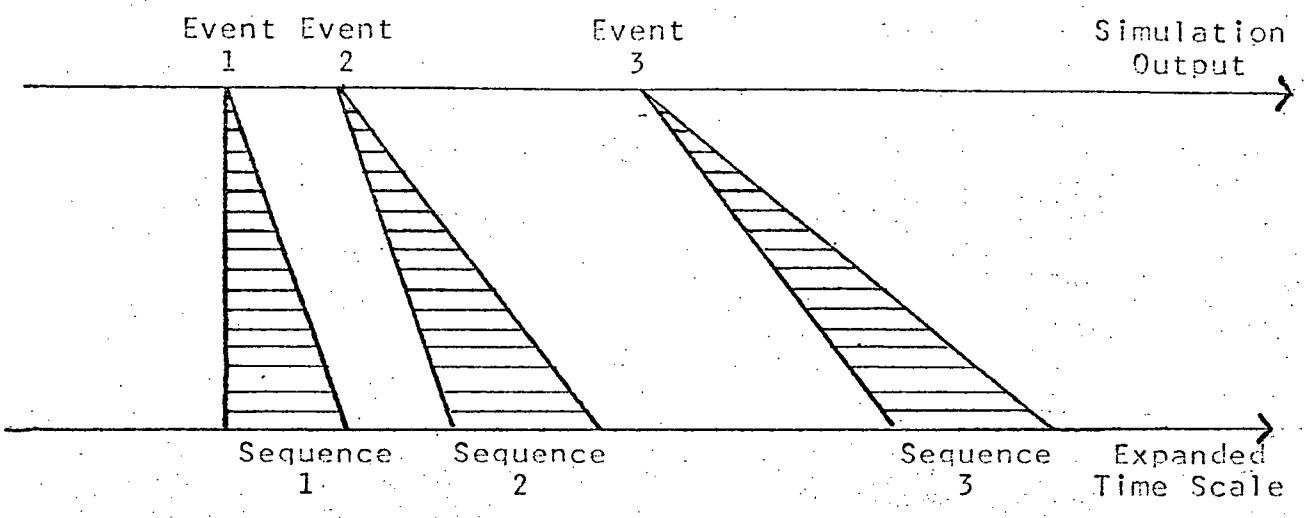


FIGURE 4: Simple Time Expansion

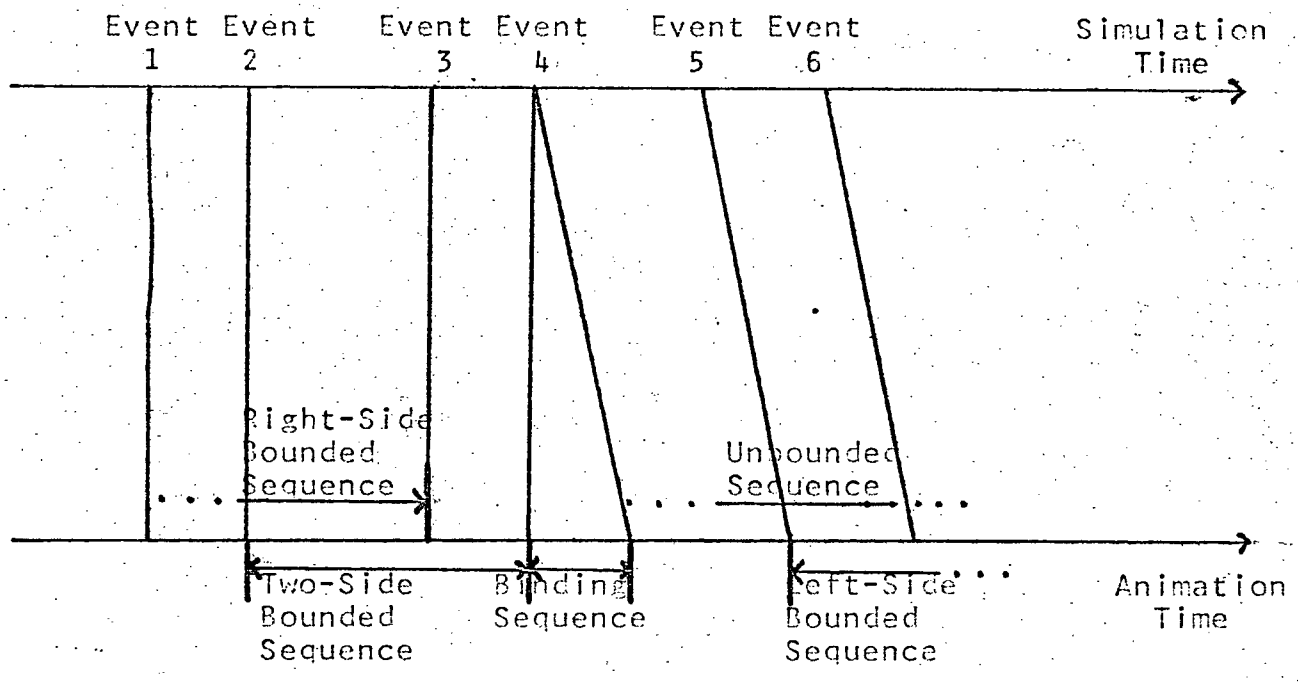


FIGURE 5: Mapping Events into Sequences

stayed in the queue for t(a)-t(b) time units. We are not conveying the parallelism of the two arrival sequences; something that is desirable for monitoring purposes. In general this approach does not restore much reality to the model.

The display of parallel sequences can, however, be achieved if a few simulation events are somehow buffered before displaying. These buffers of events, or "event lists", can thus be manipulated so that arrival sequences for example, can start their motion on the screen ahead of time and arrive at the correct spot in the queue exactly at the time the arrival event is to take place. Display sequences, which may overlap in time, can be compiled from one event list and displayed by the graphics computer while subsequent event lists are being generated. The problem, then, becomes one of editing the event lists so that the order and time relationships between events are preserved. To simplify this problem, it is useful to group display sequences into certain classes. For the following definitions, consider an animation sequence on a horizontal time axis (as in Figures 4 and 5). The left side and right side refer to the sequence's starting time and finish time, respectively.

Binding Sequence: This is the type discribed above where the simulation time is expanded by the insertion of the sequence. An example is a departure of an item from a server to a new queue.

Right-side Bounded Sequences: This type of sequence must end exactly with a simulation event, but it can be started at any time before that. The time scale is not expanded. An arrival event can be represented by a right-side bounded sequence.

Left-side Bounded Sequence: This type must start with an event, but can end at any time. An example is a departure to a sink.

Instantaneous Sequence: Not all "sequences" require a duration. Certain events can be represented partially, or sometimes completely, by merely changing the representation of the state. The arrival to a queue requires a right-side bounded sequence for the moving item and an instantaneous sequence (at the time the event happens) which changes the queue symbol to show one more item. A blocked-departure event is represented only by two instantaneous sequences which "switch on" the blinking of the full queue and the blinking of an arrow pointing from the blocked item to the full queue.

The sequence types defined above are used to represent events only. Two further types of animation sequences, which are not required in ANISIM at present, may also be useful in order to properly animate an event-oriented simulation.

Unbounded Sequence: A sequence (for example, an error or warning message) triggered by some process other than the simulation itself can be regarded as an unbounded sequence.

Two-side Bounded Sequence: This type of sequence must start and end with simulation events. It does not expand the time scale, but it can be used to further elaborate on a component of the state of the system. For example, if we wanted to animate the process that takes place during that abstract concept of a "service", we would have to co-ordinate the sequence with the event that started the service and with the event that terminated the service (e.g. a departure). In the ferry terminal simulation, a two-side bounded sequence could be used to illustrate the round trip of the ferry. Such a sequence would give the modeller no useful information and would probably provide more distraction than reality. An example of a more useful two-side bounded sequence would occur if ANISIM were extended to allow transit times between servers and queues. As pointed out in section 1.5, this extension would considerably ease the representation problem. As far as the simulation is concerned, the travelling item has entered into a delay or service of a specified duration (i.e. the state of that part of the system is static between the two events). But the viewer sees that service as a moving sequence which, of course, does not bind the other events (expand the time scale).

Figure 5 illustrates the mapping of simulation events into animation sequences. It should be noted that an instance of an event from an event class generates an instance of a sequence of a certain type. For example, an arrival event always generates a right-side bounded sequence. However, due particularly to

sequences which change the graphical representation of the state of the system, the animation routines require certain information about the system which was available in the data base at the time the simulation routine processed the event. For example, a sequence which displays the new state of a queue due to an arrival requires knowing what the state of the queue was before the arrival event occurred.

The simulation may provide this information in one of two ways: 1) Record, along with the event class, one or more subclass designators, or modifiers. For example, the state of a queue must be passed to the animation routines along with an event that causes an addition to or deletion from the queue. 2) Partition the event class into two or more distinct event classes. For example, in ANISIM a departure from a server which has been blocked must turn off the blinking of the arrow pointing from the no-longer-blocked item. It turned out to be more convenient to handle this kind of departure as a separate event class, distinct from a normal departure, due to the presence of the special, or "critical" state.

No clear generalization has become apparent as to which method requires less modification to the simulation when additional animation detail is desired.

Chapter V contains further discussion of sequence types as they relate to the modelling of processes other than queuing network simulations.

2.3.2 Internal Cycles

As explained in the previous section, it is not possible for the animation routines to process an event at the time it occurs in the simulation. Instead, the simulation routine must proceed until a sufficient number of events have happened, and then stop and pass control to an "Edit" routine. The Edit routine edits the event list produced by the simulation, calculating the animation time and duration for each event, and passes what is now called the "sequence list" to a third routine which actually compiles the display programs and sends them to the graphics computer over a high speed channel. This three-step process, called the "internal cycle", then repeats itself by returning control to the simulation routine (see figure 6). The internal cycle approach is feasible only because the display

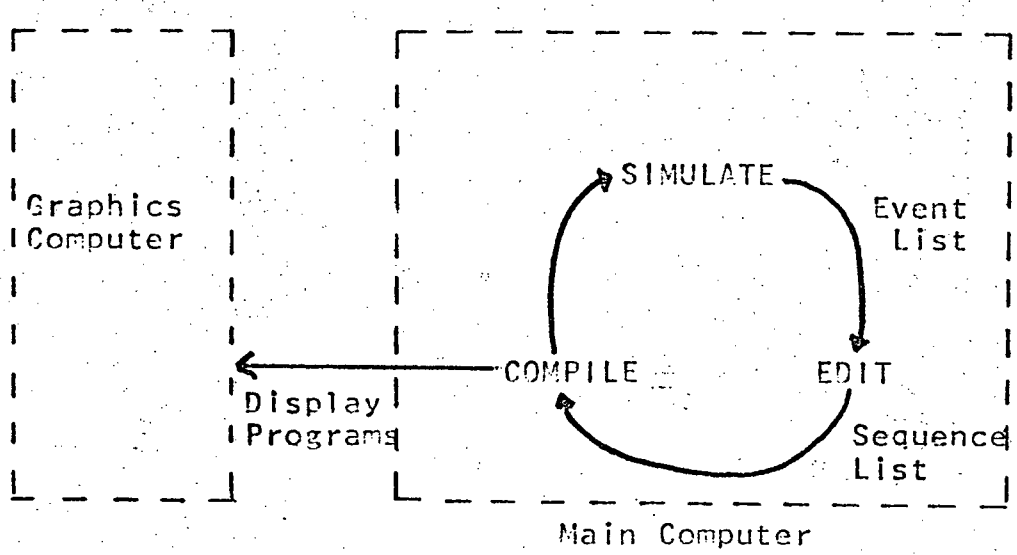


FIGURE 6: The Internal Cycle

of sequences by the graphics computer proceeds independently of the computation in the main computer (Section 2.4.2). Of course, the problem of co-ordinating the timing of the sequences in order to provide a smooth, accurate display is still a difficult one. We see below how this problem is related to the level of user interaction with the simulation.

There are two ways in which the simulation can terminate. An upper limit is imposed by the user on certain simulation variables such as the clock, the number of arrivals, and the number of terminations (departures from the system). If the user chooses to simulate without the animation, the simulation will proceed until it reaches one of these limits. At that point it will terminate, and a special routine is called to scan the data base and display the current state. However, if the user is monitoring an animation, he may wish to interrupt the simulation after he has seen enough. This interrupt is discovered by the program between internal cycles, suggesting that the number of events processed in one cycle be kept as small as possible. In this way, the user will achieve a response to his interrupt within a reasonable time and the lag between the simulation and the animation will be kept to a minimum.

The "length" of the internal cycle must not be too short, however, or the continuity of the display will be disrupted. Basically, the simulation must generate enough animation

sequences in order to provide a display which lasts long enough on the screen to allow the next cycle time to prepare the subsequent display. Several factors influence the length of the display. For a given internal cycle length, the duration of the display depends basically on the proportion of binding sequences generated, the user-controlled duration parameters (for each type of sequence), and a time conversion factor described in Section 2.4.2. User control of these parameters, and of the length of the internal cycle, is discussed in Section 3.1.2. (The simulation routine measures the cycle length by estimating the number of sequences that will be generated from the event list being produced.)

2.3.3 Editing the Event List

The Edit routine must accomplish two things. First, as mentioned earlier, it must create a sequence list with animation times from an event list with simulation times. Second, together with the Compile routine, it must coordinate the timing of the new sequence list with that of the previous sequence list and the following sequence list.

The first objective is a matter of analyzing each event in order to determine whether it is a bound of a sequence and to determine the type of the sequence. Whenever the event that bounds a right-side bounded sequence is found, the time of the event is moved up and a duration attribute is assigned in such a

way that the sequence will end exactly when it is supposed to, i.e. at the bound. Left-side bounded sequences only have a duration assigned. Two-side bounded sequences could be handled by deleting the right-side bound and assigning to the left-side bound a duration equal to the time delay between the two events. In this way, the event list is transformed into a sequence list where each sequence has two time attributes, its starting time and its duration. The Compile routine uses this information both to compile the moving sequences and also to compile any necessary instantaneous sequences which must occur at the beginning or at the end of a moving sequence.

Several problems are encountered in the process of editing the event list. The first one has to do with the insertion of binding sequences and their effect on the timing of the rest of the sequences. The solution requires the Edit routine to make two main passes through the event list. The first pass takes care of events which do not require binding sequences (as described above). When the second pass encounters an event that does require a binding sequence, two things must be done. The starting times of all sequences which begin after the start of the binding sequence must be incremented by the duration of the binding sequence. In other words, the time scale is expanded by the insertion of the binding sequence. Secondly, sequences which begin <u>before</u> the start of the binding sequence, and overlap it due to their duration attributes, must be processed as follows: right-side bounded sequences must have their

starting times incremented by the duration of the binding sequence, and two-side bounded sequences must have their durations extended by the duration of the binding sequence. This procedure allows the display of an instantaneous event as a moving sequence while preserving all time relationships between it and the other events.

A second problem with the editing process involves coordination between internal cycles. If an event which requires an right-side bounded sequence is found very near the beginning of the cycle, the edit procedure may assign it a starting time in the range already processed by the previous cycle. More seriously, the right-side bound of a two-side bounded sequence may very likely not be in the same cycle as the left-side bound. The general coordination problem is handled by dividing the edited sequences into two lists: one is composed of all of the sequences which start in the time range which has been completely resolved, and the other, referred to as the "tail" of the sequence list, or "tail list", is composed of those sequences starting in the time range which could be affected by the next cycle or starting after the left bound of an unresolved two-side bounded sequence. The first list is the sequence list that is sent on to the Compile routine and displayed. The tail list is saved and processed with the next cycle's event list. Thus it is actually possible for a cycle to produce no displayable sequences (i.e. all tail), increasing the chances of a visible lag in the animation. It has been found in

ANISIM however, that the cycle lengths and durations normally used produce a tail list of manageable porportions.

If any two-side bounded sequences were to be implemented, the problem may become more complex, depending on the nature of the sequences. If the durations of such sequences are known to be relatively short, then the edit process described above would be suitable. To prevent a two-side bounded sequence from requiring more than two cycles to be resolved, the cycle length would be specified long enough so that the sequences generated in any one cycle normally span a range of time that is longer than the duration of the two-side bounded sequence. If this duration is long, however, then some other technique is required. For example, it may be possible to break up the sequence into two or more sequences such that the first components can be displayed before the remainder of the time span is resolved. This method is very dependent on the particular graphics of the sequence, and would be generally awkward for the Edit and Compile routines to process. If the duration of the sequence is not known until the right-side bound is found, then chances are the sequence can be re-formulated to be a special representation of a state, which can be switched on and off with instantaneous sequences (e.g. blinking a blocked item until it is able to depart). In the case of ANISIM, the two-side bounded sequences required to implement transit times (discussed earlier) would probably be short enough that the entire sequence could be displayed in one cycle (i.e. the tail

is not resolved until the right-side bound is found).

2.4 The Display Process

Appendix A describes the system architecture on which ANISIM was implemented. Some general discussion of display methods used however, is necessary for fully understanding the animation technique.

2.4.1 The Display Buffer

Briefly, the processing program, written mostly in ALGOLW and run on an IBM 370/168, makes use of a basic graphics subroutine package [6] in order to communicate with a monitor program [14] in the Adage Graphics Computer. This graphics computer has a 6000 word buffer where a word may contain either a display vector, or a control instruction. The monitor continually scans this buffer to generate a display on the Adage Model 10 Graphics Display Scope. The significant feature of this graphics monitor is that it scans the entire display buffer at a fixed rate (40 scans per second), allowing the accurate control of the timing of animation sequences. In ANISIM, the instructions for the display of the network structure, and most potential state representations, are contained in a region at the top of the buffer, and thus are continuously scanned and displayed. The remainder of the buffer, during the animation phase, is free to contain the individual display programs

necessary for each animation sequence (see next section). The dynamic loading of these display programs into the buffer is described in Section 2.4.3. Note that this scheme avoids the necessity of sending a series of "frames" (as in movie frames) to the graphics computer, each containing an entire description of the display. Once the network has been constructed, only the descriptions of the changing components need be sent to the display buffer. Thus a much longer sequence of apparent frames may be displayed with considerably less time and space required for the transfer. Also, the capability of updating the display 40 times per second allows moving sequences of very high resolution.

2.4.2 Compiling Sequences into Display Programs

The display buffer may contain a combination of vector words and control words. One or more contiguous buffer words (properly formated by the basic graphics subroutines) may be sent to the Adage in any one transfer. The Compile routine thus constructs all the displayable sequences for a cycle in an array and sends this entire batch of small display programs to the Adage, where they are effectively executed in parallel, much in the manner described by Baecker [2].

Several buffer control words are critical in making this scheme work. For example, control of the scan in any one pass is achieved through relative and absolute jump instructions.

Thus, the number of items appearing in a queue is altered merely by changing a single relative jump instruction in the static region at the top of the buffer (see Figure 7). This type of instantaneous sequence can be achieved by an animation display program containing a buffer command word which moves the following buffer word (the new jump command) to a specified location in the buffer. It is desirable then to compile such a program that executes the move instruction at exactly the right time, and only once. Alternatively, a moving sequence consists of buffer words which must begin being scanned at the proper time and continue to be scanned for a specified number of scans before they are finally skipped again. (To move a box in a straight line, the vectors for the box are preceded by a set of two control words for each dimension. These control words consist of a command that adjusts, by a small amount each scan, the value of the following word, which will be the control word that displaces the vectors along an axis.)

Therefore, either type of display program must start with a set of "timer words" which are keyed to the scan. In addition, these timer words are all compiled with times relative to the starting time of the first sequence in that batch to begin displaying (i.e. the earliest sequence in the cycle). The two key control words used in the timers each have an integer counter which is tested at each scan and decremented if not already zero. In one case, the following word is skipped only if the counter is zero, and in the other case it is skipped only

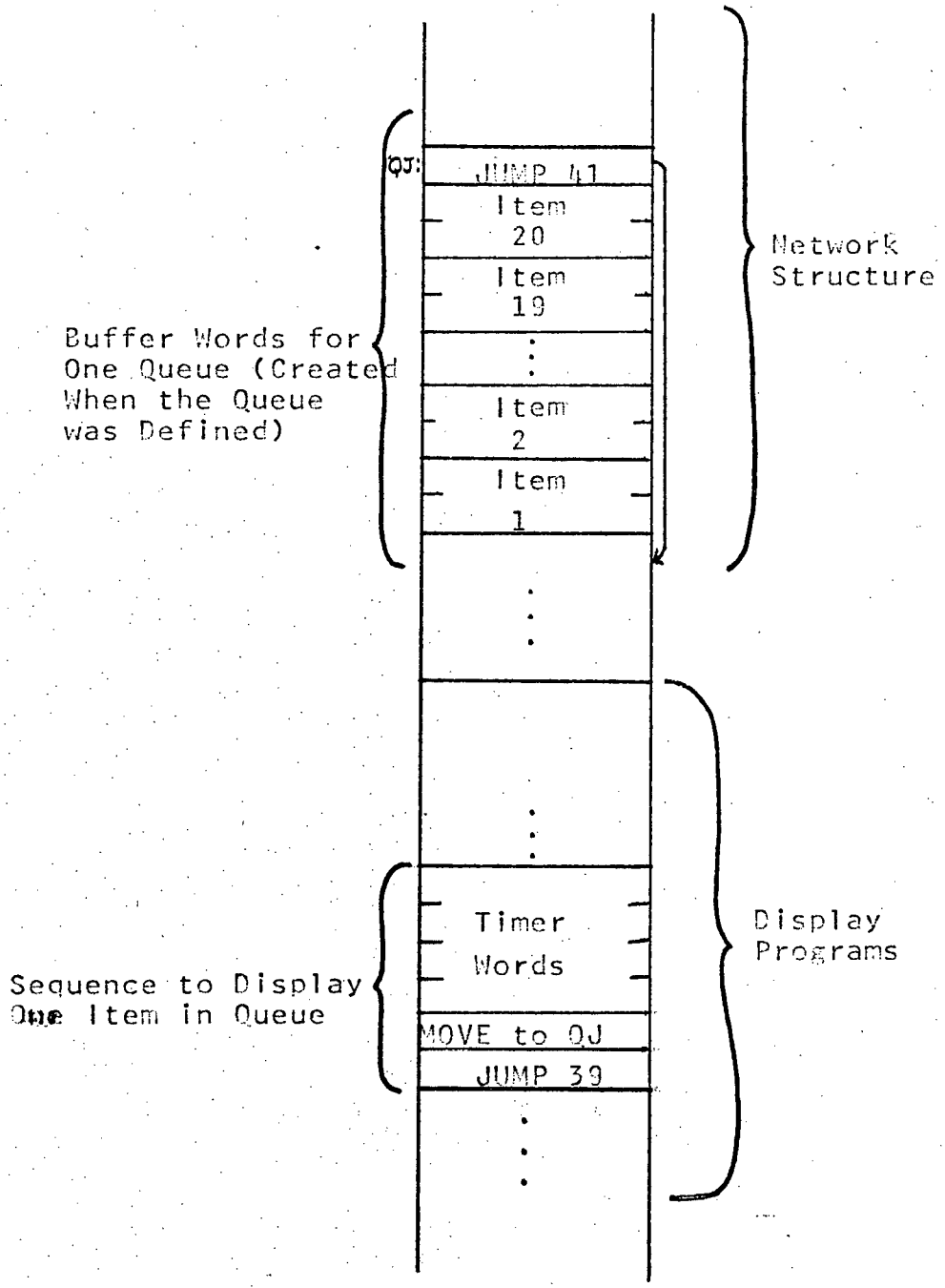
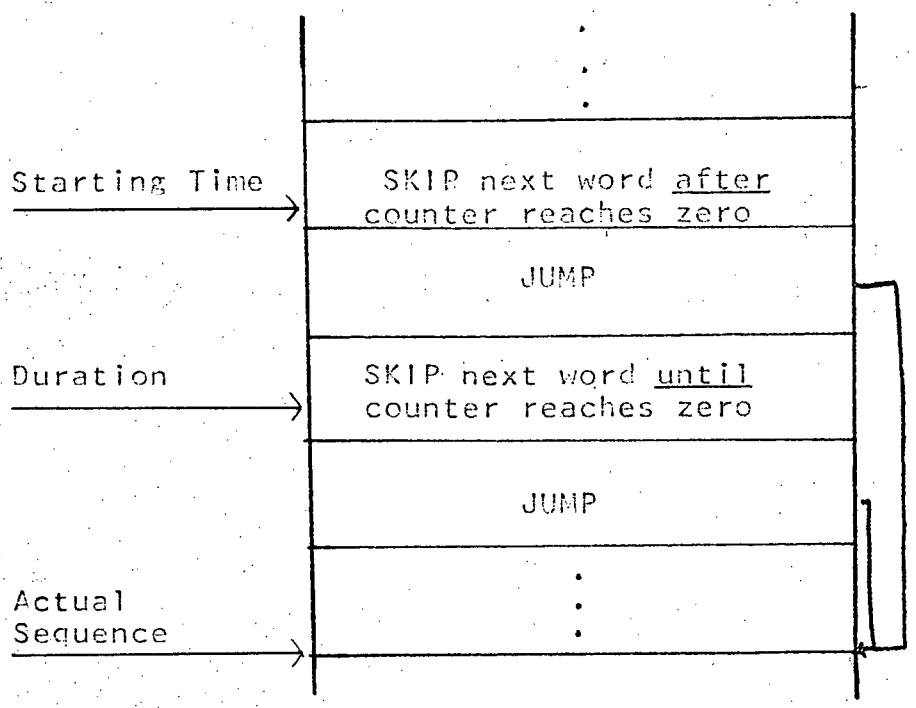


FIGURE 7: Display Program to Alter the State of a Queue

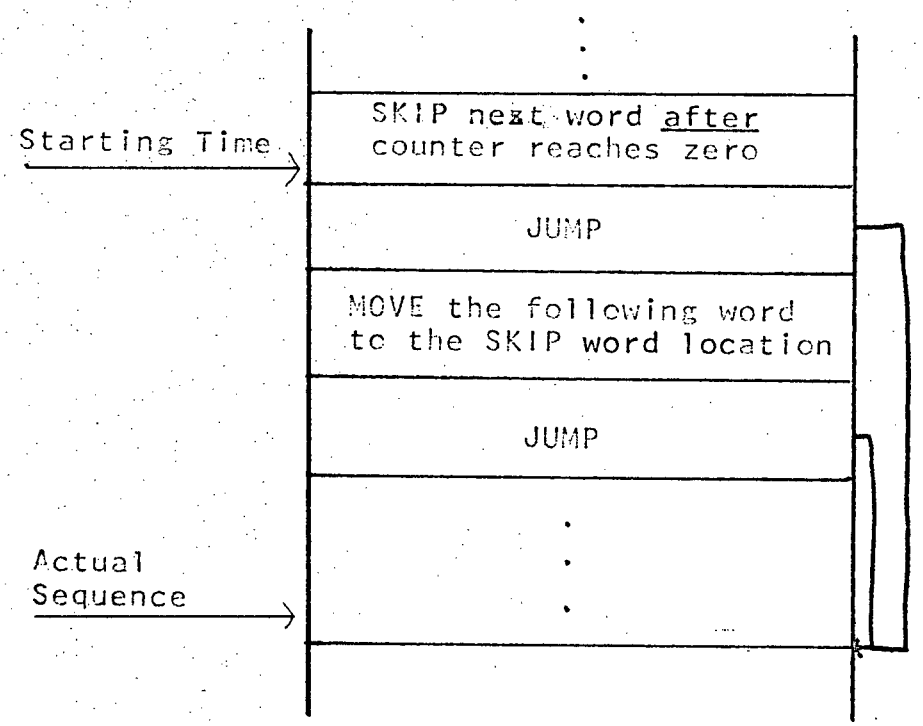
if the counter is not yet zero. In the former case, if the following word is a jump relative, the counter becomes the number of scans before a sequence begins (figure 8a). Likewise, in the latter case, the counter becomes the duration of the sequence. An instantaneous sequence (i.e. completed in one pass of the scanner), once it is allowed to execute, moves a jump relative on top of the first timer word in order to prohibit any subsequent execution (figure 8b).

The basic animation time unit, then, is always a single scan of the display buffer. This brings up one additional requirement of the Edit routine. When mapping simulation time into animation time, the speed of the display can be controlled by multiplying all times by a suitable "factor." This factor is available as a system variable for both user and program control. For example, if the factor is set to ten, then an interval of eight simulation time units will last 80 animation time units (scans), or two seconds.

The manner in which the Compile routine procedes through the sequence list, compiling each display program into an array, leads to an additional problem in the Edit routine. An earlier version of ANISIM, when editing the event list, actually moved the event records around in order to always maintain the list in animation time order. In this way, the tail could be easily determined and designated by a single pointer to the start of the tail. However, it is not uncommon for two instantaneous



a. Timer for a Sequence With a Duration.



b. Timer for an Instantaneous Sequence.

FIGURE 8: Display Program Timer Words

sequences to occur at the same animation time --- a fact which, in the general case, requires them to appear in the display buffer in exactly the same order as their associated events were processed in the simulation. For example, consider a queue/server system with a state of ten. The simulation processes both a departure from and an arrival to the system at the same time and in that order. The state of the system should remain ten. But, due to the right-side bounded sequence, the Compile routine will process the arrival first. The instantaneous sequence that changes the state to ten precedes in the buffer the sequence that changes the state to nine. Thus, after the scan, the state will appear to be nine. The only reasonable solution was to re-write the Edit routine in order that the simulation order of events is always maintained and only the time parameters are edited. Two passes through the list are required in order to identify and separate the tail list from the sequence list.

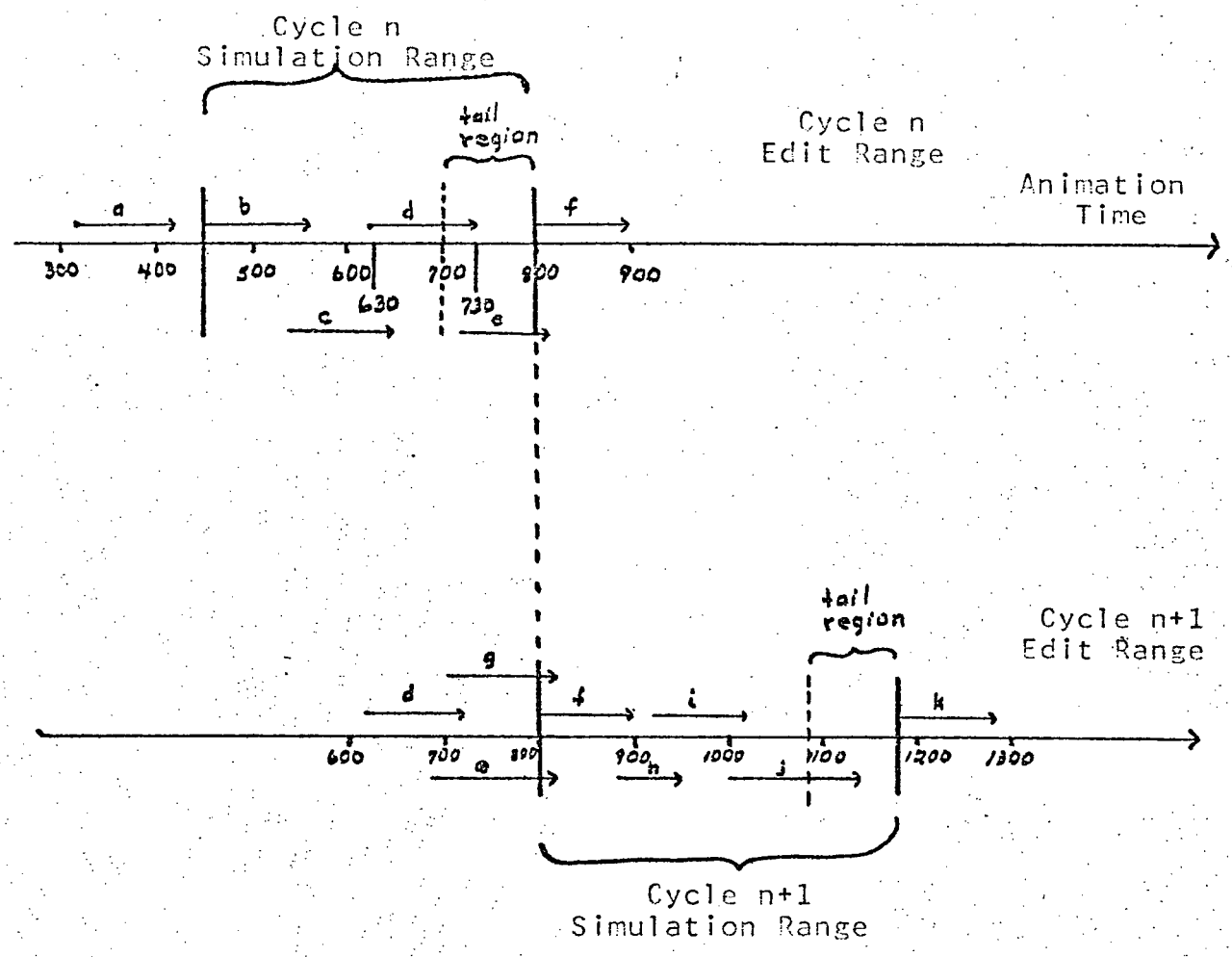
2.4.3 Double Buffering

Once the tail is determined, and the display sequences are compiled into an array, the problem remains of how to send the array to the Adage buffer and start displaying the new sequences in perfect co-ordination with the sequences of the previous cycle.

First of all, the available buffer space is divided into

two buffers of equal length. The basic scheme is to send up a new set of sequences as soon as the older of the two buffers finishes its display. Again, this scheme is possible due to a very useful buffer control word, called a "Notify", which works as follows. When the Compile routine has finished preparing the new sequences it issues a Read operation to the Adage computer. Each buffer in the Adage has one additional timer sequence in it that executes a Notify after the duration of that cycle has expired. The Notify causes the graphics monitor to issue an I/O interrupt which in effect cancels the pending Read from the 370. The program is then allowed to proceed with sending the new sequences up to the Adage. If, for some reason, the 370 had not yet issued a Read when the Notify is executed, the control words are set up so that the finished buffer will continue to be scanned and thus keep issuing a Notify until it is successful.

Now this newly loaded buffer must be co-ordinated with the sequences in the currently displaying buffer. Consider the simple example in figure 9. The internal cycle length is assumed to be five sequences. Suppose that cycle n is displaying in buffer one and cycle n+1 has just been loaded into buffer two. Initially, the scanner is unconditionally branching around buffer two. Note that when the tail of the sequence list of cycle n was determined, it was composed of all sequences whose <u>starting times</u> were in the region which might contain sequences from cycle n+1 (i.e. sequences e and f). Other sequences may start before that tail region but overlap with it



	Sequence List	Tail List
Cycle n	a,b,c	d,e,f
Cycle n+1	d,e,g,f,h,i	j,k

FIGURE 9: Buffer Co-ordination

due to their durations (sequence d). In other words, it is necessary for the two buffers to be displaying simultaneously for a short period. This is handled by actually including in the tail the last sequence which starts before the "real" tail (sequence d). The maximum length of cycle n's display is assigned as if sequence d were to be displayed in buffer one. But instead of that sequence, another sequence is compiled to be executed at that time (time 630 in figure 9). This special sequence removes the branch around buffer two and allows it to begin displaying. Of course, the sequence in buffer two with a relative starting time of zero is simply that pseudo-tail sequence (sequence d) from cycle n. Thus, buffer one and buffer two are displaying simultaneously from time 630 to time 730. At time 730, cycle n executes the Notify command, allowing the 370 to send up cycle n+2 to buffer one. The co-ordination between cycles is now complete. The fact that the old cycle expects the new cycle to be loaded and ready to start as soon as the branch is removed is, in fact, the reason why a lag in the animation will be seen if the old cycle has a very short display.

2.5 The Overall Visual Effect

It has been shown in this chapter why internal cycles are necessary and how the edit procedure transforms the event list into a sequence list and a tail list. The mapping of simulation time into animation time has been described, and the details of a carefully co-ordinated double buffering scheme have been

explained for the loading into the graphics computer, of display programs compiled from the sequence list. The techniques described in this chapter were somewhat painful to develop and implement, but they are actually quite logical. The classification of sequences and the edit procedures are general enough for other applications. Many of the display techniques are quite dependent on the specific hardware and software available, but make efficient use of these resources. The real test is in the quality of the animation. The overall visual effect is quite impressive. The animation proceeds in a smooth, continuous manner and the visual aids, for the most part, succeed in adding enough reality to make the simulation easy and informative to watch (see Chapter IV).

Some problems do still exist, however, with these techniques. Consider the sequences generated by ANISIM when a departure event occurs. The event is processed in Edit as a binding sequence, due to the desired move from the server to the new queue/server system. The Compile routine generates that move sequence and if the queue is not empty, it also generates an instantaneous sequence to display the new state of the queue. This method is an arbitrary simplification that was decided on early in the development of the program. The flow of the animation would look even smoother and more realistic if an additional binding sequence, a move from the queue to the server, were added such that it started at the same time as the departure move started. Two additional instantaneous sequences

would be required to switch off, then on, the symbol of the item in the server. In general, it seems that when one event generates two moving sequences, the Edit routine should make a separate copy of the event record so that both the sequences can be edited independently. In this case, a better solution would be to always assign a duration to the new sequence that is less than or equal to that of the departure sequence and let the Compile program use the information in the single event record to generate both binding sequences at the same time. In this way, the Edit routine can treat all event classes which generate binding sequences the same. This leaves the peculiarities of certain events to the Compile routine, which must examine the various parameters of the event record anyway.

The real problem is more basic than the above situation indicates. In a way, the animation does certainly distort the time frame of the simulation. Two events which occur within a short time of each other in the simulation may or may not do so in the animation, depending on how many binding sequences come between them. The time between every pair of (timewise) adjacent events remains accurate, as does the actual progression of states of the network. This type of "distortion" really seems to be insignificant, then, as long as one remembers that the beginning of a binding sequence is the "same time" as the end. It would be nice to try to reduce this distortion by allowing two binding sequences to have the same starting time if they correspond to the same event time in the simulation. It is

easy to find certain examples, however, of situations which require that the Edit routine increment the starting times even of binding sequences whose events are simultaneous with that of the binding sequence being processed.

III. INTERACTIVE FEATURES

An interactive modelling system is of little practical use unless special attention is paid to the design of the dialogue between the user and the system. According to Newman and Sproull [16], the three main qualities that the programmer should attempt to optimize are 1) simplicity of operation of the program, 2) consistency in the overall construction of the command language and error recovery, and 3) economy from both the user's and program's standpoint. The following discussion of these qualities, and other important features of the dialogue, distinguishes between the overall system and simulation control and the actual model construction and modification.

3.1 Simulation Monitoring and Control

3.1.1 System control

ANISIM provides a simple, but effective, command language for top level interaction with the system. All commands at this level are entered on the IBM 3270 Display Terminal which is located next to the graphics terminal. Each command is defined explicitly, rather than being implicit in the sequence of inputs. Furthermore, it is easy for the programmer to add new commands to the system, although no extensibility is provided to the user. The objective throughout the system has been to allow

the user maximum freedom of control of the sequence of operations, wherever feasible. This flexibility reduces the reliance on frustrating questions which must be answered by the user, but places more emphasis on error recovery when he attempts to enter inappropriate commands or data. Also, in the interest of simplicity, a minimum of information about the system or the simulation is displayed (on either terminal) unless otherwise requested.

In order to aid in achieving these objectives, the program must help keep the user aware of what operations are available and what responses are required. The HELP and MORHELP commands, shown in figure 10, list all available commands along with a brief reminder of their use. (MORHELP lists the less frequently used commands.) In addition, default values are used whenever applicable. This is useful in cases where the user does not yet know what value is appropriate (e.g. display parameters), and keeps the number of mandatory inputs to a minimum. The current value (and thus the default value) of a system or display parameter can be found by entering the appropriate command without specifying any arguments. For example, entering "CYCLE" will result in a print out of the current values of the four limits on the simulation. The command "CYCLE * * * 100" will change the limit on the number of terminations to 100 and print out the four values. The PRINT command allows a selection of 17 options for printing out full or specific details about the state and the statistics of the simulation. The TRACK command

```
# $run walk:anisim.o+dhir:camadd+agt:basic par=size=150k
# EXECUTION BEGINS


  ARE YOU USING THE ADAGE? TRUE OR FALSE
  false

  ENTER COMMAND OR HELP
  help
  BUILD     TO BUILD OR MODIFY THE ACTIVE NETWORK
  NEWNET    TO BUILD A NEW NETWORK
  EDITNET   TO USE BUILD FOR SMALL CHANGES TO A NETWORK
  GO  N     TO SIMULATE AND TO DISPLAY THE ACTIVE NETWORK
               FOR N TIME UNITS FROM CURRENT TIME
  FASTER    TO SPEED UP THE DISPLAY
  SLOWER    TO SLOW DOWN THE DISPLAY
  SCAN      TO INHIBIT MOVING SEQUENCES
  DESPEED   TO RESTORE THE DEFAULT SPEED
  NODISP    TO INHIBIT DISPLAY UNTIL NEXT GO COMMAND
  RESET     TO RESET SIMULATION CLOCK AND STATS
  SAVE      TO SAVE THE NETWORK
  RESTORE   TO RESTORE A SAVED NETWORK
  LABEL     TO DISPLAY LABELS AT EACH NODE
  CYCLE     TO CHANGE SIMULATION LIMITS
             #OF TIME UNITS, #OF GEN,#OF ENTRIES,#OF TERM
  PRINT I   TO DUMP RESULTS, FOR CODES SEE DOCUMENT
             20 GIVES STATE, 1 STATS, 13 PARAMS, 7 QSTATS, ETC
  MORHELP   ADDITIONAL COMMANDS
  STOP OR END  TO TERMINATE EXECUTION

  ENTER COMMAND OR HELP
  morhelp
  UNLABEL   TO REMOVE ALL NODE LABELS
  FACTOR     DEFAULT=10; INCREASE TO SLOW DISPLAY
                         DECREASE ONLY IF USE RESET
  INTERCY    DEFAULT=15; INCREASE WHEN DISPLAY IS FAST
  SCALE     TO CHANGE THE SCALE AND 'SCALER'
  DUR       TO CHANGE THE 4 SEQUENCE DURATIONS
  TRACK I   TO SET DEBUG FLAGS ON. 0=<I<7, SEE DOCUMENT
  PLOT S     TO GET A HARDCOPY OF ADAGE DISPLAY
                S IS MAXIMUM PLOT SIZE, IN INCHES
  STOP OR END  TO TERMINATE EXECUTION

  ENTER COMMAND OR HELP
  stop

  000.03 SECONDS IN EXECUTION
# EXECUTION TERMINATED
```

FIGURE 10: Commands Available

turns on one of six debug flags in the program, providing detailed traces of the simulation, edit, and compile routines. In the case of both PRINT and TRACK, an optional second argument can be used to route the output to a disk file of the user's choice.

Most top-level commands require no further response from the user. The exceptions to this are the BUILD, NEWNET, and EDITNET commands, which envoke an entirely new dialogue (section 3.2), and the SAVE, RESTORE, PRINT, and TRACK commands, which may require further clarification of file-handling operations. For example, if the user tells the program to "SAVE NET1", and the file NET1 already exists, then the program must ask the user if it is alright to empty the file and store the current network on it. The user may also return temporarily to the MTS operating system by issuing an attention interrupt. At this point he has full access to any disk files created by the program, as well as run-time statistics. The ANISIM program may then be re-entered, using the MTS $RESTART command.

3.1.2 Simulation Monitoring

Two of the most important advantages of an interactive graphics system are the fast turnaround and the immediate graphical display of complex information. Smith [22] lists as an equally important advantage the capability to allow the user to try "various attacks" on a particular problem during a single

session with the computer. With ANISIM, this can be interpreted in two ways. The user should be able to quickly simulate several versions of a model, and he should be able to view the results of a simulation in several different ways.

The first goal requires not only the capability to edit the model structure and parameters, but also quick access to the description of such information. It may be possible to display all of the details of the model on the screen, including rates, capacities, and routing data. This, however, would not be desirable, both from a simplicity and an economy point of view. It is instead desirable to keep a minimum of information on the screen and make other data available on demand. For example, for each symbol in the network, a label is created internally. At present, all statistics are printed on the 3270 terminal and the labels are used to reference specific simulation entities. The user, in order to interpret these statistics, can use the LABEL command to actually display the labels at each symbol (see figure 11). The labels are automatically removed during the building or animating of the model, to save space in the Adage buffer. Other information available on demand includes the rate and capacity parameters, obtained by the PRINT 13 command. The display of routing assignments poses some difficult problems and has not yet been attempted. The user should try to construct the network so that its graphical representation conveys much of the routing information. Section 3.2 describes an additional access to current parameters while actually editing the network.

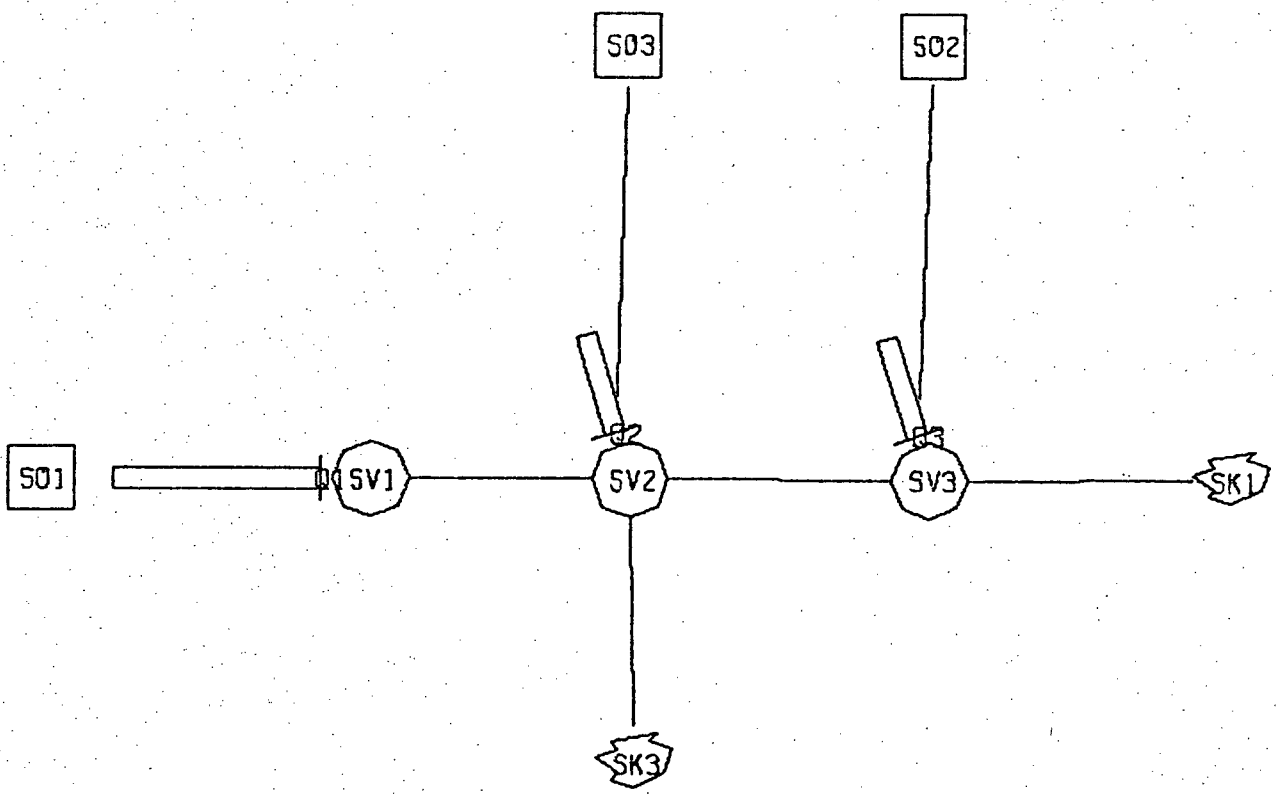


FIGURE 11: Labelled Entities

The second goal cited above--that of providing several ways of analysing the output of the simulation--has been met to a certain extent. A NODISP command, given before starting the simulation (with the GO command), allows the simulation to proceed quickly with no animation. This is useful when it is desired to watch the animation after the model has advanced to a more steady state. The user is not required in this case to watch the animation from the beginning. Whenever the simulation stops, the current state is displayed and the PRINT command may be used to find certain standard statistics such as the average and maximum queue lengths and buffer sizes, the average time-in-system of items between source-sink pairs, and the number of arrivals and terminations at each source and sink. Blocked items are an important aspect of queuing networks. An extended version of ANISIM might allow several options for the handling of blocked arrivals and departures. The existing protocol is that blocked arrivals are lost (they move halfway to the queue and disappear) and blocked departures remain in the server and attempt to depart again after waiting the user-defined "re-send time". Thus, statistics must also be available giving the number of lost arrivals and the number of blocked departures. The RESET command is used to set the clock to zero, reset all statistics, and restore the initial random number seeds.

If NODISP is not specified, then the animation will automatically accompany the simulation, providing additional valuable information about the model. Two areas of interaction

are essential in order to help the user monitor the animation: specifying how much to display, and how long to display it. From any particular state, the length of the simulation can be controlled in one of three ways. If the GO command is given without the argument, the simulation will proceed either until one of the limits in the CYCLE command is exceeded or until the user hits the attention interrupt. If the argument is given, then its value is the number of time units for which the simulation is to proceed, provided no cycle limit is exceeded or interrupt issued.

If the user wishes to monitor the simulation, not in detail, but to get a general feel for the progression of states of the model, then he may use the SCAN command. This command inhibits moving sequences by making all durations zero. Thus, the animation time frame is not expanded by binding sequences, resulting in a considerably faster display of events. (The factor is also decreased and the internal cycle is made longer, as discussed in Chapter II.) The SCAN mode is slightly less pleasing to watch, but provides a quick, clear way of observing the general trends of the model. It is especially useful in a model that would, otherwise, slowly build up to a congested state. The following special state representations are also envoked by instantaneous sequences and thus appear in SCAN mode: the blinking of a full queue or buffer due to an item attempting to get in, and the blinking of a blocked item with an arrow attached which points to the full queue or buffer (see figure

12).

Moving sequences, although artificial, provide valuable visual continuity between states. The user has access to the basic variables which control the speed of the display and of each sequence. Briefly, the DUR command controls the duration of each type of sequence, the FACTOR command controls the time conversion factor explained in section 2.4.2, and the INTERCY command controls the length of the internal cycle (i.e. the number of sequences generated each cycle). For example, the DUR command may be used to shorten the duration of binding sequences with respect to the other sequences. The inexperienced user should not have to use FACTOR to control the speed of the animation, for two reasons: 1) it is not always clear how to adjust the internal cycle length (INTERCY) to compensate for the change in speed, and 2) if the simulation is not RESET, the tail of the sequence list using the old factor will no longer be coordinated with the new sequence list. Thus, ANISIM provides a set of "speed" commands which make the necessary changes for a smooth transition to a faster or slower speed. Basically, a set of seven standard speeds are available, three slower than the default and three faster.

No provision has been made to store animation sequences on disk for later viewing without simulating, as in DYNIS [20]. This would probably be more expensive than re-simulating the saved model. One feature that can aid in the quick re-creation

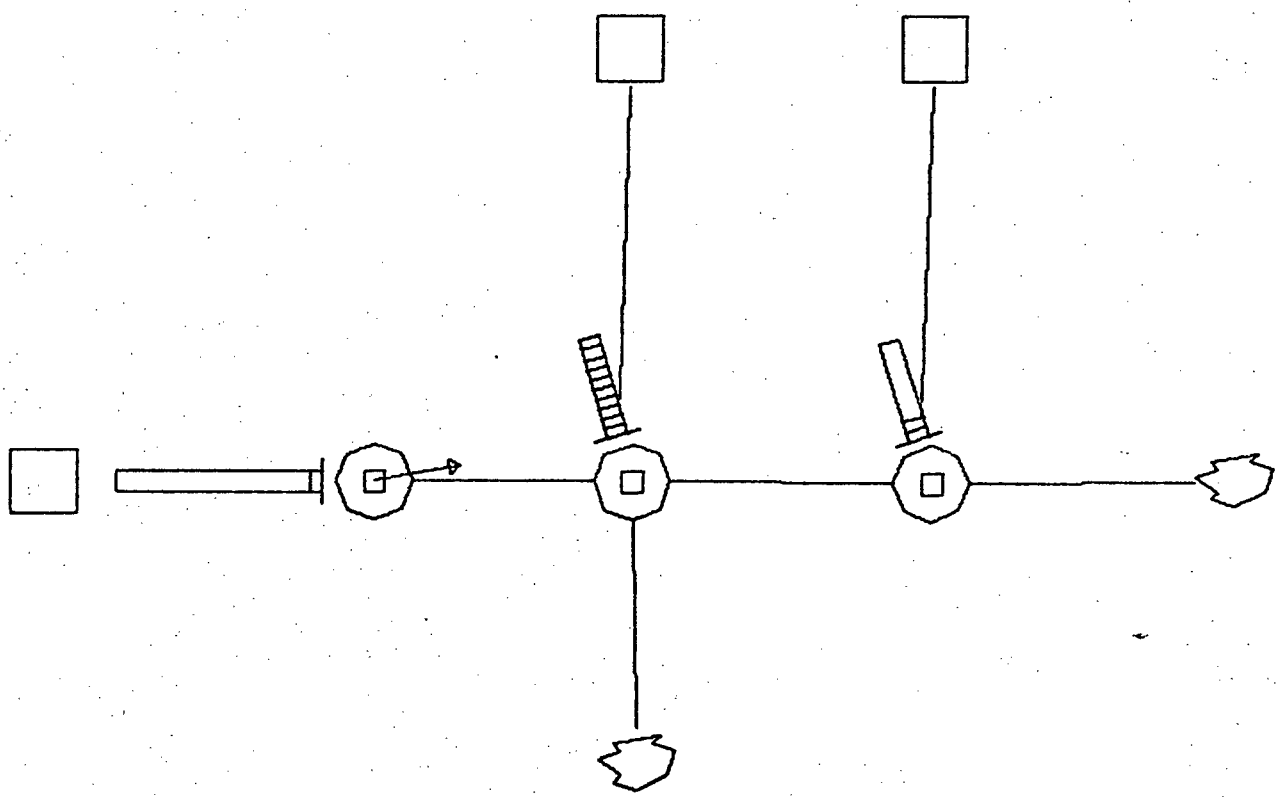


FIGURE 12: Display of Blocked Items

of a previously displayed sequence, as well as provide a powerful design tool, is the ability to save and restore a simulation state. In this way, the modeller can simulate forward from a given state several times, experimenting with various parameter values.

Extensions which would provide additional ways of analysing simulation results are discussed in Chapter V.

3.2 Model Design and Modification

The BUILD, NEWNET, and EDITNET commands all cause the program to enter a distinct phase with a dialogue of its own for constructing or modifying a network. All three commands envoke the same routine, although EDITNET appears somewhat different to the user, due to the setting of an "edit flag" in the program. Only NEWNET will destroy any existing active network (i.e. a RESTORE'd or newly constructed network) before proceeding. Otherwise, BUILD and NEWNET are identical.

3.2.1 Network construction

A considerable amount of information must be provided to the system in the network definition phase. It is here that the quality of the dialogue becomes very important. The task of entering input must not become awkward and tedious and a trade-off must be made between allowing the user to decide what to input or leading him through a fixed sequence of inputs to

ensure a properly defined mode. Also, it is important to provide feedback to the user in order that he may verify what has been entered.

Upon entry into the build phase, then, a list of possible commands, or "modes", are displayed on the graphics scope in the recommended order of use (when building a new network). This list (see figure 13) is called a "MENU", and the prompting message "MODE?" is actually blinking in order to indicate to the user that it is time to select a mode by pointing to its name with the lightpen. At this point the user still has freedom of control. He may enter or re-enter any mode at any time. Smith [22] calls this "interaction by anticipation", in the sense that all possible desires of a user are anticipated. These possibilities are presented as choices for the user to select rather than specify, thereby allowing a simple lightpen hit rather than requiring a correctly spelled alphabetic command. On any given entry into the build phase, the program helps the user remember which modes he has already entered by displaying these names at a different intensity. A further convenience is that the user may position the menu anywhere on the screen, using a pair of dials. The important thing is that the modes break up a lengthy task into easier, distinct subtasks.

In his book, Design of Man-Computer Dialogues , Martin [13] states that short-term memory is heavily utilized in complex problem solving and creation. He further states that humans

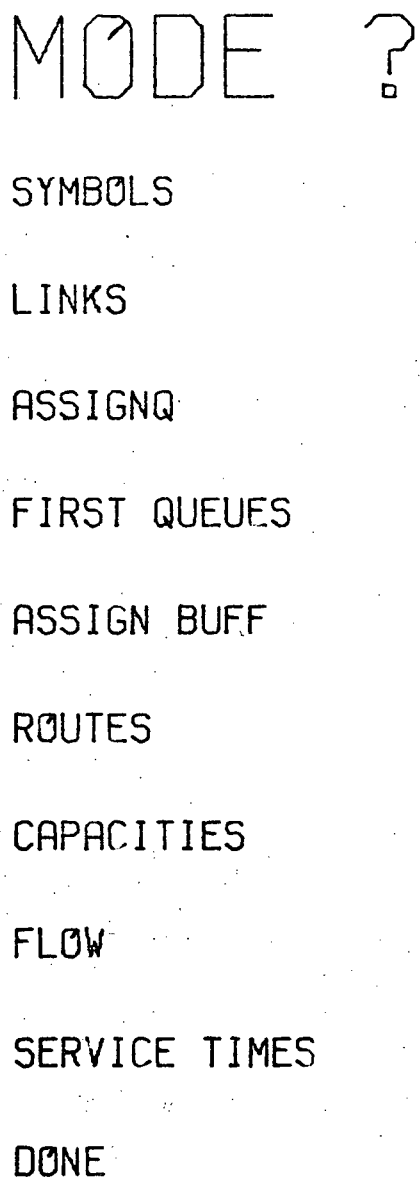


FIGURE 13: Menu for Network Design

tend to organize activity into "clumps" that can be easily completed. Thus, the modes in ANISIM are designed so that the user need only worry about a fairly simple or distinct aspect of the network at one time. In fact, when building a new network, the modes act as a series of Martin's "conversational checkpoints". The termination of a mode can be regarded as providing "mental closure"--i.e. the user can assure himself that he has completed that phase of the input. If he gets confused or makes a mistake, he need only re-enter the current mode and start over from that point.

When working in a particular mode, the user still might forget what is expected of him next, or even which mode he is in, if he is at all distracted. To avoid this situation, the program always prompts the user with some kind of mental cue whenever a response is expected. This prompt, often a one or two word message displayed at the bottom of the screen, serves only as a reminder rather than a detailed explanation. The blinking of symbols is used to signify which particular entities of the network are in question.

Proceeding through the menu, then, a typical dialogue would begin by entering the SYMBOLS mode. The menu disappears (as in all modes) and crosshairs appear. The crosshairs and a set of function buttons are used to position and select any one of the five available symbols. An entry (ALGOLW record) is created in the data base for the corresponding simulation entity. Two

symbols require interaction beyond the initial selection: queues require a second crosshair setting to define the orientation (angle) of the tail of the queue (see figure 14), and buffers are usually sketched, although a default is available. Figure 15 shows a buffer symbol being sketched around two queues which will later be assigned to the buffer entity. The lightpen may be used in SYMBOLS mode in order to delete a symbol or a "link" from the screen and the data base. A special function button terminates SYMBOLS mode, causing the re-appearance of the menu. LINKS mode allows the user to use the lightpen to connect pairs of symbols. These lines are visual aids to help portray the network structure, but no entry is made in the data base. Also, the intensity of the links is separate from the rest of the network and can be turned down by the twist of a dial. The mode is terminated by a lightpen hit on the prompting message.

The next four modes (ASSIGNQ, FIRST QUEUES, ASSIGN BUFF, and ROUTES) are necessary to make a series of assignments of simulation entities that define the structure of the network. In each case, both the programmer and the user distinguish these entities graphically, rather than trying to refer to them by labels or entering numbers in matrix form. The advantage of this approach is that the user can actually "see what he is doing." There is no problem of identifying labels or numbers with symbols. For example, in ASSIGNQ, the program goes through the list of servers that has been created and for each server, blinks the symbol and prompts the user to designate a queue.

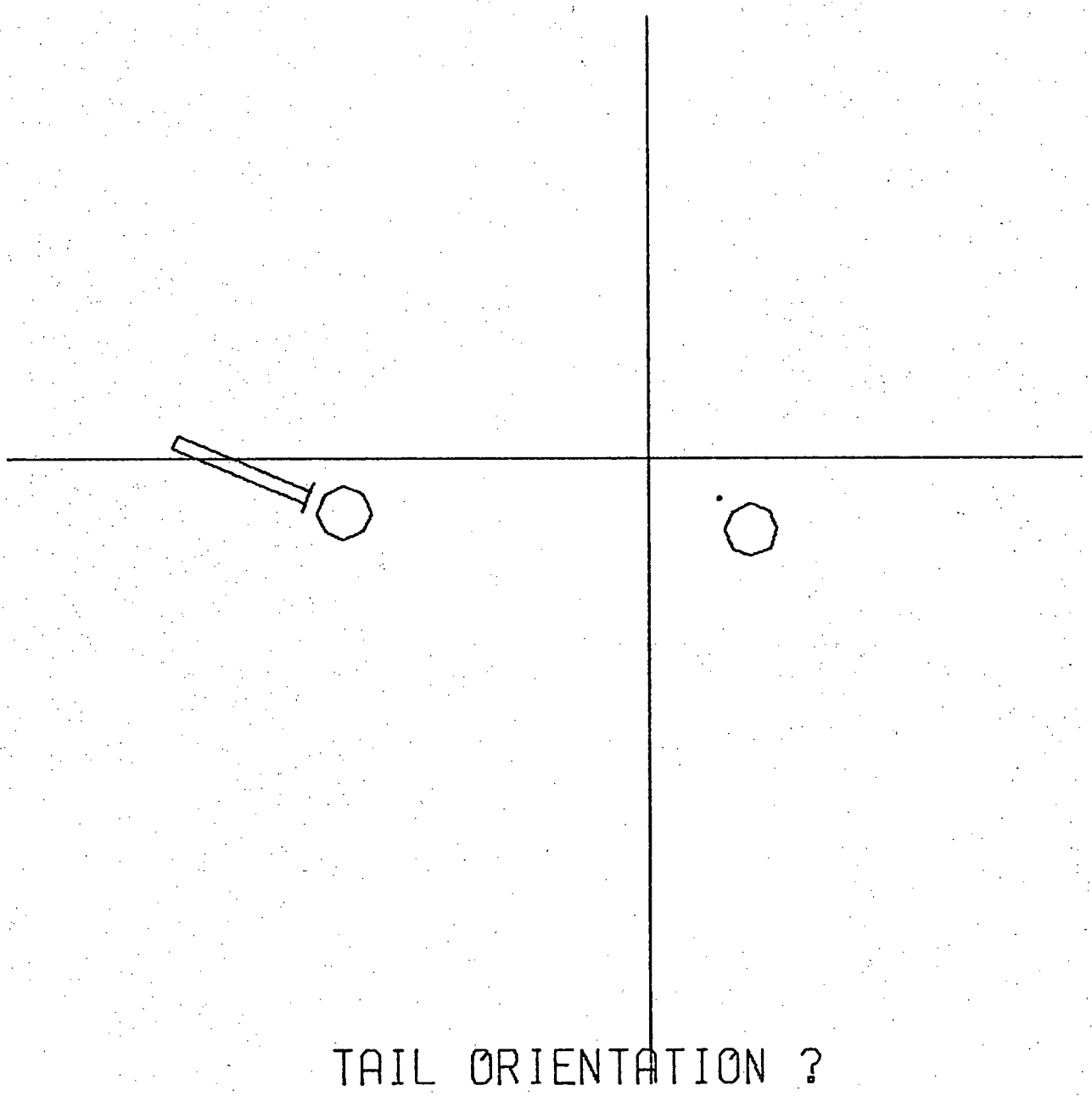


FIGURE 14: Specifying Queues

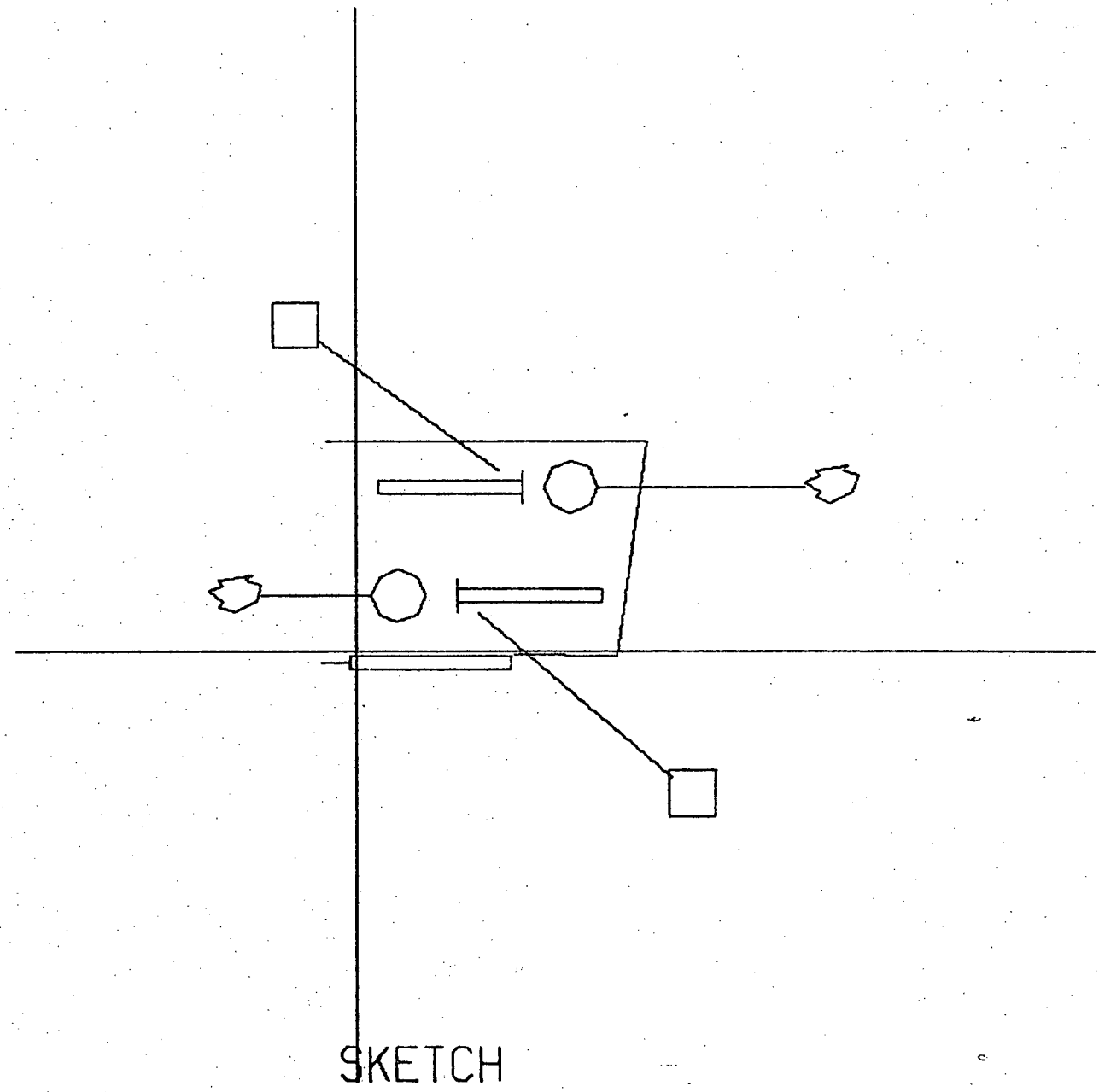


FIGURE 15: Buffer Sketching

The user need only point with the lightpen to the queue he wishes to be assigned to the blinking server. Clearly at this point it is advantageous to lead the user through all possible assignments in order that none are left out. This policy is followed in all the remaining modes as well. When the edit flag is set, however, it is assumed that all assignments have been made once already, and that the user wishes to change one or two assignments only. In this case, he must select with the lightpen a server, in the example of ASSIGNQ, before the program will prompt for a queue. He may proceed to select other servers or terminate the mode by hitting the prompting message.

In the same manner, FIRST QUEUES is used to assign a unique queue/server system to each source, and ASSIGN BUFF is used to assign buffers to queues. ANISIM currently provides one form of routing--a fixed routing scheme, although other schemes are possible (see Chapter V). Each item is assigned a destination sink when it is generated at the source. At any particular queue/server system, the next queue/server system (or sink) in the route is dependent only upon this destination. Thus, in ROUTES mode, the user fills an internal routing matrix by pointing to the next queue (or the sink) for each blinking queue-sink pair.

The final three modes require user input on the 3270 terminal. Newman and Sproull [16] advise that, in general, a single-device approach often leads to simpler, more easily

learned command languages. However, in this case it was decided that it would be easier for the user to switch from the lightpen to the keyboard than it would be to enter numerical data at the graphics terminal. Also, the 3270 screen offers the opportunity to easily display more detailed prompting messages, whereas the graphics approach would require valuable Adage buffer space. Figure 16 shows an example of the dialogue on the 3270. Of course, the program still designates entities by blinking symbols on the Adage display. The Adage scope prompt message of "TO 3270-->" reminds the user that his next response will be at the keyboard. CAPACITIES mode allows the assignment of queue and buffer capacities other than the defaults. In FLOW mode, for each source, the inter-arrival time distribution is specified (similar to the service times dialogue), and also the percentage of arrivals from that source that are destined for each sink must be given, in order to fill in an internal "flow" matrix.

In any mode requesting parameters, if the edit flag is on, the current value of the parameter is first printed on the 3270. If the edit flag is not on, the current value is printed only for those parameters which are initially assigned default values (e.g. queue capacities).

3.2.2 Model Verification

The network construction dialogue, as mentioned earlier,

```
FOR THE BLINKING SERVER, ASSIGN THE SERVICE
 DISTRIBUTION CODE:
1=EXPONENTIAL, 2=UNIFORM
1
ENTER MEAN  TIME ( > 1) FOR EXP. DSTBN
4
ENTER RE-SEND TIME FOR BLOCKED ITEMS FROM THIS NODE
  ELSE DEFAULT VALUE IS                4
2

DISTRIBUTION CODE (1 OR 2):
1
ENTER MEAN  TIME ( > 1) FOR EXP. DSTBN
7
ENTER RE-SEND TIME FOR BLOCKED ITEMS FROM THIS NODE
  ELSE DEFAULT VALUE IS                7
1

DISTRIBUTION CODE (1 OR 2):
2
ENTER MEAN TIME (>1) AND DEVIATION FOR UNIFORM DISTR.
4 5
DEVIATION IS TOO LARGE. TRY AGAIN.
ENTER MEAN TIME (>1) AND DEVIATION FOR UNIFORM DISTR.
4 2
ENTER RE-SEND TIME FOR BLOCKED ITEMS FROM THIS NODE
  ELSE DEFAULT VALUE IS                4

BACK TO ADAGE
```

FIGURE 16: Dialogue for Specifying Service Times

uses a menu to guide the user through the necessary steps of making a well-defined model, while allowing him sufficient freedom of control. In other words, it is still quite possible to create a network with missing assignments and parameters--especially after editing the network. This problem is attacked in several ways. First of all, when the user fails to hit the proper type of symbol with the lightpen, the message "NO ASSIGNMENT" appears on the screen briefly. Also, the user is asked to re-enter any numerical information that is not in the proper range such as a deviation that is greater than the mean for the uniform distribution. After exiting from the build phase and before a new simulation is allowed to begin, all servers and sources are checked to make sure then have been assigned a queue (in order to prevent a terminating error in the simulation program). The routing assignments are not checked at this time, but any undefined routes will be caught in the simulation. In that case, the item is sent to its destination sink, a message is printed on the 3270, and the simulation is allowed to continue. The DYNIS system [20] makes use of yet another method of reducing errors due to careless modification of the original model. Its editing phase is divided into two commands: REVISE and MODIFY. Their related functions in ANISIM might be as follows: REVISE would allow the user to add or delete symbols, analyze the effects of this change, and then prompt him to re-enter any modes necessary in order to make the network well-defined again. MODIFY on the other hand, would

allow only the altering of parameter values, none of which could make the network ill-defined. MODIFY would be more economical for the user (quick, safe changes; minimal dialogue), and also for the program (abbreviated menu; less checking and prompting.) This approach would in fact be feasible in ANISIM, using the basic BUILD-EDITNET structure.

Once we have insured that the model is well-defined, there still remains the possibility that a mistake has been made in the model design, i.e. the model built is not exactly the model intended. However, compared to any non-graphical simulation environment, this possibility is minimal. First of all, the modeller can see on the screen the effects of much of what he does. Also, he is less likely to specify a wrong entity when he can actually point to its representation on the screen. Furthermore, assignment and parameter information can be verified by using the PRINT command, and the animation itself should expose most unintended routing specifications. On the other hand, anyone who has written a non-trival simulation program in a language such as GPSS [9] knows that the structrue of the model may easily become obscured in the program statements and matrices. The input data is largely numerical and highly subject to mistakes as well.

A second side effect of a graphically described a model, especially in an interactive environment that helps the modeller create a well-defined model, is that the process of building the

network may very well force the modeller to re-evaluate his conception of the model. To some extent he may more quickly see the inadequacies of his initial formulation as a valid abstraction of the real-world situation.

Bracchi and Somalvico [3] emphasize that a software system for computer-aided design should provide both a strong computational capability and a flexible interaction with the designer during the design process. Likewise, ANISIM's usefulness as a simulation tool depends both on the power of the simulation routine and the quality of the user dialogue. The first part of this chapter described the techniques used to provide simple, yet flexible control over the simulation and the presentation of its results. Moreover, experience with users during the development of the preliminary versions of ANISIM has indicated that the quality of the network definition dialoque is critical to insuring that the user will have a successful session with the system. It is this phase where a large amount of information must be supplied to the system by a person who may be nervous, confused, or intimidated by conversations with machines. If the dialogue is awkward or otherwise inadequate, the user is not free to concentrate on such things as evaluating the conceptual validity of his model or verifying the accuracy of the network he is building. There will always be room for improvements in the dialogue, but the current version of ANISIM seems to meet most of the criteria for a smooth and effective interaction.

IV UTILITY OF ANIMATION

One of the points that Martin [13] makes about graphic systems, is that the use of moving or changing images can clarify certain ideas. Animation, like color, is a type of encoding that can increase the amount of information the limited human mind can grasp and ponder at one time. A reasonable question to ask, then, is when and how should we use animation in order to best take advantage of this capability.

4.1 Simulation Tool

Traditional simulation programs generally produce output in the form of statistical averages, variances, maxima, etc. They are generally expensive to execute and are often run in a batch environment. Model design and testing is done by analyzing the statistics, altering the model or its parameters, re-compiling if necessary, and then running the program again. Clearly, an interactive simulation program could improve this turn-around time by producing results quickly (while the modeller still remembers what he was trying to do) and by providing convenient means for altering the model. This is true, provided that the modeller can also analyze the simulation output quickly. If he is looking for the changes that appear in a few simple statistics, the improved turn-around may be sufficient for his purpose. However, if he is trying to understand the significance of the statistics with respect to a fairly complex

model, the program must provide him with additional aids.

One such aid available in an interactive environment is a monitoring capability. The modeller not only gets the final results, but he sees some sort of representation of the state of the model while the simulation is progressing. This may allow the modeller to quickly zero-in on the time range or parameter range that he is interested in, as well as providing information about the starting conditions and other transient effects. Alos, monitoring is likely to provide more information about the behavior of the model when it reaches certain critical states.

One simulation monitor system that has proven quite useful for ecological models is SIMCON--a Simulation Control Command Language [10]. SIMCON allows the user to plot during execution selected variables of a simulation program written in FORTRAN. He may interrupt the simulation and display or alter variables, and he may restart the simulation from several states. Thus, in many cases, a model can be adequately represented, for monitoring purposes, by line plots of the levels of certain entities (such as populations). In other cases, however, the real understanding of the model lies not in monitoring the value of a variable (or the length of a queue), but monitoring a process or relationship that determines the value of the variable. An example of a continuous simulation model where this would be true might be a study of changing boundaries between several territorial populations. The simulation

statistics may begin to make sense if the modeller can actually see on a graphics screen a population being crowded out of existance by neighboring populations. Similarly, a queue that becomes too long may be analyzed by monitoring an animation showing just where the items are arriving from and where (and when) they are going next. Without some movement of items between entities, the viewer, in Martin's terminology, is unable to keep enough information about the relative states of these entities in the 'foreground' of his mind all at the same time.

This issue of movement brings up an interesting point about ANISIM. When we think of the system being modelled as a queuing network, then animating with moving sequences can be thought of as distorting the model--particularly with respect to the expansion of time caused by binding sequences. But if we think of the system being modelled as some real-world system that is only approximated by a queuing network, then animating with moving sequences can be thought of as restoring some reality to the model. The user of ANISIM can use the SCAN mode for a true representation of the queuing network, but he does not have the benefit of that extra information provided by movement. Or, he can select varying degrees of distortion/reality by controlling the durations of the moving sequences. In either case, animation techniques can at least be used to represent the progression of states, to show the changing interrelationships amongst model components, and to signal the occurrence of critical states.

The concept of allowing the model designer optional degrees of "reality" is not unique to ANISIM. An interactive graphics system developed at IBM for designing and testing logic circuits allows the user to specify one of three modelling abstractions regarding the timing of pulses [13]. The trade-off is between a fast approximation and a slower, more realistic simulation. In this case, however, it is the actual simulation that is affected, not just the graphic representation, as in ANISIM.

The ATOPPS system [5] discussed in the next section, is an example of a discrete event simulation monitor which makes its point graphically without making any attempt to realistically model the timing between events.

4.2 Educational Tool

If the use of animation provides additional insight into the understanding of a complex process, then certainly it is desirable to apply this capability toward educational purposes. This generally means placing less of the emphasis in the graphics system on a flexible design optimization approach and placing more emphasis on carefully illustrating the process or reaction that is of interest. In fact, it may be reasonable in some cases to exaggerate the more difficult features at the expense of the overall accuracy of the animation.

For example, the ATOPPS system [14] at Pennsylvania State

University is presented as a "Computer Graphic Simulation of a Discrete Time Operating System for Introducing Elementary Concepts". Films have been made of the system, which displays the contents of the memory, active hardware, and major queues of a theoretical operating system. The objective is to watch the flow of information from one job step to another in the discrete time simulation of different operating system configurations. The model appears to be fairly simple, but allows enough structural and parameter options to demonstrate many of the important principles of operating system strategies. The graphic technique does not involve animation in the sense of elements moving on the screen. Instead, arrows and intensity are used to call the viewer's attention to (and to further explain) each change of state. The state of each entity (e.g. a queue) is described by a number rather than a symbol. The most signigicant difference between ATOPPS and ANISIM is the manner in which they graphically represent the timing of discrete events. ATOPPS makes no attempt to depict parallel processes or the elapsed time between successive events. Instead, the clock time is always shown and when an event has been displayed the clock is updated to the time of the next event, which is immediately displayed. In other words, all state changes bind the display, while they are represented in some detail. To help the viewer understand the inter-relationship of events, the future events queue is displayed on the screen, allowing one to see exactly when and why each event is scheduled during the

processing of the current event. This approach seems quite reasonable for the stated objective of understanding basic operating system principles. The emphasis is on the detailed display of each event rather than an accurate overall display of the performance of the model.

ANISIM, on the other hand, has considerable potential as an educational tool, but in a slightly different sense. It would not be particularly instructive to show how an arrival event causes a new arrival event to be scheduled and put in the future events queue. It would, however, be useful in demonstrating certain concepts of queuing theory which are based heavily on timing parameters. For example, a student trying to understand how the relationship between the arrival rate and the service rate affect the queue length might benefit by watching an animation of a simple queue with various parameters. Another simple example would be the comparison of two one-server queues with one two-server queue (see Figure 17). If the service distributions have a large variance, the student can see how items tend to travel faster through the two-server system. One problem with demonstrating theoretical results is that they are based on steady state probabilities. Monitoring the simulation at any particular period of time (especially the beginning) provides no guarantee that the state will be anywhere near that specified by the long term probabilities. Of course the statistical summary is still available after stopping the simulation.

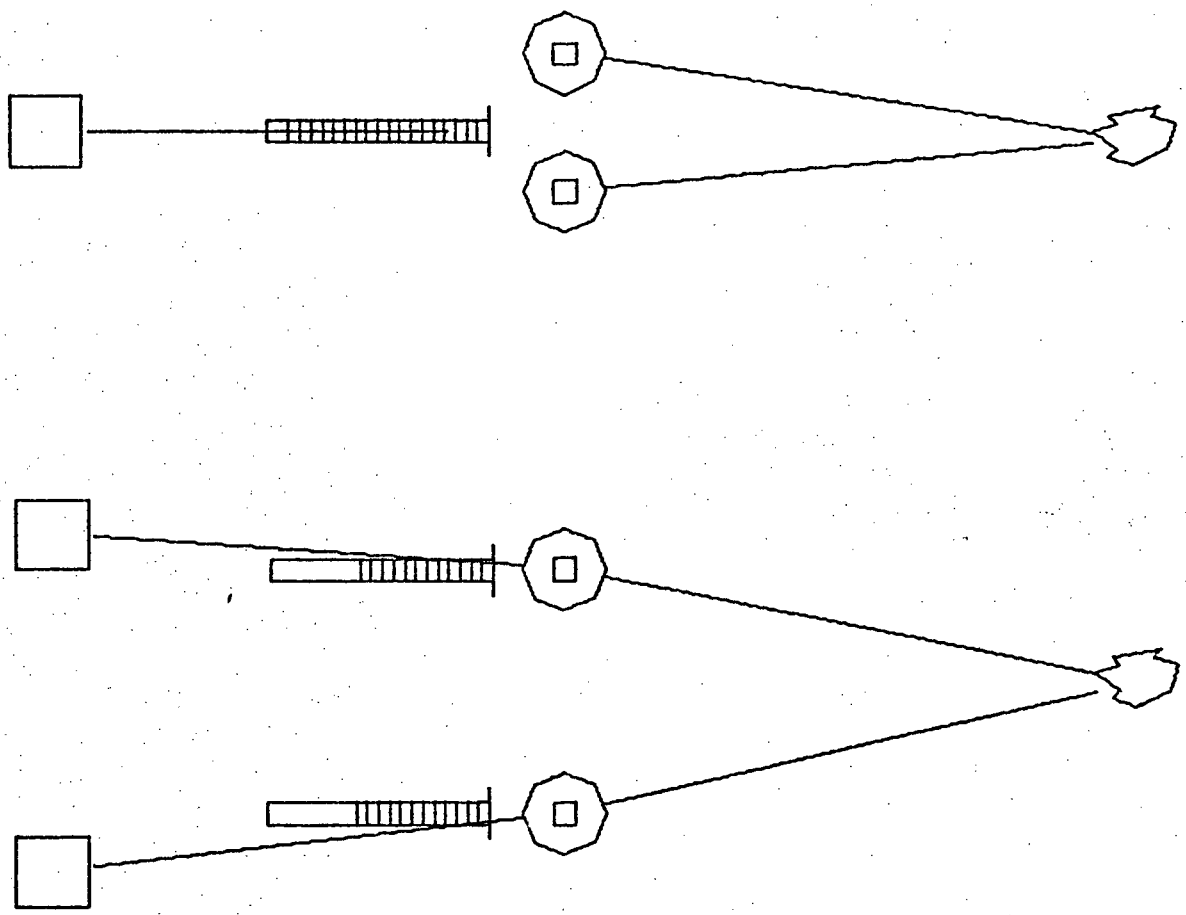


FIGURE 17: Queuing Theory Comparison

4.3 Research Tool

One of the original motivations for developing ANISIM was to study, in a general way, the processes or conditions which lead to network congestion and system deadlock. It has already been pointed out how ANISIM can be used to design or optimize a specific model to avoid undesirable states. Specifically, the animation allows the modeller to watch the chain of events leading up to the critical state. However, perhaps the animation can add additional insight to current research on the theoretical aspects of deadlock. ANISIM can be used to construct and modify networks which, when simulated, a) lead to blocking conditions (see Figure 12 in Chapter III), b) become congested (queues or buffers fill up from time to time but always eventually unblock), or c) reach deadlock (two or more items are mutually permanently blocked). One example of a simple two-node deadlock appears in Figure 18. The more complicated network shown in Figure 19 is deadlocked between the first two buffers, causing congestion in the rest of the network.

Time has not permitted the experimentation necessary in order to investigate the principles behind deadlock. A survey by Coffman, et al [5], describes various strategies for dealing with the prevention, detection (and recovery), and avoidance of deadlocks. The discussion is oriented towards operating systems design, dealing in terms of tasks and resources, but would

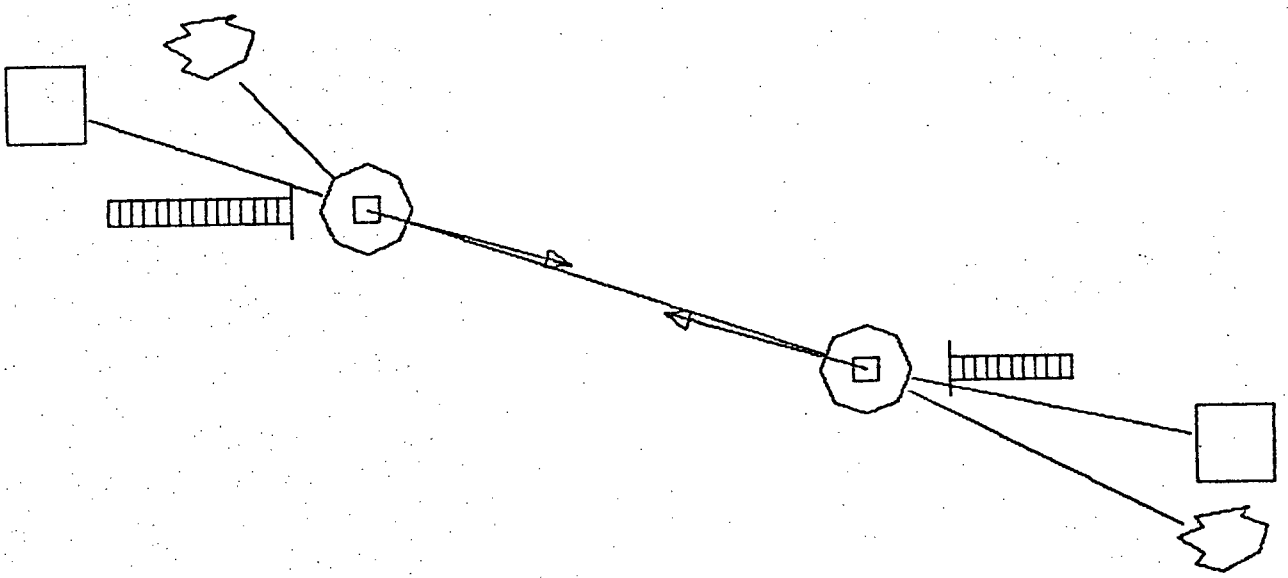


FIGURE 18: Simple Deadlock

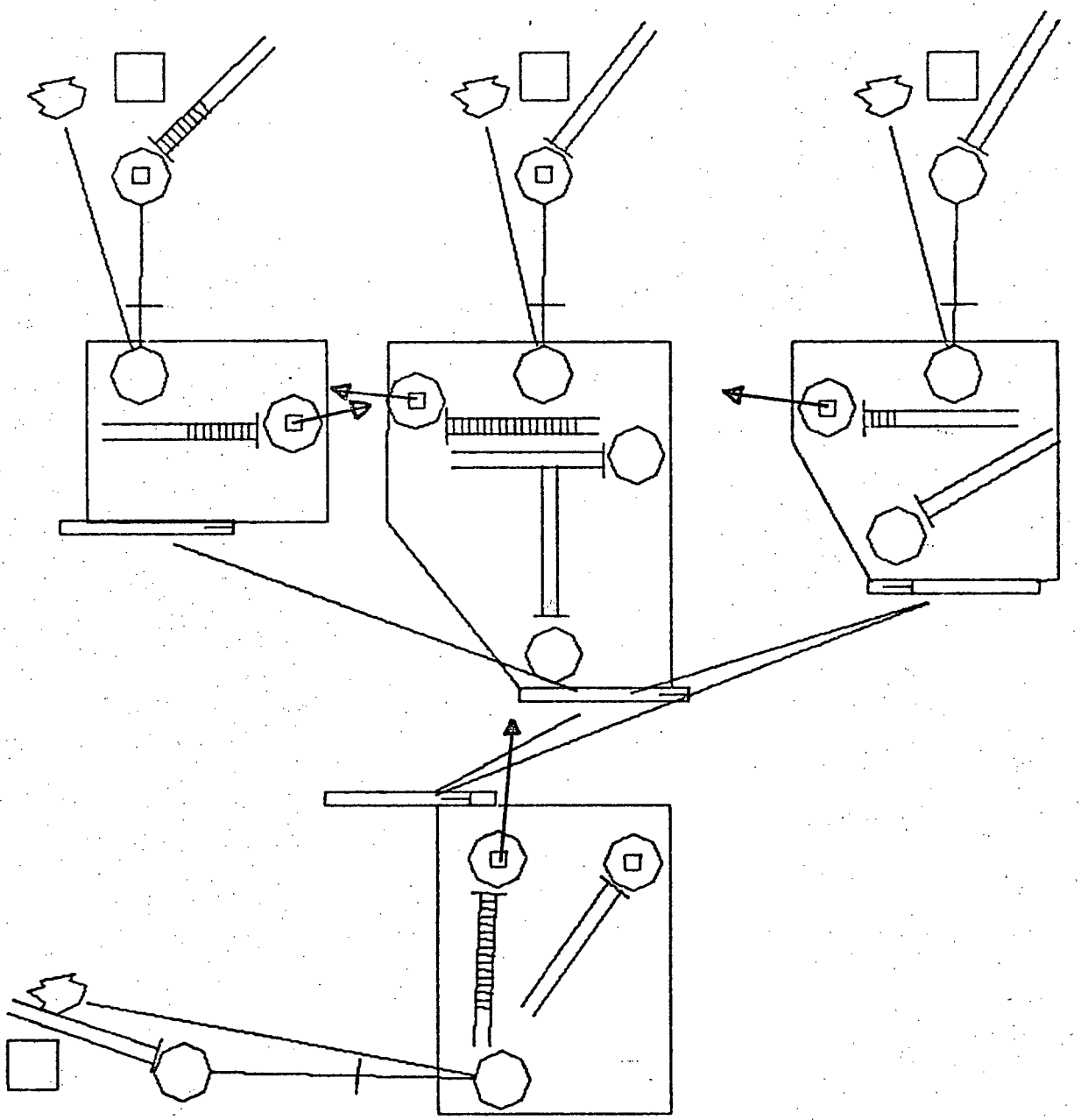


FIGURE 19: A Partially Deadlocked Network

provide an excellent starting point on which to base experimentation with ANISIM.

V CONCLUSIONS, PROSPECTS, AND EXTENSIONS

5.1 Analysis

It must be emphasized at this point that ANISIM, in its current form, can only be thought of as a research tool, and not as a finished product. It has clearly demonstrated that animation can play a very powerful and useful part in discrete event simulation modelling. Hopefully, future attempts at such systems can benefit particularly from the classification of sequences and from the event editing procedures of Chapter II.

Much of the initial effort in implementing ANISIM went into developing the simulation and animation techniques. The system more or less evolved as its capabilities became apparent, resulting in two general problems. The first problem concerns the idealogue. Martin [13], in his survey of methods, features, and psychological considerations of man-machine dialogue, emphasizes the need for comprehensive planning of the dialogue before programming begins. Such planning, in the case of ANISIM, could have eased the programming task, particularly with respect to providing simple, uniform error recovery procedures. The second problem with the lack of overall plan at the beginning is that not enough provision can be made to easily handle extensions and refinements. For example, in ANISIM, as in most graphics systems, the data structure is accessed from just about every phase of the program. Repercussions of any

changes to the structure or to the information stored in it tend to ripple throughout the program. In concentrating on the needs of the animation routines while coding the simulation routine, some basic features were overlooked, such as the gathering of certain statistics and the allowance for more transaction parameters (i.e. in addition to the destination sink and the time of arrival to the system). Adding these features, as well as further extensions (section 5.3), represent a rather tedious programming chore.

Even in light of what it does not do, however, the current system is surprisingly inexpensive to use, considering the fact that it is a simulation program, it is interactive, and it involves extensive I/O for graphics and for saving and restoring data on disk files. Furthermore, any attempt to re-write the program in a more modular fashion to perform more general tasks would be constrained by both the need for a fast, efficient simulation routine and the limited capacity of the Adage display buffer.

5.2 Limitations

One important conclusion, born out by this and the following sections, is that animation is mainly useful as an additional technique for analyzing simulations, not as a substitute for classical techniques. When one gets down to actually using ANISIM to study a real problem he finds that

1) he still needs to have certain statistics available to summarize or validate when he saw (or didn't see), and 2) most models require at least one or two simulation features that are not available in a strict queuing network formulation. This second difficulty is compounded by the limits on problem size imposed by the representation of the network. As already pointed out in the example of transit times, the availability of additional simulation power tends to simplify the required network structure, allowing a more complex situation to be represented on the screen.

In discussing potential models for ANISIM with people familiar with discrete simulations, a pattern began to emerge in the simulation features required in order to enable many models to be formulated. A few of the most important recommendations arising from (or confirmed by) these discussions are mentioned here.

In many queuing models, individual items must group together as they travel through the network. This is true of railroad cars, cargo on ships, and words in a telecommunications network. To accomodate this type of model in ANISIM, the concept of multiple servers must be changed to allow one server symbol to represent a server of unlimited (user specified) capacity. Also groups of items should be able to travel between nodes using one symbol (and generating single events). Figure 20 shows one way of animating this. In the case of trains or

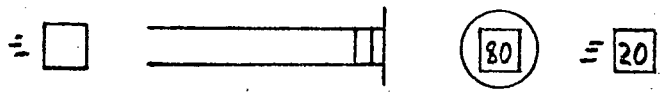


FIGURE 20: Transaction Grouping

ships, it would also be desirable to allow some flexibility in departure protocols. For example, there is currently no way of specifying periodic or otherwise scheduled departures of a set number of items (i.e. departures independent of when service begins).

Another necessary capability of many queuing models is to measure the average delay of items travelling between arbitrary points of the network. The current system only measures the delay between source-sink pairs. The only drawback of adding transaction parameters (as this would) is the increased storage requirement for the ALGOLW records which contain the information for each transaction in a queue.

Two additional features which would considerably widen the class of models that could be simulated are transit times (discussed earlier) and logic switches, or gates. The latter would involve testing on user-specified transaction parameters and would considerably complicate the dialogue for defining the network.

5.3 Extensions

In addition to those discussed in the previous section, the following potential extensions to ANISIM merit consideration.

1) As pointed out earlier, the animation facility does not preclude the need for statistical summaries. Statistics which should be added include the mean and variance of the delay at each queue and buffer, as well as the variance of the queue lengths and buffer contents. Of course, the animation then allows the viewer to watch the variations happening, in order to get a better feel for the nature of the fluctuations and the relevance of the averages. Furthermore, the animation together with the statistics on blocking, provides a better understanding of how occasional blocking (congestion) may distort the signigicance of the other statistics.

2) It may be possible to provide more meaningful ways of presenting statistics to the user. One suggestion is to provide a "third level" of information using a command which allows the modeller to view a plot of a statistic, such as queue length, over time. Another suggestion is to provide a command which displays the average state of each entity rather than just the final state in which the simulation stopped. A more difficult to implement feature would be some type of continuous display during the animation of, for example, the average queue length. (This would involve a separate symbol, or number, displayed near

the queue symbol.) It would also be helpful to display the simulation clock time during the animation.

3) Currently, only two probability distributions have been implemented for arrival and service rates. Others, such as Erlang distribution, must be added, as well as the ability to create a distribution from user-specified empirical data. Also, more flexibility with the random number seeds would allow the user to start two or more streams with the same seed, for comparison purposes.

A further type of arrival rate that would be very useful for modelling closed systems is a finite calling population. A limited number of arrivals are generated, possibly all at once, and new arrivals appear at sources only when items depart to sinks.

4) The routing mechanism used in ANISIM represents one type of route selection. Other possible types include a pre-defined routing scheme, where an item's route is fully specified at the source, and a stochastic routing scheme, where the next node from any given node is determined from a probability distribution. Other possible features than control routing are the transaction splitting and logic switching features of GPSS [9].

5) As well as allowing arrival and service time options, ANISIM

should provide optional policies for queue disciplines and blocked transactions. Priority queues (i.e. allow the generation of different types of transactions) would be somewhat difficult to illustrate, however, given the current representation. An example of an alternate blocked transaction policy is that a blocked item go back to the end of its own queue, instead of waiting in the server.

6) Some typical network configurations occur often enough that it might be useful to provide a "network macro" facility for the automatic creation of previously defined sub-networks. Even better, there may be some models which can be suitable represented with entire sub-networks of the model replaced in the animation by special, or user created, macro symobls. For example, in Figure 19 if each buffer (with its associated queues and servers) were replaced by a network macro symbol, or in this case by a small version of the buffer symbol, the representation of the model would be considerably simplified. Of course this technique would allow ANISIM to handle larger models, since the main constraint on problem size is the number of Adage buffer words used to represent the network. Also for this reason, and for viewing simplicity, it may be worthwhile to attempt a "windowing" capability, where only one portion of a large network is displayed at one time. Windowing may not prove to be of too much value in this case however, since the modeller would never be able to view the entire network at once.

7) One problem with the animation of a non-trival network is that all of the moving items look alike. It is difficult, at times to follow the progress of an item through the network, especially if it spends time in queues. A partial solution to this problem would be to shade, or otherwise mark, certain items so that they can be distinguished from the others as they travel through the network. The drawback is the space required by the additional transaction parameter field. Two methods of using this shading feature are a) mark all items arriving from a specified source, and b) mark every tenth item, or whatever , generated by the system.

5.4 Prospects for Further Work

Although it is possible to formulate a fairly large number of models in terms of networks of queues, the question arises as to what other types of discrete models or real processes might be animated using the techniques described in Chapter II.

It seems that most descrete event simulation models can make use of the internal cycle concept, the classification of sequences, and the editing procedure. The deciding factors, then, for animation feasibility, would be whether the model structure can be suitable represented on a graphics scope and whether meaningful animation sequences can be compiled and displayed using the graphics software available. (The length of any two-side bounded sequences might be a problem in some cases,

as noted in Chapter II.) One type of simulation with a slightly different emphasis from that used in ANISIM would utilize the graphics capabilities to study models involving spatial layout problems. Consider, for example, a model of a warehouse operation, where the cost associated with an item travelling between two points in the model is derived internally from the physical distance between symbols on the screen. The user is able to optimize the model, with respect to cost and space, by simulating and animating it with various spatial arrangements. This type of model would also require queuing facilities, only the representation of the length of the queue would now become important.

The question of animating real processes rather than simulations is a more difficult one. For example, assume it is desirable to monitor an animation of some sort of industrial or scientific process which cannot be directly observed (e.g. due to the location or size of the components of the process). Further assume that the process must be monitored in terms of discrete events, rather than continuous updating. This may be due to the method of measuring its progress, or perhaps only the general stages of the process are important to the observer. If the events are merely being recorded for later study (say the process happens too fast or too slow for real-time monitoring), then there is no problem. The events can be edited and compiled as done for simulations. If, however, it is desirable to display the animation simultaneously with the ongoing process,

then two basic problems arise. For one thing, the process can't be stopped every few events in order to wait for the editing and compiling of sequences. Furthermore, the time expansion due to binding sequences would result in a continually increasing lag between the time of the event and the time it is displayed.

First of all, the problem of time expansion may not be a problem at all. Recall that binding sequences were required in order to add some sort of reality to the modelling abstraction of an instantaneous movement. It seems likely, however, that most real processes monitored in real time will require two-side bounded sequences rather than binding sequences to properly animate movements. Let us assume then, that we wish to animate a real-time process requiring no binding sequences and no two-side bounded sequences of unmanageable length. A system configuration that would probably make this task feasible requires three parallel operations instead of the current two (370 program and Adage monitor). The first processor would continuously record the events of the ongoing process and make the event list available to the second processor, which would then be able to use the internal cycle approach to edit events and compile sequences for the graphics computer to display. The animation would of course lag behind the actual events by a fixed start-up time, but the speed of the display could be made equal to that of the real process.

This discussion is quite abstract and there would certainly

be bugs to work out in this approach. The point is that the potential exists for applying the general techniques of Chapter II to the animation of systems other than simulations.

BIBLIOGRAPHY

1. Baecker, R.M.
"Toward Animating Computer Programs: A First Progress Report," Proceedings of the Third Man-Computer Communications Seminar, NRC, Ottawa, Canada, May 1973.

2. Baecker, R.m.
Interactive Computer-Mediated Animation, MAC-TR-61 (THESIS), June 1969, Project MAC, MIT.

3. Brocchi, G.; Somalvico, M.
"An Interactive Software System for Computer-Aided Design: An Application to Circuit Project," CACM, Vol.13, No.9, September 1970.

4. Chheda, D.P.
"CAM, A Computer-Aided Modelling Program for Systems Dynamics Models," Master's Thesis, Department of Computer Science, U.B.C., 1974.

5. Coffman, E.G.; Elphick, M.J.; and Shoshani, A.
"System Deadlock," Computing Surveys, Vol. 3, No. 2, June 1971.

6. Coulthard, W.J.; Dekleer, J.
"UBC:AGTBASIC - Basic Communication Package for the Adage Graphics Terminal," Computing Centre, U.B.C., 1973.

7. Coulthard, W.J.
"UBC:GRAPH - A Simple Interactive Graphics Package," Computing Centre, U.B.C., 1973.

8. Forrester, J.W.
World Dynamics, Wright Allen Press, 1971.

9. General Purpose Simulation System V User's Manual, IBM Publication No. SH20-0851.

10. Hilborn, R.
Simulation Control Command Language--SIMCON, Institute of Animal Resource Ecology, U.B.C., October 1972.

11. Hillier, F.S.; Lieberman, G.J.
Introduction to Operations Research, Holden-Day 1967.

12. Lafata, P.; Rosen, J.B.
"An Interactive Display for Approximation by Linear Programming," CACM, Vol. 13, No. 11, November 1970.

13. Martin, James
Design of Man-Computer Dialogues, Prentice-Hall, 1973.

14. McIntosh, J.F.
"GRAPHIC," Computing Centre, U.B.C., 1973.

15. Merchant, M.
"Interactive Spline Approximation," Master's Thesis, Department of Computer Science, U.B.C., 1974.

16. Newman, W.M.; Sproull, R.F.
Principles of Interactive Computer Graphics, McGraw-Hill, 1973.

17. Okaya, Y.
"Interactive Aspects of Crystal Structure Analysis," IBM Systems Journal, Vol.7, Nos. 3 and 4, 1968.

18. Prince, M.D.
Interactive Graphics for Computer-Aided Design, Addison-Wesley, 1971.

19. Richardson, F.K.; Oestreicher, D.R.
"Computer Assisted Integrated Circuit Photomask Layout," in Pertinent Concepts in Computer Graphics, Faiman, M., Nievergelt, J., eds., University of Illinois Press, 1969.

20. Savage, G.J.; Andrews, G.C.
"DYNIS: A Dynamic Interactive Simulation Program For Three-Dimensional Mechanical Systems," Proceedings of the Third Man-Computer Communications Seminar, NRC, Ottawa, Canada, May 1973.

21. The SIMSCRIPT II.5 Reference Handbook,
Consolidated Analysis Centers Inc., 1971.

22. Smith, L.B.
"The Use of Interactive Graphics To Solve Numerical Problems," CACM, Vol. 13, No. 10, October 1970.

23. Smith, L.B.
"A Survey of Interactive Graphical Systems for Mathematics," Computing Surveys, Vol. 2, No. 4, December 1970.

24. Sutherland, W.R.
"On-Line Graphical Specification of Computer Procedures," MIT Lincoln Laboratory, TR 405, May 1966.

APPENDIX A -- PROGRAM DESIGN

ANISIM is implemented on an IBM 370/168 using the MTS (Michigan Terminal System) operating system, and an Adage Corporation Graphics Computer, as shown in Figure 21.

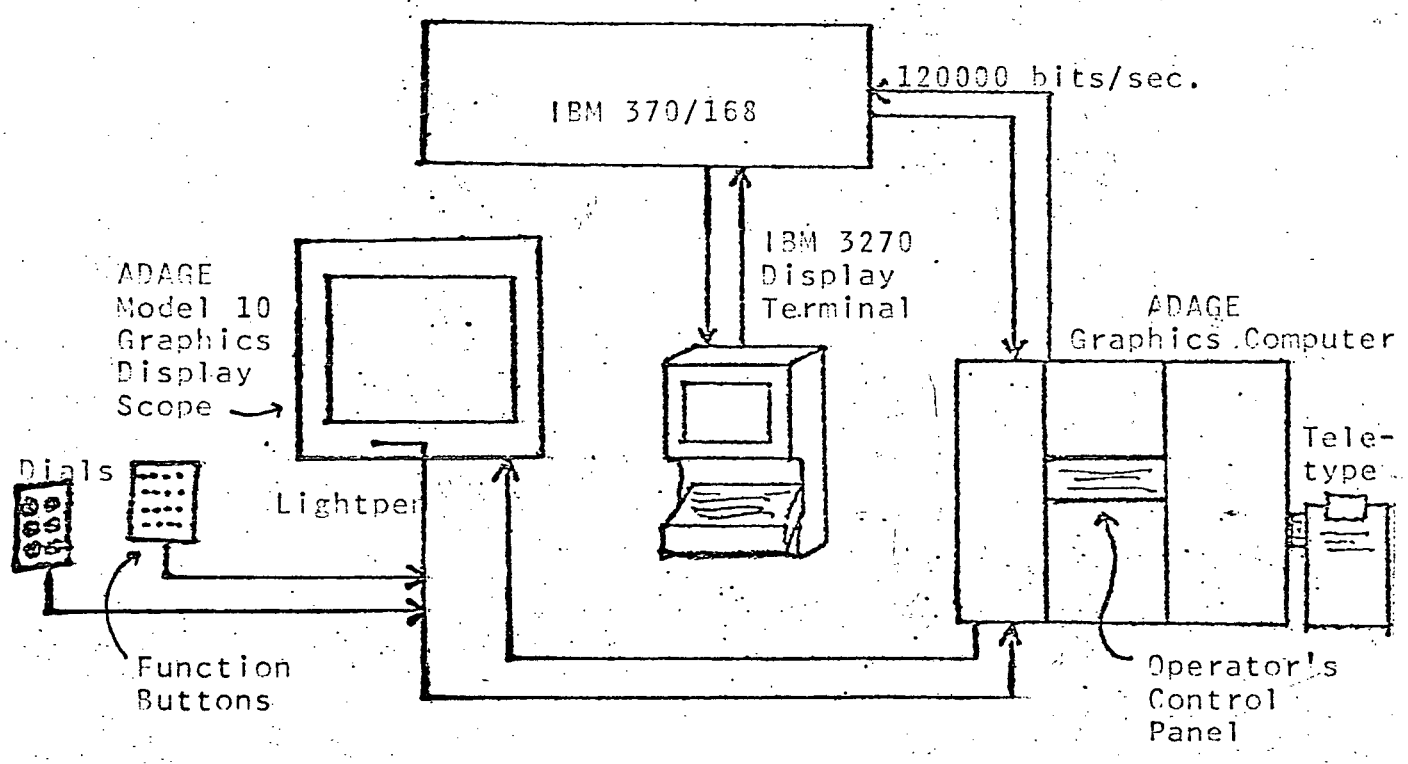


FIGURE 21: System Configuration

The program is written chiefly in ALGOLW, with a few I/O operations in FORTRAN. Software provided by the U.B.C. Computing Centre consists of the MTS file handling routines, the basic subroutine package for communicating with the Adage [6], and the graphics monitor which resides in the Adage Computer [7]. Following is a brief description of the

major procedures in the ALGOLW program.

MAIN: The main body of the program is basically the command monitor. Upon initial execution, the procedure INITMAIN is called to initialize various system variables, and the procedure SETUP is called to generate the Adage buffer words for the crosshairs, menu, messages, and symbols. The monitor then asks the user to enter a command, as shown in Figure 10. The input string is compared to each possible command until a match is found. Commands which allow parameters for changing the value of system variables always print out the new values of the variables. Thus, if the parameters are ommited, the current values will be printed. Simple commands are executed in line by the command processor, while other commands are executed by calling one or more procedures.

BUILD: The BUILD procedure monitors all network construction and modification. Upon entry, it loads the words for the menu and prompting messages into the display buffer. The main loop consists of turning on the display of the menu, issuing a lightpen read, turning off the menu, and executing a procedure (depending on the location of the lightpen hit). Also, upon termination of the procedure executed, control of the mode name in the menu is switched from dial two to dial five.

NODES: This procedure is called when the SYMBOLS command in the

menu is hit. Its main loop, after turning on the crosshairs, consists of issuing a read to the function buttons and executing the appropriate code. The buttons are used to create sources, servers, sinks, queues, and buffers. For each such entity, the symbol is displayed at the location of the crosshairs, the buffer words for the label are created, and a record is created and added to the list of records for that entity. Further use of the buttons is required in order to position a queue or to sketch a buffer. A special button terminates the SYMBOLS mode. If a lightpen hit is read instead of a button, the symbol or link pointed to is no longer displayed. If a symbol is pointed to, its associated record is also deleted from the data base.

LINKS: The LINKS mode makes use of the lightpen in order to draw connecting lines between pairs of symbols. No entry is made in the simulation data base. The intensity of the links is controlled by dial E, while the intensity of the symbols is controlled by dial B.

ASSIGNQ: This procedure is called, with different arguments, for the ASSIGNQ, FIRST QUEUES, and ASSIGN BUFF modes. For example, if the arguments are the list of queues and the list of servers (ASSIGNQ mode), then the process is as follows: a server is made to blink; a prompting message asking for a queue is displayed, and the lightpen is read. If a queue symbol was hit, then a pointer to the queue record is placed in the server record,

otherwise the message "NO ASSIGNMENT" is displayed. If the edit flag is not on, then the process starts over with the next server in the list until all servers have been processed. If the edit flag is on, then the user is first prompted to point to the server he is interested in.

ROUTING: The ROUTING mode requires a procedure similar to that used in ASSIGNQ in order to fill a matrix of pointers to queue or sink records. Each entity is assigned a number (unique within the entity type) when created in SYMBOLS mode. The number is used to form the label and to reference the entity whenever a pointer to its record is not appropriate. In the case of the routing matrix, the first dimension is indexed by queue numbers, and the second dimension is indexed by sink numbers. Thus for each blinking queue-sink pair, the user is asked to point to the next queue in the route, or to the sink. This is done by stepping through the list of sinks and, for each sink, stepping through the list of queues. If the edit flag is set, the user is asked for a sink but every queue is processed for than sink.

CAPACITIES: This procedure steps through the list of queues and then the list of buffers, blinking each in turn and prompting on the 3270 for the capacity. Each queue and buffer is given a default capacity (20 and 100, respectively) in SYMBOLS mode. The default or otherwise current capacity is printed with the

prompt. If a null line is entered by the user, the field in the queue or buffer record remains unchanged. If something other than a number is entered, the capacity is set at 1,000,000 and the queue symbol is altered to represent an infinite queue. If a queue capacity less than twenty is entered, the queue symbol is shortened proportionately.

SOURCEFLOW: The SOURCEFLOW procedure is used for both FLOW and SERVICE TIMES modes, where the arrival and service distribution parameters, respectively, are entered into their appropriate fields in the source and server records (actually the DIS records--see Appendix B). The method of blinking each symbol under consideration is used here as well. Since defaults are not assigned, the current parameters are only printed with the prompt if the edit flag is on. In SERVICE TIMES mode the re-send time parameter is also requested for each server. In FLOW mode the flow matrix, indexed by source number and sink number, is filled in the same manner as the routing matrix. If the fractions of flow to each sink from a source do not add up to one, a message is printed and that step is repeated.

This completes the major procedures within the BUILD procedure.

SAVENET: This procedure is invoked by the SAVE command and allows a complete definition of a network model to be written onto an MTS file of the user's choice. The filename is prompted

for if it was not entered as a parameter to the command. MTS subroutines are used to create or empty the file and to open it. The information saved consists of 1) the buffer words for the network display, 2) an encoded description of the relevant fields of the records for each entity, 3) an encoded description of the routing matrix, and 4) the flow matrix.

RESTORE: The inverse of SAVENET, this procedure reads in and decodes the saved information, re-creating the data base and display. The restored network replaces any currently active network that may exist, and the simulation statistics are initialized to zero.

LABEL: The LABEL procedure is used to display or remove the labels at each symbol in the network. The buffer words for all of the labels are kept in an array and only loaded into the buffer when required for display. This routine is automatically called to remove the labels before entering BUILD or starting the simulation.

GO: GO is a small procedure which monitors the Simulate-Edit-Compile cycle. It is envoked by the GO command to start or re-start the simulation. When starting a simulation, GO first checks the data base in order to make sure each source and each server has been assigned a queue. Between internal cycles, GO checks to see if an attention interrupt has been issued or one

of the simulation limits has been exceeded. If not, the SIMULATE procedure is called. The EDIT and COMPILE procedures are then called, unless a NODISP command was issued before the current GO.

SIMULATE: The main body of SIMULATE checks between the processing of each event to see if a simulation limit has been exceeded, an attention interrupt has been issued, or the required number of potential sequences for the internal cycle has been achieved. If none of these conditions hold, the GETEVENT procedure is called, otherwise control is returned to the GO procedure.

GETEVENT: The collection of routines comprising the simulation program maintain two lists of event records: the future events list (or queue) and the list of processed events to be passed on to the EDIT procedure. GETEVENT processes the event at the head of the future events queue and puts the altered record in the output event list. (If there are no future events scheduled, then GETEVENT first calls GENARR.) The event record is described in Appendix B. Basically, it contains the event class, the time of the event, and various parameters further defining the specific simulation entities involved. The record also contains fields later used for the animation time and duration.

GETEVENT begins by updating the clock to the time of the new event. It then tests on the event class and uses the more

specific information to appropriately update the state and statistics of the model in the data base. For each future event that must be scheduled as a result of the current event, the procedure GETEVENT is called in order to create the proper event record in the future events queue. The aspects of the current state of the simulation that will be required by the COMPILE procedure are then added to the current event record and it is placed in the output event list.

GENARR: This procedure generates one arrival event at each source to initialize the simulation. (An arrival event always causes the scheduling of the next arrival event at that source.) A pseudo-random number generator is used to derive independent random number streams for each source (and each server). Thus the time of arrival is determined by the stream and the user-specified probability distribution for that source. GENARR must also use the flow matrix and a separate random number stream in order to determine the destination sink for the new arrival.

GENEVENT: GENEVENT is called from GETEVENT with an argument specifying the event class desired for the new event. GENEVENT then uses the information in the current event record and in the data base to create the new event record. This record is placed in the future events queue.

SENDSTATE: When the simulation stops, the data base contains the

current state of the model. SENDSTATE is automatically called at this time to scan the relevant records and display the current state of each queue, buffer, and server. (This is desirable since either NODISP was specified, or the tail of the last internal cycle was not displayed.)

EDIT: The EDIT procedure edits the list of event records output by the simulation. These same records are used to form the sequence list and tail list. Chapter II describes the editing process in detail.

COMPILE: The operations performed by the COMPILE procedure are also described in Chapter II. The first part of the procedure steps through each record in the sequence list. Depending on the event class, selected smaller procedures are called to actually compile the display programs for each sequence. The second part of the routine then compiles the buffer timer words and sends the completed array to the Adage.

RESET: The RESET routine goes through the data base and, for each entity, changes the state and statistics to their initial value. It also displays a fresh copy of the network, restores the random number seeds, eliminates any remaining event or sequence lists, and resets the simulation clock.

SPEED: The SPEED procedure uses a small array of pre-defined

settings for the animation time conversion factor and the length of the internal cycle. The argument to the procedure specifies whether to assign the values for the next faster display, the next slower display, the default speed display, or the scan mode display (which also requires setting the sequence durations to zero). Also, if a tail list exists, the SPEED procedure will go back and adjust the animation times in the tail according to the new factor (see sections 2.3.3 and 2.4.2).

APPENDIX B -- DATA STRUCTURE

The basic record describing each model entity is the NODE record (see figure 22). These records are grouped into five separate lists of sources, servers, sinks, queues, and buffers.

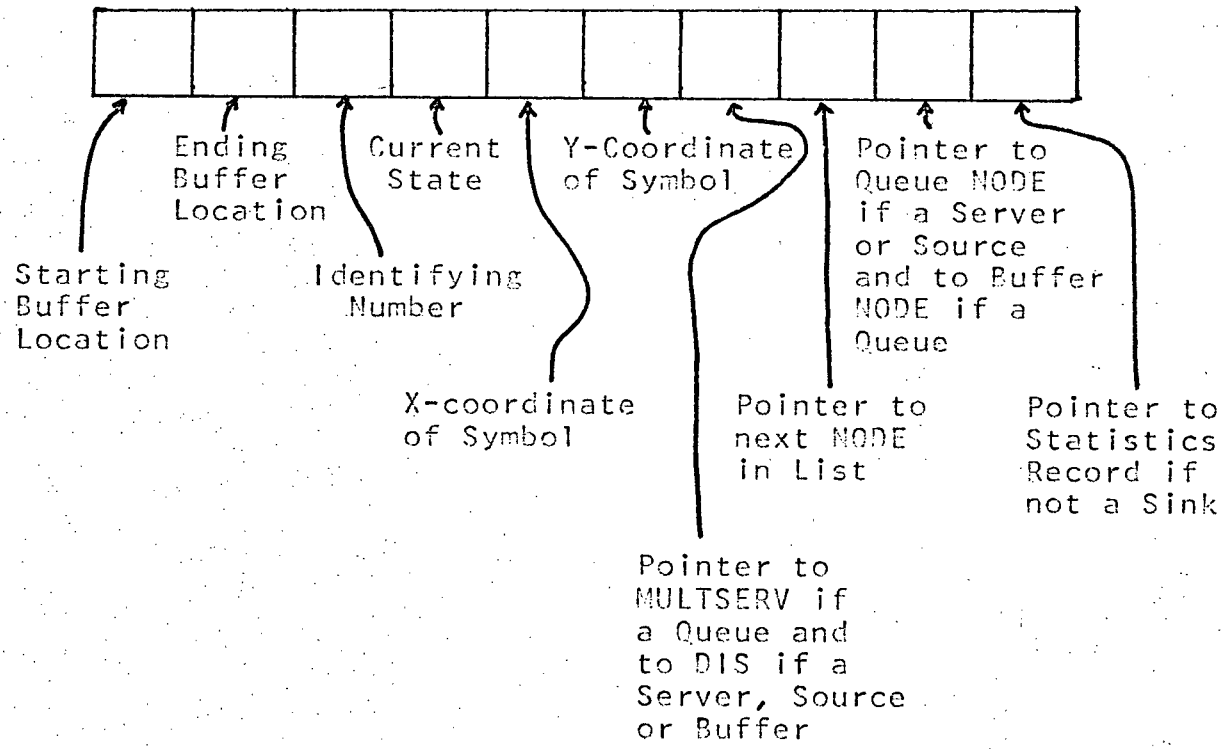


FIGURE 22: The NODE Record

The NODE record is augmented, when necessary, by the DIS, MULTSERV, and STATISTICS records. The DIS record has three fields and contains the arrival or service time distribution code and parameters, or the buffer capacity. The STATISTICS record also has three fields. For queues and buffers, it contains the statistics for computing the average and maximum queue length or buffer occupancy. For sources it contains the number of lost arrivals and an integer identifying the random

number stream. For servers it contains the stream identifier, the re-send time, and the number of times found full. Figure 23 shows the use of the MULTSERV record and the associated TRANPAR and MSPTR records. For each queue, there is one TRANPAR record for every item in the queue (not including items being served). Future transaction parameters would require expansion of this record. Also, for every server using the queue, there is one MSPTR record.

The network definition is completed by the ROUTE matrix consisting of pointers to NODE records (queues or sinks), and the FLOW matrix of real numbers.

The EVENT record contains several fields which take on various meanings at different points in the processing. Figure 24 summarizes this record.

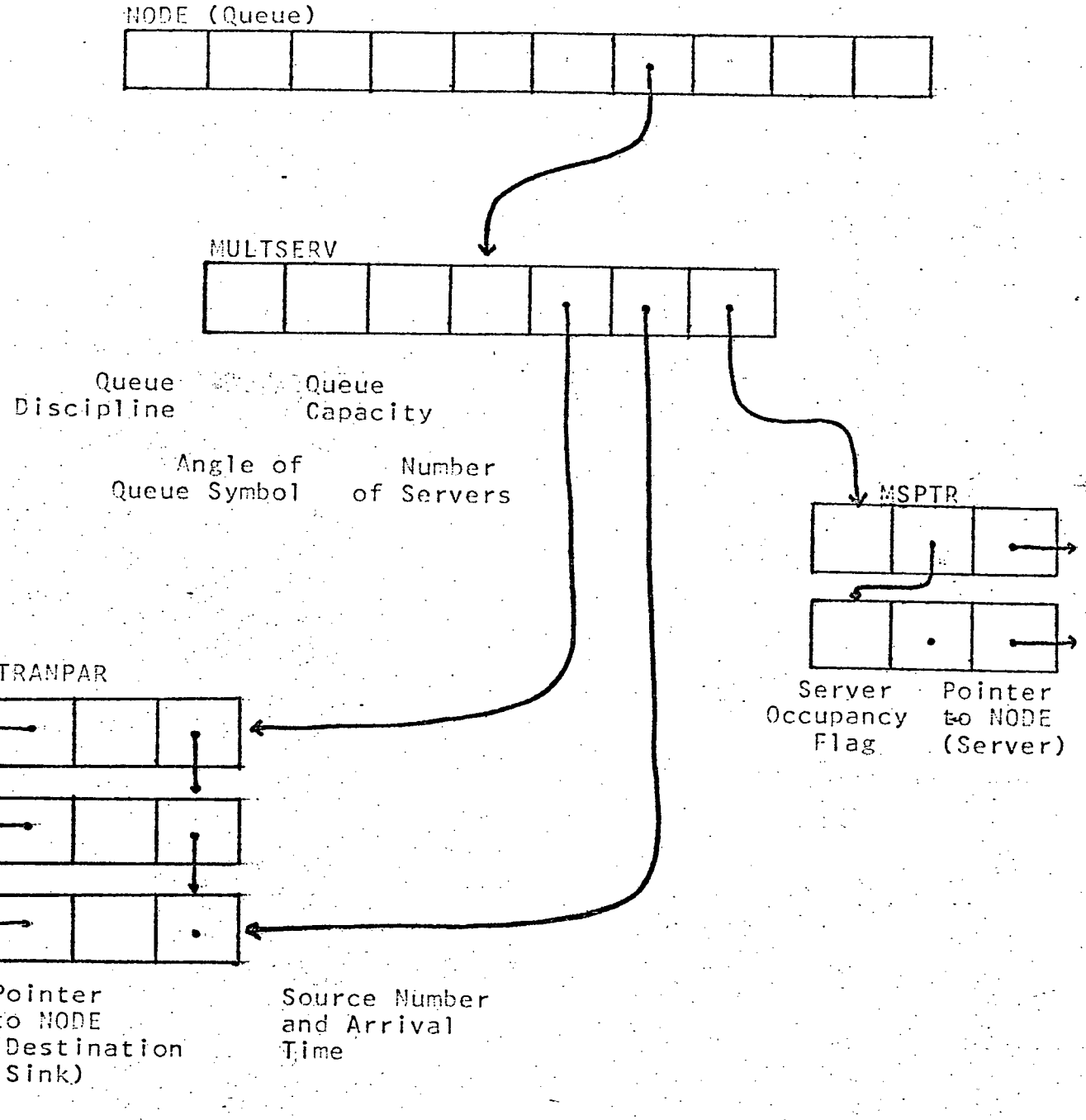


FIGURE 23: Data Structure for Queues

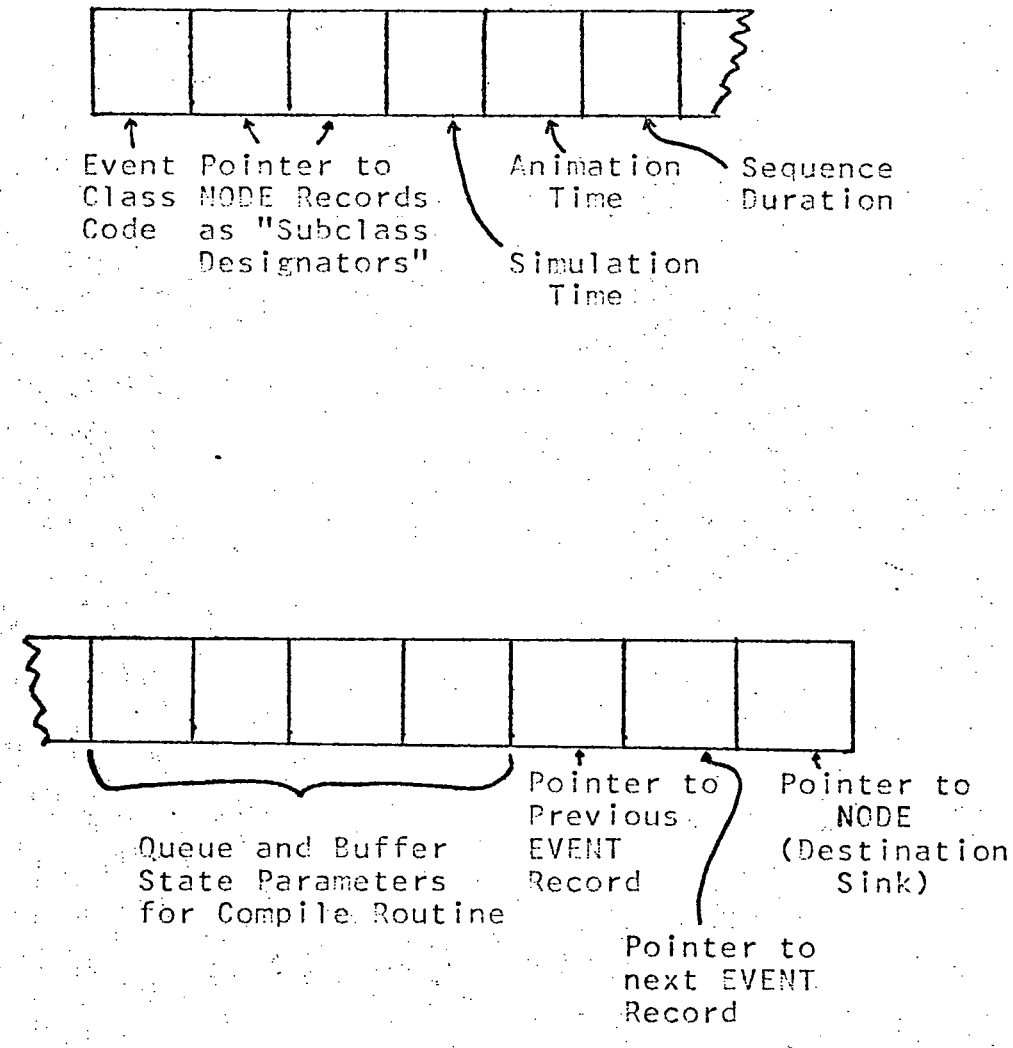


FIGURE 24: The EVENT Record

APPENDIX C -- USER'S GUIDE

Purpose

ANISIM provides a command language for creating, simulating, and animating arbitrary queuing network models. There are two main phases of execution: the normal command mode at the 3270, and the menu dialogue for network definition and modification using the Adage and the 3270.

Running the Program

Before attempting to run ANISIM, make sure that the graphics monitor has been loaded into the Adage computer. See the Computing Centre writeup UBC GRAPH for details. The following command may then be used to start execution of ANISIM:

```
$SOURCE WALK:ANISIM
```

The program will first ask the question "ARE YOU USING THE ADAGE--TRUE OR FALSE," which should be replied to by spelling out the word "TRUE" (or "FALSE"). The program will then clear the Adage screen and enter the command mode with the message "ENTER COMMAND OR HELP." On entering "HELP," a list of available commands will be presented along with a brief reminder of their purpose.

Adage Input

The six dials connected to the Adage are used in the following way:

DIAL A	DIAL D
the horizontal crosshair	the vertical crosshair
DIAL B	DIAL E
intensity of symbols and names of un-entered modes	intensity of links and labels and names of entered modes
DIAL C	DIAL F
X-co-ordinate of menu	Y-co-ordinate of menu

To use the function buttons (required during SYMBOLS mode of the menu dialogue) the yellow overlay card marked "ANISIM" should be used. If this card cannot be located, the function of each button is as follows:

BUTTON 1: Creates a source.

BUTTON 2: Creates a server.

BUTTON 3: Creates a sink.

BUTTON 5: Creates a queue.
Orients the tail of the queue symbol.

BUTTON 6: Creates a buffer.

Sketches the buffer symbol.

BUTTON 7: Terminates buffer sketching.

Terminates SYMBOLS mode.

BUTTON 4: Selects the default buffer symbol.

BUTTON 8: Allows a sketched buffer symbol.

When using the lightpen, the display will blink when a hit has been accepted, and the button should then be released to avoid an unintended second hit. Also, as a general rule in the menu dialogue, a lightpen hit on the prompting message is used to terminate a mode or avoid an assignment.

Available Commands (brackets denote optional parameters):

BUILD, NEWNET, or EDITNET

Each causes the program to enter the menu dialogue phase. NEWNET first destroys any active network, and EDITNET should be used for small changes to the active network. The menu dialogue phase is described later in more detail.

CYCLE [n1] [n2] [n3] [n4]

Changes the limits on the simulation and prints the resulting values. If any parameter is missing or given as an asterisk (position holder), the current value is retained and printed. The parameters are 1) the simulation clock time limit, 2) the maximum number of arrivals generated, 3) the maximum number of entries into queue/server systems, 4) the maximum number of

terminations (departures to sinks). Default values are 1000, 200, 200, and 50, respectively.

DESPEED

Returns to the default speed for subsequent animations.

DUR [n1] [n2] [n3] [n4]

Alters the sequence durations (in simulation time) to the new values if specified, and prints the values. The sequences are 1) arrivals, 2) lost arrivals, 3) departures (binding), and 4) departures to sinks. All default durations are 10 simulation time units, except during SCAN mode when they are zero.

EDITNET

See BUILD.

END

Terminates the program. An attention interrupt also returns control to MTS but the program can be resumed with a $RESTART command.

FACTOR [n]

Alters the animation time conversion factor to the new value if specified, and prints the value. The smaller the factor, the faster the display. The default factor is set to 10.

FASTER

Increases the speed of subsequent animations, unless the current speed is the fastest, and prints the new factor and cycle length values.

GO [n]

Simulates the active network for n time units from the current

state or until interrupted, or until a simulation limit is exceeded (see the CYCLE command). The animation accompanies the simulation unless a NODISP command is first issued.

INTERCY [n]

Alters the length of the internal cycle to the new value if specified, and prints the value. For fast displays, the internal cycle should be made larger to avoid delays in the animation between cycles. Default value is 15 sequences.

LABEL

Displays unique labels at each symbol. These labels allow reference to specific network entities by information available from the PRINT command. The intensity of the labels is controlled by dial E.

MORHELP

Prints additional commands not listed by HELP.

NEWNET

See BUILD.

NODISP

Turns off the animation for the duration of the next GO command.

PLOT [s]

Provides a hardcopy plot of the display. The maximum dimension is 10 inches unless the parameter otherwise specifies. PLOT:Q must be run after termination ANISIM.

PRINT n1 [n2]

Prints the following information, according to the first argument. The second argument, if not zero, will cause the

program to ask for the name of an MTS file on which to print the information. The codes are:

0) All of the information available from codes 2-8 and 16.
1) All of the information available from codes 3-8 and 16 (i.e. all statistics).
2) A description of the future events list.
3) All of the information available from codes 4-8.
4) Source numbers and statistics.
5) Sink numbers and statistics.
6) Server numbers and statistics.
7) Queue numbers and statistics.
8) Buffer numbers and statistics.
11) A description of the current sequence list.
12) The most recently compiled animation buffer (for debugging).
13) A summary of the current model entities and parameters.
14) The current value of each random number stream (for debugging).
15) The entire Adage display buffer (for debugging).
16) The average time-in-system statistics by source-sink pairs.

Any other number will result in the printing of the current status of the simulation.

RESET

Resets the simulation variables, the statistics, the model state, the random number seeds, and the display to their initial status.

RESTORE [filename]

Destroys the current active network and makes active the network saved on the MTS file specified.

SAVE [filename]

Saves the active network on the MTS file specified. If the file does not already exist, it will be created.

SCALE [x]

Alters the scale of the display to the new value if specified, and prints the value. The default is 0.6.

SCAN

Allows the animation to be displayed at an increased speed with no moving items (i.e. all "sequence durations" are zero). The new factor and cycle length values are printed out.

SLOWER

Reduces the speed of subsequent animations, unless the current speed is the slowest, and prints the new factor and cycle length values.

STOP

Same as END.

TRACK n1 [n2]

Sets debug flags for the programmer. The second argument causes the program to ask for the name of an MTS file on which to print

the output. The flags are 0) PRINTCYC, 1) PRINTSTEPOUTP, 2) LITDUMP, 3) TRACE, 4) DUMPER, 5) COMPTRACE, 6) COMTRACE, and 7) BUFTRACE. Flag 3 may also be of value to the user, as it provides a detailed trace of the simulation.

UNLABEL

Removes the labels from the screen. (This is done automatically before simulating and before entering the menu dialogue.)

The Menu Dialogue

Once this phase of the program is entered, commands are entered by pointing with the lightpen to a mode name in the menu. The menu should be positioned by dials C and F to a convenient location on the screen. It will disappear during the execution of each mode. Also, Dials B and E should be adjusted such that un-entered modes are at normal intensity and entered modes are at a reduced intensity. A mode may be re-entered by turning the intensity back up in order that the lightpen hit will take. When building a new network, a recommended order of execution is the order in which the mode names are listed. The purpose of each mode is outlined below, along with any instructions for use not obvious from the dialogue.

SYMBOLS: This mode is used to create model entities with the function buttons and position their symbols at the locations defined by the crosshairs. Button 7 is used to terminate the mode. When creating a queue, a second button hit is requested.

The head of the queue will be located at the first crosshair location and the tail will be oriented in the direction of the second crosshair location. Buffer symbols may be sketched in any shape. Upon selecting a buffer (button 6), the bargraph appears at the crosshairs location and the user is asked whether he wants to sketch (button 8) or use the default shape (button 4). If it is sketch, button 6 is used with the crosshairs to specify line endpoints. No line will be drawn to the first point specified. Button 7 terminates the sketching. The lightpen may be used in this mode to delete a symbol or link by pointing to it. A maximum of 19 queues, 19 servers, 10 sources, 10 sinks, and 6 buffers are currently allowed, including any deleted ones.

LINKS: This mode is used to connect any pair of symbols with a line by pointing to the symbols with the lightpen. The intensity of the links is controlled by dial E.

ASSIGNQ: Every server must be assigned a unique queue. These, and other assignments, are made by pointing to the symbol with the lightpen. The assignment is made to the server that is blinking.

FIRST QUEUES: Every source must be assigned a unique queue to designate the first queue/server system in the route for items generated at that source.

ASSIGN BUF: This mode is required in order to tell the program where to send an item next, given that it is at a queue/server system (blinking queue) and is destined for the blinking sink.

The user should point to the next queue or to the sink.

CAPACITIES: This mode is required if the default capacities of 20 for queues and 100 for buffers are not appropriate. The capacity of a queue does not include the servers, although the state of a queue refers to the queue/server system.

FLOW: This mode is required in order to define the inter-arrival time distribution type and parameters. (Currently, the exponential and uniform distributions are available.) The mode is also used to define the flow of items generated at each source, by entering the fraction (decimal fraction between 0.0 and 1.0, inclusive) of those items to be destined for each sink. If the fractions do not add up to 1.0 for each source, the user will be asked to try again.

SERVICE TIMES: This mode is required in order to specify the distribution and parameters of the service time for each server. It is also used to specify the (constant) time a blocked item must wait in the server before attempting to depart again. (An item is blocked by either the next queue or its buffer being filled to capacity.)

DONE: This is not actually a mode, but causes the menu to disappear and control to be returned to the normal command phase at the 3270. Since there may still be bugs in ANISIM which could cause an unexpected termination of the program, any newly created or modified network should be saved at this time using the SAVE command.

This user's guide is not intended to provide a thorough

understanding of the uses and capabilities of ANISIM. The interested user is referred to the main body of this thesis for further details.