Proceedings of the 1997 Winter Simulation Conference
ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson

AWESIM: THE INTEGRATED SIMULATION SYSTEM

A. Alan B. Pritsker
Jean J. O'Reilly

Pritsker Corporation
P.O. Box 2413
Lafayette, Indiana 47906, U.S.A.

ABSTRACT

AweSim® is a totally new general-purpose simulation system. AweSim takes advantage of the latest in Windows® technology to integrate programs and provide componentware. AweSim includes the Visual SLAM® simulation language to build network, subnetwork, discrete event, and continuous models. Network models require no programming yet allow user-coded inserts in Visual Basic or C. Discrete event and continuous models can be created using the object-oriented technology of Visual Basic, C or Visual C++ and can be combined with network models. This paper will demonstrate the process of using AweSim's componentware, describe examples of user interfaces that allow integration with other applications, and present examples of models used for problem solving using AweSim.

1 INTRODUCTION

AweSim is a program that supports the range of tasks required to perform a simulation project. AweSim also provides integrating capabilities to store, retrieve, browse and communicate with externally written software applications. The architecture of AweSim is shown in Figure 1. The most fundamental feature of the AweSim architecture is its openness and interconnectivity to databases, spreadsheets and word processing programs such as Microsoft Office. AweSim is built in Visual Basic and C/C++ and programs written in these languages are easily incorporated into its architecture. The details on the capabilities of AweSim are contained in the *AweSim User's Guide* (Pritsker Corporation, 1997).

An AweSim project consists of one or more scenarios, each of which represents a particular system alternative. A scenario contains component parts. Software programs called builders are provided by AweSim to create each component.

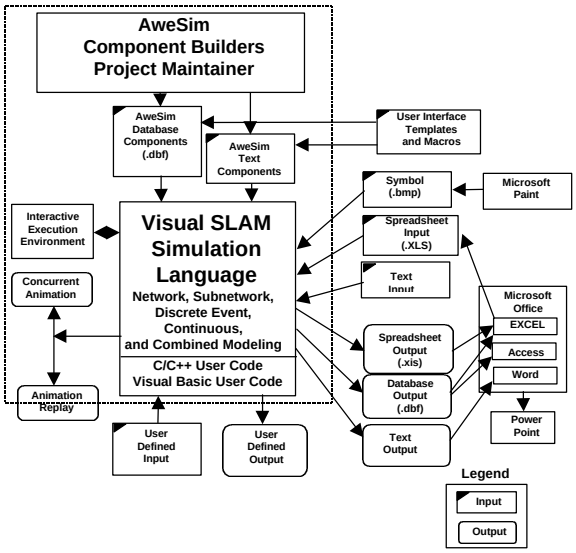

Figure 1: The AweSim Architecture

AweSim's Project Maintainer determines if a model translation or compilation is required. Each time the modeler requests a run, the Project Maintainer examines changes made to the current scenario to determine if any components have been modified and indicates whether translating tasks should be performed. The Project Maintainer then allows the user to specify whether these tasks should be done prior to performing the requested function. Multiple tasks may be performed in parallel while a simulation is executed in the background. The simulation modeler can switch between tasks by using a mouse to click in the appropriate window.

The use of the MS Windows interface simplifies learning and provides the foundation for combining application programs using that interface.

2 MODEL OUTPUT

AweSim provides the ability to compare output from various scenarios both graphically and textually. A report "browser" allows alternative textual outputs to be compared side by side. Graphically, output maybe viewed within AweSim in the form of bar charts, histograms, pie charts, and plots. Bar charts can be used to display the value of a statistic across scenarios. It is possible to view multiple windows of graphical output at the same time, as shown in Figure 2. Graphical and textual information from the AweSim database can be exported to other Windows packages such as EXCEL or Word for additional analysis and for documentation.

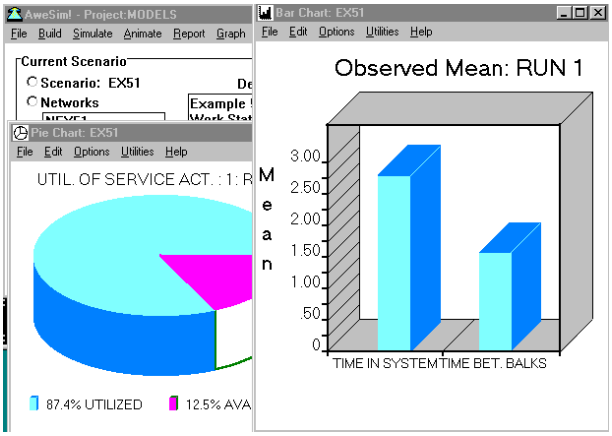

Figure 2: Multiple Report Windows

3 ANIMATION

With the AweSim animator, one may develop and display multiple animations for a single scenario (Figure 3).

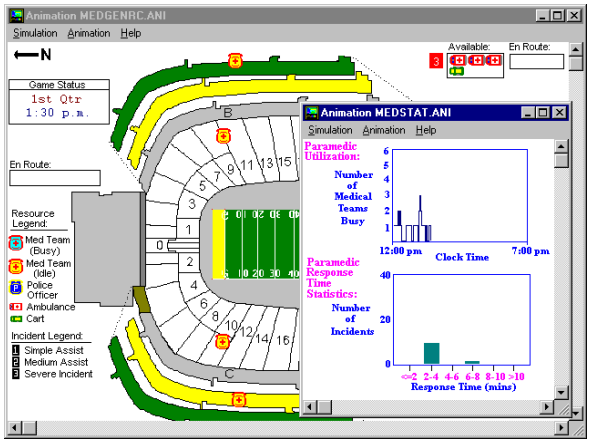

Figure 3: Animation with Multiple Windows

For example, the modeler can create one animation of a system at an aggregate level and another at a department level, side by side in separate windows. The two views may then be displayed by associating the animations with the current scenario and running the simulation. Animation constructs, called actions, correspond to elements in an AweSim network model. For example, the ACTIVITY action shows movement of a symbol. It requires that the modeler define a symbol, a graphical path location where movement of the symbol will be shown, and the number of an activity in the AweSim model to which to tie the movement. Other actions are provided to display resource status, items in a queue, variable values, and other system status conditions. The symbols manipulated by the animator are of two types: graphical items one wants to display or move, and the background on which they will appear. These symbols are stored in standard Windows bitmap format, allowing them to be exchanged between programs using the Windows clipboard.

4 INTERACTIVE EXECUTION

The AweSim Interactive Execution Environment (IEE) provides an interface with an executing simulation. The modeler may examine, modify, save, or load the current system status using the IEE. The IEE supports the debugging of a model under construction and verification of a completed model. The analyst may use the IEE to develop and analyze alternative control strategies for a system or to demonstrate the operation of the model to non-modelers. The IEE control panel (Figure 4) lets you advance through the model step-by-step or by setting breakpoints. At any point you may examine variable values, statistics, or queue entries or save the system status in order to reload and experiment with alternative scenarios.

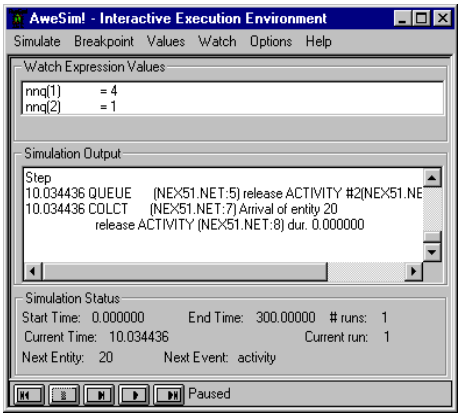

Figure 4: Interactive Execution Environment

5 MODELING AGENTS AT AN AIRPORT COUNTER

At an airport counter, there are two lines for passengers waiting to check in or to purchase tickets. One of the two lines is for first class passengers and passengers who receive priority treatment due to the number of miles they fly with the airline. The second line is for coach passengers. The airline maintains 6 agents to process passengers during a peak period. Two agents process priority passengers but will serve coach passengers if no priority passengers are waiting. Two of the agents select their passengers from either of the two lines with a preference for passengers waiting in the priority line if both dedicated agents are busy. If no one is waiting in the priority line, then these agents select the next passenger waiting from the coach line. The last 2 agents are dedicated to coach passengers. If more than 1 of the coach agents are idle, the agents have an informal rule that the agent that has been idle the longest serves the next incoming passenger. Passengers in the priority line are served by the closest available agent that serves priority passengers.

Priority passengers arrive during the peak period according to an exponential distribution with a mean time between arrivals of 5 minutes. Coach passengers also arrive with an exponential distribution but with a mean time between arrivals of 2 minutes. For priority passengers, the service time is uniformly distributed between 2 and 20 minutes as they require different types of service. Coach passengers have a processing time that is triangularly distributed with a modal value of 6 minutes, a low value of 3 minutes and a high value of 12 minutes. It is desired to estimate the amount of time each type of passenger waits in the system and to assess the utilization of the 6 agents, both individually and as part of the first class and coach groups. A schematic of the system is shown in Figure 5.

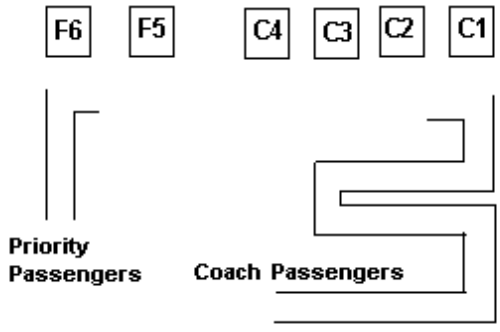

Figure 5: Schematic Diagram of Airport Counter.

The Visual SLAM network model consists of two disjoint network segments involving the arrival of passengers, the waiting for an agent, the processing by the agent, the freeing of the agent and the collection of time-in-the-system statistics. These functions are modeled by a CREATE node, AWAIT node, ACTIVITY, a FREE node, a COLCT node and a TERMINATE node. Consider first the resource definitions and the resource GROUP definitions which graphically are shown in blocks (Figure 6).

RESOURCE Blocks

Res#	Name	Cap	File#	
1	C1	1	2	
2	C2	1	2	
3	C3	1	1	2
4	C4	1	1	2
5	F5	1	1	2
6	F6	1	1	2

GROUP Blocks

Group #	Name	Resources	Rule
1	ALLAGNTS	{6,5,4,3,2,1}	ORDER
2	COACHAGNTS	{1,2,3,4}	LIDLE
3	FCAGNTS	{6,5}	ORDER

Figure 6: Resource and Group Blocks

A RESOURCE block defines each resource individually by number and name and resource capacity. Then the file numbers where entities wait for the resources are listed in a preferred order for allocation. The GROUP block defines a group number and name, the order in which individual resources are allocated from the group and the rule by which they are allocated. For example, for the GROUP FCAGNTS, agent F6 is allocated before agent F5 if both are idle. At an AWAIT node, multiple GROUPS can be listed so that combinations of sets of resources can be allocated to entities.

In Figure 7, the flow of entities is shown. The arrival time is assigned at the CREATE nodes and placed in ATRIB[0] for each arriving passenger entity. The entity is then routed to an AWAIT node where it waits, if necessary, in File 1 for priority service or File 2 for coach service. For priority service, a selection is made of one of the resources in GROUP FCAGNTS and the coach agents C4 and C3. The coach passenger entities wait in File 2 for one of the resources in GROUP COACHAGNTS or for an idle FCAGNTS. The agent or resource number allocated to a passenger is stored in LTRIB[0] as indicated at the AWAIT nodes. Activities 1 and 2 model the agent processing time. For priority passengers the processing time is uniformly distributed between 2 and 20. For coach passengers the processing time is triangularly distributed with parameters 3, 6 and 12.

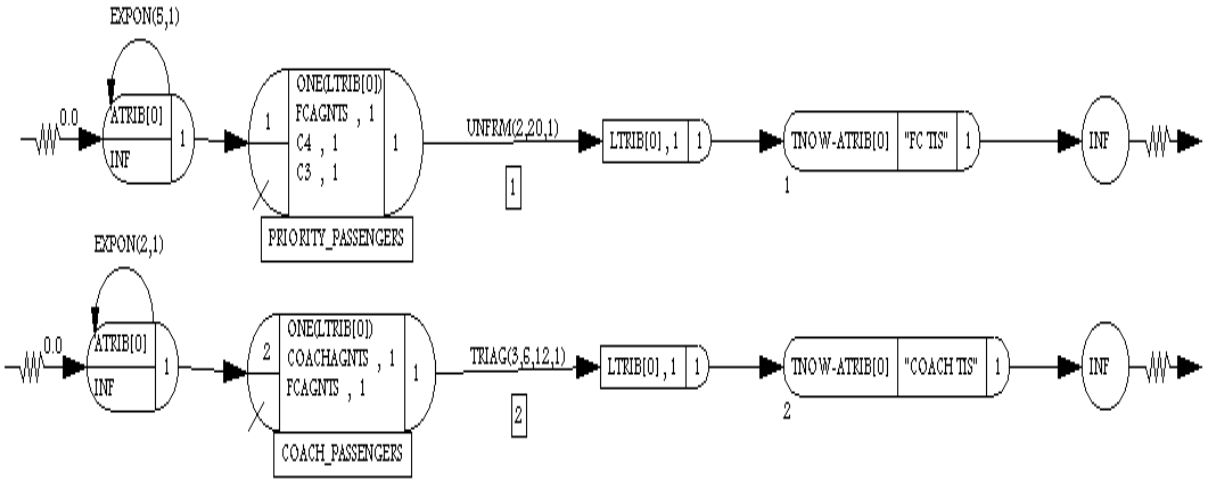

Figure 7: Network Model for Airport Agent Model.

After being processed, the passenger entity arrives at a FREE node where the resource allocated at the AWAIT node is freed. Next, the time in the system, TNOW-ATRIB[0], is collected at COLCT nodes which are numbered as 1 and 2. The entities are then terminated as they have received service from an agent.

6 MODELS WITH A SUBNETWORK

The Visual SLAM subnetwork (VSN) shown in Figure 8 is used to demonstrate the basic capabilities and concepts related to subnetworks. The VSN block provides a name, COMPRES, a computer resource for the subnetwork. COMPRES is used to make a computer resource available at different locations in a calling network or subnetwork. The parameters of COMPRES are COMPCAP, a computer resource capacity, and PROCTIME, a processing time for the computer activity.

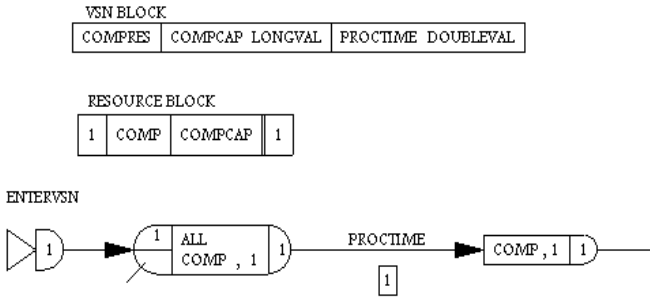

Figure 8: Subnetwork for Computer Processing

The RESOURCE block in the VSN specifies the label for resource 1 as COMP which has a capacity as defined by Parameter 1, that is, COMPCAP. The value of COMPCAP on the RESOURCE block for the instance CPU is set on the first call to VSN COMPRES through the process of instantiation. The first node of the VSN is an ENTERVSN node which is the start of the path of the entity in the subnetwork. Next the entity waits at an AWAIT node in file 1 for one unit of the resource COMP. When the entity is allocated the resource, it proceeds to activity 1 where Parameter 2, PROCTIME, provides a value for the processing time. Following the processing time activity, resource COMP is freed at a FREE node and the entity is returned from the VSN by the RETURNVSN node.

The VSN is called from the main network or a subnetwork when an entity arrives to a CALLVSN node. An example of a calling network with two calls to the VSN COMPRES from nodes CALL1 and CALL2 is shown in Figure 9.

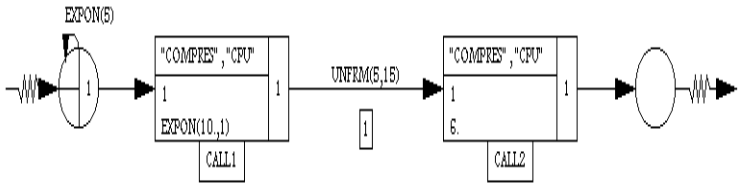

Figure 9: The Main Network.

At both CALLVSN nodes, the resource is identified as CPU which provides an instance of the VSN whose name is COMPRES. At node CALL1, the processing time is specified as EXPON(10.,1) and at node CALL2 the processing time is specified as 6. In this illustration, the entities arriving to the CALLVSN nodes both wait in file 1 for resource COMP with instance CPU in accordance with the priority rule for file 1. The processing time is taken as parameter 2 of the VSN. The resource capacity, COMPCAP, is taken from parameter 1. Since no special exit numbers are prescribed, the entity is returned to the CALLVSN node from which it came and normal branching from the CALLVSN node occurs.

A second calling network that uses the VSN COMPRES is shown in Figure 9 involving two resources, CPU1 and CPU2. The CALLVSN nodes CALL3 and CALL4 specify the instance of the subnetwork as CPU1 and CPU2, respectively. Each different instance of the subnetwork is a separate object with its own capacities, files, etc.

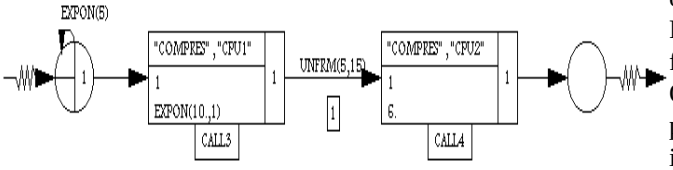

Figure 10: A Second Main Network.

In Figure 10, parameter 1 at both CALLVSN nodes indicates a capacity of 1. Parameter 2 specifies the processing times. Entities wait in file 1 but in a different instance of file 1. Statistics for file 1 would be collected separately for entities waiting for CPU1 from those waiting for CPU2.

If CPU2 has a capacity of 2 then parameter 1 at node CALL4 would be changed to a 2. Since there are two instances of the VSN COMPRES, the resource capacity COMPCAP is set on the first call of VSN COMPRES for each instance.

7 LIVER TRANSPLANTATION POLICY EVALUATION

Organs for transplantation are a scarce life-saving resource. The number of cadaveric donors has not kept pace with increasing demand. The United Network for Organ Sharing (UNOS) operates a nationwide system to procure and allocate human organs to potential recipients. Pritsker Corporation in conjunction with UNOS developed the UNOS Liver Allocation Model, ULAM, for assessing proposed policies for allocating livers (Pritsker et al., 1995, 1996). The material presented here referred to as ULAMjr is based on ULAM with major changes made to reduce the size and scope of the model.

ULAMjr illustrates the use of the AweSim integration capabilities to: provide input data for a model; store model outputs for presentation using Excel; and build animations illustrating the allocation of organs to waiting patients. Outputs are stored in an AweSim database to facilitate the computation and display of performance measures. A MATCH node is used to determine the patient to which an arriving liver is to be allocated. A ranking of entities in a queue based on primary and secondary priority variables is demonstrated with the first priority being status and the secondary ranking based on waiting time.

The ULAMjr model is shown in Figure 11. Only the novel features will be described in this paper (see Reference 1 for more details).

The arrival time of the liver entity is established as ATRIB[0] and the time between arrivals of liver organs is specified at the CREATE node. READ node READ0 is used to read a file which contains attributes for liver entities. The liver entity is then routed to QUEUE node Q2 where it is matched with a waiting patient. The coordinates of the donor hospital are read in as ATRIB[4] and ATRIB[5] and these are used to collect statistics on the donated liver travel distance and in an animation that shows the transfer of livers to patients throughout the United States.

The arrival of patients is separated into arrivals by status type. Prior to running the Visual SLAM program, files were constructed with the attributes necessary to describe the type of patient arriving for a given status. The random sampling from these files to represent a set of characteristics for patient entities is referred to as “bootstrapping.”

The patient entity is sent to an ASSIGN node where the patient’s current status is set. The value of the next status for the patient, and the time at which the status change will occur, are set in a user function.

For each status code, patient entities are routed to QUEUE node Q1 where they are inserted into the file 1 waiting list in accordance with the file's priority ranking rule. A MATCH node MAT is used to match an arriving liver entity with a waiting patient with the same blood type who has the highest value of the ranking expression. The matched entities are routed from the MATCH node to an ACCUMULATE node where the attributes of the two entities are combined. The transplanted patient entity is then routed to COLCT node DISP where the distance between the donor hospital and the patient transplant center is collected and the patient entity is routed according to

the status code at which the patient was transplanted. Three COLCT nodes are used to obtain the waiting time until transplant by status type. Following each COLCT node is an activity which has a condition that represents the 2-year survivability of the transplanted patient.

A disjoint network is used to model the status change process. To accomplish this reviewing of patients, a CREATE node creates one entity at 4 a.m. of the first simulated day and routes it to FINDAR node SEARCH. The FINDAR node initiates a search through file 1 and a testing of the value of LTRIB[2] to determine whether it is less than or equal to TNOW.

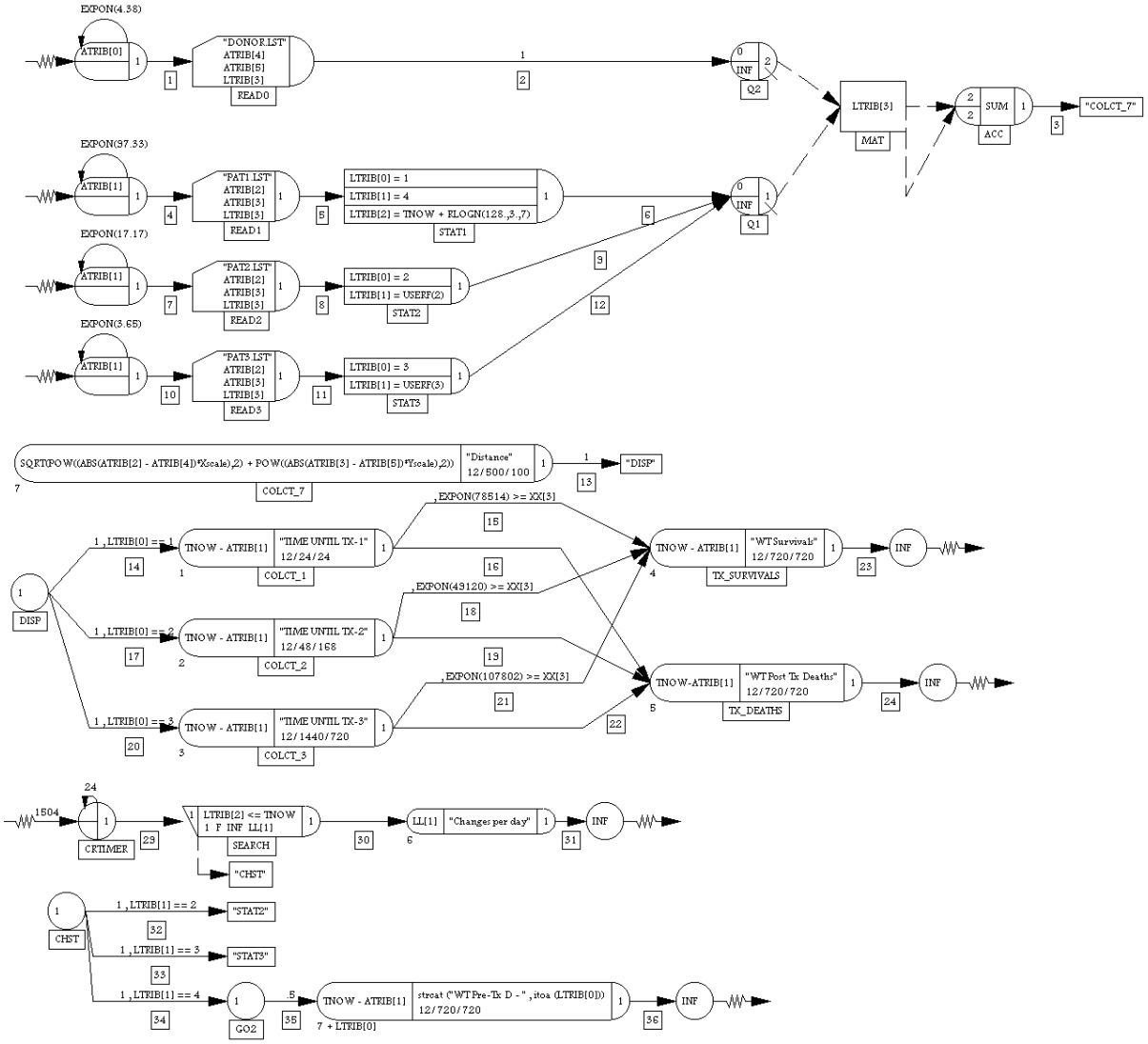

Figure 11: Visual SLAM Network Model for Transplantation Analysis.

If the value is less than or equal to TNOW, the patient entity that is making a status change is removed from file 1 and routed to GOON node CHST where it is further routed to ASSIGN nodes, STAT2 or STAT3, based on its changed status code, LTRIB[2]. At nodes STAT2 and STAT3, the current status of the patient is reset and the next transition state and time are established. If the patient is making a transition to Status 4, the death state, then statistics are collected on the number and waiting time of a pre-transplant death categorized by status at death. The pre-transplant death waiting times are collected at a single COLCT node by making the COLCT node number an expression, 7+LTRIB[0], and concatenating a string with LTRIB[0] as the output identifier. This entity is then terminated from the simulation.

Function INTLC is used to provide an initial list of 200 patients in file 1. Fifteen hundred hours was

selected as the start time of the simulation because the earliest arrival time of the 200 patients was within this 1500 hour time interval.

The policy being simulated involves a single national list in which patients are allocated a liver based on their medical status first and patients of the same status selected based on longest waiting time. The allocation requires identical blood matching between the organ donor and the patient. Livers were available to transplant approximately 66% of the arriving and initial patients. Of the patients transplanted, 3,156 survived for greater than two years to give a transplantation survival fraction of 0.778.

There were 900 patients who died after transplantation and 699 patients who died while waiting for a transplant. This constitutes 26% of the patients.

Although the data in this example is hypothetical, it does present the dilemma involved in allocating a scarce life-saving resource. In actuality, there is a great need to increase the number of donors. This can be done by making the general public aware that each donor can save as many as five lives of terminally ill patients. Donation of organs needs to be discussed among family members.

There are many performance measures that need to be taken into account in selecting a liver allocation policy. A policy must balance the utility of an organ placement with the equity involved in treating all patients in the same manner. Simulation is a tool that has supported policy selection in this complex medical environment.

8 INTEGRATION OF AWESIM WITH OTHER SOFTWARE

AweSim was designed to integrate easily with other Windows applications. AweSim is built on a relational database which is accessible with standard tools such as Dbase, Access, FoxPro and Excel. Input data is easily moved from an Excel worksheet to the AweSim input tables. Output data is stored in AweSim output tables, available for creating custom reports using a favorite tool. In addition to standard output data, raw data from the simulation can be stored in standard database or Excel format for analysis, manipulation, or presentation. Data used to create AweSim output graphics can also be exported “on the fly” to an output file for use in any tool accepting comma-delimited input.

An AweSim animation can use graphics created from other programs. As mentioned in the discussion of animations, the graphical elements manipulated by the animator can be created using CAD, drawing or paint programs and loaded into AweSim by using the Windows bitmap format. The output charts and plots created by AweSim can be exported via the clipboard to other applications. For example, a pie chart created by AweSim may be copied to the clipboard and pasted into a word processing document describing the results of the model.

9 SUMMARY

AweSim is a new simulation support system. which takes advantage of the latest Windows technology in order to provide a simulation support system able to interface with a variety of familiar tools. It incorporates the Visual SLAM modeling methodology. AweSim is distributed by Pritsker Corporation, which offers regularly scheduled training classes as well as applications support.

REFERENCES

Pritsker, A.A.B. et al, "Organ Transplantation Policy Evaluation", *Proceedings, 1995 Winter Simulation Conference*, pp. 1314-1323.

Pritsker, A.A.B., O.P. Daily and K.P. Pritsker, "Using Simulation to Craft a National Organ Transplantation Policy", *Proceedings, 1996 Winter Simulation Conference*, pp. 1163-1169.

Pritsker, A.A.B., J.J. O'Reilly and D.K. LaVal, *Simulation with Visual SLAM and AweSim*, John Wiley and Systems Publishing Corporation, 1997.

Pritsker, A.A.B., J.J. O'Reilly, *Solutions Manual for Simulation with Visual SLAM and AweSim*, Systems Publishing Corporation, 1997.

Pritsker Corporation, *AweSim! User's Guide*, West Lafayette, IN, 1997.

Pritsker Corporation, *Visual SLAM Quick Reference Manual*, West Lafayette, IN, 1997.

AUTHOR BIOGRAPHIES

A. ALAN B. PRITSKER is Chairman of the Board of Pritsker Corporation. He obtained a Ph.D. from The Ohio State University in 1961. In 1967, he was awarded the IIE Distinguished Research Award for his pioneering work on network simulation languages. In 1985, he was elected to the National Academy of Engineering for the basic concepts underlying discrete/continuous simulation languages. Dr. Pritsker served twice as a member of the Board of Directors of the WSC and as Board Chairman in 1984 and 1985.