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

ALPHA/SIM SIMULATION SOFTWARE TUTORIAL

Kendra E. Moore
John E. Brennan

ALPHATECH, Inc.
50 Mall Road
Burlington, MA 01803-4562, U.S.A.

ABSTRACT

ALPHA/Sim is a general-purpose, discrete-event simulation tool. ALPHA/Sim allows a user to graphically build a simulation model, enter input data via integrated forms, execute the simulation model, and view the simulation results, within a single graphical environment. In this paper, we introduce ALPHA/Sim and describe how to use ALPHA/Sim to build, simulate, and analyze a simple manufacturing system. In addition, we briefly describe some advanced features and list some sample applications.

1 INTRODUCTION

ALPHA/Sim is a general-purpose, discrete-event simulation tool. With ALPHA/Sim you can graphically build a simulation model, enter input data (timing delays, routing rules, initial conditions, and other data) via integrated forms, execute the simulation model, and view the simulation results, within a single graphical environment.

ALPHA/Sim provides a hierarchical modeling capability that allows models to be built from the bottom-up, top-down, or both. Models can be built without seeing or writing a single line of code; it is also possible to link to external software. ALPHA/Sim automatically collects statistics on populations (queues), delays, activity rates, and attributes.

ALPHA/Sim has been used in a wide number of applications including computer hardware systems, manufacturing systems, queuing systems, and military command and control. ALPHA/Sim currently runs on the PC (Windows NT) and Sun Workstation (SunOS and Solaris under the X Window System or Motif).

The modeling paradigm used in ALPHA/Sim is based on Petri nets (PNs). PNs were developed in the early 1960s to model concurrent operations in computer systems. Over the years PNs have been extended and applied to a wide range of systems characterized as being concurrent, asynchronous, distributed, parallel, and stochastic. PNs are a mathematical and graphical modeling tool. As a mathematical tool, PNs can be used to set up state equations, algebraic equations, and simulation models. As a graphical tool, PNs provide a visual modeling technique.

In this paper, we present a brief overview of PNs and describe how to use ALPHA/Sim to implement a simple manufacturing model. Specifically, we describe how to build the graphical model and define attributes, token types, timing delays, decision rules, output attribute definitions, and statistics collection. In addition, we briefly describe some advanced features and list some sample applications.

2 PETRI NETS

Petri nets (PNs) are a graphical and mathematical modeling technique originally developed by C.A. Petri in the early 1960s to characterize concurrent operations in computer systems (Petri 1962). PNs have been extended to capture many important aspects of large-scale systems, including attributes, timing relationships, and stochastic events (Moore and Lynch 1990, Moore et al. 1986, Murata 1989, Peterson 1981). The greatest appeal of PNs is their conceptual simplicity.

PNs consist of four primitive elements (tokens, places, transitions, and arcs) and the rules that govern their operation (Figure 1). PNs are based on a vision of *tokens* moving around a network. Tokens appear as dots and represent the objects or entities in a system. *Places* are shown as circles and represent the locations where objects await processing. Location can be either a physical location (e.g., the queue where a message waits to be processed) or a state (e.g., an idle resource). *Transitions* appear as rectangles and represent processes or events (e.g., processing a message or machining a part). Finally, *arcs* represent the paths of objects through the system. Arcs connect places to transitions and transitions to places; the direction of the path is indicated by an arrowhead at the end of the arc.

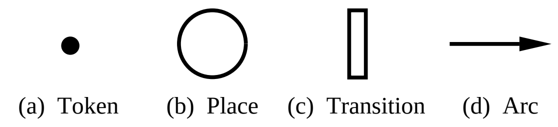


Figure 1: Depiction of PN Primitives

PN firing rules specify the behavior of transitions; i.e., the conditions under which processes or events can occur. Three rules govern transition firing:

1. When all upstream places are occupied by at least one token, the transition is *enabled*.
2. Once enabled, the transition *fires*.
3. When a transition fires, exactly one token is removed from each upstream place and exactly one token is placed in each downstream place.

Figure 2 depicts these rules for a transition (Assemble) with two upstream places (Part A, Part B) and one downstream place (Assembly).

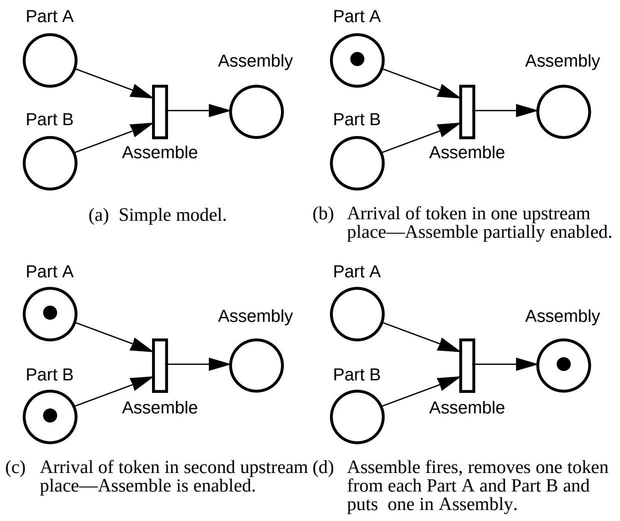

Part A
Assembly
Part B
Assemble
(a) Simple model.
(b) Arrival of token in one upstream place—Assemble partially enabled.
(c) Arrival of token in second upstream place—Assemble is enabled.
(d) Assemble fires, removes one token from each Part A and Part B and puts one in Assembly.

Figure 2: Transition Firing

Timing rules are associated with transitions and represent the time required to complete some activity. A timing rule may be stochastic, based on an assigned probability function, a computed value, or a constant. *Decision rules* are associated with places and resolve cases where more than one transition is enabled by the same token or set of tokens. There are three types of decision rules: priority, probability, and constructed. The *priority* decision rule (shown in Figure 3) states that, if all other firing rules are met, the token will leave by the path with the highest priority. The *probability* decision rule states that if all other firing rules are met, the token will select a path based on assigned probabilities. The *constructed* decision rule allows the user to specify the conditions under which the token will select a particular path, given that the firing rules are met.

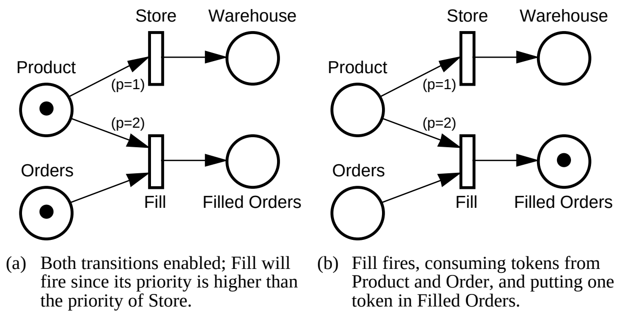

Store
Warehouse
Product
(p=1)
(p=2)
Orders
Fill
Filled Orders
(a) Both transitions enabled; Fill will fire since its priority is higher than the priority of Store.
(b) Fill fires, consuming tokens from Product and Order, and putting one token in Filled Orders.

Figure 3: Effect of a Priority Decision Rule

Attributes on tokens are used to specify a set of characteristics associated with a token (e.g., size, type, priority, identity, etc.). The values of the attributes may be changed at transitions. They can also be used to determine timing and decision rules. Finally, the values of the attributes can be passed to external algorithms and the results incorporated into the PN model.

There are two other types of arcs, in addition to the standard arc, which provide complex transition logic. The *enable* arc is depicted as a line with a solid filled circle at the end where the arrowhead normally appears. The enable arc enables a transition when the upstream place has a token, but does not consume the token (the token remains in the upstream place). The *inhibit* arc is depicted as a line with a hollow circle at the end where the arrowhead normally appears. The inhibit arc disables a transition when the upstream place has a token in it (the token is not consumed along the inhibit arc).

Box nodes are used to encapsulate portions of a PN model and to provide a hierarchical modeling capability. Box nodes are used to group or cluster PN fragments that relate to various subsystems, functions, or organizational units.

3 BUILDING MODELS WITH ALPHA/Sim

With ALPHA/Sim you can: build and debug your models graphically; build models from the top-down, bottom-up, or both; easily modify model parameters and structure; navigate through the model; monitor results at any point in the simulation run; and save any model component for reuse in other models.

In the remainder of this section we illustrate how to use ALPHA/Sim to implement a simple manufacturing system which produces two types of parts. Type 1 parts are turned, milled, and plated, in that order; Type 2 parts are turned and milled. Input parameters include part mix and part processing times. Output parameters include buffer sizes (queue lengths), machine utilization, part latency, and throughput.

3.1 Drawing the Graphical Model

We begin by drawing the graphical model. Figure 4 shows an ALPHA/Sim screen. The screen has a menu bar at the top, an icon palette on the left, and a drawing window with scroll bars. We create the graphical model by using the mouse and icon palette to drop icons in the drawing window and connect them with arcs. Icons are automatically assigned default names (see Figure 4); these can be changed to be more meaningful.

Figure 5 shows the complete manufacturing model. The place-transition combination in the upper left corner periodically generates new parts into the place labeled Lathe_Q. Arriving parts wait in the Lathe_Q, until the Lathe becomes available. They are then turned and enter the Mill_Q, where they wait for the Mill to become available. Once the parts are milled, they are passed to the Plate_Q. Since Type 2 parts are done, they are immediately routed to the stock of Finished_Type_2 parts; meanwhile, Type 1 parts remain in the Plate_Q until the Plating_Machine becomes available. Once plating is complete, the Type 1 parts enter the Finished_Type_1 stock.

3.2 Defining Token Types

Once we have built the graphical model, we can define the token types using the Token Type Edit Form. For this model, we will use two token types: Part and Machine. Figure 6 shows the Token Type Edit Form for the Part token type. This form contains a field for identifying the token type and allows us to define the attributes associated with the token type. Each attribute definition consists of a name, class, type, and an initial range.

The attribute's class is a scalar (single) value, an array of values, or a matrix of values. If an attribute's class is array or matrix, we must also specify its size (rows and columns). The attribute's type refers to its format. Valid types include: Boolean, integer, real, string, or another (previously defined) token type. If the type is not another token type, we have the option of specifying an initial range for the attribute. Table 1 summarizes the token type definitions for the manufacturing model.

3.3 Place and Transition Forms

Once we have defined the token types, we can use the place and transition forms to assign token types to places and specify timing, routing, and other logical rules. Figure 7 shows a sample place form. The top of the place form lists the input and output transitions, and

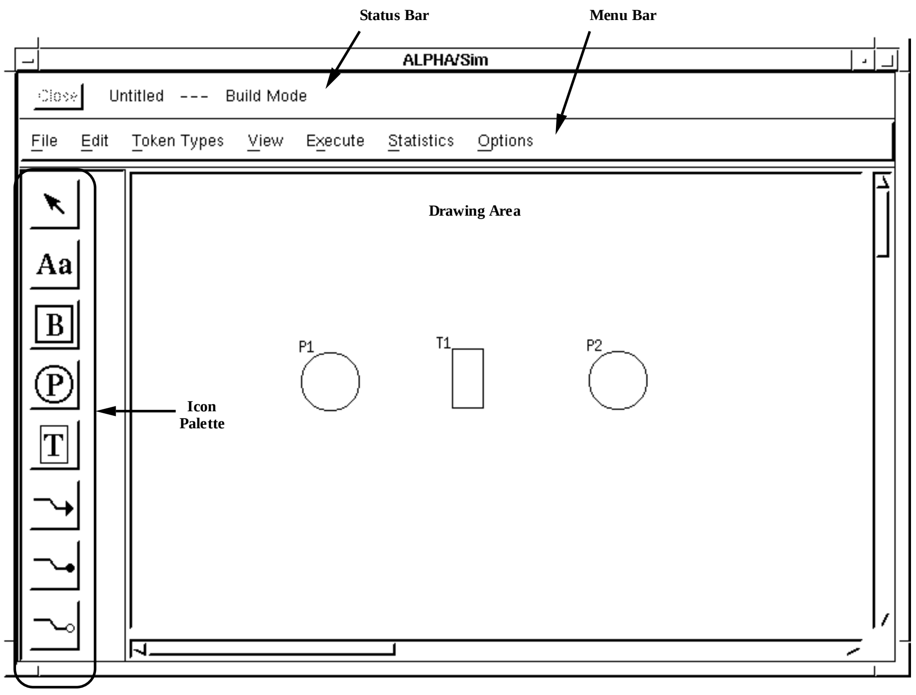


Figure 4: Sample ALPHA/Sim Screen

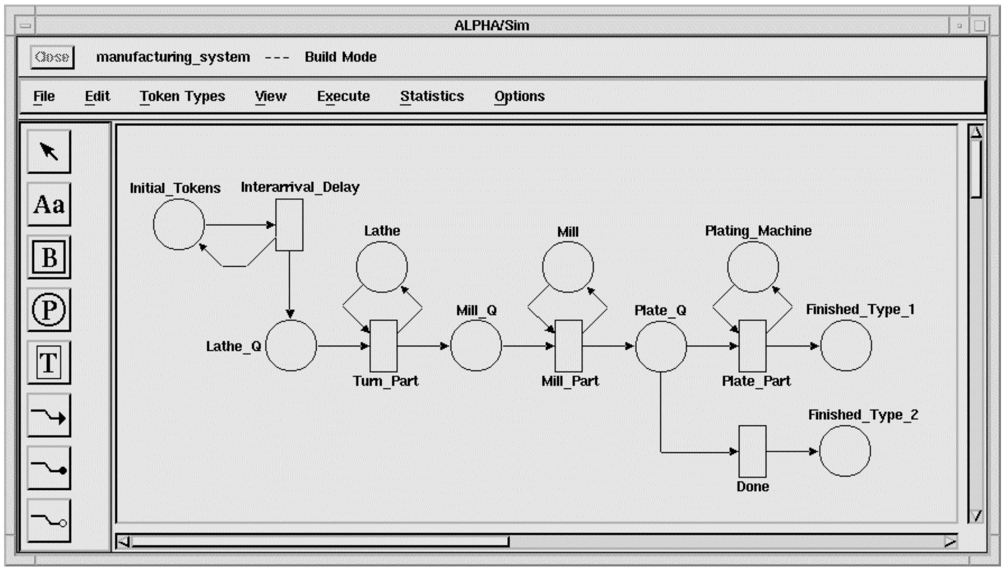
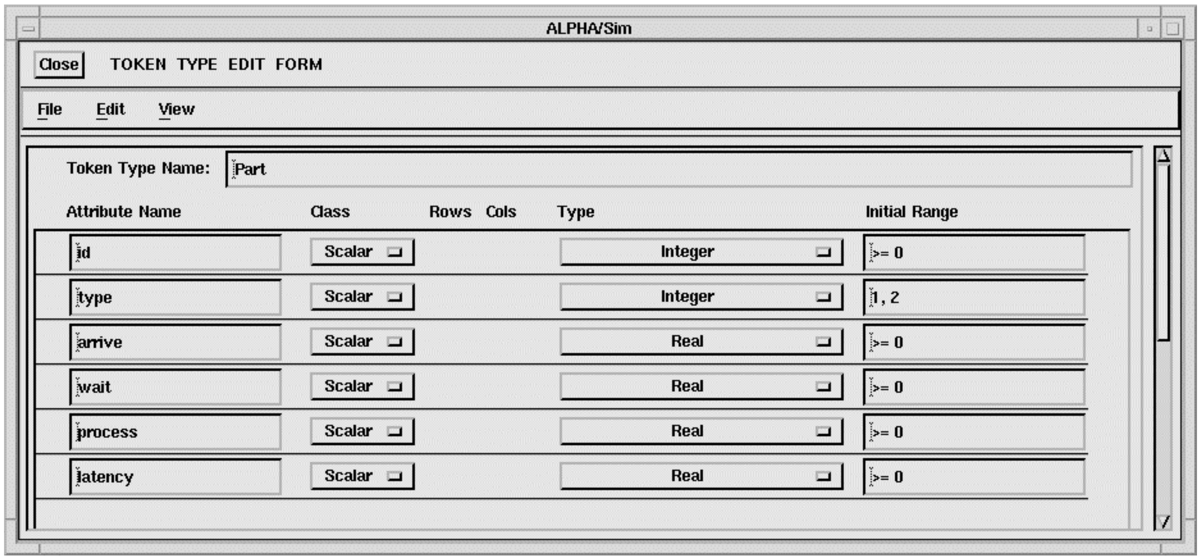


Figure 5: Completed Model



Token Type Name: Part

Attribute Name	Class	Rows	Cols	Type	Initial Range
id	Scalar			Integer	>= 0
type	Scalar			Integer	1, 2
arrive	Scalar			Real	>= 0
wait	Scalar			Real	>= 0
process	Scalar			Real	>= 0
latency	Scalar			Real	>= 0

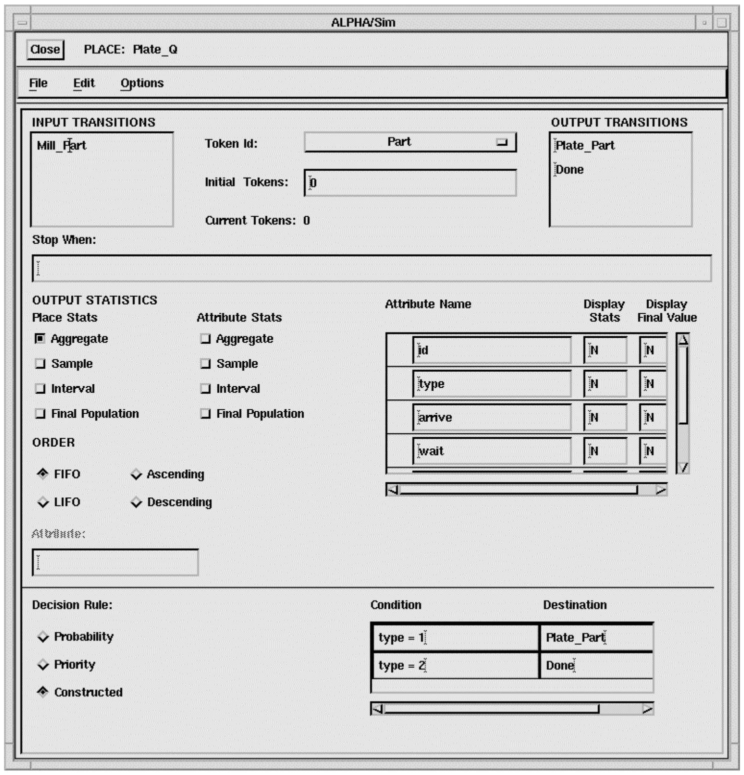
Figure 6: Token Type Edit Form for the Parts Token Type

Table 1: Token Type Definitions for Model

Token Type	Attributes	Class	Type
Part	id	Scalar	Integer
	type	Scalar	String
	arrive	Scalar	Real
	wait	Scalar	Real
	process	Scalar	Real
	latency	Scalar	Real
Machine	id	Scalar	Integer
	type	Scalar	String
	p_time	Scalar	Real

allows us to specify the token type and the number of initial tokens. The middle of the form allows us to specify statistics collection and set the queuing order (FIFO, LIFO, or ascending/descending on an attribute value). The bottom of the form allows us to set decision rules (priority, probability, or constructed) for routing tokens out of the place.

Figure 8 shows a sample transition form. The left side of the transition form lists the input places; clicking on one of these places opens the input token profile displaying the input token type definition. Similarly, the right side of the form lists the output places and clicking


ALPHA/Sim

Close PLACE: Plate_Q

File Edit Options

INPUT TRANSITIONS: Mill_Part

Token Id: Part

Initial Tokens: 0

Current Tokens: 0

OUTPUT TRANSITIONS: Plate_Part, Done

Stop When:

OUTPUT STATISTICS

Place Stats	Attribute Stats
[✓ Aggregate]	Aggregate
Sample	Sample
Interval	Interval
Final Population	Final Population

Attribute Name	Display Stats	Display Final Value
id	N	N
type	N	N
arrive	N	N
wait	N	N

ORDER

[✓ FIFO] Ascending

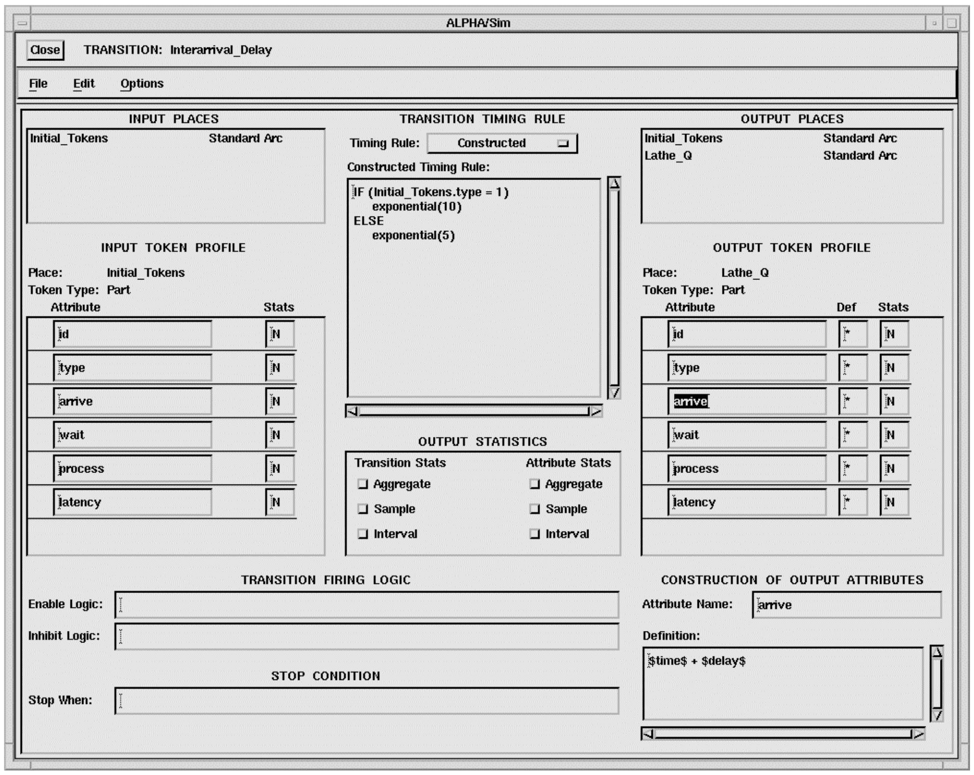
LIFO Descending

Attribute:

Decision Rule: Probability, Priority, [✓ Constructed]

Condition	Destination
type = 1	Plate_Part
type = 2	Done

Figure 7: Place Form for the Plate_Q Place


ALPHA/Sim

Close TRANSITION: Interarrival_Delay

File Edit Options

INPUT PLACES

Initial_Tokens	Standard Arc

TRANSITION TIMING RULE

Timing Rule: Constructed

Constructed Timing Rule:

```
IF (Initial_Tokens.type = 1)
    exponential(10)
ELSE
    exponential(5)
```

OUTPUT PLACES

Initial_Tokens	Standard Arc
Lathe_Q	Standard Arc

INPUT TOKEN PROFILE

Place: Initial_Tokens

Token Type: Part

Attribute	Stats
id	N
type	N
arrive	N
wait	N
process	N
latency	N

OUTPUT STATISTICS

Transition Stats	Attribute Stats
Aggregate	Aggregate
Sample	Sample
Interval	Interval

OUTPUT TOKEN PROFILE

Place: Lathe_Q

Token Type: Part

Attribute	Def	Stats
id	*	N
type	*	N
arrive	*	N
wait	*	N
process	*	N
latency	*	N

TRANSITION FIRING LOGIC

Enable Logic:

Inhibit Logic:

STOP CONDITION

Stop When:

CONSTRUCTION OF OUTPUT ATTRIBUTES

Attribute Name: arrive

Definition:

```
$time$ + $delay$
```

Figure 8: Transition Form for the Interarrival_Delay Transition

on one opens the output token profile. Ordinarily, the input attribute values are mapped to the output attribute values; however, we can assign new values to these attributes using the area below the output token profile.

The center of the transition form is used to set a timing rule and to specify statistics collection. Regarding the timing rules, we can choose None, Selected Distribution, or Constructed. If we choose Selected Distribution, we are prompted to select one of the available distributions and provide the appropriate parameters. Table 2 lists these distributions and their parameters. If we choose Constructed, we can enter an expression utilizing other distributions or attribute values. The language used for the expression is English-like; e.g., the timing rule in the Interarrival_Delay transition is:

```
IF (Initial_Token.type = 1)
   exponential(10)
ELSE exponential(5)
```

“Initial_Token.type” is the value of the attribute “type” on the token coming from the place “Initial_Token”. The lower left corner of the transition form is used to set specific enabling or inhibit logic (using attributes) or conditions for stopping the simulation.

Table 2: Built-In Timing Distributions

Distribution	Parameters
Constant	Value
Exponential	Mean
Gamma	Alpha, Beta
Normal	Mean, Std Dev, Min, Max
Triangular	Min, Mode, Max
Uniform	Min, Max

3.4 Specifying the Model Logic via the Forms

We use these forms to associate the token types with places, and specify the initial tokens, the decision (routing) rules, timing rules, and output attribute definitions. First, we associate token types with places using the Token Id option menu in the Place Forms. At this time, we set two initial tokens in Initial_Token (one for each type of part), and one initial token in each of the machine places (Lathe, Mill, and Plating_Machine). Table 3 lists the token type assignments and initial populations for each place in the model; Table 4 lists the initial values of the attributes for those places with initial token populations.

Next, we specify a constructed decision rule for the place labeled Plating_Q to route Type 1 and Type 2 parts. Figure 7 shows the decision rule used for routing parts as it appears in the Plate_Q place form. Alternatively, we can use enabling logic in the Plate and Done transitions. If desired, we can set a queuing order in the Lathe_Q and Mill_Q places.

Table 3: Token Type Assignments and Initial Populations

Token Type	Places	# Init. Tokens
Part	Initial_Token	2
	Lathe_Q, Mill_Q, Plate_Q, Finished_Type_1, Finished_Type_2	0
Machine	Lathe, Mill, Plating_Machine	1

Table 4: Initial Token Values

Place	Attributes	Initial Value
Initial_Token	type	1 or 2
	all others	0
Machine	id	Unique integer
	type	“lathe”, “mill”, “plating”

Finally, we set various output attribute definitions to collect information on the machines and on individual parts as they pass through the system. ALPHA/Sim provides four system variables that can be used in expressions; these are \$time\$ (the current simulation time), \$delay\$ (the timing delay of an individual transition firing), \$count\$ (the number of times a specified transition has fired), and \$pop\$ (the current number of tokens in a place). Since we are interested in the queuing, service, and system times for the customers, we will make use of the \$time\$ and \$delay\$ variables. Table 5 lists the expressions that are used in output attribute definitions; Fig. 8 shows a sample output attribute definition in a transition form (lower right). In addition, we can turn on statistics collection for places (average population), transitions (firing rates), and attributes, using the place and transition form statistics panels (see Figs. 7 and 8).

3.5 Controlling the Simulation Run

Additional forms are available to set the simulation run time, the number of replications and random number seeds, and statistics collection preferences. ALPHA/Sim has facilities for collecting aggregate, interval, and sample statistics. At runtime, ALPHA/Sim checks all

Table 5: Sample Output Attribute Definitions

Transition	Attribute	Definition
Interarrival_ Delay	id	\$count\$
	arrive	\$time\$ + \$delay\$
Turn	wait	\$time\$ - arrive
	process	\$delay\$
Mill	wait	\$time\$ - arrive - process
	process	process + \$delay\$
Plate	wait	\$time\$ - arrive - process
	process	process + \$delay\$
	latency	\$time\$ - \$delay\$ - arrive
Done	latency	\$time\$ - arrive

expressions to make sure that there are no errors and executes the simulation. The results can be observed on-screen or sent to a file for further analysis. The simulation can also be run in batch mode.

4 ADVANCED FEATURES

ALPHA/Sim incorporates a number of additional features. These include: functions, enable and inhibit logic, stop when conditions, boxes, show tree, and various printing and file handling features. ALPHA/Sim includes over thirty built-in mathematical functions as well as arithmetic and logical operators that can be used in timing rules, decision rules, output attribute definitions, and other expressions. In addition, it is possible to incorporate user-defined functions and interact with external code. Enable and inhibit logic can be used in transition forms to specify which combinations of tokens will cause a transition to fire. Stop when conditions are logical expressions that can be used to halt the simulation if a specified condition is reached. Boxes provide a hierarchical modeling capability. Show tree allows you to view a model's hierarchy in a tree structure and provides an easy way to navigate through a model. The graphical model and the information contained in the forms can be printed out to a laser printer or sent to a file.

5 SAMPLE APPLICATIONS

ALPHA/Sim has been used to develop a wide array of discrete-event simulation models: computer components and systems (Ethernet system (Brennan, Walenty, and Moore 1995), client-server system, and high-speed disk systems), manufacturing systems (Moore and Gupta 1996), large-scale military command and control systems (Moore and Lynch 1990), and business process reengineering and workflow models for a charter air cargo and passenger service system.

The client-server system consists of several data processing nodes connected via a local area network (LAN). The model evaluates the impact of changing the number of hardware components and their capabilities on throughput and latency for individual processes. It also identifies bottlenecks in the system, thereby indicating good candidates for increasing capacity.

The manufacturing system of (Moore and Gupta 1996) is constructed from a series of predefined modules to generate a serial-line just-in-time (JIT) production system. A JIT model consists of four basic types of modules: a Supply module, a Demand module, some number of workstation modules, and an optional kanban control module. These modules may be combined to form both single- and multiple-product JIT systems of arbitrary length. The resulting model can be used to determine the impact of the demand, processing times, line length, the number of kanbans, and kanban control policies on system performance. In addition, the model can be used to explore the impact of variations in processing and demand on system performance. System performance measures include: throughput, part time in system, order completion time, and work-in-process.

The charter air cargo and passenger service model depicts the workflow for a thirty person office responsible for handling and scheduling domestic and international transportation. This workflow is unique in that the staff's activities are frequently interrupted by higher priority tasks and phone calls or delayed due to communications interruptions. The model was used to determine the impact of automation and task redefinition on staffing requirements, particularly overtime requirements, and throughput.

6 SUMMARY AND CONCLUSIONS

In this paper, we described a general-purpose, discrete-event simulation software tool called ALPHA/Sim. With ALPHA/Sim you can: build and debug your models graphically; build models from the top-down, bottom-up, or both; easily modify model parameters and structure; navigate through the model; monitor results at any point in the simulation run; and save any model component for reuse in other models.

With ALPHA/Sim's graphical modeling and simulation environment it is possible to develop and exercise simulation models without having to see or write a line of code. The graphical interface allows you to design the model using the mouse and icons. Integrated forms provide the means for specifying logic and input parameters for the model. ALPHA/Sim also provides the ability to interface with external software.

We described how to use this tool via a simple example of a manufacturing system. This example illustrates the key features of ALPHA/Sim. In addition, we briefly listed some of the advanced features of the tool. We also listed a number of sample applications and briefly described three of these, namely a client-server performance model, a serial-line JIT production system, and a business process workflow model.

ACKNOWLEDGMENTS

ALPHA/Sim is a trademark of ALPHATECH, Inc. Motif is a registered trademark of Open Software Foundation, Inc. Sun OS is a trademark of Sun Microsystems, Inc. Sun Workstation is a registered trademark of Sun Microsystems, Inc. UNIX is a registered trademark of UNIX Systems Labs, Inc. Windows NT is a trademark of Microsoft Corporation. X Window System is a trademark of the Massachusetts Institute of Technology.

REFERENCES

Brennan, J.E., M.E. Walenty, and K.E. Moore. 1995. Simulating a UNIX processor using Petri nets and ALPHA/Sim. In *Proceedings of the 1995 Summer Computer Simulation Conference*, ed. T. I. Oren and L. G. Birta, 63-69. Society for Computer Simulation, San Diego, CA.

Moore, K.E. and S.M. Gupta. 1996. Petri net-based analysis and simulation of traditional and flexible kanban control policies, ed. K. Stelson In *Proceedings of the Japan-USA Symposium on Flexible Automation*. American Society of Mechanical Engineers, New York, NY.

Moore, K.E. and J.P. Lynch. 1990. Stochastic, timed, attributed Petri net (STAPN) modeling of antisubmarine warfare C^3 architectures. In *Proceedings of the 1990 Symposium on Command and Control Research*, 311-325. SAIC, McLean, VA.

Moore, K.E., R.R. Tenney, P.A. Vail. 1986. Systematic evaluation of command and control systems. Technical Report TR-284, ALPHATECH, Inc., Burlington, MA.

Murata, T. 1989. Petri nets: properties, analysis and applications. *Proceedings of the IEEE* 77: 541-580.

Peterson, J. L. 1981. *Petri net theory and the modeling of systems*. Englewood Cliffs, NJ: Prentice-Hall.

Petri, C.A. 1962. *Kommunikation mit automaten*. Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 3, Bonn, Germany. Also, English translation, “Communication with automata.” Technical Report RADC-TR-65-377, vol. 1, Suppl. 1, January 1966, Griffiss Air Force Base, NY.

AUTHOR BIOGRAPHIES

KENDRA E. MOORE is the Simulation Products and Services manager at ALPHATECH, Inc., in Burlington, Massachusetts. She received her MS degree (1989) in operations research from the Department of Mechanical, Industrial, and Manufacturing Engineering at Northeastern University in Boston, Massachusetts. She is currently pursuing a PhD in operations research at Northeastern; she is affiliated with the Laboratory for Responsible Manufacturing and her research is focused on using Petri nets to model flexible manufacturing systems. She has a MA degree (1985) in philosophy of religion from Columbia University in New York City, and a BA degree (1981) in philosophy and religion from Stephens College in Columbia, Missouri. Since joining ALPHATECH in 1985, Ms. Moore has been actively involved in developing and applying simulation and modeling techniques and tools. Her areas of interest are Petri nets, discrete-event simulation, optimization, manufacturing systems, business process reengineering, and performance analysis.

JOHN E. BRENNAN is a senior simulation engineer at ALPHATECH, Inc., in Burlington, Massachusetts. He holds a BS degree (1981) from Catholic University in Washington, DC and an MS degree (1989) in operations research from the College of Business and Management at the University of Maryland at College Park. Since joining ALPHATECH in 1994, he has been involved in modeling and analysis of hardware and software architectures to support business process reengineering. Prior to joining ALPHATECH, Mr. Brennan was an engineer with the American Red Cross Biomedical Services division involved in systems design and analysis to support process improvement efforts in the collection, testing, manufacture, and distribution of blood and blood products. His areas of interest include simulation, decision support systems, process improvement, and health care applications.