Proceedings of the 1999 Winter Simulation Conference
P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds.

ALPHA/SIM SIMULATION SOFTWARE TUTORIAL

Kendra E. Moore
Jack C. Chiang
Scott D. Hammer

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 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 can also 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 variety 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 vice-versa; the arrowhead at the end of the arc indicates the direction of the path.

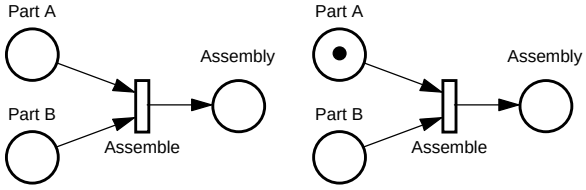
(a) Token (b) Place (c) Transition (d) Arc

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 consumed from each upstream place and exactly one token is deposited 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*).



(a) Two upstream places and one downstream place connected to a single transition.

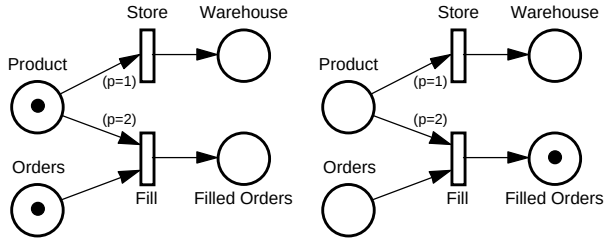
(b) Arrival of token at *Part A* partially enables *Assemble*.

(c) Arrival of token at *Part B* fully enables *Assemble*.

(d) *Assemble* fires, consumes one token each from *Part A* and *Part B*, and deposits one token 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 via the path having 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 condition under which the token will select a particular path, if all other firing rules are met.



(a) Both transitions enabled; *Fill* will fire since it has a higher priority than *Store*.

(b) *Fill* fires; consumes one token each from *Product* and *Order*, and deposits 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 output 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.

In addition to a standard arc, there are two other arc types which allow for more complicated 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 if the upstream place is occupied by 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 if the upstream place is occupied by a token, but does not consume the token (the token remains in the upstream place).

Box nodes are used to encapsulate portions of a PN model and to provide a hierarchical modeling capability. Box nodes group or cluster PN fragments of related 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 structures; navigate through the model; monitor results at any point in the simulation run; and save any model component for reuse in other models.

The remainder of this section illustrates how to use ALPHA/Sim to implement a simple manufacturing system that 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 with the status bar and the menu bar at the top, an icon palette to the left, and the drawing window with scroll bars at the bottom and to right. We create the graphical model by using the mouse to drag-and-drop icons from the icon palette onto the drawing area, then connecting the icons with arcs. Icons are automatically assigned unique default names (see Figure 4) which can be changed to something more meaningful.

Figure 5 shows the complete manufacturing model. The place-transition combination in the upper-left corner periodically generates new parts that are deposited into the *Lathe_Q* place. Arriving parts wait at 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 routed to the stock of *Finished_Type_2* parts. "Type 1" parts remain in the *Plate_Q* until the *Plating_Machine* is available. Once plating is complete, the "Type 1" parts enter the *Finished_Type_1* stock.

3.2 Defining Token Types

Once the graphical model is built, we use the Token Type Edit Form to define the token types; for this model, we have two: "Part" and "Machine". Figure 6 shows the Token Type Edit Form for "Part". This form contains a field for naming the token type and a field for defining the associated attributes. Each attribute definition consists of a name, class, type, and an initial range.

The attribute's class can be a scalar (single) value, an array of values, or a matrix of values; if it is an array or matrix, we must specify its size (the number of rows and columns). The attribute's type can be Boolean, integer, real, string, or another (previously defined) token type. If the type is not another token type, we can specify an initial range for the attribute, if desired. Table 1 lists the token type definitions for the manufacturing model.

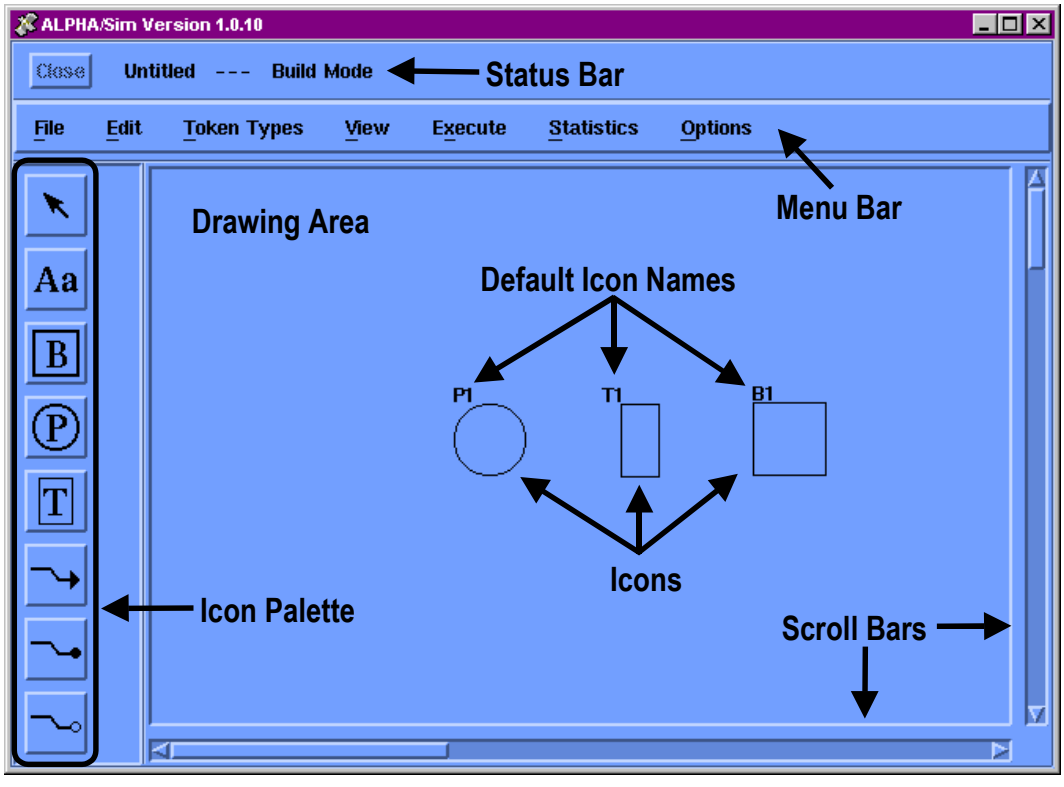


Figure 4: Sample ALPHA/Sim Screen

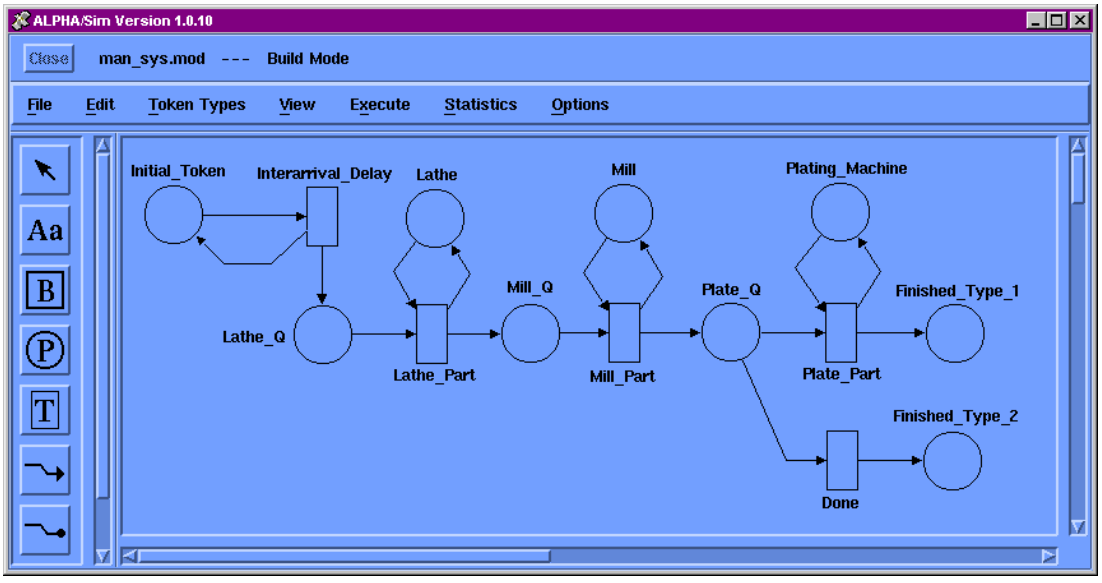


Figure 5: Manufacturing Model

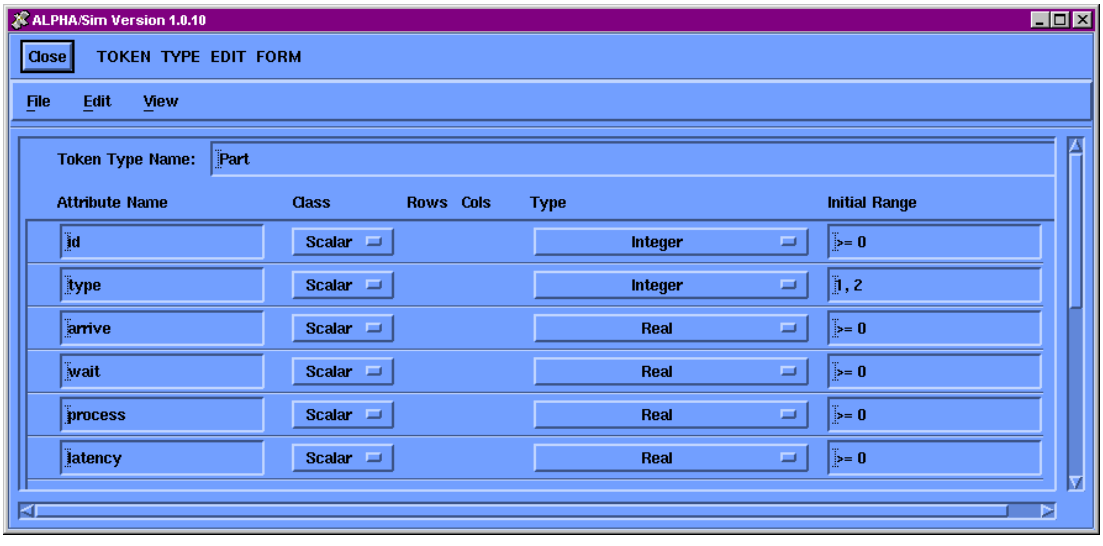


Figure 6: Token Type Edit Form for the "Part" Token Type

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

3.3 Place and Transition Forms

Once the token types are defined, we use the place and transition forms to assign token types, initial token populations, and 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 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.

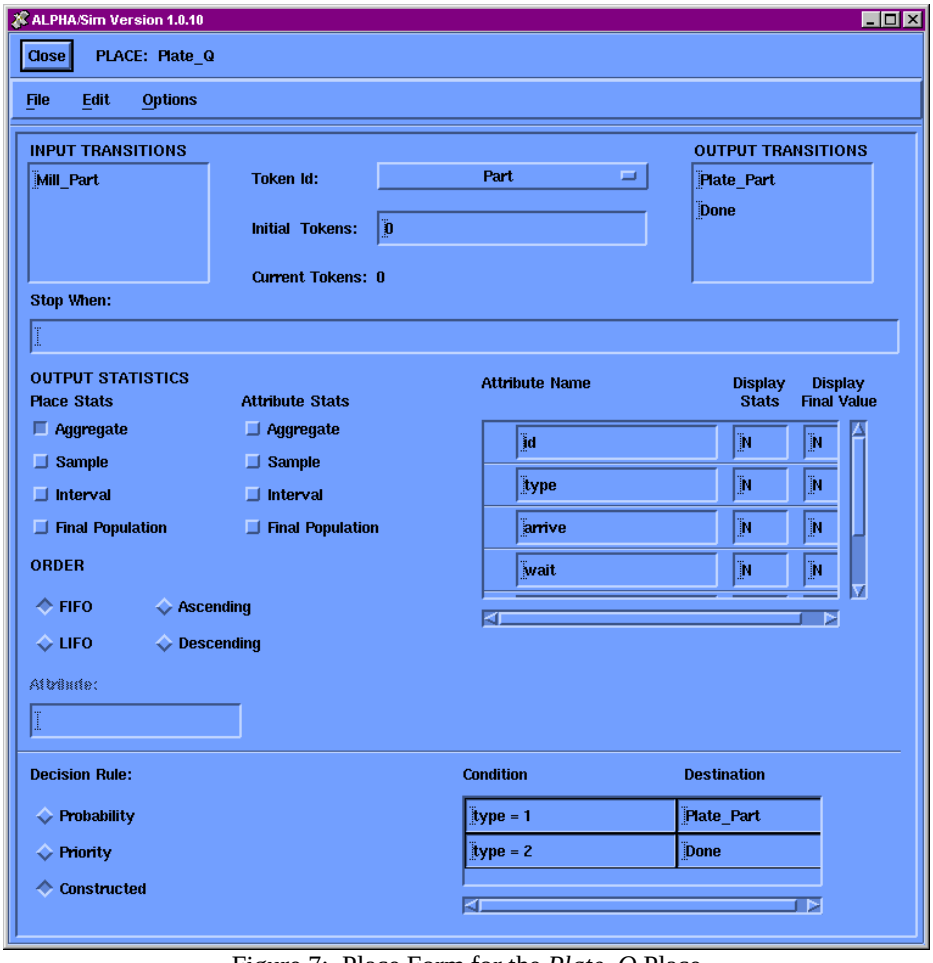


Figure 7: Place Form for the *Plate_Q* Place

Figure 8 shows a sample transition form. The left side of the form lists all the input places, and clicking on one opens the input token profile to display the input token type definition for that place. Similarly, the right side of the form lists the output places and clicking on one opens the output token profile. Ordinarily, the values of the input attributes are mapped to their corresponding output attribute; however, we can assign new output attribute values 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 preferences. For timing rules, we can specify "None", "Selected Distribution", or "Constructed". If we choose "Selected Distribution", we are prompted to select from one of the six available probability distributions and provide the appropriate parameters; Table 2 lists these distributions and their parameters. If we choose "Constructed", we can enter an expression that utilizes other distributions or attribute values. The language used for the expression is English-like; e.g., the timing rule at the *Interarrival_Delay* transition is:

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

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

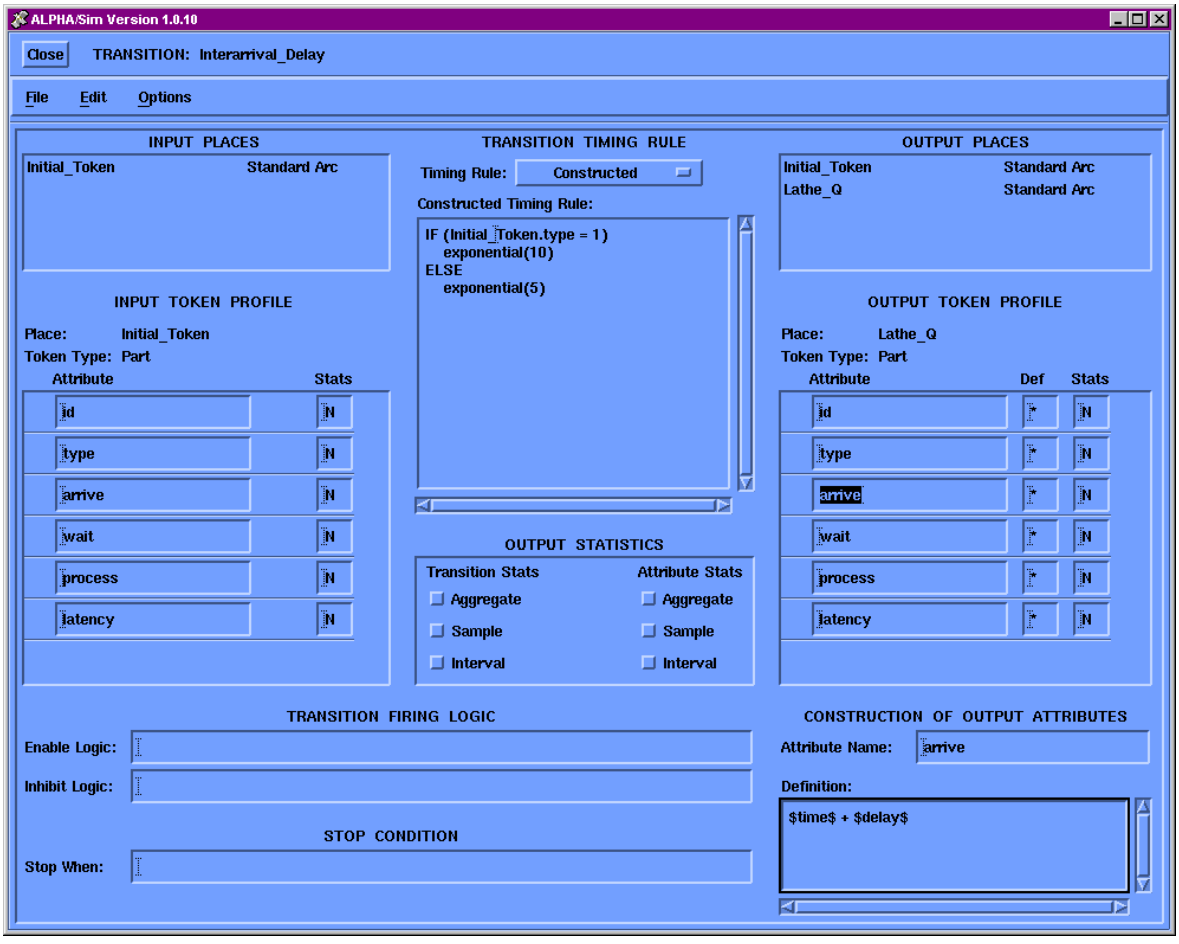


Figure 8: Transition Form for the *Interarrival_Delay* Transition

"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 define enable or inhibit transition firing logic or conditions to stop the simulation.

3.4 Specifying the Model Logic via the Forms

We use the Place form to assign token types to places, specify initial token populations, define decision (routing) rules, and specify conditions to stop simulation runs. We use the Transition form to assign timing rules, define output attribute expressions, specify enable and inhibit transition firing logic, and specify conditions to stop simulation runs. First, we assign token types to places using the Token Id option menu on each Place Form. Next, we specify two initial tokens at the *Initial_Token* place (one for each part type) and one initial token in each machine place (*Lathe*, *Mill*, and *Plating_Machine*). Table 3 lists the token type assignment and initial token population for each place in the model. Table 4 lists the attribute values for places having an initial token population, which is assigned using the Initial Tokens for Place Form.

Table 3: Token Type Assignments and Initial Populations

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

Table 4: Initial Token Values

Place	Attributes	Initial Value
Initial_Token	type	1, 2
	all others	0
Machine	id	unique integer
	type	"lathe", "mill", "plating"

Next, we specify a constructed decision rule at the *Plating_Q* place so that "Type 1" and "Type 2" parts are routed to the appropriate downstream places. If it is "Type 1", we route the part to the *Plate_Part* transition; conversely, if it is "Type 2", we route the part to the *Done* transition. The bottom portion of Figure 7 shows the decision rule as it appears on the *Plate_Q* Place Form. Alternatively, we can assign enabling logic at the *Plate_Part* and *Done* transitions to control the flow of part types out of the *Plate_Q* place. At the bottom of each transition form, there is a Transition Firing Logic section; under that section is a field for entering an Enable Logic expression. For "Type 1" parts, we enable *Plate_Part* transition firing only if the value for the *type* attribute of the incoming token from the *Plate_Q* place equals "1". Conversely, for "Type 2" parts, we enable *Done* transition firing only if the value of the *type* attribute equals "2". If desired, we can set a queuing order for the *Lathe_Q* and *Mill_Q* places.

Finally, we define output attribute expressions to collect information on the machines and on individual parts as they pass through the system. ALPHA/Sim provides four simulation variables that can be used in the expressions: $time$, $delay$, $count$, and pop. Table 5 lists the definition for each simulation variable. Since we are interested in the queuing, service, and system times for the parts, we utilize the $time$ and $delay$ variables. Table 6 lists the output attribute definitions for the transitions. The bottom-left portion of Figure 8 shows the output definition for the *arrive* attribute as tokens are fired from the *Interarrival_Delay* transition out to the *Lathe_Q* place. We can also specify statistics collection for places (average population), transitions (firing rates), and attributes, using the place and transition form statistics panels (see Figures 7 and 8, respectively).

Table 5: ALPHA/Sim Simulation Variables

Variable	Description
$time$	current simulation time prior to transition firing
$delay$	time delay of a transition firing
$count$	number of times a transition has fired
pop	current number of tokens in a place

Table 6: Output Attribute Definitions

Transition	Output	Attri-bute	Definition
Interarrival_Delay	Lathe_Q	id	$count$
		arrive	$time$ + $delay$
Turn_Part	Mill_Q	wait	$time$ - Lathe_Q.arrive
		process	$delay$
Mill_Part	Plate_Q	wait	$time$ - Mill_Q.arrive - Mill_Q.process
		process	Mill_Q.process + $delay$
Plate_Part	Finished_Type_1	wait	$time$ - Plate_Q.arrive - Plate_Q.process
		process	Plate_Q.process + $delay$
		latency	$time$ + $delay$ - Plate_Q.arrive
Done	Finished_Type_2	latency	$time$ - Plate_Q.arrive

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. Before run execution, ALPHA/Sim automatically checks all expressions for errors and if none exists, executes the simulation run. 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, such as: 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 allow a transition to fire. Stop when conditions are logical expressions that can be used to stop the simulation run when a specified condition is reached. Boxes provide a hierarchical modeling capability. Show tree displays the model 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 and its predecessor, Modeler, have been used to develop a wide array of discrete-event simulation models, such as computer components and systems (e.g., Ethernet system (Brennan, Walenty, and Moore 1995), client-server system, and high-speed disk systems), manufacturing systems (Moore and Gupta 1999), 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 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 delays. The model was used to determine the impact of automation and task redefinition on staffing requirements and throughput.

6 SUMMARY AND CONCLUSIONS

In this paper, we described a general-purpose, discrete-event simulation software tool called ALPHA/Sim. The modeling paradigm used by ALPHA/Sim is based on Petri nets, a concept developed in the early 1960s to model concurrent operations in computer systems. Over the years, this concept has been extended to incorporate attributes, timing relationships, and stochastic events.

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.

ALPHA/Sim's graphical modeling and simulation environment makes it 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 to drag-and-drop icons onto a drawing area. The logic and input parameters for the model are entered using the appropriate integrated forms. ALPHA/Sim also provides the ability to interface with external software.

We described how to use ALPHA/Sim and illustrated its key features via a simple example of a manufacturing system. In addition, we listed some of the advanced features of the tool. We also listed a number of sample applications and briefly described two of these, namely a client-server performance model 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. 1999. Stochastic, colored Petri net (SCPN) models of traditional and flexible kanban systems. *International Journal of Production Research* 37: 2135-2158.

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, and 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 an Associate Division Manager in the Information and Decision Systems Division at ALPHATECH, Inc., in Burlington, Massachusetts. She received her Ph.D. (1998) and MS (1989) degrees in operations research from the Department of Mechanical, Industrial, and Manufacturing Engineering at Northeastern University in Boston, Massachusetts. Her research focused on using Petri nets to (1) model and analyze kanban-based production systems, and (2) perform disassembly process planning. 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, Dr. Moore has been actively involved in developing and applying simulation and modeling techniques and tools. Her areas of interest include Petri nets, discrete-event simulation, manufacturing systems, business process reengineering, and decision making analysis.

JACK C. CHIANG is a senior engineer at ALPHATECH, Inc., in Burlington, Massachusetts. He received his MSE degree (1996) in Environmental Engineering and his BS degree (1995) in Biomedical Engineering, both from the Johns Hopkins University in Baltimore, Maryland. He is currently pursuing an MS degree in operations research from Northeastern University. Since joining ALPHATECH in 1997, Mr. Chiang has worked on various large- and small-scale model development and analysis projects, including discrete-event simulations of military and civilian command and control systems, linear programming, and optimization.

SCOTT D. HAMMER is a senior simulation engineer at ALPHATECH, Inc., in Burlington, Massachusetts. He received his MS degree (1991) in Mechanical Engineering (thermofluids) from Northeastern University in Boston, Massachusetts. He is currently pursuing an MS degree in operations research from Northeastern University. He has a BS degree (1986) in Aerospace Engineering from Boston University in Boston, Massachusetts. Mr. Hammer served in the US Air Force from 1987 to 1992, where he worked as a project manager in the Joint Tactical Information Distribution System (JTIDS) program office at Hanscom AFB, MA. In 1993, Mr. Hammer joined Ball Aerospace in Sacramento, CA, where he performed flight test data analysis and systems engineering support for USAF and Royal Australian Air Force F-111 aircraft modification programs. Since joining ALPHATECH in 1995, Mr. Hammer has worked on large-scale modeling and analysis of military and civilian command and control systems. This work includes simulation, linear programming, optimization, and perturbation analysis. He is also the lead technical support engineer for ALPHA/Sim. His areas of interest include search heuristics, genetic algorithms, discrete-event simulation, and model development.