Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

INTRODUCTION TO SAS SIMULATION STUDIO

Ed Hughes
Emily Lada
Phillip Meanor
Hong Chen

SAS Institute Inc.
100 SAS Campus Drive
Cary, NC 27513, USA

ABSTRACT

An overview is presented of SAS Simulation Studio, an object-oriented, Java-based application for building and analyzing discrete-event simulation models. Emphasis is given to Simulation Studio's hierarchical, entity-based approach to resource modeling, which facilitates the creation of realistic simulation models for systems with complicated resource requirements, such as preemption. Also discussed are the various ways that Simulation Studio integrates with SAS and JMP for data management, distribution fitting, and experimental design.

1 INTRODUCTION

SAS Simulation Studio is a SAS application that uses discrete-event simulation to model and analyze systems. Simulation Studio is based on the Java programming language and is a flexible, general purpose, object-oriented package designed to provide the necessary modeling and analysis tools for users ranging from novice to advanced. To facilitate the construction of simulation models, a visual programming environment based on a flow chart paradigm is provided, along with a programmatic interface for running models in batch mode.

Simulation Studio provides a comprehensive set of model-building blocks and features; however, it is not designed as a black box that takes model inputs and autonomously produces model outputs. Instead, it includes features that enable you to customize your models and tailor them to meet your specific modeling needs. One modeling feature in particular that can be easily customized in Simulation Studio is resource management. In general, a resource is a system component that provides service. Examples of resources in manufacturing systems include machines, operators, space in storage for finished products, cranes, and forklifts. Resources in a hospital include nurses, doctors, operating rooms, and beds in a recovery room. The users (or consumers) of resources are entities (Schriber and Brunner 1998). The available resources in a model might be unlimited, limited, or fixed. In the latter cases, entities may be required to wait for use of a resource. The number of available resource units may vary throughout a simulation run, perhaps governed by a predefined schedule or random failure.

Resources are an essential part of most simulation models because they often control or restrict the flow of entities. In Simulation Studio, resources may be modeled as special types of hierarchical entities that can be seized and released by other entities to fulfill varying resource demands. Furthermore, resource entities can be assigned attributes and can flow through the model. Using entity type and attribute values, a specific resource entity can easily be located in a model, thereby giving the user fine-level control over resource behavior, such as preemption.

978-1-4799-2076-1/13/$31.00 ©2013 IEEE

While an extensive collection of modeling tools is important in simulation software, advanced analysis capabilities are critical as well. Because analyzing the data generated by discrete-event simulation models often requires the use of advanced statistical methods, SAS Simulation Studio is designed to interact with both SAS (SAS Institute Inc. 2013a) and JMP statistical discovery software (SAS Institute Inc. 2012b) for analysis of simulation results. Data generated by a simulation model can easily be saved as a SAS data set or a JMP table, and it is possible to run a SAS or JMP program and utilize its output during a simulation run. Simulation Studio also integrates seamlessly with JMP for design of experiments and input analysis.

The purpose of this paper is to provide an overview of Simulation Studio and highlight its modeling and analysis capabilities. In Section 2, an overview of Simulation Studio's organizational structure is provided, while Section 3 describes Simulation Studio's hierarchical, object-based approach to resource management. Section 4 highlights Simulation Studio's data management and analysis capabilities, including distribution fitting and experimental design. Section 5 summarizes the paper.

2 OVERVIEW OF SIMULATION STUDIO

The fundamental modeling objects in Simulation Studio include entities, data values, blocks, ports, and links. During a simulation, entities and data values can travel among blocks to satisfy various processing needs. Data values designate information such as numbers, character strings, and boolean values. Entities are discrete objects that can traverse a simulation model and be assigned attributes, or properties. Simulation Studio also enables the user to define new entity types possibly with their own default attributes, in which the primary usage of each new type can be as either a *regular* entity or a *resource* entity. Both regular and resource entities can be used to represent physical or conceptual components in a model, such as telephone calls in a telecommunications system, customers in a retail store, or ships in a harbor.

In Simulation Studio, blocks are the most fundamental units used to build a model. Each block typically encapsulates some well-defined and specialized functionality. Communication between blocks occurs via input and output ports, classified as value ports and entity ports. For example, the OutLength output value port on a Queue block communicates the length of the associated queue of waiting entities. In Simulation Studio, the user creates a link between the ports on blocks to define a path for values or entities to flow (SAS Institute Inc. 2013b). The OutLength port on a Queue block can be connected, for example, to an input port of a Formula block so that the current queue length is used as part of an expression whose result is used to route entities to different parts of the model.

After a model is built in Simulation Studio, an *experiment* is created to control the initialization and running of the model. Simulation Studio includes an Experiment window that by default contains columns for controlling the system parameters (start time, end time, and number of replications) for a model. The Experiment window can also be used to control the initialization of block parameters, thereby providing an efficient means for investigating the effects of different input parameters (or *factors*) on model outputs (or *responses*).

Models and experiments in Simulation Studio are organized into *projects*. A project must contain at least one model and one experiment but may contain multiple models and experiments. Projects also provide a storage mechanism for factor and response definitions so that they can be shared across all models and experiments in the project. When a model/experiment pair is executed, Simulation Studio must map the factors and responses included in an experiment to specific block parameters in the model. This process is accomplished using *anchors*, each of which defines the link between a factor or response defined on a project and an actual block parameter in a specific model. This mapping technique provides an efficient and effective means of reusing models and experiments, because multiple models can be linked to the same experiment and a single model can be linked to multiple experiments.

3 MODELING RESOURCES IN SIMULATION STUDIO

For building models, Simulation Studio provides a comprehensive set of tools that includes standard blocks for modeling fundamental concepts, such as queueing, switching (or branching), cloning, and batching. In addition, Simulation Studio provides unique and highly flexible tools for modeling resources. Because resources are a fundamental part of most simulation studies, we devote this section to an overview of Simulation Studio's resource modeling capabilities and, in particular, the use of both stationary and mobile resource objects.

3.1 Stationary Resources

Entity holding blocks (such as Queue, Server, and Delay blocks) represent stationary resources in a Simulation Studio model. These stationary resources are created at model-building time and are used to model one type of resource. Holding blocks have a capacity (which may be infinite), and they hold or delay entities for some period of time. Furthermore, entities may compete for available space in a holding block. This contrasts with nonholding blocks (such as a Switch block) in which entities flow through without the simulation clock advancing.

To illustrate the use of stationary resources, Figure 1 shows a Simulation Studio model of a bank lobby in which there are three tellers and one queue for waiting customers. Customer arrivals to the bank are modeled using an Entity Generator block (labeled Arriving Customers). A Numeric Source block (labeled Interarrival Time) generates a sample from a specified distribution, and the Entity Generator block pulls that value through its InterArrivalTime value port to schedule the arrival time of the next customer. When an entity (representing a bank customer) leaves the Entity Generator block, it is pushed to a Queue block (labeled FIFO Queue). The Queue block in this model has infinite capacity and a first-in-first-out queueing discipline. When an entity arrives at the Queue block, it attempts to push the entity to a Server block (labeled Tellers). The Server block has a specified capacity of three and represents the three bank tellers. If a unit of the Server is available (that is, one of the bank tellers is idle), then the Server block accepts the customer entity; otherwise, the entity waits in the Queue. When a unit of the Server becomes available, it requests an entity from the queue. When an entity arrives at the Server block, a service time is sampled from a Numeric Source block (labeled ServiceTime) and pulled by the Server through its InServiceTime value port. Once the entity completes service, it is pushed out to the Disposer block (labeled Departing Customers) and leaves the system.

In the model shown in Figure 1, there are two stationary resources: the Queue block (with infinite capacity) and the Server block (with finite capacity). Each block holds entities for some time period and represents one type of resource. Using a holding block such as a Queue or a Server block is the simplest way to model resources in Simulation Studio. However, if the system being modeled has a complex resource structure (perhaps so that several different types of resources are required simultaneously to fulfill a demand), then mobile resources are needed.

3.2 Mobile Resources

Mobile resources, which are dynamic and created during the simulation run, are resource objects that flow in the model. Mobile resources are a special type of entity (called a resource entity) and possess all of the capabilities and attributes of regular entities. They can be processed and managed by the blocks for regular entities. All resource entities in Simulation Studio have a predefined entity attribute named ResourceUnits, which is the capacity (number of units) of the resource. In addition to the ResourceUnits attribute, each resource entity also has run-time state information, such as seized status and resource state, that is used by the simulation system to perform resource management during the run. From a user's point of view, the resource state can be either functional or nonfunctional.

Most importantly, functional resource entities fulfill resource requirements by being *seized* by other entities (including other resource entities) in a simulation model. Once resource entities have been allocated

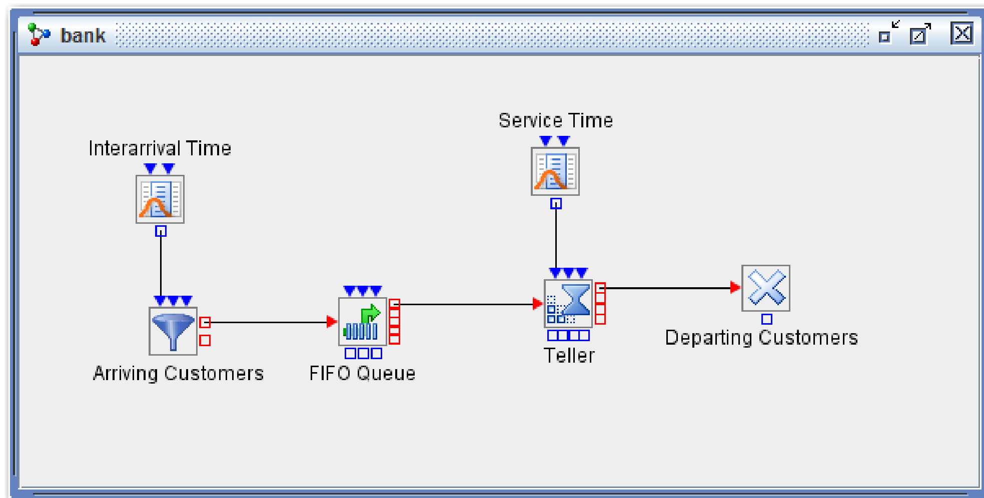

Figure 1: The banking system model in Simulation Studio using stationary resources.

and seized by a controlling entity, an entity hierarchy is formed with the controlling entity at the top level and each seized resource at the next level. The controlling entity then typically continues to flow through the model, along with its seized resource entities. In Figure 2, an alternative Simulation Studio model of the same bank lobby system described in Figure 1 is shown. Recall that in the model in Figure 1, the bank tellers are modeled as a stationary resource (a Server block), unable to flow or move through the model. In Figure 2, the bank tellers are modeled as resource entities, created at model run-time. An Entity Generator block (labeled Create Teller) generates three resource entities (one for each teller) at time zero and sends those entities to a Resource Pool block (labeled Teller Pool) to wait until needed. The arrival of customers to the bank is modeled in the same way as in Figure 1. However, as shown in Figure 2, a Seize block (labeled Seize Teller), a Resource Pool block (Teller Pool), a Delay block (Hold Teller), and a Release block (Release Teller) work together to reproduce the functionality of the Server block in the model in Figure 1.

When a customer entity arrives at the FIFO Queue block, the Queue block notifies the Seize block (labeled Seize Teller) that a customer is waiting. The Seize block then checks to see if a bank teller resource entity is available in the Resource Pool block. If one is not available, then the customer entity remains in the queue. If a bank teller resource entity is available, the Seize block accepts the customer entity from the Queue block, pulls a bank teller resource entity from the Resource Pool block, and attaches it to the customer entity, forming a hierarchy of entities. As the customer entity flows through the simulation model, it brings the teller resource entity along with it. After seizing a teller resource entity, the customer entity is sent to a Delay block (labeled Hold Teller) where it is held (along with the teller resource entity) until its service is completed. It is then routed to a Release block where the teller resource entity is extracted from the customer entity. The two entities flow out of different ports on the Release block and are sent to different locations: the customer entity is routed to a Disposer block (labeled Departing Customers) and the teller resource entity is routed back to the Resource Pool block, where it waits to be seized by another customer entity.

A hierarchical entity-based approach to resource modeling greatly facilitates the modeling of scenarios that require multiple types of resources simultaneously. Any entity (including a resource entity) can seize multiple resources of different types simultaneously and then release them (perhaps partially) as needed. For this simple banking system, using a stationary resource (that is, a Server block) to model the bank tellers is sufficient, and mobile resources are not really required. However, suppose at some point a bank teller requires the assistance of a manager in order to service a customer. For this scenario, the bank

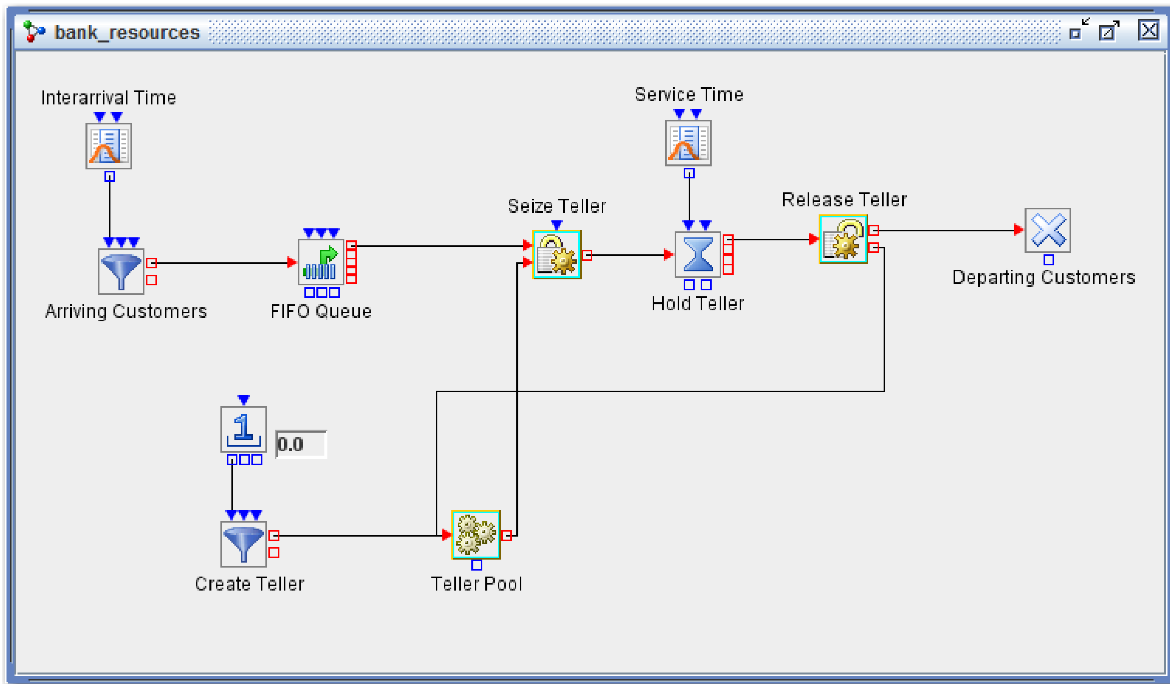

Figure 2: The banking system model in Simulation Studio using mobile resources.

tellers must be modeled as mobile resource entities as in Figure 2. After seizing a teller resource entity, a customer entity could then seize a manager resource entity. Following a delay (representing service time), the customer entity could then release both the teller resource entity and the manager resource entity simultaneously, or it could release them at different points in the model. Because the released resources are also entities, each can flow either to a holding block (like a Resource Pool) to wait to be seized by another entity or to other blocks in the model before returning to a Resource Pool block. For example, suppose a teller needs to complete clerical work before assisting the next customer. This scenario is easily modeled in Simulation Studio by sending the teller resource entity to a Delay block (representing the clerical work completion time) before sending it back to a Resource Pool where it can subsequently be seized by another customer entity.

Even though the individual resource entities may be scattered throughout the model during a simulation run, it is fairly easy to locate a specific resource entity using *resource entity rules*, which are defined using entity characteristics such as type and attribute values. Once the resource entity is located, its state or capacity can be adjusted as needed according to a specified resource schedule, or it can be allocated to another entity to fulfill a demand.

3.3 Preemption

Simulation Studio supports two types of resource preemption: priority-based and scheduled. Priority-based preemption is used primarily for preempting entities occupying stationary resources (entity holding blocks), including the Queue, Server, and Delay blocks. An entity attempting to enter a holding block is considered to be a consumer of the stationary resource represented by that block. Allocation of stationary resources usually involves the acceptance of entering entities into the holding block to occupy space. Preemption of stationary resources forces out one or more entities currently holding a space in the block. The preempted entity is pushed out a dedicated OutPreempt port and can be routed to any part of the model, as dictated by the system logic. For example, an entity preempted from a Server block can be routed first to another

block that computes its remaining processing time and then back into a queue to wait for space in the Server block to complete its processing time.

To handle priority-based preemption, Simulation Studio provides an Entity Group object that is a collection of entity references. An entity reference contains information that uniquely identifies a particular entity. Thus, an Entity Group holds information about a collection of entities, but not the actual entities themselves. Each Simulation Studio holding block has an OutHoldings port that other blocks can use to access an Entity Group object that contains a collection of references to entities held by the block. The Simulation Studio holding blocks also each provide an InPreempt port that accepts an Entity Group object as input. These blocks compare the entity references in the Entity Group object to the entities currently held by the block and preempt any matches. With this design, it is possible to preempt any number of units of a stationary resource. Also, determining which entities to preempt from service is specific to the system being modeled, and the Entity Group construct allows the user to control exactly which entities are preempted.

Scheduled preemption is used for preempting mobile resources (resource entities) and is based on the requirements of a resource schedule. Within a defined resource schedule, the user indicates whether a capacity or state change should be preemptive. Resource entity rules can be specified as part of a schedule to precisely indicate which resource entities (according to type and attribute values) the schedule should be applied to. Sometimes an allocated and seized resource entity needs to be preempted from its current controlling entity because (i) the resource entity needs to be reallocated to a different controlling entity, (ii) the resource entity needs to be sent to some other part of the model for processing, or (iii) the resource entity has a scheduled capacity reduction or state change. The entity holding blocks provide an OutResource output port for routing the preempted resource entity so that the user can decide (through modeling) how to handle the post-processing of preempted entities.

4 DATA INPUT, COLLECTION, AND ANALYSIS

The subject matter of a simulation investigation or the sophistication of a model often dictates what type of data needs to be collected from each simulation run and the amount of data required to perform an appropriate analysis. Simulation Studio is well-integrated with both SAS and JMP to take advantage of the rich and powerful data processing and analysis capabilities available in each package. In this section, we provide an overview of the various ways in which Simulation Studio interacts with both SAS and JMP for data management, distribution fitting (input analysis), and experimental design.

4.1 Input Data Management

Simulation Studio provides two special data object types to manage the collection of data during a simulation run. The first, a *data model object*, can be viewed as an in-memory representation of a SAS data set or JMP table during a simulation run. It contains information and/or values specified in rows, columns, and cells. The second type, an *observation object*, represents one row from a data model object. It can be viewed as the simulation-time representation of a data observation from a SAS data set or a data row from a JMP table. The data model and observation objects are used in Simulation Studio blocks to represent data for various access and collection tasks. For example, the Dataset Holder block can be used as a holding facility for a data model object, making it useful for matrix computations, as well as for modeling scenarios that require repeated access to a data set (or look-up table) to perform a particular computation. During a simulation run, the contents of a data model object (such as individual data cell values and observation objects) stored in a Dataset Holder block can be pulled through user-defined output ports and passed to other blocks in the model.

In a Simulation Studio model, the Numeric Source, Text Source, and Observation Source blocks can be used to input data to a model. The Numeric Source and Text Source blocks can be used to read a column of numbers or strings from a SAS data set or JMP table, while the Observation Source block provides a

stream of data observation (row) objects from a SAS data set or JMP table. For example, the Observation Source block can be used to read the rows from a data set and either assign an entire row as attributes on an entity or assign a subset of the data cell values in the row as attributes. The Observation Source block can also be used to read in an entire SAS data set or JMP table.

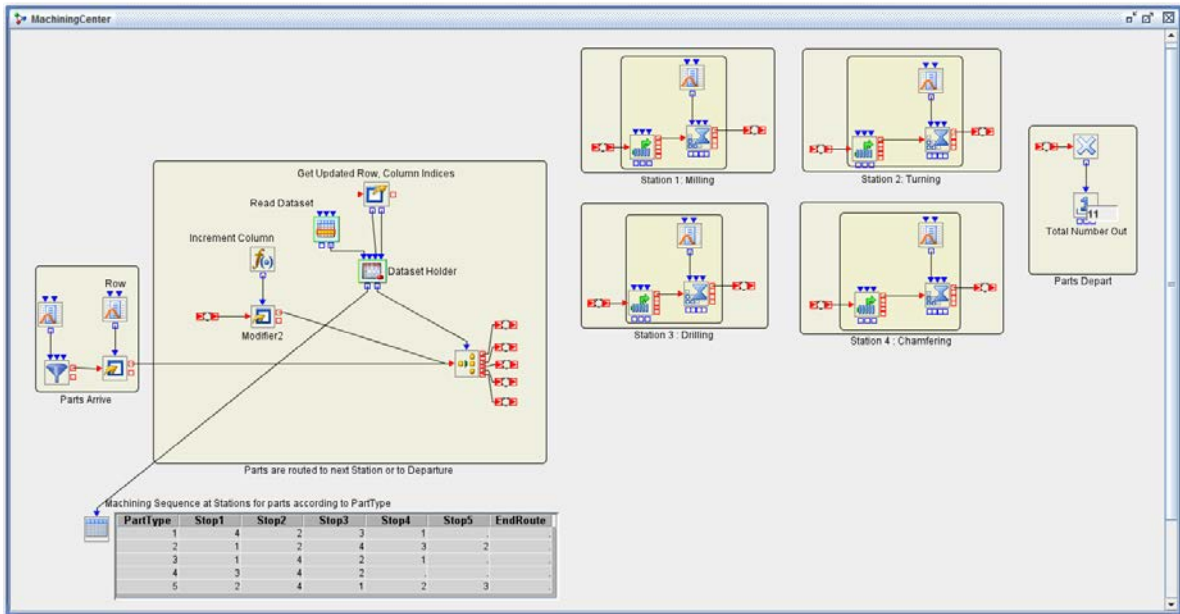

Figure 3: The machining center model in Simulation Studio using the Dataset Holder block.

Figure 3 shows a Simulation Studio model of a machining center in which an Observation Source block (labeled Read Dataset) reads in a SAS data set and stores it in a Dataset Holder block for use as a look-up table indicating the routing of parts of various types through the stations in the machining center. In this system, each part type is processed in a distinct sequence at some or all of four different stations. In this example, a Dataset Holder block with one user-defined output port (located at the bottom right of the Dataset Holder block) is used to hold the machining sequence data set, which is displayed by using a Table block (located at the bottom left of Figure 3). The data set value that is pulled from the bottom right output port is a particular cell value based on part type; it indicates the next station in the processing sequence. In this example, the Dataset Holder block holds a data set that is used repeatedly by all entities. An alternative is to store the information in the machining sequence data set as entity attributes, but that would result in the same data being stored multiple times. The Dataset Holder block enables the data to be stored once and accessed as needed for each part type.

4.2 Output Data Storage and Analysis

Simulation Studio has a number of blocks that can accumulate data and store it as a data model object. A data model can be accessed by other blocks in the simulation model via an OutData port. For example, a plot or table block can be connected to the OutData port of a Queue Stats Collector block to visually display the queue statistics (such as average waiting time) while the simulation model is running. The contents of a data model can also be saved at the end of a run as a SAS data set or a JMP table. Furthermore, a Dataset Writer block can be used to save the contents of a data model object as either a SAS data set or a JMP table at any point during a simulation run. The data saving operation is triggered by a Boolean signal that is sent to the Dataset Writer block from another block in the model.

In addition to the data collection blocks, Simulation Studio includes a SAS Program block that can be used to execute a SAS program or a JMP script at any point during a simulation run. For example, in a simulation model of an inventory system, it may be necessary to update a production plan data set based on the current state of the system. If the number of backlogged orders exceeds a certain level, a SAS Program block can be signaled to execute a SAS program that generates a new production plan data set that is used to set production levels downstream in the model.

4.3 Input Analysis

The process of building a simulation model may include the need to identify probability distributions that faithfully represent the behavior of the random input processes driving the system under study. Given a data set of values that represent observations of a particular random input process, it is first necessary to identify an appropriate distribution family and then estimate the corresponding distribution parameters. The accuracy of the simulation results depends on the quality of the distribution fit, making input modeling one of the critical problems in the design and construction of a simulation model. The automatic distribution-fitting procedure of JMP provides a list of candidate distributions and corresponding estimated parameters for a specified data set. The distribution fits are ranked using the Akaike information criterion (Akaike 1974). Simulation Studio is integrated with JMP for input modeling capabilities so that the JMP automatic distribution-fitting procedure can be easily accessed through the Numeric Source block in Simulation Studio.

Figure 4 shows the results of JMP automatic distribution fitting applied to a column of data labeled *bvar*. The first distribution listed in the `Compare Distributions` section of the output (Weibull) is the top-ranked fit. After analyzing the fit results in JMP and selecting a distribution, the `Commit to Simulation Studio` button can be used to pass the selected distribution and its parameter values back to a Numeric Source block in Simulation Studio.

4.4 Design of Experiments

After building a simulation model and ensuring the system under investigation is accurately reflected, a typical next step is to systematically study the impact of various model input parameters on the simulation output. Experimental design techniques can be used to generate an efficient and effective plan to guide your simulation runs. Generating an experimental design begins by defining the input parameters, or factors, for a particular model. Possible factors for a simulation experiment might include staffing levels, rates of work, or maximum lengths for queues. In general a simulation experiment may have many factors and each factor is defined on a range of values, called levels. Next the simulation performance measures, or responses, are defined. Examples of responses include average waiting times in queues, utilization of resources, and total cost.

A design is a matrix in which each column corresponds to a factor and each row (called a design point) corresponds to a particular combination of factor levels. Examples of classic experimental designs include factorials, fractional factorials, central composite, and Latin hypercube. After establishing a design, the simulation model is run for each design point and values of the responses are recorded. The primary goals of experimental design are (i) to identify those factors that have the greatest impact on the responses; (ii) to categorize the nature of the impact of a particular factor on the responses (for example is the effect increasing, linear, or quadratic); and (iii) to determine if factor interactions exist, that is, to determine if the levels of some factors influence the effects that other factors have on the responses (Sanchez and Wan 2009).

Once the factors and responses are defined in Simulation Studio for a particular model, JMP can be used to generate an experimental design. Simulation Studio interfaces with JMP so that given the factor and response definitions, a default design is created by the JMP custom designer (SAS Institute Inc. 2012a) and automatically passed back to the Experiment window in Simulation Studio. The user can alter the default JMP design by adding, for example, additional design points, replicates, or interaction terms. Figure 5

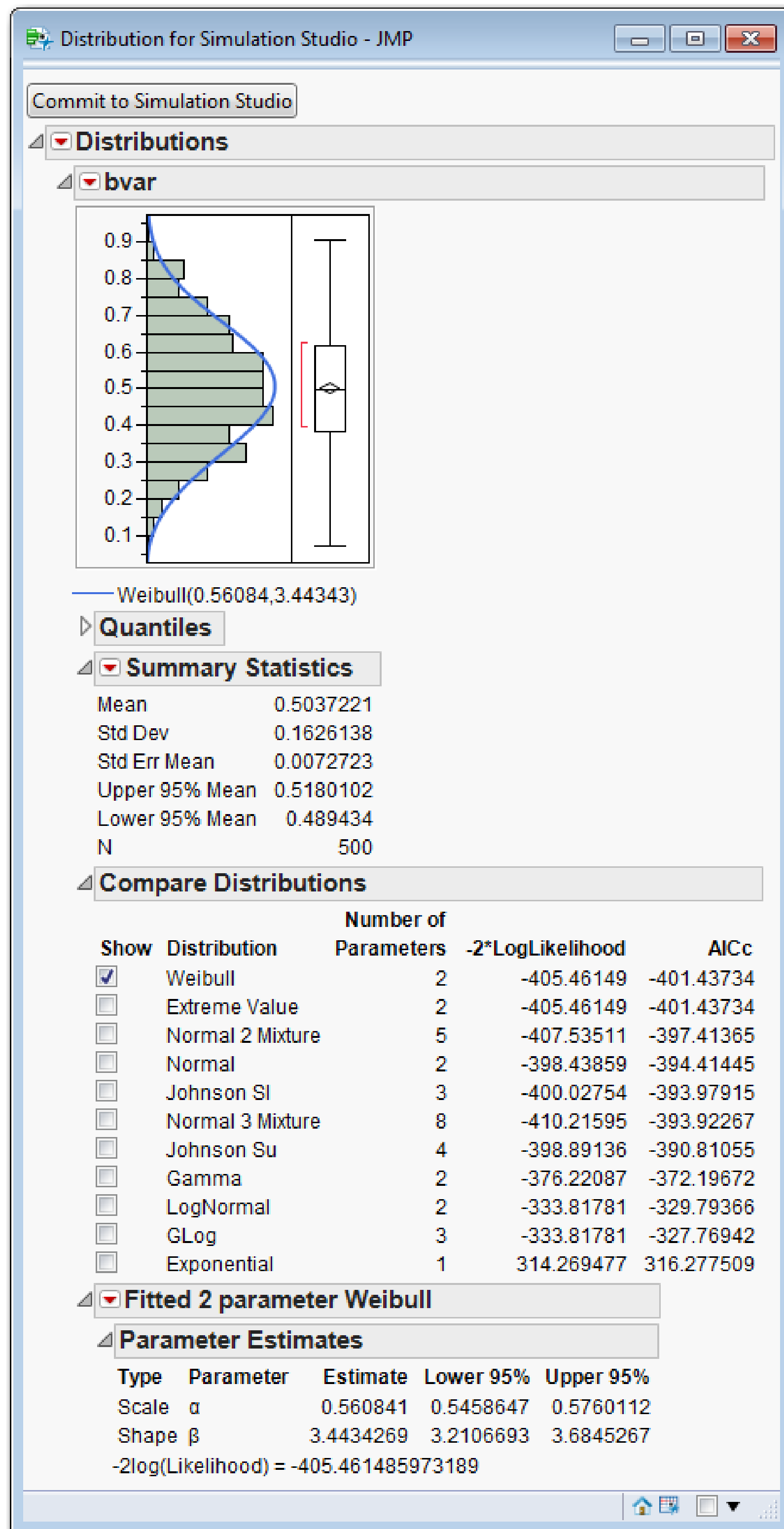

Figure 4: Automatic distribution-fitting in JMP.

shows the Experiment window for a model of a repair shop. The three factors (yellow columns) denote staffing levels at the Quality Control, Repair, and Service locations. The responses (pink columns) include the number of units fixed and the average wait at the Quality Control, Repair, and Service locations. The default design generated by JMP has 12 design points and five replications are run for each design point. Design point number 4 has been expanded to show the results for each of its five replications. The remaining design point rows display the average for each response over the five replications. After running an experiment, the results (that is, the entire contents of the Experiment window) can be passed directly back to JMP for analysis. For example, the simulated results can be used to estimate a statistical model, which in turn can be used to determine optimal levels of the factors so that a particular response is maximized or minimized. The results can also be saved as a SAS data set or a JMP table for later analysis.

experiment0

PointName	StartTime	EndTime	NumService	NumRepair	NumQC	Replicates	NumFixed	AvgWaitServi...	AvgWaitRepair	AvgWaitQC
point 1	0	2,700	3	3	2	5	105.4	0.11444080...	1.13945013...	0.28406588...
point 2	0	2,700	1	2	1	5	104.8	16.7647729...	5.75633988...	3.22633454...
point 3	0	2,700	2	3	3	5	105.4	0.83741026...	0.96856203...	0.00199815...
point 4	0	2,700	1	2	2	5	105.2	16.7647729...	5.81418541...	0.03291412...
						1	114	9.14545120...	7.67442277...	0.01134718...
						2	98	7.64456804...	4.47389647...	0.05961856...
						3	95	7.12897199...	6.17055115...	0.03560460...
						4	92	9.07937810...	5.13200419...	0
						5	127	50.8254955...	5.62005249...	0.05800025...
point 5	0	2,700	3	2	3	5	105.2	0.11444080...	10.2670209...	0
point 6	0	2,700	1	1	1	5	77.4	16.7647729...	322.056083...	0.20429683...
point 7	0	2,700	3	2	2	5	105.2	0.11444080...	10.2687899...	0.03888053...
point 8	0	2,700	3	3	1	5	105.2	0.11444080...	1.08450340...	6.26674826...
point 9	0	2,700	3	1	3	5	77.4	0.11444080...	339.008908...	0
point 10	0	2,700	2	1	2	5	77.4	0.83741026...	338.241586...	0
point 11	0	2,700	1	3	2	5	105.4	16.7647729...	0.40958337...	0.29823839...

Figure 5: An experimental design in the Simulation Studio Experiment window.

5 CONCLUSIONS

SAS Simulation Studio is an object-oriented, Java-based application for discrete-event simulation that features a hierarchical, entity-based approach to resource management. These resource entities can be processed by the modeling blocks for regular entities, and they can be seized by other entities to fulfill resource demands. There are many advantages to an entity-based approach, including greater control over complicated resource management issues such as scheduling and preemption.

SAS Simulation Studio is closely integrated with SAS and JMP for data management and analysis capabilities. Special data model and observation objects are used to manage the collection of data during a run and facilitate the inputting of SAS data sets or JMP tables to a model. Data model objects can be saved as either a SAS data set or JMP table at any point during a simulation run. To systematically study the effect of specific input parameters on the simulation model output, experimental designs can be created in JMP and passed back to the Experiment window in Simulation Studio. JMP can also be used to fit a distribution to a specified data set, and the distribution, along with the corresponding estimated parameters, can easily be passed back to Simulation Studio for use as a model input.

SAS Simulation Studio is available as a component of SAS/OR software for operations research and is also available separately as an add-on to JMP. SAS Simulation Studio is supported on 32-bit and 64-bit Microsoft Windows platforms.

REFERENCES

Akaike, H. 1974. "A New Look at the Statistical Model Identification." *IEEE Transactions on Automatic Control* AC-19 (6): 716–723.

Sanchez, S. M., and H. Wan. 2009. "Better Than a Petaflop: The Power of Efficient Experimental Design." In *Proceedings of the 2009 Winter Simulation Conference*, edited by M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, 60–74. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

SAS Institute Inc. 2012a. *JMP 10 Design of Experiments Guide.* Available online via http://support.sas.com/documentation/onlinedoc/jmp/10.0.1/DOE_Guide.pdf [accessed July 15, 2013].

SAS Institute Inc. 2012b. *Using JMP 10.* Available via http://support.sas.com/documentation/onlinedoc/jmp/10.0.1/Using_JMP.pdf [accessed July 15, 2013].

SAS Institute Inc. 2013a. *SAS 9.4 Language Reference: Concepts.* Available via http://support.sas.com/documentation/cdl/en/lrcon/64801/PDF/default/lrcon.pdf [accessed July 15, 2013].

SAS Institute Inc. 2013b. *SAS Simulation Studio 12.3: User's Guide*. Available via http://support.sas.com/documentation/cdl/en/simsug/66111/PDF/default/simsug.pdf [accessed July 15, 2013].

Schriber, T. J., and D. T. Brunner. 1998. "How Discrete-Event Simulation Software Works." In *Handbook of Simulation*, edited by J. Banks, 765–811. New York: John Wiley & Sons, Inc.

AUTHOR BIOGRAPHIES

ED HUGHES is a product manager at SAS Institute. He is a member of INFORMS and his e-mail address is Ed.Hughes@sas.com.

EMILY LADA is an operations research specialist at SAS Institute. She is a member of INFORMS and her e-mail address is Emily.Lada@sas.com.

PHILLIP MEANOR is a research and development manager at SAS Institute. His e-mail address is Phillip.Meanor@sas.com.

HONG CHEN is a software developer at SAS Institute. His e-mail address is Hong.Chen@sas.com.