Editor: Michael A. Gray, gray@american.edu

DISCRETE EVENT SIMULATION

A REVIEW OF SIMEVENTS

By Michael A. Gray

A new entrant in the field of discrete event simulation systems, SimEvents has several desirable system features. It still has some catching up to do in some areas, but SimEvents is particularly useful for existing Matlab and Simulink users looking to construct complex hybrid systems of discrete/continuous processing.

Discrete event simulation is very effective in providing quantitative results for analyses of stochastic flow systems. In a *flow system*, tasks enter and flow through a graph of interconnected servers, which perform various services on the tasks. I use the term "task" here in the broadest sense because the things that flow through these systems can be as diverse as humans flowing through a bank, materials flowing though a plant, or message packets flowing through a communications network. Given that the tasks usually share servers, queues form to provide a holding function for tasks waiting for busy servers. The tasks exit the system when all servicing is complete. A *stochastic* flow system is one in which the individual task entrance times and service times aren't deterministic; the fact that we can describe them only in the aggregate by statistical measures implies a stochastic variation in the queues, as well.

Researchers have developed discrete event simulation systems (DESSs) to assist in creating and running simulations, and several commercial packages are available.[1] MathWorks, the makers of Matlab, have released a new entrant, called SimEvents, a demonstration version of which I recently evaluated. To ground my evaluation in something concrete, I also contrasted the features in SimEvents with those in the academic version of Rockwell's Arena system (www.arenasimulation.com).

I concentrated on system features that I view as desirable in a DESS. Most DESSs provide various ways to model system components, but users often prefer direct ways because they have little time to master DESS concepts. That said, I've tried to evaluate the indirect ways of doing things, as well.

Desirable Features

I began by organizing desirable DESS features; Table 1 shows my results (p. 64). The first two columns list the categories and a further breakdown by features. The third column labels each feature (*Xn*, where *X* indicates the feature category and *n* is an integer) that I examine later. The last two columns show the final evaluation results for SimEvents. Due to space limitations, I'll restrict myself to remarking on only those features that might not be obvious.

In the environment category, *replication tools* (feature G4) serve model builders' needs for automatically building large models. Manually constructing a 10,000-node network is so tedious that it's impossible without tools for the job. Furthermore, these replication tools should let users write and execute a controlling program of some kind—rather than forcing them to manipulate by hand—leading to *replication under program control* (G5) for programmatically constructing large numbers of similar servers and channels.

Occasionally, modelers might want a queue without an associated server, perhaps for delay or storage purposes. In this case, the queue should be an *individual mechanism* (Q6) rather than rigidly attached to a server. Under the server category, *multiplicity* (S5)—the ability to simultaneously service multiple entities—is desirable. We must have an adequate set of *logic and control* (L1) mechanisms, and we certainly must have *conditional testing* (L2) mechanisms that can sample an entity's properties or model conditions and route flows to alternate paths. We need *feedback* (L3) mechanisms to allow loopback paths in the models, and there must also be some *communication* (L4) means to let model components send runtime messages to each other to change model operation.

An important requirement for any DESS is to provide complexity control devices for model builders. Given that a moderately complex system can easily overwhelm a model builder's ability to manage it, a DESS should include *abstraction* (A1) mechanisms to reduce complexity at different model levels. Specifically, we want *embedding* (A2) functionality to let us embed systems within other systems. Model users need *user control* (U1) mechanisms that require *appropriate skills* (U2) for users who might not be as knowledgeable as model builders.

SimEvents Environment

SimEvents is installed as part of the Simulink extension for Matlab, and it appears in the Simulink environment as a block library. SimEvents immediately meets important environment requirements because it presents a *graphical drag-and-drop interface* (G1, G2) for assembling and using models. The fact that it supplies its blocks in the form of *libraries* (G3) satisfies another requirement. It's disappointing, though, that SimEvents has no replication tools for building large models and no runtime programmatic mechanisms for assembling such models.

Entities

The SimEvents direct mechanism for entities is also called *entity* (E1). The event-based or time-based entity generators create entities based on object blocks from the Generators|Entity Generators library. The builder inserts the generator blocks into the model and, during model execution, creates entities according to the conditions defined. The entities might have both presupplied and *builder-defined properties* (E4), called attributes, specified for the entity class by means of a parameter sheet. An attribute is a name–value pair that can be defined to supply an entity with a property. This is a familiar approach to Arena modelers because it uses an analogous interface to define and create entities.

A SimEvents entity can be one of two types: `standard` or `blank` (E3). The standard type is presupplied with two properties—the `count` attribute indexes the entities as they're generated and serves as an identifier for *individual entities* (E2); the `priority` attribute (E6) is available for logic and control. Model builders must define any additional entity properties in builder-supplied attributes. Because the `blank` type has no presupplied properties, all its properties must be builder-defined. However, this only minimally satisfies the requirement for a builder-definable entity type because the builder must entirely define the type via attributes, implying additional work. By comparison, Arena lets builders define their own entity types by supplying new names and attribute sets. Such entity types then become one of the available types in the system.

The builder supplies the statistical distribution, describing entity-creation times as a parameter for the *time-based entity generator* (E5). The presupplied distributions are

- `constant`, for constant interarrival time;
- `uniform`, for randomly chosen interarrival times uniformly distributed between builder-supplied minimum and maximum times; and
- `exponential`, for randomly chosen interarrival times distributed according to an exponential distribution with a builder-supplied mean.

Builders can also use other distributions by supplying the generator with an externally defined sequence of interarrival times. Arena provides a much richer set of directly available presupplied distributions, so this is an area in which we can hope for improvements in future SimEvents releases.

Channels

The SimEvents direct mechanism for providing channels is the *connection line* (C1), a unidirectional channel between two blocks along which entities or signals flow from the supplier block to the consumer block. Constructed inside the model-building graphical interface rather than using a block from a library, connection lines don't have unique identifiers. SimEvents differentiates entity flow from control-signal flow by defining entity connection lines and signal connection lines: entities flow through a model along entity connection lines, whereas signals propagate around a model on signal connection lines. A builder has no direct means for defining a new connection type. With no properties controlling bandwidth or speed, connection lines have properties a model builder can use. The Routing library provides blocks that we can use to construct complex

About the Editor

Given that this is my first column as Technologies editor, I'd like to introduce myself. While a physics graduate student at Penn State, I was introduced to computing by writing Fortran II programs to simulate underwater acoustic environments, and I've worked in some part of computing ever since.

I have a great fondness for simulation as a working technique, so it's an appropriate subject for my inaugural column. I first encountered simulation languages in the 1970s in the form of Simscript for continuous simulation and GPSS for discrete simulation. I've watched since then as these relatively low-level, primitive facilities have grown into impressively rich and powerful simulation systems such as Simulink and Arena. The underlying technologies for these and other scientific computation uses have grown in ways I never foresaw, and the expansion pace seems unabated, so it's a real pleasure to have the opportunity to help cover this area for our readership. Please feel free to email me at gray@american.edu with your comments and suggestions for column topics.

Table 1. Summary of SimEvents evaluation.

Category	Desirable features	Label	Direct	Indirect
Environment	Interactive, graphical	G1	Yes	
	Drag-and-drop action	G2	Yes	
	Customizable libraries	G3	Yes	
	Replication tools	G4	No	
	Replication under program control	G5	No	
Entity		E1	Yes	
	Unique identifier	E2	Yes	
	Customizable type	E3	Minimal	Yes
	Customizable property list	E4	Yes	
	Customizable creation statistics	E5	Minimal	Yes
	Entity priority	E6	Yes	
Channel		C1	Yes	
	Customizable type	C2	No	Yes
	Customizable bandwidth	C3	No	Yes
	Customizable speed	C4	No	Yes
Queue		Q1	Yes	
	Customizable type	Q2	No	Yes
	Customizable size	Q3	Yes	
	Customizable priority policy	Q4	Minimal	Yes
	Customizable overflow policy	Q5	No	Yes
	Individual	Q6	Yes	
Server		S1	Yes	
	Customizable type	S2	No	Yes
	Service time	S3	Minimal	Yes
	Customizable service time statistics	S4	No	Yes
	Multiplicity	S5	Yes	
	Customizable state	S6	No	Yes
	Customizable state statistics	S7	No	Yes
	Customizable scheduling	S8	No	Yes
Logic, control		L1		
	Testing	L2	Yes	
	Feedback	L3	Yes	
	Communication	L4	Yes	
Abstraction		A1	Yes	
	Embedding	A2	Yes	
User control		U1	Yes	
	Appropriate skills	U2	Yes	
	Graphical display	U3	Yes	
	Client presentation	U4	No	
	Summary reports	U5	No	Yes
	Detail reports	U6	No	Yes
	Statistical analysis tools	U7	Minimal	Yes

structures representing connections with specified bandwidths and speeds. This imposes an additional burden on model builders, but it's similar to Arena's treatment of connections.

Queues

The SimEvents direct mechanisms for providing queues are the *FIFO* (first in, first out), *LIFO* (last in, first out), and *priority queues* (Q1), which are in the Queues library. SimEvents can provide other queue protocols when builders *combine priority queues with server preemption* (Q5). This differs from Arena's approach of providing a single, customizable queue data-block type for which we can alter parameters to select the desired queue protocol. SimEvents doesn't provide builder-definable queue types in a direct manner. We can control queue size through the `capacity` parameter (Q4), but no parameters control queue logic for handling overflow. Instead, the supplying block accomplishes that by blocking output when the consuming block (the queue) is full. Queues are stand-alone mechanisms (Q6).

Servers

The SimEvents direct mechanisms for providing servers are the *infinite server*, *N-server*, and *single server* (S1) from the Servers library. The application doesn't provide builder-definable server types. We can define a fixed service time internal to a server (S3), but there's no direct means for specifying a varying service time within the server itself. This differs from Arena's Process module, in which server time and statistical variation is completely definable. In SimEvents, the only ways to define varying service times are by indirect, external mechanisms. The builder can specify server multiplicity through the server block used or through the `number` parameter for an *N*-server (S5). Because SimEvents doesn't supply servers with state directly, builders must create this property indirectly. The application also offers no way to directly provide server-scheduling policies.

Logic and Control

SimEvents has an extremely rich set of direct mechanisms for providing logic and control. First, it offers the *signal* (L3) for runtime communication between model blocks. Signals propagate along signal connections from block to block and carry values encoding information. Most blocks in SimEvents accept appropriate signals on input and provide for sending signals on output. With other control mechanisms, such as enabled gate, release gate, and replicate, these signals give model builders a set of control mechanisms. Testing mechanisms consist of the *input and output switches* (L1), which let the builder test model conditions or entity properties and route entities to alternate paths. The *path combiner* (L2) provides the loopback mechanism by combining multiple input streams of entities into single output streams. This lets us loop an output stream from a downstream block back into an upstream block as input.

Abstractions

SimEvents includes three powerful abstraction mechanisms: the *subsystem*, *masked subsystem*, and *discrete event subsystem* (A1). The first two are in the underlying Simulink environment and let the builder create subsystems embedded in a system. In turn, they can have *embedded lower-level subsystems* (A2) as their components. The subsystem block is analogous to Arena's subsystem module. SimEvents provides a more powerful abstraction in the masked subsystem block, which not only combines blocks into visually simplified single blocks but also lets model builders use masks to set variables within the new abstraction from outside the structure. The discrete event subsystem lets the builder create structures that react to events in a precisely timed manner.

User Customization and Control

Users might need to customize a completed simulation model for a particular system before running the analysis. SimEvents provides a display that lets users set important *run parameters* such as start and stop times (U1). In addition, model builders can encapsulate the entire system in a single masked subsystem and provide all user controls for the internals in a mask. This makes a very user-friendly customization and control mechanism that doesn't require detailed understanding of SimEvents modeling (U2). This is an improvement over Arena's user facilities, in which users have to customize modules in a more complicated way.

Graphical Displays and Reporting

SimEvents provides graphical displays for users through its interactive execution capability coupled with the *scope block* (U3). The scope block receives data during a run and graphs important simulation information versus time in an oscilloscope-like manner. Model builders can use *display blocks* to display the running value of signals and statistical summary information as numbers, but unlike Arena, SimEvents doesn't offer the ability to display entities flowing through the graph. That's a valuable tool for small systems because a visual inspection of the entity flow often reveals model errors and bottlenecks without the need for a more exhaustive analysis. It's also very valuable for client presentation because it lets the customer clearly follow how the system changes in time.

The reporting facilities in SimEvents are disappointing. The system includes no report-type blocks, so modelers must take the final data outside the system and summarize it by other means. This stands in contrast to Arena, in which the production of detailed, comprehensive reports is an automatic part of model execution.

Statistical Tools

In this first release, the statistical tools directly available in SimEvents consist mainly of output signals from blocks that report *totals or time averages of relevant properties* (U7). The builder must often mathematically process the output of these signals to obtain other kinds of statistics, which generally requires skills in Simulink or Matlab programming.

An important issue in simulation is ensuring that the results are statistically valid. This typically involves multiple runs of the model with random draws from the distributions describing the stochastic variables. SimEvents doesn't provide an easy, direct way to organize and conduct such multiple-run experiments. Instead, builders (or users) must initialize and control the experiment's operation from the Matlab environment using Matlab programs. Although it's certainly advantageous for SimEvents to leverage the existing Matlab system, this approach again places an extra burden on users to be skilled in Matlab. SimEvents doesn't compare well to Arena in this area, but we should remember that this is the first release, whereas Arena is a mature system that's benefited from multiple improvements.

Indirect Mechanisms

SimEvents supports three basic approaches to constructing model mechanisms indirectly. First, we can combine SimEvents blocks into subsystems to create single, specialized mechanisms or provide functionality missing in direct mechanisms. Given the rich environment of signal-processing and event-manipulating mechanisms within SimEvents, it's possible to construct a library of subsystem blocks to correspond to any target.

Given that modelers can use the surrounding Simulink environment's features in SimEvents models, a second approach is to combine Simulink blocks with SimEvents blocks to create desired subsystems when combinations of SimEvents blocks alone can't achieve the desired functionality.

Finally, it's relatively simple to send and receive data from the underlying Matlab environment with its extensive function library and general programming language for creating specialized functions. The *discrete event signal to workspace* block makes it possible to send data directly to the Matlab workspace. Modelers with time and Matlab expertise can thus use the extensive Matlab facilities to create specialized libraries of blocks.

Table 1 presents my overall evaluation of SimEvents. A "Yes" under the "Direct" heading means that a mechanism exists to provide the feature; a "Yes" under the "Indirect" heading means that it's possible to construct such a feature by combining two or more mechanisms, possibly including ones that are external to SimEvents.

Although it's missing some direct mechanisms for important features, SimEvents is a good all-around DESS, provided that the resources are available to use indirect mechanisms to supply the missing direct mechanisms. It's especially useful for existing Matlab and Simulink users. Its ease of integration with the existing system is a really strong feature because we can use SimEvents to construct complex hybrid systems of discrete/continuous processing.

Acknowledgments

I thank MathWorks for providing an evaluation copy of SimEvents (available from www.mathworks.com).

Reference

1. J. Banks et al., *Discrete-Event System Simulation*, 3rd ed., Prentice-Hall, 2001.

Michael A. Gray is an associate professor at American University. His research interests include computer science and physics. Gray has a PhD in physics from Pennsylvania State University. He is a member of the ACM, the IEEE, and the IEEE Computer Society. Contact him at gray@american.edu.