Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

RESOURCE MANAGEMENT IN SAS SIMULATION STUDIO

Hong Chen

SAS Institute Inc.
100 SAS Campus Drive, R5418
Cary, NC 27513, USA

Emily Lada

SAS Institute Inc.
100 SAS Campus Drive, R5313
Cary, NC 27513, USA

ABSTRACT

We present an overview of resource management in SAS Simulation Studio, an object-oriented, Java-based application for building and analyzing discrete-event simulation models. In Simulation Studio, resources are modeled as special types of hierarchical entities that can be assigned attributes and flow through the model. Furthermore, resource entities can be seized and released by other entities to fulfill resource demands. Flexible resource entity rules are used to specify these demands, as well as the requirements of other resource operations, such as state and capacity changes. The hierarchical, entity-based approach in Simulation Studio allows the user more control over resource behavior and provides many advantages over alternative resource management techniques, especially in the areas of resource scheduling and preemption.

1 INTRODUCTION

SAS Simulation Studio is a SAS application that uses object-oriented, 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 discrete-event simulation package designed to provide the necessary modeling and analysis tools for both novice and advanced users. To facilitate the construction of simulation models, a visual programming environment based on a flow chart paradigm is provided with various fundamental modeling objects or constructs, including entities, data values, blocks, ports, and links. During a simulation, entities and data values can travel among blocks for various processing needs. In general, values are information such as numbers, character strings, and boolean values. An entity is a discrete object that can traverse a simulation model and be assigned attributes, or properties. 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 usually encapsulates some well-defined and specialized functionality. Communication between blocks is accomplished through ports. Blocks have two types of ports: value ports and entity ports. Furthermore, each port can either be an input port or an output port. For example, an output value port on a Queue block is the length of the associated queue of waiting entities. Generally, an input port is used to get information into the block and an output port is used by the block to either push information out or used by another block to pull information from the block. In Simulation Studio, a link is created between the ports on blocks to define a path for values or entities to flow (SAS Institute Inc. 2009).

Simulation Studio attempts to avoid being simply a black box that takes model inputs and mysteriously produces model outputs. Rather, it includes features that enable you to customize your models and tailor the application to meet 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 on a conveyor, 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). In some instances, the available resources in a model might be unlimited, while in others the number of units of a resource might be limited, or fixed. In the latter case,

entities are required to wait for use of the resource. The number of available resource units may also vary throughout a simulation run, perhaps governed by a predefined schedule. Resources are an essential part of most simulation models since they often control or restrict the flow of entities. It can even be said that most discrete-event simulation models contain a resource of some type, whether it be a runway at an airport, a cashier at a grocery store, or a parking garage with a finite number of spaces for cars.

The purpose of this paper is to provide an overview of resource management in Simulation Studio and highlight the advantages of a hierarchical, entity-based approach for modeling complex, resource-driven systems. In Section 2 an overview of alternative resource management techniques is given and in Section 3 an overview of Simulation Studio's object-based approach to resource management is provided. In Section 4 a common resource usage pattern is proposed to describe the various ways in which resource entities are used in Simulation Studio. Sections 5 and 6 address the topics of resource scheduling and preemption, respectively, and conclusions are provided in Section 7.

2 GERNERAL RESOURCE MANAGEMENT TECHNIQUES

Resource management schemes are largely dictated or influenced by how resources are actually represented in the modeling software. There are several possible ways to define resources in a general purpose simulation environment. The simplest way is to model resources with a number, as in Arena (Kelton, Sadowski, and Sturrock 2007). The number usually represents resource capacity and can be managed, for example, by a resource pool. This approach, however, assumes the resources are homogeneous and have no individual identity or attributes. This may force a user to make simplifying assumptions in complex situations, especially when trying to model preemption.

Resources can also be defined as special objects, as in ProModel (Harrell and Price 2002), SimProcess (Jones 1995), and AnyLogic (XJ Technologies 2009). These resource objects usually contain a number for capacity and possibly other attributes as well, but they are not entity objects. As a result, dedicated constructs or special treatments must be provided for the delivery and management of these resource objects. In ProModel, dynamic resource objects are allowed to move along the pathways in a simulation model and compete for node usage. As a result, ProModel's dynamic resources have much in common with its entities, although the resources are not actual entities.

Finally, resources can be represented as entities. In some simulation systems, it is possible to model resources using regular entities (without any explicit resource representation). These entities can be stored in a queue and later matched with other entities using some batching mechanism to mimic the seizing of resources to fulfill a demand. This approach is one of the resource representation techniques used in ExtendSim (Diamond et al. 2007). Since in this case the modeling is primarily based on regular entities and their processing constructs, users may have to design, implement, and maintain a more sophisticated resource management subsystem based on the constructs for regular entities in complicated scenarios. ExtendSim attempts to facilitate the use of their regular entities as resources by providing some dedicated blocks, such as the Resource block, which acts like a special queue that allows its entities to be pulled and used as resources.

On the other hand, the object-oriented simulator YANSL (Joines and Roberts 1998) proposes to define resources as a *special* type of entity since entities and resources have a great deal of overlapping characteristics and behavior in an object-oriented simulation environment. The concept of a *resource* entity takes advantage of the rich entity processing capability in a simulation environment, resulting in less special treatment and required support for resources. In addition, YANSL introduces the resource group and resource team constructs to depict the relationships of related resource entities. YANSL, however, does not have a graphical user interface and as a result, a simulation model (including resource management) must be programmed in the C++ programming language by extending the YANSL C++ library. Because YANSL does not have any visual/graphical support, it may be difficult for users without programming skills to use.

As in YANSL, Simulation Studio resources are also defined as a special type of entity. However, Simulation Studio offers a graphical user interface so that programming is not required by the user. In addition, since the entities in Simulation Studio can be hierarchical, the resource entities can be used to organize many levels of child resources. The "resource team" in YANSL can be modeled directly in Simulation Studio without introducing any additional constructs. Furthermore, the resource entities can seize other resource entities and the resulting entity hierarchy can be used like a regular resource entity (that is, it can be seized, released, scheduled, and preempted), which greatly facilitates the modeling of a collection of resources in complex situations.

3 MODELING RESOURCES IN SIMULATION STUDIO

In Simulation Studio, an object-based approach to resource management is taken. In particular, systems modeled in Simulation Studio can use two kinds of resource objects: stationary resources and mobile resources. The following subsections introduce both types of resource objects.

3.1 Stationary Resources

Entity holding blocks (such as Queues, Servers, and Delay blocks) represent stationary resources in a Simulation Studio model. These stationary resources are static, created at model building time, and 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 is opposed to 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 simple banking system in which there are three tellers and one queue for customers to wait. Customer arrivals to the bank are modeled using an Entity Generator block (labeled Arriving Customers). The 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 to the bank. 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 the Numeric Source block labeled ServiceTime and pulled by the Server through the InServiceTime value port. Once the entity completes service, it is pushed out to the Disposer block (labeled Departing Customers) where it leaves the system.

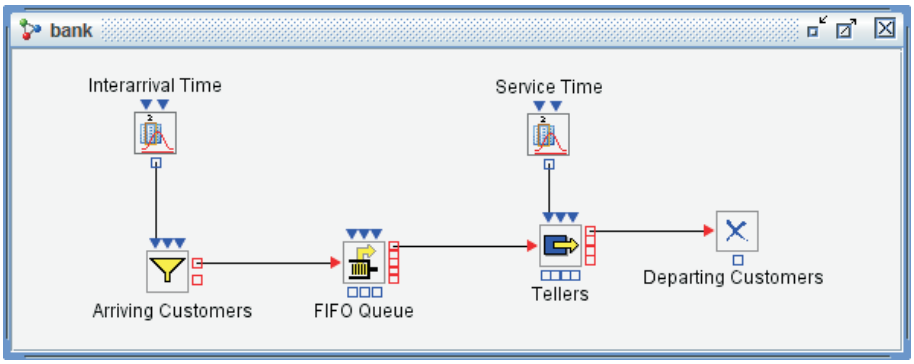

Figure 1: The banking system model in Simulation Studio using stationary resources.

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). Both blocks hold entities for some time period and represent 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 must be used.

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 and possess all the capabilities and attributes of regular entities. They can be processed and managed by the blocks for regular entity objects. All resource entities in Simulation Studio have a predefined entity attribute named ResourceUnits, which is the capacity (number of units) of the resource. While the ResourceUnits attribute has special uses for resource entities, it can also be used as an ordinary numeric entity attribute for modeling purposes. 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. The functional resource entities can be allocated to other entities to fulfill resource requirements.

Most importantly, resource entities can be *seized* by other entities (including other resource entities) in a simulation model. In Figure 2, a Simulation Studio model of the same banking system described in Figure 1 is shown where the

tellers are now modeled as mobile resource entities instead of with a static Server block. In the model in Figure 1, the bank tellers are a stationary resource and they never flow or move through the model. In Figure 2, the bank tellers are modeled as resource entities and they are created at model run-time. The Entity Generator block labeled Create Teller generates three resource entities (one for each teller) at time zero and sends those entities into a Resource Pool block (labeled Teller Pool) to wait until needed. The arrival of customers to the bank is modeled 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 mimic the functionality of the Server block in the model in Figure 1.

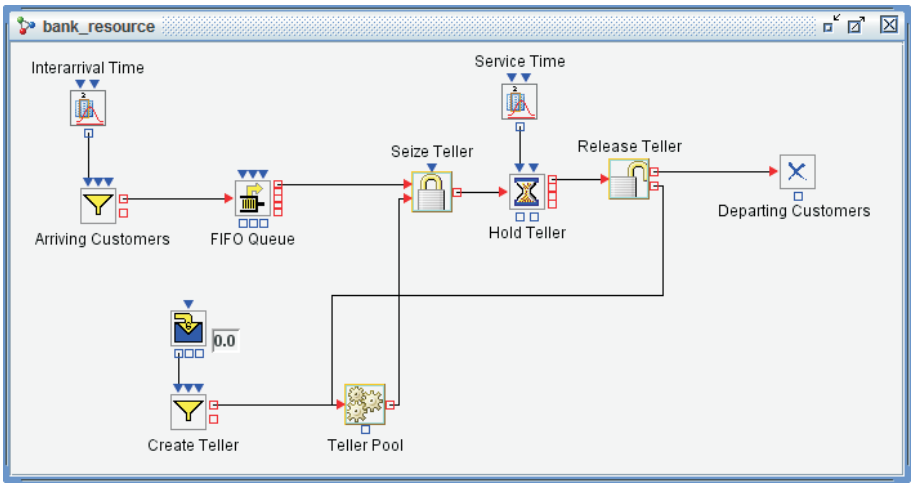

Figure 2: The banking system model in Simulation Studio using mobile resources.

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 stays in the queue. If a bank teller resource entity is available, then the Seize block accepts the customer entity from the Queue block, pulls the bank teller resource entity from the Resource Pool block (labeled Teller Pool), and attaches it to the customer entity. At this point, a hierarchy of entities is formed. At the top level is the customer entity (called the parent, or root entity) and below it is the teller resource entity (called the child entity). When 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 bank teller resource entity is extracted from the customer entity. The two entities flow out different ports on the Release block: 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.

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. For this scenario, the bank 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. By modeling the resources as entities in this example, the user has more flexibility and it is possible to model complex scenarios.

An entity-based approach to resource modeling facilitates the modeling of scenarios that require multiple types of resources simultaneously. Any entity can seize multiple resources of different types simultaneously and then release them (perhaps partially) as needed. Because the released resources are also entities, they can either flow to a holding block (like a Resource Pool) where they wait to be seized by another entity, or they can flow to other blocks in the model before returning to a Resource Pool block. For example, suppose a doctor needs to complete paperwork before seeing the next patient. This scenario is easily modeled in Simulation Studio by sending the doctor resource entity to a Delay block (representing the paperwork completion time) before sending it back to a Resource Pool where it can subsequently be seized by another patient 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 by 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 some resource schedule. With resource scheduling comes the issue of resource preemption, which becomes relatively straightforward to model with an entity-based approach to resource management. Simulation Studio provides the user with constructs to control resource preemption, including the specification of which resource to preempt, the number of units to preempt, and where the preempted resource should be routed in the model (if necessary).

4 COMMON RESOURCE USAGE PATTERN

In object-oriented computer programming, identifying and applying design patterns to handle various reoccurring situations has greatly improved the design, implementation, understanding, and usage of computer software systems. Inspired by the concept of design patterns (Gamma et al. 1995), we propose a common resource usage pattern to describe various usages of resource entities in Simulation Studio. This common usage pattern for resource entities consists of the following fundamental steps:

1. Creating,
2. Storing,
3. Locating,
4. Allocating,
5. Using,
6. Deallocating, and
7. Disposing.

Each of these steps might use one or more Simulation Studio blocks. In addition to all the regular modeling blocks, there are six resource-specific blocks available in Simulation Studio: the Seize, Release, Resource Pool, Resource Scheduler, Resource Agenda, and Resource Stats Collector blocks. These six blocks, together with the other regular blocks, provide all of the powerful resource management capabilities discussed in this paper.

Resource entities in Simulation Studio are usually created by an Entity Generator block, just as regular entities are. The Entity Generator block provides the user with options to select what type of entity it will create. Within the Entity Generator block, it is possible to select the DefaultResourceEntity type with a predefined ResourceUnits attribute for generation, or the user can define a new resource entity type that includes additional custom attributes. The initial value of all attributes, including ResourceUnits, can also be set in the Entity Generator block.

After a resource entity is created, it is typically sent to a Resource Pool block for storage before it can be seized and allocated to meet a resource demand. The Resource Pool block performs resource management tasks for resource entities, such as maintaining seized/unseized status, and processing resource requests. A resource entity is considered *unseized* if it resides in a Resource Pool block. It is considered *seized* if it leaves the pool and is not directly held by any other Resource Pool block. A newly created resource entity is also considered unseized before it enters a Resource Pool block.

Resources need to be located, requested, and allocated to fulfill other entities' resource demands. Locating resources is also essential for other resource operations, such as scheduling, statistics collection, and preemption. Simulation Studio uses resource entity rules (that is, boolean expressions that the attributes of the targeted resource entities must satisfy) to locate resource entities. The resource needs or constraints of an entity that enters a Seize block (referred to as a *controlling* entity) can be specified as resource entity rules in the Seize block. A Seize block provides an input resource entity port for each resource need or constraint. The input resource ports can be connected with Resource Pool blocks. During a simulation run, the Seize block uses the links to its input resource ports to locate and request resource entities from resource storage blocks to satisfy the resource needs of the controlling entity.

After locating the resources in the Resource Pool blocks, the Seize block requests the resources. The Resource Pool blocks process the requests and allocate the resources to the Seize block. In the Resource Pool block, only those resource entities with a resource state set to functional can be allocated. To decrease the likelihood of a resource deadlock, a single Seize block in Simulation Studio does not support partial allocation of the resources it is requesting. All resource constraints must be satisfied before resources are actually allocated to a controlling entity. Otherwise, the Seize block does not accept the controlling entity's resource request and the controlling entity must wait in a Queue block for all required resources to become available. If partial allocation is required, then a chain of Seize blocks may be used to seize different subsets of resources, one after another, as they become available.

Once the resource entities have been allocated to a controlling entity, an entity hierarchy is formed with the controlling entity at the top (parent) level and each seized resource at the next (child) level. The controlling entity

then typically continues to flow through the model, along with its seized resource entities. As a simple example of how resources are used, the controlling entity might move to a Delay block to represent a processing time and then move to a Release block to have the resource entities deallocated. In a more complicated system, such as an emergency room, it is not hard to imagine resource entities staying with a controlling entity as it flows through various parts of the model. When a patient enters an emergency room, the patient might be assigned a nurse, a doctor, and a surgery room for some time period. After the surgery, the doctor and surgery room might be released from the patient, but the nurse might stay with the patient and a recovery room might be added as a resource. In another example, parts could be modeled as resource entities and they could be continually added to a controlling entity as it progresses down an assembly line. In this case, the controlling entity will never release the resource entities since they are essentially consumed to build the final product.

Resources seized by controlling entities can be released (deallocated, unseized) by using a Release Block. Resource constraints (resource entity rules) can be defined on the Release block to locate targeted resource entities within the controlling entity hierarchy. The Release block provides an output resource entity port for each constraint defined. For each controlling entity that enters the Release block during the simulation run, the user-defined resource constraints are used to locate and deallocate the targeted resources among all resources held by the controlling entity. The deallocated resources flow out the appropriate output resource ports on the Release block. Released resources can be routed to any block in the simulation model as dictated by the system logic. For example, a resource entity can be processed like a regular entity and have its attributes adjusted in a Modifier block or be held in a Server block for a period of time. Routing the resource entities back to a Resource Pool block indicates that they are available to be seized by other entities.

Like regular entities, resource entities can be disposed by the Disposer block in Simulation Studio. Resource entities that are attached to a controlling entity that enters a Disposer block are destroyed along with the controlling entity. Disposing resource entities with a Disposer block affects the total resource capacity available in the simulation model.

5 SCHEDULING RESOURCE ADJUSTMENTS

Resources often undergo routine adjustments, or changes, and the effects of such adjustments can last for a limited period of time. Examples include a truck that must undergo routine maintenance, a worker that requires a lunch break, and the addition of sales people for the holiday shopping season. The scheduling of resource adjustments should address the following issues:

a. What kind of adjustment to make (capacity or state change);
b. What resources to adjust (locate the targeted resources);
c. When to make the adjustment and for how long;
d. Whether or not the adjustment is preemptive;
e. When to proceed to the next related adjustment; and
f. Whether or not to repeat the adjustment in the future (is it repeatable or not).

In Simulation Studio, the scheduling of mobile resources is supported by the Resource Agenda and Resource Scheduler blocks together. The Resource Agenda block addresses the first issue above, and the Resource Scheduler block addresses the rest.

Resource adjustments are often related and happen in an orderly fashion. In Simulation Studio, related adjustment actions can be grouped together. A special type of value object, called a resource agenda, defines a sequence of related resource adjustment actions based on a relative simulation time. Each resource adjustment action (called an agenda entry) includes a change to either the resource capacity value or the resource state value in targeted resource entities over a certain time period. The Resource Agenda block provides a resource agenda to describe what kind of resource adjustments to make during a simulation run. In the banking system example, the tellers may have staggered lunch breaks. An agenda could be defined over one day to indicate the number of tellers available and for how long. Figure 3 shows the same banking system model as in Figure 2 with the addition of the Resource Agenda and Resource Scheduler block to model the teller lunch breaks.

The Resource Scheduler block accepts and stores resource agenda objects during a simulation run and handles items (b)–(f) in the above list. During a simulation run, the Resource Scheduler block locates the targeted resource entities among the resources in the system dynamically using entity search rules (which may be, for example, an attribute value). These targeted resource entities are either unseized and located in resource storage blocks (such as a Resource Pool block) or seized by a controlling entity. The seized resource entities are busy and in use. In the nonpreemptive case, the seized resource entities should not be adjusted according to the schedule until they become unseized. In the

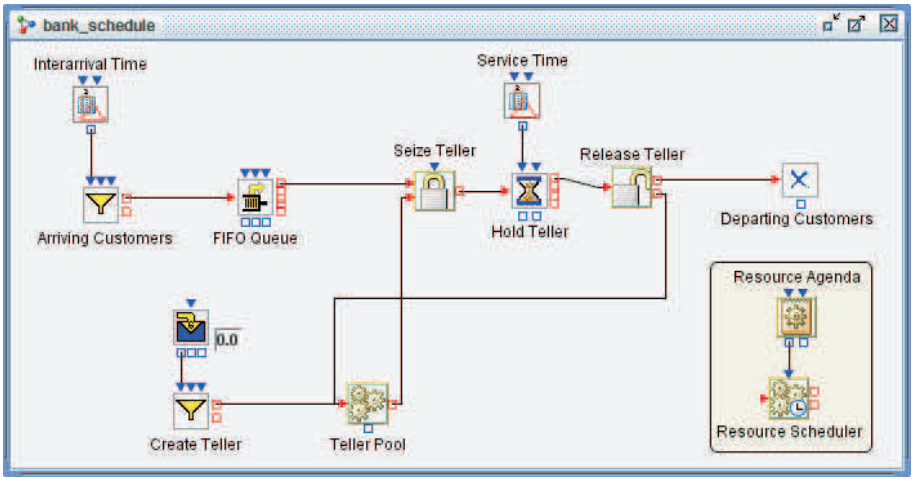

Figure 3: The banking system model in Simulation Studio using scheduled mobile resources.

preemptive case, the capacity of the seized entities is decreased (or the state is changed to nonfunctional) according to the resource schedule and the scheduled adjustment is considered *disruptive* to those resources. In this case, the seized resources are deallocated from their current controlling entity for further processing or allocation. Options are available (called "Immediate Actions" rules) in the Resource Scheduler block to allow the user to specify how a specific agenda entry is to be processed for a targeted set of resource entities and whether or not it should be preemptive.

The Resource Scheduler block uses several heuristics to process agenda entries. For an increase in capacity, the Resource Scheduler block divides the increased units evenly among all targeted resource entities (where the resource entities affected are specified in the Resource Scheduler block based on entity type or attribute value). For a total decrease of capacity, the Resource Scheduler block attempts to decrease as much capacity as possible from a first targeted resource entity before locating the next target, and so on. For example, if there are four bank teller resource entities in the model each with capacity one, and at a specific time the agenda entry indicates the total capacity for the bank teller resource should be two, then the first bank teller resource entity will be located and its capacity will be decreased from one to zero. At that point, another teller resource entity will be located and its capacity will also be decreased from one to zero. So the total capacity of the teller resource is decreased from four to two. In general, the Resource Scheduler block always attempts to finish processing a resource agenda entry without preemptive changes. For capacity changes, the currently unseized resources are located and adjusted first. This "unseized first" heuristic decreases the waiting time for the seized resources to become unseized and avoids unnecessary preemptive adjustments.

6 PREEMPTING RESOURCE ENTITIES

Currently in Simulation Studio, two types of resource preemption are supported: priority-based and scheduled. Priority-based preemption is primarily used for preempting stationary resources (entity holding blocks), which include the Queue, Server, and Delay blocks. An entity that wants to enter a holding block is considered a consumer of the static resource represented by that block. Allocation of static resources usually involves the acceptance of entering entities into the holding blocks to occupy space. Preemption of static resources forces out one or more entities currently holding a space in the block so that the space can be given to some other entity. The preempted entity is pushed out a dedicated preemption port and can be routed to any part of the model, as dictated by the system logic.

To handle priority-based preemption, Simulation Studio provides an object named Entity Group that is a collection of entity references. An entity reference contains information that uniquely identifies a particular entity. Therefore an entity group holds information about a collection of entities, but not the actual entities themselves. The Simulation Studio holding blocks (including Queue, Server, and Delay) provide an InPreempt input port that accepts an Entity Group object as input. These blocks compare the entity references in the Entity Group 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. This type of preemption is often triggered by the higher priority of a new entity attempting to enter a holding

block that does not have any more space. Determining which entities to preempt from service is very specific to the system being modeled, and the entity group construct allows the user to control exactly which entities are preempted.

Figure 4 shows an example of how priority-based preemption can be modeled in Simulation Studio. In this model, higher-priority customers can preempt lower-priority customers who are already receiving service if all units of the server are busy. The preempted customers wait for a server to become available again so they can complete their remaining service time. Entities with a priority attribute with value 1 are generated and sent immediately into a Queue block labeled Priority Queue that is ordered based on the value of the attribute priority such that entities with high priority values are at the front of the queue and will receive service first. Entities in the Priority Queue block wait for service from one of three servers (modeled with a Server block with capacity three). After entities with a priority attribute with value 2 are generated, they are routed to the Preemption Logic section of the model. A priority 2 entity first enters a Switch block where the availability of a server is checked. If a server is available, then the priority 2 entity is routed to the Priority Queue block (via the Connector labeled To Entrance). If all units of the Server are busy, then we check to see whether a priority 1 entity is currently in service (so that it can be preempted). At this point, the priority 2 entity is cloned and the original entity is sent to the front of the Priority Queue block to wait for the possible preemption of a priority 1 entity. The cloned priority 2 entity is sent to a Gate block where the Entity Group block labeled Targets is triggered to pull information on all the entities currently being held in the Server block. Also within the Entity Group block are entity rules specifying which entity should be preempted (in this case, a single entity with priority attribute equal to 1). The Entity Group block attempts to find an entity among all the entities being held in the Server block that satisfies the specified rules. The resulting entity group (which is a reference to either one entity or none if there are currently no priority 1 entities in service) is passed to the Gate block which then pushes the entity group to the InPreempt port of the Server block, indicating to the Server block which entity (if any) should be preempted. Preempted Priority 1 entities are sent to the OutPreempt port of the Server block and routed to a different part of the model (labeled Preempted Customers) where the remaining service time is computed. If the remaining service time is greater than zero, then the Priority 1 entity is sent back into the Priority Queue to wait to complete service. Otherwise, the preempted priority 1 entity is routed to the Disposer block and the number serviced is updated.

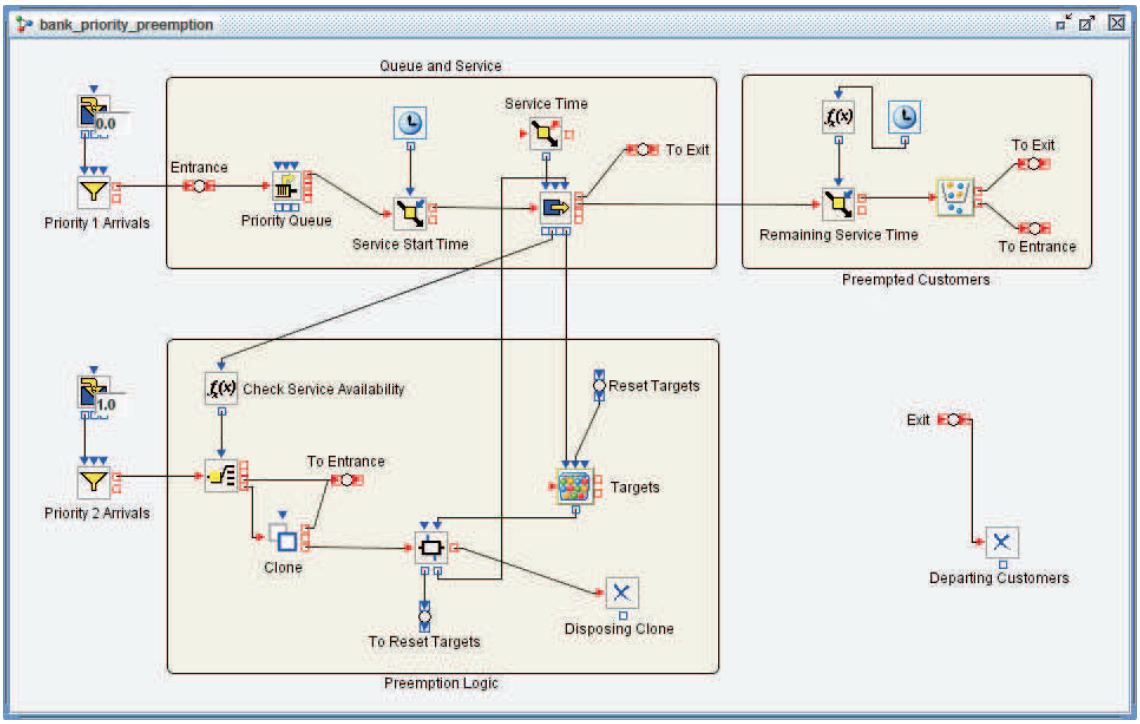

Figure 4: An example of priority-based preemption in Simulation Studio.

Scheduled preemption is primarily used for preempting mobile resources (resource entities) and is based on the requirements of a resource agenda. Sometimes allocated and seized resource entities need to be preempted from their

current controlling entities so that the resource entities can be reallocated to other controlling entities or sent to some other part of the model for processing. This type of preemption can be triggered by the Immediate Actions option in the Resource Scheduler block (that is, the user would select to adjust seized resources for the agenda). Figure 5 shows an example of how scheduled preemption can be modeled in Simulation Studio using the banking example from Figure 3. The entity holding blocks, including the Queue, Server, and Delay blocks, provide OutPreempt and OutResource output ports. If a teller resource entity allocated to a customer (controlling) entity that is currently held in the Delay block (labeled Hold Teller) is adjusted preemptively, the Delay block attempts to force the customer entity out of the block's OutPreempt port and the resource entity out of the OutResource port. In the model in Figure 5, the preempted customer entity is routed to a Disposer block (labeled Preempted Customers) and the teller resource entity is routed back to the Teller Pool. In general, if the OutPreempt port is not connected to a port of another block, the controlling entity remains in the entity holding block until it completes its service time. If the OutResource port is not connected, the adjusted resource entity remains allocated to its controlling entity.

The post-processing of preempted entities and resources is often highly specific to the system being modeled. For example, when an entity is preempted from a service, it may first be required to finish its remaining service time (as shown in Figure 4), it might be required to restart its service time from the beginning, or it may be required to move on to some other part of the model. Simulation Studio provides the necessary modeling constructs to handle all of these possible situations.

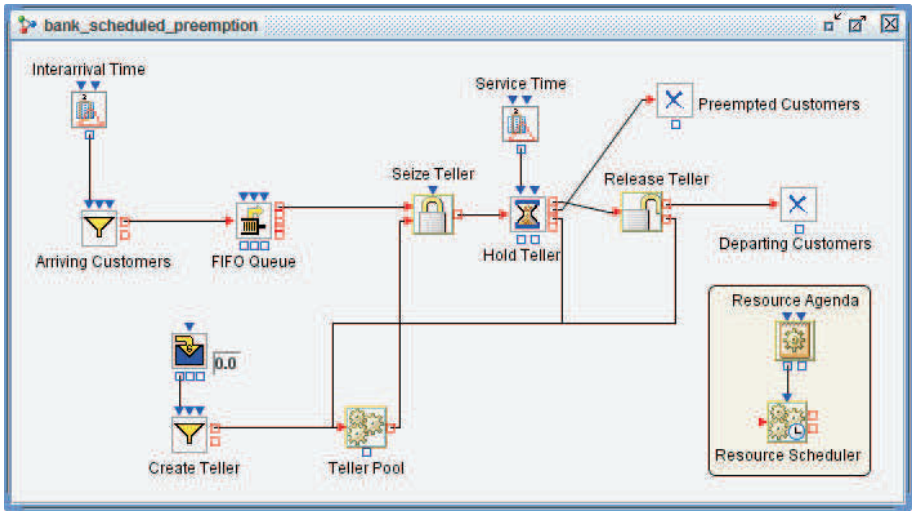

Figure 5: An example of scheduled preemption in Simulation Studio.

7 CONCLUSIONS

SAS Simulation Studio is an object-oriented, Java-based application for discrete-event simulation that uses a hierarchical, entity-based approach to resource management. In Simulation Studio, mobile resources can be created at simulation run-time that are special types of entities. 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. Resource entity rules are used extensively to locate resource entities for various resource management tasks. There are many advantages to an entity-based approach, including greater control over complicated resource management issues such as scheduling and preemption. The combination of (i) hierarchical, entity-based resources and (ii) resource entity rules for locating targeted resource entities provides a powerful paradigm for modeling resources in a simulation environment.

In the future, we plan to continue expanding the resource management capabilities in Simulation Studio. In particular, we are implementing a statistics collection block that will allow the user to compute statistics for specific groups of resource entities. The groups are defined using resource entity rules based on entity type and attribute value. We also plan to further enhance Simulation Studio's resource scheduling capabilities by determining a method to handle multiple and potentially conflicting scheduling requests for the same targeted resource entities. Furthermore, we are investigating additional heuristics for determining how to distribute a request (provided, perhaps, from a Resource Scheduler block) for increasing or decreasing the total number of resource units across multiple resource entities.

Currently, Simulation Studio employs a heuristic that evenly distributes the units across all resource entities. Ideally, the user would have control over how the increase/decrease in units should be distributed.

ACKNOWLEDGMENTS

The authors would like to thank professors James Wilson, Stephen Roberts, and Jeff Joines at North Carolina State University for many enlightening discussions on this paper. We would also like to thank Phil Meanor, Ed Hughes, and Manoj Chari at SAS Institute for their support of this work.

REFERENCES

Diamond, B., J. S. Lamperti, D. Krahl, A. Nastasi, and C. Damiron. 2007. *ExtendSim User Guide*. San Jose, California: Im-agine That Incorporated.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1995. *Design patterns: Elements of resusable object-oriented software*. Reading, Massachusetts: Addison-Wesley.

Harrell, C.R., and R.N. Price. 2002. Simulation modeling using ProModel technology. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yücesan, C.-H. Chen, J.L. Snowdon, and J.M. Charnes, 192–198. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Joines, J.A., and S.D. Roberts. 1998. Object-oriented simulation. In *Handbook of simulation*, ed. J. Banks, 397–427. New York: John Wiley & Sons, Inc.

Jones, J. 1995. SIMPROCESS III: Object-oriented business process simulation. In *Proceedings of the 1995 Winter Simulation Conference*, ed. C. Alexopoulos, K. Kang, W.R. Lilegdon, and D. Goldsman, 548–551. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Kelton, W.D., R.P. Sadowski, and D.T. Sturrock. 2007. *Simulation with Arena*. 4th ed. Boston: McGraw-Hill.

SAS Institute Inc. 2009. SAS Simulation Studio 1.5: User's guide. Available via ⟨http://support.sas.com/documentation/cdl/en/simsug/62554/PDF/default/simsug.pdf⟩ [accessed April 13, 2010].

Schriber, T.J., and D.T. Brunner. 1998. How discrete-event simulation software works. In *Handbook of simulation*, ed. J. Banks, 765–811. New York: John Wiley & Sons, Inc.

XJ Technologies. 2009. Enterprise Library Reference Guide. Available via ⟨http://www.xjtek.com/anylogic/help/⟩ [accessed April 13, 2010].

AUTHOR BIOGRAPHIES

HONG CHEN is a developer in the operations research group at SAS Institute. His email address is ⟨Hong.Chen@sas.com⟩.

EMILY LADA is an operations research development tester at the SAS Institute. She is a member of INFORMS and her email address is ⟨Emily.Lada@sas.com⟩.