Proceedings of the 1987 Winter Simulation Conference
A. Thesen, H. Grant, W. David Kelton (eds.)

THE SMALLTALK SIMULATION ENVIRONMENT, PART II

Verna E. Knapp
Computer Research Laboratory, MS 50-662
Tektronix Laboratories
P.O. Box 500
Beaverton, Oregon

ABSTRACT

This paper discusses the use of Smalltalk as a discrete event simulation environment. SimTalk, which is an extention of Smalltalk, is also discussed. Smalltalk is an object oriented language with interactive programming support, multiple process support, and interactive graphics. SimTalk adds queueing support, statistics gathering, simulation oriented graphics, and an interactive user interface which greatly simplifies modification of running simulation experiments.

1. Simulation with Objects and Processes

The real world can be modeled of as a set of objects with concurrent processes associated with them. The objects interact with each other, and this can be modeled as communicating processes, with the process communication taking the form of messages sent to the objects together with the resulting replies.

In a simulation, objects of the same kind can be said to belong to a "class" of objects which behave according to some particular protocol. Each object is an instance of the class. For example, in a simulation of a ski resort, each skier would be an in instance of a Skier class. All skiers would then behave according to the same protocol. Each execution of the protocol would be a process associated with a skier.

Time is modeled in such a simulation by maintaining a simulated clock and a time queue, and suspending the execution of a process until it reaches a place in the time queue which corresponds with the current simulated time. When this occurs it is removed from the time queue and execution of the process resumes until the protocol requires it to wait until a later time. Then the process is suspended and put back in the time queue at a point corresponding to the time at which it will again be resumed.

Processes may also be suspended while waiting for other types of events to occur. An example of this is queueing. For instance, the skier joins a lift line. This is modeled by suspending the skier process until the skier reaches the front of the line and is picked up by the ski lift.

2. Smalltalk Simulation

In Smalltalk, everything is an object. There are one or more processes, and at any given point in time, a process is associated with one object. Communication takes the form of an object sending a message to another object, and receiving a reply.

Objects in Smalltalk are instances of classes. Each class of objects has a set of methods associated with it which are invoked by sending a message to the object. These methods would correspond to the simulated object's protocols.

Smalltalk supports class hierarchies and subclassing. This means that a class can be defined as a subclass of another class. An object which is an instance of a subclass understands messages which invoke methods of its superclass as well as its own methods.

Smalltalk also supports multiple concurrent processes. A process can be created, suspended, resumed, and terminated. Thus Smalltalk needs only a time queue and other queueing with process control to correspond to the real world model as described above. SimTalk implements this additional support plus a large number of user interface and simulation support features in Smalltalk.

3. Interactive Simulation

An interactive simulation environment would have graphic displays of variables which are of interest in the simulation. It would allow the user to suspend the simulation processes, modify variables, add or delete simulated objects, and continue the simulation. This allows the user to develop a better intuition as to what is going on in the simulated system, and to experiment with the system to quickly determine which ranges of the simulated variables will be of the most interest. These features are also of use in debugging a simulation, since abnormal results often show up very quickly on the displays.

Smalltalk includes an interactive programming environment. This means that it is possible to suspend a process in Smalltalk, modify its variables, recompile some or all of the methods, and continue execution of the process. It is also possible to use the debugger to single-step through a program and examine the variables at each step if necessary.

Smalltalk also includes substantial support for graphic displays. This graphic support has been extended in SimTalk to provide graphs, gauges, and animation.

4. SimTalk

SimTalk defines a simulation control class which maintains the time queue, the simulated clock, and a set of queueing points. This class also acts as a central communication point for the simulation, and controls the interactive user interface. This class is called SimTalk.

SimTalk also defines an abstract simulated object class called SimTalkObject. This class has methods for commonly required simulation activities such as entering the time queue or one of the other queues, communicating with the user interface, and creating and scheduling other simulated objects.

A user of SimTalk will generally create a subclass of SimTalk to represent the simulated world, and one or more subclasses of SimTalkObject to represent the classes of objects to be simulated in that world. Each SimTalkObject subclass will have an actions method defined which defines the protocol for the simulated object to follow when it arrives in the simulated world. The user will write these subclasses and methods in Smalltalk.

Normally these are the only two class in SimTalk for which the user will need to define subclasses. There are a large number of other classes in SimTalk which the user will normally use without adding any subclasses or methods to them. These classes include random number generators, probability distributions, statististics gatherers, statistics analysis, customer/server queueing points for coordinating coprocesses, inventory queueing points for the use of producers and consumers, boolean electronic gates, and classes which provide graph display and animation functions and an interactive user interface.

5. How to Write a SimTalk Simulation

5.1 SimTalk

SimTalk is the class which defines the simulated world. It maintains the simulated clock and the time queue, and it controls creating, suspending, resuming, and terminating processes. It contains a list of queueing points other than the time queue. These queueing points are known as "resources". SimTalk also controls the various windows which may be opened on a simulation. These windows include an interactive user interface which allow the user to suspend the simulation, modify it, and continue it quite easily. There are also windows for graphs and gauges, animated display of icons of the simulated objects, and interactive networks of boolean gates. The user may wish to define variables which are global to the simulated world in a subclass of SimTalk.

Smalltalk allows the user to group classes by category. The interactive user interface uses this feature of Smalltalk by allowing the user to interactively instantiate those subclasses of SimTalkObject which are defined in the same category as the subclass of SimTalk which has been instantiated and is controlling the interface.

Thus, when the user subclass of SimTalk which will control the simulation is defined, it should be defined in a user-defined category which will contain all of the user-defined subclasses of SimTalkObject which will pertain to this simulation. In our example of writing a simulation, we have defined category SimTalk-SkiResort-Interactive to contain all of these classes. The subclass of SimTalk which is being defined is called SimTalkSkiResort.

5.1.1 Initialization

There are some standard methods which the user will supply as needed to initialize a subclass of SimTalk. If the user does not need a particular method, it may be omitted. They are:

- initializeVariables
- initializeObjectArrivals
- initializeResourcePool
- initializeOptionalViews
- initializeGatesOption

If the user has defined variables in the subclass of SimTalk, the method initializeVariables must be defined to initialize these variables. If other objects will require access to the contents of these variables, the user must write methods to provide that access. In our example, there are five variables. To keep count of the number of lifts in the resort, there is liftCount. It is accessed by skiers so they will know when they have reached the top. The other four variables are probability distribution streams which were centralized for efficiency's sake. Each skier will get a pointer to these streams upon arrival at the ski resort. Thus the method initializeVariables looks like:

```
initializeVariables
  "Initialize instance variables and create the probability
   distribution streams which will be used by the skiers"

  liftCount ← 0.
  bernoulli5 ← SimTalkBernoulli parameter: 0.5.
  bernoulli8 ← SimTalkBernoulli parameter: 0.8.
  uniform40100 ← SimTalkUniform from: 40 to: 100.
  uniform2080 ← SimTalkUniform from: 20 to: 80
```

The access methods for these variables are addLifts:, lifts, bernoulli5, bernoulli8, uniform2080, and uniform40100.

If the user wants to specify the creation of simulated objects under program control and at the beginning of the simulation, the method initializeObjectArrivals would be used. Alternatively, simulated objects may create and schedule other simulated objects at any time, or the user may use the interactive user interface to create and schedule the arrival of simulated objects. In our example simulation, the ski resort creates a stream of skiers. The ski lifts will be created interactively by the user. Thus the method initializeObjectArrivals looks like:

```
initializeObjectArrivals
  "Create an arriving stream of new skiers"

  self
    queueCreationOf: SimTalkSkier
    timeDistribution: (SimTalkExponential mean: 1)
    startTime: self simTime + (SimTalkExponential mean: 1) next
```

If the users wants to specify a set of initial resources (queueing points, discussed below), the method initializeResourcePool may be defined. Alternatively the user may specify resources interactively, or resources may be created at any time by simulated objects. In our example program, each ski lift at its own initialization time creates the customer/server resource and the slope inventory resource (count of skiers on that slope) which are associated with it. Thus initializeResourcePool was not defined in this simulation.

If the user wants views of the simulation other than the interactive user interface lists, the method initializeOptionalViews is used to specify what views will be required. For our example only the graph view will be needed. The other options are a switch view (for logic simulation and interactive switches), and an animation view (for icons of the objects in the simulation). In our example initializeOptionalViews looks like:

```
initializeOptionalViews

"This simulation will require a graph view"

    self initializeGraphView
```

If logic simulation is being done and the user wants to define gates interactively, the method initializeGatesOption must contain the statement "self initializeGates". Our example does not use logic simulation.

5.1.2 Time Queue Access

The instance of SimTalk which controls a given simulation maintains a queue of waiting processes which will resume execution at a specified simulated time. The queue is sorted according to the simulated time at which the process is to resume. The SimTalk object maintains a count of the processes which are in execution.

This count is incremented when a process is resumed or created, and it is decremented when a process is suspended or terminated. When the count reaches zero, the SimTalk finds the first process in the queue and sets the simulated clock to the simulated time at which that process was to resume execution. Then it removes that process from the time queue and resumes execution of that process. It also resumes execution of all processes bearing the same time stamp as that first process and removes them from the queue. Having resumed execution of all eligible processes, it then yields possession of the processor to them. It will regain control when all of them have either terminated or been suspended.

Any simulated object's executing process can request to be suspended and put into the time queue with a request to resume execution at a specific simulated time. Any process can request the arrival of any simulated object in the system at a specific simulated time. When a simulated object arrives in the system, it receives the message actions. The method actions must be specified by the user for each class of simulated object, since it specifies the simulated actions of the object. In addition to these normal scheduling activities, SimTalk also understands requests to reschedule queued processes or to terminate them.

5.1.3 Queueing Points (Resources)

In addition to the time queue there are other queueing points at which suspended processes may wait. These queueing points are known as resources. There are two main classes of resources, SimTalkCustomerServer, and SimTalkInventory. The user requests SimTalk to create instances of the appropriate type of resource, specifying a name for each instance. This may be done interactively, or it may be done under program control. Simulated objects request access to a resource either through the name, or through a pointer to the resource. The SimTalk object keeps a collection of resources and can provide a pointer to a resource if it is requested by name. Queueing at resources is first-come, first served within priority. A queued process may specify a time limit on how long it will wait before being resumed with its request possibly unsatisfied. A resource will also respond to queries as to queue length, inventory size, and the position of a particular object in the queue.

When a request is queued, the resource requests the SimTalk object to suspend the process which made the request, and to decrement the count of active processes. When the condition for which the request was queued is satisfied, the resource requests the SimTalk object to resume the process which was waiting and to increment the count of active processes.

SimTalkCustomerServer, and its subclass SimTalkGroupCustomerServer, coordinate coprocesses. A server process requests access to one or more customers, and when the appropriate customers request service, the server process is resumed and given pointers to the customers. The server may request parameters which each customer has passed, and it may pass parameters to the customers when it resumes the customer process(es). It may either continue execution or suspend itself when it resumes the customer process(es). Likewise, the customers may request and pass parameters to the server(s), and may either continue execution or suspend themselves when they resume execution of the server process(es). In general, one or more servers may be coordinated with one or more customers in an extended conversation with parameter passing if necessary.

SimTalkInventory, and its subclass SimTalkLimitedInventory, maintain a count of an inventory. This count is incremented by requests to create, and decremented by requests to consume. Queueing occurs among consumers if the inventory is smaller than the requested amount, and it occurs among producers if the size of the inventory is limited and the produce request would cause the inventory to exceed the limit.

In our example, a SimTalkCustomerServer resource is used for each lift to coordinate the skiers with the lift in a customer server relationship. A SimTalkInventory resource is used to count the number of skiers on a ski slope. The skier produces 1 of the resource when entering the slope, and consumes 1 of the resource when leaving it. The resources are created when the lift, which is a SimTalkObject as discussed below, is initialized.

5.2 SimTalkObject

For each class of object to be simulated, the user must define a subclass of class SimTalkObject. This subclass will define the behaviour of objects of that class in the simulation. Each SimTalkObject is created having a pointer to the SimTalk control object which controls it. This pointer is in the instance variable "simControl". It also is numbered sequentially in the order of creation and the number is stored in the instance variable "numbered". There are also instance variables for the icon and its position if animation is being used.

5.2.1 Initialization

There are two methods which the user may define to initialize an instance of a SimTalkObject. If there are instance variables to be initialized, use the method initialize. If animation is being used, initializeIcon can be used to initialize the icon and sets its initial position, and to tell the SimTalk that this object will appear in the animation.

In our example, there are two subclasses of SimTalkObject. They are SimTalkLift and SimTalkSkier.

When an instance of SimTalkLift is created, it must create its own customer/server resource and its own inventory resource. It also notifies the SimTalk that there is one more lift. Animation is not being used in our example. Thus only the method initialize is defined for our lift, and it looks like:

```
initialize
  "Create a customer server resource to coordinate this lift
  with its customers"

  liftLine ← self makeCustomerServerNamed: 'lift' ,
                    numbered printString.

  "Create an inventory resource to keep a count of the
  number of skiers on the ski slope associated with this lift.
  The skier will produce one item for this inventory when
  he starts down the slope, and consume one item from this
  inventory when he reaches the bottom of this slope."

  self makeInventoryNamed: 'slopeUser' , numbered printString.

  "Notify the control that another lift has been created"

  simControl addLifts: 1
```

SimTalkSkier initializes a skier to start by going up on lift 1. It gets pointers to the probability distributions which the skier uses from the SimTalkSkiResort control object. Thus the initialize method looks like:

```
initialize
  "initialize instance variables, and get access to the
   probability distributions"

   atLift ← 1.
   lastDirection ← #up.
   bernoulli5 ← simControl bernoulli5.
   bernoulli8 ← simControl bernoulli8.
   uniform40100 ← simControl uniform40100.
   uniform2080 ← simControl uniform2080
```

5.2.2 Actions

For each class of simulated object, its simulated actions must be defined. This is done by defining a method called "actions", which may of course call other methods as needed.

A SimTalkLift schedules itself to arrive in the simulation with a pair of empty chairs at regular simulated time intervals. Each time it arrives, it schedules the next arrival, picks up at most 2 skiers, holds onto the skiers long enough for them to reach the top of the lift, and then resumes the skier processes. Thus the actions method for a SimTalkLift looks like:

```
actions
  | skierCount skiers |

"Schedule the arrival of the next pair of chairs"

   simControl queueArrivalOf: self afterWait: 5.0.

"Count the skiers in the line"

   skierCount ← self countCustomersAt: liftLine.

"Pick 0, 1, or 2 skiers without making the chair wait"

   skierCount > 0
    ifTrue:
     [skierCount > 1
      ifTrue:
        [skiers ← self findCustomers: liftLine count: 2]
      ifFalse:
        [skiers ← self findCustomer: liftLine].

"Hold the skiers in the chairs for 12 time units"

   self waitFor: 12.0.

"Resume the skier processes when they get off the chairs at the top"

   skiers resumeAllProcesses]
```

A SimTalkSkier arrives each time it leaves the top of a lift, each time it arrives at the bottom of a slope, and when it arrives for the first time at the ski resort. It must maintain the count of skiers on the slopes by incrementing and decrementing the inventory resources appropriately, decide whether to go up or down based on its position on the slope and the length of lift lines, and decide whether to go home, based on being at the bottom lift with the line at that lift being too long. Thus the SimTalkSkier actions method looks like:

```
actions

lastDirection = #down
 ifTrue:
  ["remove self from slope user count
    for the slope I just came down"

   self consume: 1 ofResourceNamed:'slopeUser' , atLift printString.
   lastDirection ← #up.

   "Choose next direction to go. Consider line length at next lift
    and at lower lift, and consider a preference for going to the
    top most of the time."

   bernoulli5 next = 0 & (atLift > 1)
     ifTrue: [lastDirection ← #down]].
atLift > simControl lifts
 ifTrue: ["if at top, must go down" lastDirection ← #down]
 ifFalse:
  [bernoulli8 next = 0 & (atLift > 1)
     ifTrue: [lastDirection ← #down].

"Consider line lengths and choose direction"

atLift > 1 ifTrue:
 [(self countCustomersAtResourceNamed:'lift' , (atLift - 1)
    printString) > uniform40100 next
       ifTrue: [lastDirection ← #up]].
(self countCustomersAtResourceNamed:'lift' , atLift printString)
    > uniform40100 next
      ifTrue:
       ["The line is too long"
         atLift = 1 ifTrue:
           ["And I am at the bottom lift so go home"
             ↑self].
  lastDirection ← #down]].
lastDirection = #up
 ifTrue:
  ["Going up, so I must wait in lift line"
   self findServerAtResourceNamed: 'lift' , atLift printString.
   atLift ← atLift + 1.
   simControl queueArrivalOf: self afterWait: 0.0]
 ifFalse:
  ["Going down. Count myself as a slope user for that slope
    and enter time queue at the time at which I will arrive at
    the bottom of that slope"
   atLift ← atLift - 1.
   self create: 1
       ofResourceNamed: 'slopeUser' , atLift printString.
simControl queueArrivalOf: self afterWait: uniform2080 next]
```

5.2.3 Interactive User Interface

In our example, the user has chosen to interactively specify the creation of ski lifts. The user has the power to create new lifts, to create additional streams of arriving skiers, to remove specific skiers from the slope, to suspend the activity of a skier or a ski lift and later to resume that activity, to display graphs of the lift lines and the number of slope users, and to change the priority of a skier in a lift line. This is all quickly and easily done using the interactive user interface.

The interactive user interface creates a list view for the running simulation. There are popup yellow button (middle button on the mouse) menus for each of the lists in the view. Each of these

menus will contain a series of list specific functions, plus entries to allow the user to suspend, continue, stop, or inspect the simulation. Whenever the simulation is suspended, the lists are updated to match the current values of the simulation.

The list view will contain a number of lists. The leftmost one will be a list of all subclasses of SimTalkObject which are in the same category as this subclass of SimTalk. A yellow button menu may be popped up on this list. This menu is used to create new objects of the classes in the list. These objects are numbered sequentially as they are created. There is also a yellow button menu selection to create a stream of new instances of the selected object class. It will cause a view of a set of lists to pop up which can be used to specify starting time for creation of the stream, number of objects to be created (finite or infinite), how many times each object will arrive in the simulation (once or infinitely many times), and the probability distributions for the intercreation times and the interarrival times of the objects in the stream. Another yellow button menu selection will allow the user to display all existing objects of the class selected. A yellow button menu can be popped up on this resulting display and used to suspend, resume, kill, or inspect the objects in the display. There is also a yellow button menu entry to pop up a list of gates which can be created, interconnected, and simulated. These gates are displayed in the switch view. It is also possible to inspect any entry in the class list.

The second pane in the list view contains a list of the entries in the time queue. A yellow button menu on this list allows the user to reschedule or terminate processes which are waiting in the time queue. A small pane below this list contains the current simulated time when the simulation has been suspended.

The third pane in the list view contains a list of resources in the simulation. A yellow button menu on this list allows the user to create and name more resources and to create graph displays on various aspects of the resources. These displays are automatically placed in the graph view.

The fourth pane in the list view contains a list of the queue elements for the resource selected in the third pane. A yellow button menu on this list allows the user to inspect, kill, or change the priority of processes which are queued for a resource.

6. Other Simulation Services of SimTalk

6.1 Animation

Animation is done by scheduling an animation control simulated object to arrive at regular intervals and update the animation view by querying those simulated objects which have requested animation. These objects return their current icon and current position in the animation view. Simulated objects may send requests to be animated or removed from the animation to the SimTalk object in control of the simulation.

6.2 Random Number Generators and Probability Distributions

There are 5 pseudo-random number generators supplied with SimTalk. One of them supplies 16 bit numbers, three supply 31 bit numbers, and one provides 32 bit numbers. All of them can be duplicated to provide antithetical variates. If none of them are satisfactory to the user, it is quite easy for the user to add another random number generator to the system and use it.

All probability distributions supplied with the system provide standard and antithetical variates. The following distributions are supported: bernoulli, binomial, exponential, gamma, geometric, normal, poisson, uniform, and sample space. The user may easily add any others which may be required.

6.3 Statistics Gatherers and Observers

Several classes of simulated objects have been defined to gather statistics on the simulation. They can be scheduled to arrive at fixed intervals of simulated time, at time intervals which are drawn from a user specified probability distribution, or whenever some user specified condition based on that statistic is met. Whenever one of these objects arrives, it records or displays the statistical values it is monitoring, and it may schedule a user specified process to be executed.

Statistics gatherers and observers may monitor a queue at a resource, or the size of an inventory. Alternatively they may accumulate numbers sent to them by the simulated objects using +, -, setValue:, totalPlus:, totalMinus:, setTotalValue:, countPlus:, countMinus:, setCountValue:, totalCountPlus:, totalCountMinus:, and setTotalCountValue: messages. A single statistics gatherer thus can monitor the value of total attempts, the value of successful attempts, the count of total attempts, and the count of successful attempts for a single variable of interest. Statistics gatherers and observers also can report moving averages, rates, and time intervals between conditions.

Output can be displayed as a graph or bar gauge, or written to a file or a SimTalkSampleStream. The SimTalkSampleStream expects to contain a series of values for a single variable, and it understands requests for largest and smallest values, mean, variance, standard deviation, frequency distribution, and cumulative frequency distribution.

7. Summary

Smalltalk, together with SimTalk, provides a very powerful language supporting interactive discrete event simulation. The user can write a small amount of code defining the simulated world and the objects in it, and gain access to an experimental world which is easily monitored and modified. This allows the user to quickly zero in on the simulated variables which will be of interest. The Smalltalk programming environment provides the user with very powerful coding and debugging tools, leading to high productivity in writing and modifying simulations. SimTalk provides powerful queueing discipline support, extensive statistics gathering and monitoring capabilities, graphic display and animation support, and an effective interactive user interface, thereby saving the user much time and effort when these facilities are required. Thus SimTalk and Smalltalk allow a simulationist to quickly and easily write, debug, and modify simulation experiments.

AUTHOR'S BIOGRAPHY

VERNA E. KNAPP is a Senior Hardware Software Engineer in the Computer Research Laboratory of Tektronix Laboratories, Tektronix, Inc. She received a B.S. in mathematics from the University of Washington in 1969, and M.S. and Ph.D. degrees in Computer Science from the University of Washington in 1970 and 1985 respectively. She worked as a systems programmer, performance analyst, and capacity planner for Pacific Northwest Bell Telephone from 1970 to 1982. She has worked for Tektronix as a computer architect since 1982. Her current research focuses on applying Smalltalk and modern workstation technology to interactive discrete event simulation.

Verna E. Knapp
Computer Research Laboratory, MS 50-662
Tektronix Laboratories
P.O. Box 500
Beaverton, Oregon