The Bank[2]: More examples of SimPy Simulation

G A Vignaux

2005 Jan 4

SimPy Version: 1.5 or later

Table Of Contents

Introduction

The first Bank tutorial, The Bank , developed and explained a series of simulation models of a simple bank using SimPy. In various models, customers arrived randomly, queued up to be served at one or several counters, modelled using the Resource class, and, in one case, could choose the shortest among several queues. It demonstrated the use of the Monitor class to record delays and showed how a *model()* mainline for the simulation was convenient to execute replications of simulation runs.

In this extension to The Bank, I provide more examples of SimPy facilities for which there was no room before and for some that were developed since it was written. These facilities are generally more complicated than those introduced before. They include plotting, interrupts, waituntil and waitfor. The programs are roughly arranged in sections to do with Processes, Resources, and Monitors.

The acronym PEM stands for Process Execution Method, the method containing the *yield* statements in the Processes.

This tutorial should be read with the SimPy Manual or Cheatsheet at your side for reference.

The programs are available without line numbers and ready to go, in directory *bankprograms*. Some have trace statements for demonstration purposes, others produce graphical output to the screen. Let me encourage you to run them and modify them for yourself

Some facilities require SimPy version 1.5 or later.

Processes

Two new *yield* facilities were introduced in SimPy-1.5: *waituntil* and *waitevent* to allow processes to wait for a condition set up by other processes.

waituntil the Bank door opens

Customers arrive at random, some of them getting to the bank before the door is opened by a doorman. They wait for the door to be opened and then rush in and queue to be served.

This model uses the *waituntil* yield command. In the program listing the door is initially closed (line 5) and a function to test if it is open is defined at line 6.

The Doorman class is defined starting at line 9 and the single doorman is created and activated at at lines 56 and 57. The doorman waits for an average 10 minutes (label 14) and then open the door.

The Customer class is defined at 27 and a new customer prints out "Here I am" on arrival. If the door is still closed, he adds "but the door is shut" and settles down to wait (line 36), using the *yield waituntil* command. When the door is opened by the doorman the *dooropen* state is changed and the customer (and all others waiting for the door) proceed. A customer arriving when the door is open will not be delayed.

```
 1 """bank14: *waituntil* the Bank door opens"""
 2 from SimPy.Simulation  import *
 3 from random import *
 4
 5 door = 'Shut'
 6 def dooropen():
 7     return door=='Open'
 8
 9 class Doorman(Process):
10     """ Doorman opens the door"""
11     def openthedoor(self):
12         """ He will open the door when he arrives"""
13         global door
14         yield hold,self,expovariate(1.0/10.0)
15         door='Open'
16         print "%7.4f Doorman: Ladies and "\
17               "Gentlemen! You may all enter."%(now(),)
18
19 class Source(Process):
20     """ Source generates customers randomly"""
21     def generate(self,number,rate):
22         for i in range(number):
23             c = Customer(name = "Customer%02d"%(i,))
24             activate(c,c.visit(timeInBank=12.0))
25             yield hold,self,expovariate(rate)
26
27 class Customer(Process):
28     """ Customer arrives, is served and leaves """
29     def visit(self,timeInBank=10):
30         arrive=now()
31
32         if dooropen(): msg = ' and the door is open.'
33         else:  msg = ' but the door is shut.'
34         print "%7.4f %s: Here I am%s"%(now(),self.name,msg)
```

```
35
36          yield waituntil,self,dooropen
37
38          print "%7.4f %s: I can  go in!"%(now(),self.name)
39          wait=now()-arrive
40          print "%7.4f %s: Waited %6.3f"%(now(),self.name,wait)
41
42          yield request,self,counter
43          tib = expovariate(1.0/timeInBank)
44          yield hold,self,tib
45          yield release,self,counter
46
47          print "%7.4f %s: Finished    "%(now(),self.name)
48
49 counter = Resource(1,name="Clerk")
50
51 def model(SEED=393939):
52     seed(SEED)
53
54     initialize()
55     door = 'Shut'
56     doorman=Doorman()
57     activate(doorman,doorman.openthedoor())
58     source = Source()
59     activate(source,source.generate(number=5,rate=0.1),0.0)
60     simulate(until=400.0)
61
62 model()
```

An output run for this programs shows how the first three customers have to wait until the door is opened.

```
 0.0000 Customer00: Here I am but the door is shut.
 5.6941 Customer01: Here I am but the door is shut.
15.8155 Customer02: Here I am but the door is shut.
22.2064 Doorman: Ladies and Gentlemen! You may all enter.
22.2064 Customer00: I can  go in!
22.2064 Customer00: Waited 22.206
22.2064 Customer01: I can  go in!
22.2064 Customer01: Waited 16.512
22.2064 Customer02: I can  go in!
22.2064 Customer02: Waited  6.391
22.4603 Customer03: Here I am and the door is open.
22.4603 Customer03: I can  go in!
22.4603 Customer03: Waited  0.000
24.8884 Customer00: Finished
30.8017 Customer04: Here I am and the door is open.
30.8017 Customer04: I can  go in!
30.8017 Customer04: Waited  0.000
57.5367 Customer01: Finished
58.6695 Customer02: Finished
91.0096 Customer03: Finished
93.5228 Customer04: Finished
```

Wait for the doorman to give a signal: *waitevent*

Customers arrive at random, some of them getting to the bank before the door is open. This is controlled by an automatic machine called the doorman which opens the door only at intervals of 30 minutes (it is a very secure bank). The customers wait for the door to be opened and all those waiting enter and proceed to the counter. The door is closed behind them.

This model uses the *yield waitevent* command which requuires a *SimEvent* to be defined (line 5). The Doorman class is defined at line 6 and the doorman is created and activated at at labels 51 and 52. The doorman waits for a fixed time (label 11) and then tells the customers that the door is open. This is achieved on line 12 by signalling the *dooropen* event.

Tthe Customers class is defined at 23 and in its PEM, when a customer arrives, he prints out "Here I am". If the door is still closed, he adds "but the door is shut" and settles down to wait for the door to be opened using the *yield waitevent* command (line 31). When the door is opened by the doorman (that is, he sends the *dooropen.signal()* the customer and any others waiting may proceed.

```
 1 """ bank13: Wait for the doorman to give a signal: *waitevent*"""
 2 from SimPy.Simulation  import *
 3 from random import *
 4
 5 dooropen=SimEvent("Door Open")
 6 class Doorman(Process):
 7     """ Doorman opens the door"""
 8     def openthedoor(self):
 9         """ He will opens the door at fixed intervals"""
10         for i in range(5):
11             yield hold,self, 30.0
12             dooropen.signal()
13             print "%7.4f You may enter"%(now(),)
14
15 class Source(Process):
16     """ Source generates customers randomly"""
17     def generate(self,number,rate):
18         for i in range(number):
19             c = Customer(name = "Customer%02d"%(i,))
20             activate(c,c.visit(timeInBank=12.0))
21             yield hold,self,expovariate(rate)
22
23 class Customer(Process):
24     """ Customer arrives, is served and leaves """
25     def visit(self,timeInBank=10):
26         arrive=now()
27
28         if dooropen.occurred: msg = '.'
29         else:  msg = ' but the door is shut.'
30         print "%7.4f %s: Here I am%s"%(now(),self.name,msg)
31         yield waitevent,self,dooropen
32
33         print "%7.4f %s: The door is open!"%(now(),self.name)
34
35         wait=now()-arrive
36         print "%7.4f %s: Waited %6.3f"%(now(),self.name,wait)
```

```
37
38          yield request,self,counter
39          tib = expovariate(1.0/timeInBank)
40          yield hold,self,tib
41          yield release,self,counter
42
43          print "%7.4f %s: Finished    "%(now(),self.name)
44
45 counter = Resource(1,name="Clerk")
46
47 def model(SEED=393939):
48     seed(SEED)
49
50     initialize()
51     doorman=Doorman()
52     activate(doorman,doorman.openthedoor())
53     source = Source()
54     activate(source,source.generate(number=5,rate=0.1),0.0)
55     simulate(until=400.0)
56
57 model()
```

An output run for this programs shows how the first three customers have to wait until the door is opened.

```
 0.0000 Customer00: Here I am but the door is shut.
22.2064 Customer01: Here I am but the door is shut.
27.9005 Customer02: Here I am but the door is shut.
30.0000 You may enter
30.0000 Customer02: The door is open!
30.0000 Customer02: Waited  2.099
30.0000 Customer01: The door is open!
30.0000 Customer01: Waited  7.794
30.0000 Customer00: The door is open!
30.0000 Customer00: Waited 30.000
37.9738 Customer02: Finished
38.0219 Customer03: Here I am but the door is shut.
40.6558 Customer01: Finished
46.3632 Customer04: Here I am but the door is shut.
60.0000 You may enter
60.0000 Customer04: The door is open!
60.0000 Customer04: Waited 13.637
60.0000 Customer03: The door is open!
60.0000 Customer03: Waited 21.978
73.3042 Customer00: Finished
74.4369 Customer04: Finished
90.0000 You may enter
106.7770 Customer03: Finished
120.0000 You may enter
150.0000 You may enter
```

Resources

Resources can defined as priority-based queue as well as the usual first-in, first-out (FIFO or LCFS). They can also have preemptive priority.

A Resource with Priority to vary the number of units

Although we can establish a Resource with different numbers of units when the Resource is established it is not easy to adjust the number of units as the simulation proceeds. In this Bank model we use the Resource priority system to remove units from the counter for a short period at random times.

Much of the model is standard: a *Source* (line 9) generates *Customers* (line 17) at random. Each Customer requests a unit of the *counter* (i.e. a *Clerk*) at line 22. For simplicity no Monitor is used here.

The first difference from the usual model is that the *counter* Resource (line 55) is given 1 unit and establish it as a *PriorityQ* (but not one that is *preemptable*).

We create a *clerk1*, a *ClerkProcess* on line 61 and activate it (line 62). The *ClerkProcess* is defined at line 30. It requests a counter unit after a mean time of 15 minutes with the very high priority of 100 (line 42). As the *counter* is not *preemptable* it will wait in the *waitQ* until the current customer in service has finished. This has the effect of removing one of the units from the *counter* Resource. The *clerk1* holds it for 3 minutes (line 47) before releasing it for use by Customers.

```
 1 """bank18: A Resource with Priority to vary the number of units"""
 2 tracing=0
 3 if tracing:
 4     from SimPy.SimulationTrace  import *
 5 else:
 6     from SimPy.Simulation  import *
 7 from random import *
 8
 9 class Source(Process):
10     """ Source generates customers randomly"""
11     def generate(self,number,rate):
12         for i in range(number):
13             c = Customer(name = "Customer%02d"%(i,))
14             activate(c,c.visit(timeInBank=12.0))
15             yield hold,self,expovariate(rate)
16
17 class Customer(Process):
18     """ Customer arrives, is served and leaves """
19     def visit(self,timeInBank):
20         print "%8.4f %s: Arrived     "%(now(),self.name)
21
22         yield request,self,counter
23         print "%8.4f %s: Got counter "%(now(),self.name)
24         tib = expovariate(1.0/timeInBank)
25         yield hold,self,tib
26         yield release,self,counter
27
28         print "%8.4f %s: Finished    "%(now(),self.name)
29
30 class ClerkProcess(Process):
31     """ This process removes a clerk from the counter
32     after an average of 20 minutes.
```

```
33      The clerk returns after 5 minutes """
34      def serverProc(self):
35          while True:
36              # The clerk starts off working but leave after an average of 10 minutes
37              yield hold,self,expovariate(1.0/15.0)
38              print "%8.4f %s: urgent.  Free:"\
39                    "%d, %d waiting"%(now(),self.name,counter.n,len(counter.waitQ))
40
41              # The first free clerk is removed
42              yield request,self,counter,100
43              print "%8.4f %s: leaves.  Free:"\
44                    "%d, %d waiting"%(now(), self.name,counter.n,len(counter.waitQ))
45
46              #period away is 3 minutes
47              yield hold,self,3.0
48
49              #clerk returns
50              yield release,self,counter
51              print "%8.4f %s: returns. Free:"\
52                    "%d, %d waiting"%(now(), self.name,counter.n,len(counter.waitQ))
53
54 # The counter is a Priority Queue but is not preemptive
55 counter = Resource(1,name="Clerk",qType=PriorityQ)
56
57 def model(SEED=393939):
58      seed(SEED)
59
60      initialize()
61      clerk1 = ClerkProcess('Clerk')
62      activate(clerk1, clerk1.serverProc())
63      source = Source('Source')
64      activate(source,source.generate(number=20,rate=0.1),0.0)
65      simulate(until=200.0)
66
67 model()
```

This is a simple model with only one unit in the *counter* Resource. We could model a number of clerks in a several ways. The simplest would be to set up the resource with N units (these are anonymous, of course). Then establish and activate one *ClerkProcess* for each unit. This does quite model the real situation, though, because it does not handle the clerks individually. When a *ClerkProcess* requests a unit it just gets the first one to finish serving.

A more complicated way would be to have N counters each with 1 unit and having a corresponding *ClerkProcess* to remove its unit from the counter when it becomes free. The complication, then, is to decide to which *counter* the customer goes to upon arrival.

At lines 4 and 6 there are alternative *import* statements. If we use *SimPy.SimulationTrace* instead of *SimPy.Simulation* an automatic trace is added to the printed output already in place. This gives a more detailed picture of what is happening.

Using priorities to increase the clerks for long queues

In this model we increase the number of active clerks when the queue exceeds a certain level. We start with 2 clerks and immediately grab one with high priority thus reducing the number available for service

to customers. We wait until the number in the waiting queue rises above 2. At that point we release the clerk to customer service. We then wait until the number in the waiting queue falls to 0 again at which point we take it back.

This is done using a single *ClerkProcess* and two functions *queuelong*, and *queueshort* to control a clerk (line 32). It is initialised and activated at lines 63 and 64. The *serverProc* method first removes the clerk with high priority (line 38) and then waits until the queue is long enough to return (line 42). At that point it releases the corresponding unit (line 44) and waits until the queue length falls to 0 using function *queueshort* (defined on line 56 and used on line 48). When that is satisfied the *serverProc* method recyles to the top (line 36) and immediately grabs the clerk with high priority.

queuelong is defined on line 53 and returns *True* when the length of the counter's *waitQ* rises above 2. *queueshort* is defined on line 56 returns "True" when the queue length falls to 0.

```
 1 """bank19: Using priorities to increase the  clerks for long queues"""
 2 tracing=0
 3 if tracing:
 4     from SimPy.SimulationTrace  import *
 5 else:
 6     from SimPy.Simulation  import *
 7
 8 from random import *
 9
10 class Source(Process):
11     """ Source generates customers randomly"""
12     def generate(self,number,rate):
13         for i in range(number):
14             c = Customer(name = "Customer%02d"%(i,))
15             activate(c,c.visit(timeInBank=12.0))
16             yield hold,self,expovariate(rate)
17
18 class Customer(Process):
19     """ Customer arrives, is served and leaves """
20     def visit(self,timeInBank):
21         print "%8.4f %s: Arrived     "%(now(),self.name)
22
23         yield request,self,counter
24         print "%8.4f %s: Got counter "%(now(),self.name)
25         tib = expovariate(1.0/timeInBank)
26         yield hold,self,tib
27         yield release,self,counter
28
29         print "%8.4f %s: Finished    "%(now(),self.name)
30
31
32 class ClerkProcess(Process):
33     """ This process removes a clerk from the counter
34     immediately."""
35     def serverProc(self):
36         while True:
37             # immediately grab the clerk
38             yield request,self,counter,100
39             print "%8.4f %s: leaves.  Free:"\
40                   "%d, %d waiting"%(now(),self.name,counter.n,len(counter.waitQ))
41
```

```
42              yield waituntil,self, queuelong
43
44              yield release,self,counter
45              print "%8.4f %s: needed .  Free:"\
46                    "%d, %d waiting"%(now(),self.name,counter.n,len(counter.waitQ))
47
48              yield waituntil,self,queueshort
49
50 # The counter is a Priority Queue but is not preemptive
51 counter = Resource(2,name="Clerk",qType=PriorityQ)
52
53 def queuelong():
54     return len(counter.waitQ)> 2
55
56 def queueshort():
57     return len(counter.waitQ)== 0
58
59 def model(SEED=393939):
60     seed(SEED)
61
62     initialize()
63     clerk1 = ClerkProcess('Clerk')
64     activate(clerk1, clerk1.serverProc())
65     source = Source('Source')
66     activate(source,source.generate(number=20,rate=0.1),0.0)
67     simulate(until=200.0)
68
69 model()
```

Monitors

Monitors are used to track and record values in a simulation. They store a list of [time,value] pairs, one being added whenever the *observe* method is called.

They have a set of simple statistical methods such as *mean* to calculate the average of the observed values ˜˜ useful in estimating the mean delay, for example. They also have the *timeAverage* method that calculates the time-weighted average of the recorded values. It determines the total area under the time˜value graph and divides by the total time. This is valuable for estimating the average number of customers in the bank, for example.

There is an important caveat in using this method. To estimate the correct time average you must certainly *observe* the value (say the number of customers in the system) whenever it changes (as well as at any other time you wish) but, and this is important, observing the *new* value. The *old* value was recorded earlier. In practice this means that if we wish to observe a changing value, *n*, using the Monitor, *Mon*, we must keep to the the following pattern:

```
n = n+1
Mon.observe(n,now())
```

Thus you make the change (not only increases) and *then* observe the new value. Of course the simulation time *now()* has not changed between the two statements.

One useful characteristic of Monitors is that they continue to exist after the simulation has been completed. Thus further analysis of the results can be carried out.

Plotting a Histogram of Monitor results

A Monitor can construct a histogram from its data using the *histogram* method. In this model we monitor the time in the system for the customers. This is calculated for each customer in line 27, using the arrival time saved in line 17. We establish the Monitor at line 31 and the times are *observed* at line 28. The histogram is constructed from the Monitor, after the simulation has finished, at line 43.

The SimPy.SimPlot package allows simple plotting of results from simulations. Here we use the SimPlot *plotHistogram* method. The plotting routines are run after the main simulation (lines 45-49). The *plotHistogram* call is in line 46.

```
 1 """bank17: Plotting a Histogram of Monitor results"""
 2 from SimPy.Simulation  import *
 3 from SimPy.SimPlot import *
 4 from random import *
 5
 6 class Source(Process):
 7     """ Source generates customers randomly"""
 8     def generate(self,number,rate):
 9         for i in range(number):
10             c = Customer(name = "Customer%02d"%(i,))
11             activate(c,c.visit(timeInBank=12.0))
12             yield hold,self,expovariate(rate)
13
14 class Customer(Process):
15     """ Customer arrives, is served and leaves """
16     def visit(self,timeInBank):
17         arrive=now()
18         #print "%8.4f %s: Arrived     "%(now(),self.name)
19
20         yield request,self,counter
21         #print "%8.4f %s: Got counter "%(now(),self.name)
22         tib = expovariate(1.0/timeInBank)
23         yield hold,self,tib
24         yield release,self,counter
25
26         #print "%8.4f %s: Finished    "%(now(),self.name)
27         t = now()-arrive
28         Mon.observe(t)
29
30 counter = Resource(1,name="Clerk")
31 Mon=Monitor('Time in the Bank')
32 N = 0
33 def model(SEED=393939):
34     seed(SEED)
35
36     initialize()
37     source = Source()
38     activate(source,source.generate(number=20,rate=0.1),0.0)
39     simulate(until=400.0)
40
41 model()
42
43 Histo = Mon.histogram(low=0.0,high=200.0,nbins=20)
44
```

```
45 plt=SimPlot()
46 plt.plotHistogram(Histo,xlab='Time (min)',
47                   title="Time in the Bank",
48                   color="red",width=2)
49 plt.mainloop()
```

Monitoring a Resource

Now consider observing the number of customers waiting or executing in a Resource. Because of the need to *observe* the value after the change but at the same simulation instant, it is impossible to use the length of the Resource's *waitQ* directly with a Monitor defined outside the Resource. Instead Resources can be set up with built-in Monitors.

Here is an example using a Monitored Resource. We intend to observe the average number waiting and active in the *counter* resource. *counter* is defined at line 27 and we have set *monitored=True*. This establishes two Monitors: *waitMon*, to record changes in the numbers waiting and *actMon* to record changes in the numbers active in the *counter*. We need make no further change to the operation of the program as monitoring is then automatic. No *observe* calls are necessary.

At the end of the run in the *model* function, we calculate the *timeAverage* of both *waitMon* and *actMon* and return them from the *model* call (line 37). These can then be printed at the end of the program (line 39).

```
 1 """bank15: Monitoring a Resource"""
 2 from SimPy.Simulation  import *
 3 from random import *
 4
 5 class Source(Process):
 6     """ Source generates customers randomly"""
 7     def generate(self,number,rate):
 8         for i in range(number):
 9             c = Customer(name = "Customer%02d"%(i,))
10             activate(c,c.visit(timeInBank=12.0))
11             yield hold,self,expovariate(rate)
12
13 class Customer(Process):
14     """ Customer arrives, is served and leaves """
15     def visit(self,timeInBank):
16         arrive=now()
17         print "%8.4f %s: Arrived     "%(now(),self.name)
18
19         yield request,self,counter
20         print "%8.4f %s: Got counter "%(now(),self.name)
21         tib = expovariate(1.0/timeInBank)
22         yield hold,self,tib
23         yield release,self,counter
24
25         print "%8.4f %s: Finished    "%(now(),self.name)
26
27 counter = Resource(1,name="Clerk",monitored=True)
28
29 def model(SEED=393939):
30     seed(SEED)
31
```

```
32      initialize()
33      source = Source()
34      activate(source,source.generate(number=5,rate=0.1),0.0)
35      simulate(until=400.0)
36
37      return (counter.waitMon.timeAverage(),counter.actMon.timeAverage())
38
39 print 'Average waiting = %6.4f\nAverage active  = %6.4f\n'%model()
```

Plotting from Resource Monitors

waitMon and *actMon*, the two Monitors in a monitored Resource contain information that enables us to graph the output. Alternative plotting packages can be used; here we use the simple SimPy.SimPlot package just to graph the number of customers waiting for the counter. The program is a simple modification of the one that uses a monitored Resource.

The SimPlot package is imported at line 3. No major changes are made to the main part of the program except that I commented out the print statements. The changes occure in the *model* routine from lines 29 to 35. The simulation now generates and processes 20 customers (line 34). *model* does not return a value but the Monitors of the *counter* Resource still exist when the simulation has terminated.

The additional plotting actions take place in lines 39 to 42. Line 40-41 construct a step plot and graphs the number in the waiting queue as a function of times. *waitMon* is primarily a list of *[time,value]* pairs which the *plotStep* method of the SimPlot object, *plt* uses without change. On running the program the graph is plotted; the user has to terminate the plotting *mainloop* on the screen.

```
 1 """bank16: Plotting from  Resource Monitors"""
 2 from SimPy.Simulation  import *
 3 from SimPy.SimPlot import *
 4 from random import *
 5
 6 class Source(Process):
 7     """ Source generates customers randomly"""
 8     def generate(self,number,rate):
 9         for i in range(number):
10             c = Customer(name = "Customer%02d"%(i,))
11             activate(c,c.visit(timeInBank=12.0))
12             yield hold,self,expovariate(rate)
13
14 class Customer(Process):
15     """ Customer arrives, is served and leaves """
16     def visit(self,timeInBank):
17         arrive=now()
18         #print "%8.4f %s: Arrived     "%(now(),self.name)
19
20         yield request,self,counter
21         #print "%8.4f %s: Got counter "%(now(),self.name)
22         tib = expovariate(1.0/timeInBank)
23         yield hold,self,tib
24         yield release,self,counter
25
26         #print "%8.4f %s: Finished    "%(now(),self.name)
27
28 counter = Resource(1,name="Clerk",monitored=True)
```

```
29 def model(SEED=393939):
30     seed(SEED)
31
32     initialize()
33     source = Source()
34     activate(source,source.generate(number=20,rate=0.1),0.0)
35     simulate(until=400.0)
36
37 model()
38
39 plt=SimPlot()
40 plt.plotStep(counter.waitMon,
41         color="red",width=2)
42 plt.mainloop()
```

Acknowledgments

I thank Klaus Muller and the other developers and users of SimPy who have improved this document by sending their comments. I will be grateful for further corrections or suggestions. Could you send them to me: *vignaux* at *users.sourceforge.net*.

References

- Python website: http://www.Python.org
- SimPy website: http://sourceforge.net/projects/simpy
- The Bank: The Bank