The Bank part 2: More examples of SimPy Simulation

G A Vignaux

2007 October 22

Table Of Contents

1 Introduction

2 Priority Customers

- 2.1 Priority Customers without preemption
- 2.2 Priority Customers with preemption

3 Balking and Reneging Customers

- 3.1 Balking Customers
- 3.2 Reneging (or abandoning) Customers

4 Processes

- 4.1 Interrupting a Process.
- 4.2 `waituntil` the Bank door opens
- 4.3 Wait for the doorman to give a signal: `waitevent`

5 Monitors

- 5.1 Plotting a Histogram of Monitor results
- 5.2 Monitoring a Resource
- 5.3 Plotting from Resource Monitors

6 Acknowledgements

7 References

1 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 queueing with priority, possibly with preemption, reneging, plotting, interrupting, waiting until a condition occurs (`waituntil`) and waiting for events to occur.

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

SimPy itself can be obtained from: http://simpy.sourceforge.net/. It is compatible with Python version 2.3 onwards. The examples in this documentation run with SimPy version 1.5 and later.

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

2 Priority Customers

In many situations there is a system of priority service. Those customers with high priority are served first, those with low priority must wait. In some cases, preemptive priority will even allow a high-priority customer to interrupt the service of one with a lower priority.

SimPy implements priority requests with an extra numerical priority argument in the `yield request` command, higher values meaning higher priority. For this to operate, the requested Resource must have been defined with `qType=PriorityQ`.

2.1 Priority Customers without preemption

In the first example, we modify the program with random arrivals, one counter, and a fixed service time (like `bank07.py` in The Bank tutorial) to process a high priority customer. Warning: the `seed()` value has been changed to `98989` to make the story more exciting.

The modifications are to the definition of the `counter` where we change the `qType` and to the `yield request` command in the `visit` PEM of the customer. We also need to provide each customer with a priority. Since the default is `priority=0` this is easy for most of them.

To observe the priority in action, while all other customers have the default priority of 0, in lines 46 to 48 we create and activate one special customer, `Guido`, with priority 100 who arrives at time `23.0` (line 48). This is to ensure that he arrives after `Customer03`.

The `visit` customer method has a new parameter, `P=0` (line 21) which allows us to set the customer priority.

In lines 38 to 38 `counter` is defined with `qType=PriorityQ` so that we can request it with priority (line 26) using the statement `yield request,self,counter,P`

In line 24 we print out the number of customers waiting when each customer arrives.

```
 1 """ bank20: One counter with a priority customer """
 2 from SimPy.Simulation import *
 3 from random import expovariate, seed
 4
 5 ## Model components ------------------------
 6
 7 class Source(Process):
 8     """ Source generates customers randomly """
 9
10     def generate(self,number,interval,resource):
11         for i in range(number):
12             c = Customer(name = "Customer%02d"%(i,))
13             activate(c,c.visit(timeInBank=12.0,
14                                res=resource,P=0))
15             t = expovariate(1.0/interval)
16             yield hold,self,t
17
18 class Customer(Process):
19     """ Customer arrives, is served and  leaves """
20
21     def visit(self,timeInBank=0,res=None,P=0):
22         arrive = now()       # arrival time
23         Nwaiting = len(res.waitQ)
24         print "%8.3f %s: Queue is %d on arrival"%(now(),self.name,Nwaiting)
25
26         yield request,self,res,P
27         wait = now()-arrive  # waiting time
```

```
28         print "%8.3f %s: Waited %6.3f"%(now(),self.name,wait)
29         yield hold,self,timeInBank
30         yield release,self,res
31
32         print "%8.3f %s: Completed"%(now(),self.name)
33
34 ## Experiment data -------------------------
35
36 maxTime = 400.0  # minutes
37 k = Resource(name="Counter",unitName="Karen",
38              qType=PriorityQ)
39
40 ## Model/Experiment ------------------------------
41 seed(98989)
42 initialize()
43 s = Source('Source')
44 activate(s,s.generate(number=5, interval=10.0,
45                       resource=k),at=0.0)
46 guido = Customer(name="Guido     ")
47 activate(guido,guido.visit(timeInBank=12.0,res=k,
48                            P=100),at=23.0)
49 simulate(until=maxTime)
```

The resulting output is as follows. The number of customers in the queue just as each arrives is displayed in the trace. That count does not include any customer in service.

```
 0.000 Customer00: Queue is 0 on arrival
 0.000 Customer00: Waited  0.000
 2.166 Customer01: Queue is 0 on arrival
 5.121 Customer02: Queue is 1 on arrival
12.000 Customer00: Completed
12.000 Customer01: Waited  9.834
13.347 Customer03: Queue is 1 on arrival
23.000 Guido     : Queue is 2 on arrival
24.000 Customer01: Completed
24.000 Guido     : Waited  1.000
36.000 Guido     : Completed
36.000 Customer02: Waited 30.879
38.156 Customer04: Queue is 1 on arrival
48.000 Customer02: Completed
48.000 Customer03: Waited 34.653
60.000 Customer03: Completed
60.000 Customer04: Waited 21.844
72.000 Customer04: Completed
```

Reading carefully one can see that when `Guido` arrives `Customer00` has been served and left at `12.000`), `Customer01` is in service and two (customers 02 and 03) are queueing. `Guido` has priority over those waiting and is served before them at `33.162`. When `Guido` leaves at `45.162`, `Customer02` starts service.

2.2 Priority Customers with preemption

Now we allow `Guido` to have preemptive priority. He will displace any customer in service when he arrives. That customer will resume when `Guido` finishes (unless higher priority customers intervene). It requires only a change to one line of the program, adding the argument, `preemptable=True` to the `Resource` statement in line 38.

```
 1 """ bank23: One counter with a priority customer with preemption """
 2 from SimPy.Simulation import *
 3 from random import expovariate, seed
 4
 5 ## Model components ------------------------
 6
 7 class Source(Process):
 8     """ Source generates customers randomly """
 9
10     def generate(self,number,interval,resource):
11         for i in range(number):
12             c = Customer(name = "Customer%02d"%(i,))
13             activate(c,c.visit(timeInBank=12.0,
14                                res=resource,P=0))
15             t = expovariate(1.0/interval)
16             yield hold,self,t
17
18 class Customer(Process):
19     """ Customer arrives, is served and  leaves """
20
21     def visit(self,timeInBank=0,res=None,P=0):
22         arrive = now()       # arrival time
23         Nwaiting = len(res.waitQ)
24         print "%8.3f %s: Queue is %d on arrival"%(now(),self.name,Nwaiting)
25
26         yield request,self,res,P
27         wait = now()-arrive  # waiting time
28         print "%8.3f %s: Waited %6.3f"%(now(),self.name,wait)
29         yield hold,self,timeInBank
30         yield release,self,res
31
32         print "%8.3f %s: Completed"%(now(),self.name)
33
34 ## Experiment data -------------------------
35
36 maxTime = 400.0  # minutes
37 k = Resource(name="Counter",unitName="Karen",
38              qType=PriorityQ, preemptable=True)
39
40 ## Model/Experiment ------------------------------
41 seed(98989)
42 initialize()
43 s = Source('Source')
44 activate(s,s.generate(number=5, interval=10.0,
45                       resource=k),at=0.0)
46 guido = Customer(name="Guido     ")
47 activate(guido,guido.visit(timeInBank=12.0,res=k,
```

```
48                              P=100),at=23.0)
49 simulate(until=maxTime)
```

Though `Guido` arrives at the same time, `23.000`, he no longer has to wait and immediately goes into service, displacing the incumbent, `Customer01`. That customer had already completed `23.000-12.000 = 11.000` minutes of his service. When `Guido` finishes at `35.000`, `Customer01` resumes service and takes `36.000-35.000 = 1.000` minutes to finish. His total service time is the same as before (`12.000` minutes).

```
 0.000 Customer00: Queue is 0 on arrival
 0.000 Customer00: Waited  0.000
 2.166 Customer01: Queue is 0 on arrival
 5.121 Customer02: Queue is 1 on arrival
12.000 Customer00: Completed
12.000 Customer01: Waited  9.834
13.347 Customer03: Queue is 1 on arrival
23.000 Guido     : Queue is 2 on arrival
23.000 Guido     : Waited  0.000
35.000 Guido     : Completed
36.000 Customer01: Completed
36.000 Customer02: Waited 30.879
38.156 Customer04: Queue is 1 on arrival
48.000 Customer02: Completed
48.000 Customer03: Waited 34.653
60.000 Customer03: Completed
60.000 Customer04: Waited 21.844
72.000 Customer04: Completed
```

3 Balking and Reneging Customers

Balking occurs when a customer refuses to join a queue if it is too long. Reneging (or, better, abandonment) occurs if an impatient customer gives up while still waiting and before being served.

3.1 Balking Customers

Another term for a system with balking customers is one where "blocked customers" are "cleared", termed by engineers a BCC system. This is very convenient analytically in queueing theory and formulae developed using this assumption are used extensively for planning communication systems. The easiest case is when no queueing is allowed.

As an example let us investigate a BCC system with a single server but the waiting space is limited. We will estimate the rate of balking when the maximum number in the queue is set to 1. On arrival into the system the customer must first check to see if there is room. We will need the number of customers in the system or waiting. We could keep a count, incrementing when a customer joins the queue or, since we have a Resource, use the length of the Resource's `waitQ`. Choosing the latter we test (on line 25). If there is not enough room, we balk, incrementing a Class variable `Customer.numBalking` at line 34 to get the total number balking during the run.

```
 1 """ bank24. BCC system with several counters """
 2 from SimPy.Simulation import *
 3 from random import expovariate, seed
 4
 5 ## Model components ------------------------
 6
 7 class Source(Process):
 8     """ Source generates customers randomly """
 9
10     def generate(self,number,meanTBA,resource):
11         for i in range(number):
12             c = Customer(name = "Customer%02d"%(i,))
13             activate(c,c.visit(b=resource))
14             t = expovariate(1.0/meanTBA)
15             yield hold,self,t
16
17 class Customer(Process):
18     """ Customer arrives, is served and leaves """
19
20     numBalking = 0
21
22     def visit(self,b):
23         arrive = now()
24         print "%8.4f %s: Here I am "%(now(),self.name)
25         if len(b.waitQ) < maxInQueue:     # the test
26             yield request,self,b
27             wait = now()-arrive
28             print "%8.4f %s: Wait %6.3f"%(now(),self.name,wait)
29             tib = expovariate(1.0/timeInBank)
30             yield hold,self,tib
31             yield release,self,b
32             print "%8.4f %s: Finished  "%(now(),self.name)
33         else:
34             Customer.numBalking += 1
```

```
35             print "%8.4f %s: BALKING   "%(now(),self.name)
36
37 ## Experiment data -----------------------------
38
39 timeInBank = 12.0 # mean, minutes
40 ARRint = 10.0     # mean interarrival time, minutes
41 numServers = 1    # servers
42 maxInSystem = 2   # customers
43 maxInQueue = maxInSystem - numServers
44
45 maxNumber = 8
46 maxTime = 4000.0 # minutes
47 theseed = 12345
48
49 ## Model/Experiment ------------------------------
50
51 seed(theseed)
52 k = Resource(capacity=numServers,
53              name="Counter",unitName="Clerk")
54
55 initialize()
56 s = Source('Source')
57 activate(s, s.generate(number=maxNumber,meanTBA=ARRint,
58                        resource=k),at=0.0)
59 simulate(until=maxTime)
60
61 ## Results --------------------------------------
62
63 nb = float(Customer.numBalking)
64 print "balking rate is %8.4f per minute"%(nb/now())
```

The resulting output for a run of this program showing balking occurring is given below:

```
  0.0000 Customer00: Here I am
  0.0000 Customer00: Wait  0.000
  8.7558 Customer01: Here I am
 10.6770 Customer02: Here I am
 10.6770 Customer02: BALKING
 22.7622 Customer03: Here I am
 22.7622 Customer03: BALKING
 32.7477 Customer04: Here I am
 32.7477 Customer04: BALKING
 49.1642 Customer05: Here I am
 49.1642 Customer05: BALKING
 54.8556 Customer06: Here I am
 54.8556 Customer06: BALKING
 55.0607 Customer00: Finished
 55.0607 Customer01: Wait 46.305
 73.0765 Customer07: Here I am
 80.0846 Customer01: Finished
 80.0846 Customer07: Wait  7.008
 86.9980 Customer07: Finished
balking rate is   0.0575 per minute
```

When `Customer02` arrives, numbers 00 is already in service and 01 is waiting. There is no room so 02 balks. By the vagaries of exponential random numbers, 00 takes a very long time to serve (55.0607 minutes) so the first one to find room is number 07 at 73.0765.

3.2 Reneging (or abandoning) Customers

Often in practice an impatient customer will leave the queue before being served. SimPy can model this *reneging* behaviour using a *compound yield statement*. In such a statement there are two yield clauses. An example is:

```
yield (request,self,counter),(hold,self,maxWaitTime)
```

The first tuple of this statement is the usual `yield request`, asking for a unit of `counter` Resource. The process will either get the unit immediately or be queued by the Resource. The second tuple is a reneging clause which has the same syntax as a `yield hold`. The requesting process will renege if the wait exceeds `maxWaitTime`.

There is a complication, though. The requesting PEM must discover what actually happened. Did the process get the resource or did it renege? This involves a *mandatory* test of `self.acquired(`*resource*`)`. In our example, this test is in line 26.

```
 1 """ bank21: One counter with impatient customers """
 2 from SimPy.Simulation import *
 3 from random import expovariate, seed
 4
 5 ## Model components ------------------------
 6
 7 class Source(Process):
 8     """ Source generates customers randomly """
 9
10     def generate(self,number,interval):
11         for i in range(number):
12             c = Customer(name = "Customer%02d"%(i,))
13             activate(c,c.visit(timeInBank=15.0))
14             t = expovariate(1.0/interval)
15             yield hold,self,t
16
17 class Customer(Process):
18     """ Customer arrives, is served and  leaves """
19
20     def visit(self,timeInBank=0):
21         arrive = now()       # arrival time
22         print "%8.3f %s: Here I am     "%(now(),self.name)
23
24         yield (request,self,counter),(hold,self,maxWaitTime)
25         wait = now()-arrive  # waiting time
26         if self.acquired(counter):
27             print "%8.3f %s: Waited %6.3f"%(now(),self.name,wait)
28             yield hold,self,timeInBank
29             yield release,self,counter
30             print "%8.3f %s: Completed"%(now(),self.name)
31         else:
32             print "%8.3f %s: Waited %6.3f. I am off"%(now(),self.name,wait)
```

```
33
34 ## Experiment data ------------------------
35
36 maxTime = 400.0  # minutes
37 maxWaitTime = 12.0 # minutes. maximum time to wait
38
39 ## Model  ----------------------------------
40
41 def model():
42     global counter
43     seed(98989)
44     counter = Resource(name="Karen")
45     initialize()
46     source = Source('Source')
47     activate(source,
48              source.generate(number=5, interval=10.0),at=0.0)
49     simulate(until=maxTime)
50
51 ## Experiment  ----------------------------------
52
53 model()
```

```
  0.000 Customer00: Here I am
  0.000 Customer00: Waited  0.000
  2.166 Customer01: Here I am
  5.121 Customer02: Here I am
 13.347 Customer03: Here I am
 14.166 Customer01: Waited 12.000. I am off
 15.000 Customer00: Completed
 15.000 Customer02: Waited  9.879
 25.347 Customer03: Waited 12.000. I am off
 30.000 Customer02: Completed
 38.156 Customer04: Here I am
 38.156 Customer04: Waited  0.000
 53.156 Customer04: Completed
```

`Customer01` arrives after 00 but has only 12 minutes patience. After that time in the queue (at time 14.166) he abandons the queue to leave 02 to take his place. 03 also abandons. 04 finds an empty system and takes the server without having to wait.

4 Processes

In some simulations it is valuable for one SimPy Process to interrupt another. This can only be done when the *victim* is "active"; that is when it has an event scheduled for it. It must be executing a `yield hold` statement.

A process waiting for a resource (after a `yield request` statement) is passive and cannot be interrupted by another. Instead the `yield waituntil` and `yield waitevent` facilities have been introduced to allow processes to wait for conditions set by other processes.

4.1 Interrupting a Process.

`Klaus` goes into the bank to talk to the manager. For clarity we ignore the counters and other customers. During his conversation his cellphone rings. When he finishes the call he continues the conversation.

In this example, `call` is an object of the `Call` Process class whose only purpose is to make the cellphone ring after a delay, `timeOfCall`, an argument to its `ring` PEM (line 26).

`klaus`, a `Customer`, is interrupted by the call (line 29). He is in the middle of a `yield hold` (line 12). When he exits from that command it is as if he went into a trance when talking to the bank manager. He suddenly wakes up and must check (line 13) to see whether has finished his conversation (if there was no call) or has been interrupted.

If `self.interrupted()` is `False` he was not interrupted and leaves the bank (line 21) normally. If it is `True`, he was interrupted by the call, remembers how much conversation he has left (line 14), resets the interrupt (line 15) and then deals with the call. When he finishes (line 19) he can resume the conversation, with, now we assume, a thoroughly irritated bank manager (line 20).

```
 1 """ bank22: An interruption by a phone call """
 2 from SimPy.Simulation  import *
 3
 4 ## Model components ------------------------
 5
 6
 7 class Customer(Process):
 8     """ Customer arrives, looks around and leaves """
 9
10     def visit(self,timeInBank,onphone):
11         print "%7.4f %s: Here I am"%(now(),self.name)
12         yield hold,self,timeInBank
13         if self.interrupted():
14             timeleft = self.interruptLeft
15             self.interruptReset()
16             print "%7.4f %s: Excuse me"%(now(),self.name)
17             print "%7.4f %s: Hello! I'll call back"%(now(),self.name)
18             yield hold,self,onphone
19             print "%7.4f %s: Sorry, where were we?"%(now(),self.name)
20             yield hold,self,timeleft
21         print "%7.4f %s: I must leave"%(now(),self.name)
22
23 class Call(Process):
24     """ Cellphone call arrives and interrupts """
25
26     def ring(self,klaus,timeOfCall):
27         yield hold,self,timeOfCall
28         print "%7.4f Ringgg!"%(now())
29         self.interrupt(klaus)
```

```
30
31 ## Experiment data ------------------------
32
33 timeInBank = 20.0
34 timeOfCall = 9.0
35 onphone = 3.0
36 maxTime = 100.0
37
38
39 ## Model/Experiment  ---------------------------------
40
41 initialize()
42 klaus = Customer(name="Klaus")
43 activate(klaus,klaus.visit(timeInBank,onphone))
44 call = Call(klaus)
45 activate(call, call.ring(klaus,timeOfCall))
46 simulate(until=maxTime)


 0.0000 Klaus: Here I am
 9.0000 Ringgg!
 9.0000 Klaus: Excuse me
 9.0000 Klaus: Hello! I'll call back
12.0000 Klaus: Sorry, where were we?
23.0000 Klaus: I must leave
```

As this has no random numbers the results are reasonably clear: the interrupting call occurs at 9.0. It takes `klaus` 3 minutes to listen to the message and he resumes the conversation with the bank manager at 12.0. His total time of conversation is 9.0 + 11.0 = 20.0 minutes as it would have been if the interrupt had not occurred.

4.2 `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 7) and a function to test if it is open is defined at line 8.

The `Doorman` class is defined starting at line 11 and the single `doorman` is created and activated at at lines 65 and 66. The doorman waits for an average 10 minutes (label 16) and then opens the door.

The `Customer` class is defined at 29 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 40), 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 expovariate,seed
 4
 5 ## Model components ------------------------
 6
 7 door = 'Shut'
 8 def dooropen():
```

```
 9     return door=='Open'
10
11 class Doorman(Process):
12     """ Doorman opens the door"""
13     def openthedoor(self):
14         """ He will open the door when he arrives"""
15         global door
16         yield hold,self,expovariate(1.0/10.0)
17         door = 'Open'
18         print "%7.4f Doorman: Ladies and "\
19               "Gentlemen! You may all enter."%(now(),)
20
21 class Source(Process):
22     """ Source generates customers randomly"""
23     def generate(self,number,rate):
24         for i in range(number):
25             c = Customer(name = "Customer%02d"%(i,))
26             activate(c,c.visit(timeInBank=12.0))
27             yield hold,self,expovariate(rate)
28
29 class Customer(Process):
30     """ Customer arrives, is served and leaves """
31     def visit(self,timeInBank=10):
32         arrive = now()
33
34         if dooropen():
35             msg = ' and the door is open.'
36         else:
37             msg = ' but the door is shut.'
38         print "%7.4f %s: Here I am%s"%(now(),self.name,msg)
39
40         yield waituntil,self,dooropen
41
42         print "%7.4f %s: I can  go in!"%(now(),self.name)
43         wait = now()-arrive
44         print "%7.4f %s: Waited %6.3f"%(now(),self.name,wait)
45
46         yield request,self,counter
47         tib = expovariate(1.0/timeInBank)
48         yield hold,self,tib
49         yield release,self,counter
50
51         print "%7.4f %s: Finished    "%(now(),self.name)
52
53 ## Experiment data -------------------------
54
55 maxTime = 2000.0   # minutes
56 counter = Resource(1,name="Clerk")
57
58 ## Model  ----------------------------------
59
60 def model(SEED=393939):
61     seed(SEED)
```

```
62
63      initialize()
64      door = 'Shut'
65      doorman=Doorman()
66      activate(doorman,doorman.openthedoor())
67      source = Source()
68      activate(source,
69               source.generate(number=5,rate=0.1),at=0.0)
70      simulate(until=400.0)
71
72 ## Experiment  ---------------------------------
73
74 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
```

4.3 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 requires a `SimEvent` to be defined (line 7). The `Doorman` class is defined at line 8 and the `doorman` is created and activated at at labels 61 and 62. The doorman waits for a fixed time (label 13) and then tells the customers that the door is open. This is achieved on line 14 by signalling the `dooropen` event.

The `Customer` class is defined at 25 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 35). 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 ## Model components ------------------------
 6
 7 dooropen=SimEvent("Door Open")
 8 class Doorman(Process):
 9     """ Doorman opens the door"""
10     def openthedoor(self):
11         """ He will opens the door at fixed intervals"""
12         for i in range(5):
13             yield hold,self, 30.0
14             dooropen.signal()
15             print "%7.4f You may enter"%(now(),)
16
17 class Source(Process):
18     """ Source generates customers randomly"""
19     def generate(self,number,rate):
20         for i in range(number):
21             c = Customer(name = "Customer%02d"%(i,))
22             activate(c,c.visit(timeInBank=12.0))
23             yield hold,self,expovariate(rate)
24
25 class Customer(Process):
26     """ Customer arrives, is served and leaves """
27     def visit(self,timeInBank=10):
28         arrive = now()
29
30         if dooropen.occurred:
31             msg = '.'
32         else:
33             msg = ' but the door is shut.'
34         print "%7.4f %s: Here I am%s"%(now(),self.name,msg)
35         yield waitevent,self,dooropen
36
37         print "%7.4f %s: The door is open!"%(now(),self.name)
38
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 ## Experiment data -------------------------
50
```

```
51 maxTime = 400.0  # minutes
52
53 counter = Resource(1,name="Clerk")
54
55 ## Model  ---------------------------------
56
57 def model(SEED=393939):
58     seed(SEED)
59
60     initialize()
61     doorman = Doorman()
62     activate(doorman,doorman.openthedoor())
63     source = Source()
64     activate(source,
65              source.generate(number=5,rate=0.1),at=0.0)
66     simulate(until=maxTime)
67
68
69 ## Experiment  ---------------------------------
70
71 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
```

5 Monitors

Monitors (and Tallyies) are used to track and record values in a simulation. They store a list of [time,value] pairs, one pair being added whenever the `observe` method is called. A particularly useful characteristic is that they continue to exist after the simulation has been completed. Thus further analysis of the results can be carried out.

Monitors have a set of simple statistical methods such as `mean` and `var` to calculate the average and variance 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 useful 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.

5.1 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 29, using the arrival time saved in line 19. We create the Monitor, `Mon`, at line 37 and the times are `observed` at line 30.

The histogram is constructed from the Monitor, after the simulation has finished, at line 55. The SimPy.SimPlot package allows simple plotting of results from simulations. Here we use the SimPlot `plotHistogram` method. The plotting routines appear in lines 57-61. The `plotHistogram` call is in line 58.

```
 1 """bank17: Plotting a Histogram of Monitor results"""
 2 from SimPy.Simulation  import *
 3 from SimPy.SimPlot import *
 4 from random import  expovariate,seed
 5
 6 ## Model components ------------------------
 7
 8 class Source(Process):
 9     """ Source generates customers randomly"""
10     def generate(self,number,rate):
11         for i in range(number):
12             c = Customer(name="Customer%02d"%(i,))
13             activate(c,c.visit(timeInBank=12.0))
14             yield hold,self,expovariate(rate)
15
16 class Customer(Process):
17     """ Customer arrives, is served and leaves """
18     def visit(self,timeInBank):
19         arrive = now()
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          t = now()-arrive
30          Mon.observe(t)
31
32
33 ## Experiment data -------------------------
34
35 maxTime = 400.0   # minutes
36 counter = Resource(1,name="Clerk")
37 Mon = Monitor('Time in the Bank')
38 N = 0
39
40 ## Model  ---------------------------------
41
42 def model(SEED=393939):
43     seed(SEED)
44
45     initialize()
46     source = Source()
47     activate(source,
48              source.generate(number=20,rate=0.1),at=0.0)
49     simulate(until=maxTime)
50
51
52 ## Experiment  ---------------------------------
53
54 model()
55 Histo = Mon.histogram(low=0.0,high=200.0,nbins=20)
56
57 plt = SimPlot()
58 plt.plotHistogram(Histo,xlab='Time (min)',
59                   title="Time in the Bank",
60                   color="red",width=2)
61 plt.mainloop()
```

5.2 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 32 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 45). These can then be printed at the end of the program (line 49).

```
 1 """bank15: Monitoring a Resource"""
 2 from SimPy.Simulation import *
 3 from random import expovariate,seed
 4
 5 ## Model components ------------------------
 6
 7 class Source(Process):
 8     """ Source generates customers randomly"""
 9     def generate(self,number,rate):
10         for i in range(number):
11             c = Customer(name = "Customer%02d"%(i,))
12             activate(c,c.visit(timeInBank=12.0))
13             yield hold,self,expovariate(rate)
14
15 class Customer(Process):
16     """ Customer arrives, is served and leaves """
17     def visit(self,timeInBank):
18         arrive = now()
19         print "%8.4f %s: Arrived     "%(now(),self.name)
20
21         yield request,self,counter
22         print "%8.4f %s: Got counter "%(now(),self.name)
23         tib = expovariate(1.0/timeInBank)
24         yield hold,self,tib
25         yield release,self,counter
26
27         print "%8.4f %s: Finished    "%(now(),self.name)
28
29 ## Experiment data -------------------------
30
31 maxTime = 400.0    # minutes
32 counter = Resource(1,name="Clerk",monitored=True)
33
34 ## Model  ---------------------------------
35
36 def model(SEED=393939):
37     seed(SEED)
38
39     initialize()
40     source = Source()
41     activate(source,
42              source.generate(number=5,rate=0.1),at=0.0)
43     simulate(until=maxTime)
44
45     return (counter.waitMon.timeAverage(),counter.actMon.timeAverage())
46
47 ## Experiment  ---------------------------------
```

```
48
49 print 'Average waiting = %6.4f\nAverage active  = %6.4f\n'%model()
```

5.3 Plotting from Resource Monitors

Like all Monitors, `waitMon` and `actMon` 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 occur in the `model` routine from lines 37 to 44. The simulation now generates and processes 20 customers (line 43). `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 50 to 53. Line 51-52 construct a step plot and graphs the number in the waiting queue as a function of time. `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 expovariate,seed
 5
 6 ## Model components ------------------------
 7
 8 class Source(Process):
 9     """ Source generates customers randomly"""
10     def generate(self,number,rate):
11         for i in range(number):
12             c = Customer(name = "Customer%02d"%(i,))
13             activate(c,c.visit(timeInBank=12.0))
14             yield hold,self,expovariate(rate)
15
16 class Customer(Process):
17     """ Customer arrives, is served and leaves """
18     def visit(self,timeInBank):
19         arrive = now()
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 ## Experiment data -------------------------
31
32 maxTime = 400.0   # minutes
33 counter = Resource(1,name="Clerk",monitored=True)
34
```

```
35 ## Model ----------------------------------
36
37 def model(SEED=393939):
38     seed(SEED)
39
40     initialize()
41     source = Source()
42     activate(source,
43              source.generate(number=20,rate=0.1),at=0.0)
44     simulate(until=maxTime)
45
46 ## Experiment ----------------------------------
47
48 model()
49
50 plt = SimPlot()
51 plt.plotStep(counter.waitMon,
52         color="red",width=2)
53 plt.mainloop()
```

6 Acknowledgements

I thank Klaus Muller, Bob Helmbold, Mukhlis Matti and the other developers and users of SimPy for improving this document by sending their comments. I would be grateful for any further corrections or suggestions. Please send them to: *vignaux* at *users.sourceforge.net*.

7 References

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

Version: $Revision: 1.16 $