SIMPY

¿QUÉ ES SIMPY?

(Simulation in Python) Es un modulo que permite fácilmente que se creen modelos simulados de eventos discretos Su lenguaje de programación se basa en Python y fue sacado a la venta por GNU GPL. El módulo provee un modelador con componentes de una simulación modelo en la cual se incluyen procesos tanto para componentes activos como clientes, mensajes, y vehículos y recursos como para componentes pasivos que forman la capacidad limitada como puntos de congestión, cajas, y túneles. Además provee variables en el monitor que permiten recolectar estadísticas.

¿PARA QUÉ SE UTILIZA?

Áreas de aplicación:

- Simulación/modelación de epidemias
- Simulación del tráfico
- Vigilancia aérea de aviones
- Ingeniería Industrial
- Estudios del rendimiento del hardware de una computadora
- Funcionamiento del modelado
- Estudios en el volumen de trabajo
- Optimización de procesos industriales
- Enseñanza de la metodología de la simulación.
- Sincronización en tiempo real
- Simulación de manejo telescopios en observatorios
- Recolección de información (*SimulationRT*)

Principales funciones:

- activate () > para activar un proceso, marcar una proceso como ejecutable cuando es creada
- simulate > empieza la simulación
- reactivate () > reactivata un proceso suspendido
- cancel () > hace un proceso pasivo
- interrupt () > interrumpe un proceso suspendido
- yield hold > indicar el paso de cierto tiempo en un proceso. Es un operador cuyo primer operando es una función que es llamada, en este caso un código para una función que interpreta la operación hold en la librería simpy.

- yield passivate > hace pasivo el proceso, hasta ser activado por otro
- yield request > para pedir acceso a un recurso
- yield release > acaba con el recurso, indica que el proceso ha terminado de usar el recurso, habilitando el siguiente

Curso de control de las funciones (funcionamiento):

- Cuando main() llama a simulate(), main() se bloquea. Después empieza la simulación y main() no vuelve a correr de nuevo hasta que la simulación termina. (Cuando main() renauda, despliega los resultados de la simulación.)
- Siempre que un proceso ejecute yield, ese proceso se pondrá en pausa. Las funciones internas de Simpy correrán, y un proceso se reiniciará (posiblemente el mismso proceso).
- Cuando un proceso se reinicia finalmente, la ejecución se renauda justo después de que algún yield statement se ejecutó por último en este proceso.

Es importante reconocer que activate(), reactivate() y cancel no resultan en una pausa a la llamada de la función. Una pausa ocurre sólo cuando un yield es invocado.

Ejemplo del uso de la función yield

```
"""
One reader, one writer accessing a shared file.
"""
from SimPy.Simulation import *
import random

class Reader(Process):
    def readproc(self):
        while True:
            tstart=now()
            yield request,self,sharedFile
            print "%s Reading from file: %s, having waited %s"%(now(),self.name,now()-tstart)
            yield hold,self,random.uniform(1.0,5.0) #exclusive read access
            print "%s Reading complete: %s"%(now(),self.name)
            yield release,self,sharedFile

class Writer(Process):
    def writeproc(self):
        while True:
            tstart=now()
            yield request,self,sharedFile
            print "%s Writing to file, having waited %s"%(now(),now()-tstart)
            yield hold,self,random.uniform(0.5,7.0) #exclusive write access
            print "%s Writing complete"%now()
            yield release,self,sharedFile

sharedFile=Resource(capacity=1,name="File")

initialize()
reader=Reader("Reader")
activate(reader,reader.readproc())
writer=Writer("Writer")
activate(writer,writer.writeproc())
simulate(until=20)
```

OUTPUT:

```
0 Reading from file: Reader, having waited 0
4.23032952122 Reading complete: Reader
4.23032952122 Writing to file, having waited 4.23032952122
5.06564281243 Writing complete
5.06564281243 Reading from file: Reader, having waited 0.835313291211
8.4775272844 Reading complete: Reader
8.4775272844 Writing to file, having waited 3.41188447197
10.5635026183 Writing complete
10.5635026183 Reading from file: Reader, having waited 2.08597533391
14.371658521 Reading complete: Reader
14.371658521 Writing to file, having waited 3.80815590267
14.9996580982 Writing complete
14.9996580982 Reading from file: Reader, having waited 0.627999577182
18.4997624652 Reading complete: Reader
18.4997624652 Writing to file, having waited 3.50010436707.
```

Este ejemplo muestra el plano animado (animated plot) del número del visitantes de los bancos que esperan y el tiempo promedio de ese número.
Link: http://simpy.sourceforge.net/_build/html/Recipes/Recipe002.html

```
#-------------------------------------------------------------
# Scenario:      Customers visit a bank with one counter.
#                Show the number of waiting customers and
#                the time average of that number over time.
# Model:         Make the counter a Resource. Use Matplotlib
#                to plot the data.
#-------------------------------------------------------------
import pylab as p
import random as ran

## Model components ------------------------------------------
from SimPy.SimulationRT import *

class Visitor(Process):
    def visit(self,res,serveRate):
        yield request,self,res
        yield hold,self,ran.expovariate(serveRate)
        yield release,self,res

class VisitorGenerator(Process):
    def generate(self,arrRate,res,serveRate):
        while True:
            v=Visitor()
            activate(v,v.visit(res=res,serveRate = serveRate))
            yield hold,self,ran.expovariate(arrRate)

class Plotter(Process):
    def plotQ(self,tStep,res):
        moni=Monitor()
        p.ion()
        while now()<tEnd:
            yield hold,self,tStep
            moni.observe(len(res.waitQ))
            tAverage = moni.timeAverage()
            p.plot(moni.tseries(),moni.yseries(),"ro",
                   [now()],[tAverage],"b.")
            p.ylabel("nr waiting (time average = blue)")
            p.xlabel("time (hours)")
            p.title("Queuing customers. Time now = %s hours"%now())
            p.draw()

## Model -----------------------------------------------------
def bankModel(tStep,endTime,arrRate,serveRate,speed):
    res = Resource()
    initialize()
    vg=VisitorGenerator()
    activate(vg,vg.generate(arrRate = arrRate,res = res,serveRate = serveRate))
    pl=Plotter()
    activate(pl,pl.plotQ(tStep = tStep,res = res))
    simulate(until = endTime,real_time = True,rel_speed = speed)

## Experiment data -------------------------------------------
arrRate = 10    # mean arrival rate of customers per hour
serveRate = 14  # mean service rate (customers per hour)
timeStep = 0.1  # plot update interval (simulation time units)
tEnd = 8        # run time (hours)
aniSpeed = 10   # ratio simulation time to real time
ran.seed(111777)

## Experiment and output -------------------------------------
bankModel(tStep = timeStep,endTime = tEnd,speed = aniSpeed,
          arrRate = arrRate,serveRate = serveRate)
```

```
""" bank01: The single non-random Customer """
from SimPy.Simulation import *

## Model components ------------------------------

class Customer(Process):
    """ Customer arrives, looks around and leaves """

    def visit(self,timeInBank):
        print now(),self.name," Here I am"
        yield hold,self,timeInBank
        print now(),self.name," I must leave"

## Experiment data -----------------------------

maxTime = 100.0     # minutes
timeInBank = 10.0   # mean, minutes

## Model/Experiment ----------------------------

initialize()
c = Customer(name="Klaus")
activate(c,c.visit(timeInBank),at=5.0)
simulate(until=maxTime)
```

Ejemplo de La Sincronización de un proceso entre consumidor y productor. En este programa nos permite observar que el consumidor no consuma más productos de los que se han producido. Si es necesario, el consumidor puede esperar si el objeto que desea consumir no se encuentra disponible.

```
import random

class Producer(Process):
    def produce(self):
        self.produced=0
        while True:
            yield hold,self,random.uniform(1.0,5.0)      #produce
            buffer.append(Item())                        #put item into buffer
            if cons.passive():                           #reactivate consumer if passive
                reactivate(cons)
            print "%s item produced"%now()
            self.produced+=1

class Consumer(Process):
    def consume(self):
        self.consumed=0
        while True:
            if not buffer:                               #if buffer empty, passivate
                yield passivate,self
            while buffer:                                #while items in buffer
                buffer.pop(0)                            #take item out of buffer
                self.consumed+=1                         #consume item
                print "%s item consumed"%now()
                yield hold,self,random.uniform(1.0,4.9)

class Item:
    pass

buffer=[]
produced=consumed=0
initialize()
prod=Producer("Producer")
activate(prod,prod.produce())
cons=Consumer("Consumer")
activate(cons,cons.consume())
simulate(until=20)
print "produced: %s, consumed: %s, in buffer: %s"\
```

OUTPUT:

```
1.18998162499 item produced
1.18998162499 item consumed
3.04978978053 item produced
4.67639404396 item consumed
7.91845737849 item produced
7.91845737849 item consumed
9.39241701538 item produced
10.7954940036 item produced
12.5224683604 item consumed
14.2282703847 item produced
14.2545615616 item consumed
16.3758414166 item consumed
17.4764134234 item produced
17.8687107056 item consumed
19.9345355896 item produced
produced: 8, consumed: 7, in buffer: 1
```

Este ejemplo muestra un proceso de arribar. Diversos tiempos de distribuciones para arribaciones son mostradas.

```
from SimPy.Simulation import *
import random

def arrivalsUni(low,high):
    """Generator for uniform distribution between 'low' and 'high'
    """
    while True:
        yield random.uniform(low,high)

def arrivalsNormal(mean,sigma):
    """Generator for normal distribution with mu='mean'
       and sigma='sigma'
    """
    while True:
        yield random.normalvariate(mean,sigma)

def arrivalsEmpirical(valuelist):
    """Generator for drawing from a list 'valuelist' of
       empirical/observed values
    """
    while True:
        yield random.choice(valuelist)

class Arrivals(Process):
    def generateArrivals(self):
        i=0
        while True:
            yield hold,self,arrivals.next()
            i+=1
            print "%s arrival nr. %s"%(now(),i)

for arrivals in [arrivalsUni(low=10,high=20),
                 arrivalsEmpirical(valuelist=[10,12,14,19,20,20]),
                 arrivalsNormal(mean=15,sigma=2.0)]:
    initialize()
    a=Arrivals()
    activate(a,a.generateArrivals())
    print "\nNext distribution"
    simulate(until=30)
```

OUTPUT:

```
Next distribution
17.8638547229 arrival nr. 1

Next distribution
20 arrival nr. 1

Next distribution
15.8121137485 arrival nr. 1
29.6657638097 arrival nr. 2
```

Este es un ejemplo de un programa sobre el estudio de la congestión de celulares. El objetivo es encontrar un número de períodos ocupados que se observa en una celda del celular donde el número de canales es limitado. Las llamadas entrantes cuando la celda se encuentre ocupada serán tomadas como perdidas.

```
#!/usr/bin/env python
""" Simulate the operation of a BCC cellphone system using SimPy

"""
from __future__ import generators   # not needed for Python 2.3+
from SimPy.Simulation import *
from random import Random,expovariate

class Generator(Process):
 """ generates a sequence of calls """
 def __init__(self, maxN, lam):
     global busyEndTime
     Process.__init__(self)
     self.name = "generator"
     self.maxN = maxN
     self.lam = lam
     self.iatime = 1.0/lam
     self.rv = Random(gSeed)
     busyEndTime = now() # simulation start time

 def execute(self):
     for i in range(self.maxN):
         j = Job(i)
         activate(j,j.execute())
         yield hold,self,self.rv.expovariate(lam)
     self.trace("WARNING generator finished -- ran out of jobs")

 def trace(self,message):
     if GTRACING: print "%7.4f \t%s"%(self.time(), message)


class Job(Process):
    """ instances of the Job class represent calls arriving at
    random at the chnnels.
    """
    def __init__(self,i):
        Process.__init__(self)
        self.i = i
        self.name = "Job"+`i`

    def execute(self):
        global busyStartTime,totalBusyVisits,totalBusyTime
```

```
        self.trace("arrived ")
        if Nfree == 0: self.trace("blocked and left")
        else:
             self.trace("got a channel")
             Nfree -=  1
             if Nfree == 0:
                 self.trace("start busy period======")
                 busyStartTime = now()
                 totalBusyVisits += 1
                 interIdleTime = now() - busyEndTime
             yield hold,self,Jrv.expovariate(mu)
             self.trace("finished")
             if Nfree == 0:
                 self.trace("end   busy period++++++")
                 busyEndTime = now()
                 busy = now() - busyStartTime
                 self.trace("         busy  = %9.4f"%(busy,))
                 totalBusyTime +=busy
             Nfree += 1

    def trace(self,message):
         if TRACING: print "%7.4f \t%s %s "%(now(), message , self.name)


class Statistician(Process):
     """ observes the system at intervals """
     def __init__(self,Nhours,interv,gap):
         Process.__init__(self)
         self.Nhours = Nhours
         self.interv = interv
         self.gap = gap
         self.name="Statistician"

     def execute(self):
         global busyStartTime,totalBusyTime,totalBusyVisits
         global hourBusyTime,hourBusyVisits
         for i in range(self.Nhours):
             yield hold,self,self.gap
             totalBusyTime = 0.0
             totalBusyVisits = 0
             if Nfree == 0: busyStartTime = now()
             yield hold,self,self.interv
```

```
            totalBusyTime = 0.0
            totalBusyVisits = 0
            if Nfree == 0: busyStartTime = now()
            yield hold,self,self.interv
            if Nfree == 0: totalBusyTime += now()-busyStartTime
            if STRACING:  print "%7.3f %5d"%(totalBusyTime,totalBusyVisits)
            m.tally(totalBusyTime)
            bn.tally(totalBusyVisits)
        print("Busy Time:   mean = %10.5f var= %10.5f"%(m.mean(),m.var()))
        print("Busy Number: mean = %10.5f var= %10.5f"%(bn.mean(),bn.var()))


    def trace(self,message):
        if STRACING: print "%7.4f \t%s"%(self.time(), message)

totalBusyVisits = 0
totalBusyTime   = 0.0
NChannels =  4  # number of channels in the cell
Nfree   = NChannels
maxN    = 1000
gSeed   = 11111111
JrvSeed = 3333333
lam = 1.0
mu = 0.6667
meanLifeTime = 1.0/mu
TRACING  = 0
GTRACING = 0
STRACING = 1
Nhours  =  10
interv = 60.0    # monitor
gap    = 15.0    # monitor
print "lambda    mu      s  Nhours interv  gap"
print "%7.4f %6.4f %4d %4d    %6.2f %6.2f"%(lam,mu,NChannels,Nhours,interv,gap)

m = Monitor()
bn=Monitor()
Jrv = Random(JrvSeed)
s = Statistician(Nhours,interv,gap)
initialize()
g = Generator(maxN, lam)
activate(g,g.execute())
activate(s,s.execute())
simulate(until=10000.0)
```

```
lambda    mu      s  Nhours interv  gap
 1.0000 0.6667    4   10     60.00  15.00
  3.274     8
  0.444     2
  3.649     8
  0.652     3
  2.277     6
  1.372     9
  0.411     4
  0.067     2
  2.910     8
  5.073     9
Busy Time:   mean =    2.01297 var=    2.55814
Busy Number: mean =    5.90000 var=    7.49000
```

INSTRUCCIONES DE USO

1. Se importa el archivo de la biblioteca de simulación de Python.
2. Se define, por lo menos, una clase de componentes activos de simulación (procesos).
3. Se formula el modelo, el cual debe cumplir las siguientes condiciones:
    a. Inicia la máquina de tiempo de la ejecución (controlador de eventos)
    b. Genera, una ó más instancias, de sus componentes activos.
    c. Activa los casos.
    d. Establece la recogida de datos.
4. Se inicia la ejecución de la simulación.
5. Se recogen los valores del experimento, para realizar la simulación.
6. Se ejecuta la prueba.
7. Se analizan los datos recogidos.
8. Se da los resultados de salida.

Como encontrar errores (Debugging)

En este link se muestra un claro ejemplo de un erro y como se puede componer.
http://simpy.sourceforge.net/tracing.htm