Simulación con Perl de un problema tipo job shop

Jesús Lozano Mosterín, David de la Fuente, Montse Suárez

Departamento de Contabilidad y Administración de empresas. Escuela Politécnica Superior de Ingenieros Industriales. Universidad de Oviedo. Campus de Viesques s/n. 33204 Gijón- Asturias. lozano@epsig.uniovi.es

Resumen

En la simulación del flujo de trabajo realizado por una serie de máquinas o procesos, resulta necesario establecer transiciones en el tiempo de un estado a otro, como por ejemplo el paso de un estado "en cola" a un estado de "procesamiento"; y esto no se puede hacer con una sola variable continua o semicontinua de tiempo. Por ello, es necesario hacer llamadas a una subrutina o a un conjunto de ellas (eventos) que maneje los cambios de estado y marque la secuencia de nuevos eventos, a considerar en el futuro (E. C. Russell, 1999). Mediante la simulación a través de eventos discretos podemos evaluar los cuellos de botella de un taller (job shop) según la duración de las colas, evaluar la introducción de nuevas máquinas, nuevas órdenes de trabajo, cambio en la secuencia de órdenes de trabajo, etc.

Palabras clave: simulación, eventos discretos, job shop, Perl.

1. Introducción

En el presente trabajo se proporcionarán las claves para realizar simulaciones con un lenguaje de programación generalista de alto nivel (Perl) de forma que el lector pueda llevar a cabo sus propias investigaciones (Perl documentation. http://perldoc.perl.org). Entre los lenguajes específicos de simulación de eventos discretos se encuentran SIMULA y SIMSCRIPT II, ambos con larga tradición en el mercado. El trabajo está dividido en 10 apartados: En el 1 se realiza una breve introducción al trabajo, en el 2 se describe la simulación de eventos discretos; en el 3 algunos refinamientos; en el 4 se exhibe un ejemplo del manual de programación de SIMSCRIPT II.5 (CACI Products Co); en el 5 el código propuesto Perl (Perl language. http://cpan.org); en el 6 el output de SIMSCRIPT; en el 7 se describe la existencia de una solución doble; en el 8 la solución propuesta y su *output;* en el 9 mejoras propuestas a la simulación, y, por último, en el apartado 10 la conclusión obtenida.

2. Modelo básico de eventos discretos en un taller

En primer lugar, debemos referirnos a la estructura en que entran las órdenes de trabajo al taller: éstas pueden ser básicamente para iniciarse en un período determinado o bien iniciarse lo antes posible a partir del comienzo de la simulación. En este segundo caso, y dando por hecho la posible concurrencia de varias órdenes de trabajo, se puede establecer una probabilidad de entrada y unos retardos medios generales entre una entrada y la siguiente. Para simular las entradas probabilísticas, según una lista de trabajos y sus probabilidades, basta con calcular las probabilidades acumuladas y que una lectura aleatoria uniforme en ese rango determine, al situarse en un determinado intervalo, qué orden se debe procesar.

A continuación hacemos la primera llamada a la subrutina de eventos, que crea los siguientes datos: hora de inicio de la tarea *i.* Debemos descontar los recursos de la tarea *i* de los recursos totales.

Si los recursos de la tarea *i* no están disponibles se debe computar el tiempo en cola correspondiente.

En caso contrario, ya se puede fijar la hora de parada de la tarea *i,* y computar las estadísticas de tiempo en la máquina correspondiente.

Regularmente, se produce una llamada a los eventos como revisión de estado. Esto es, sólo deben revisarse aquellos eventos activos, que no hayan aún finalizado. Se pueden producir varios casos: que la tarea siga en cola, que no se haya superado el tiempo de parada y se está en proceso o bien que se haya superado la hora de parada. En este último caso se debe discernir cuál es la siguiente tarea *i+1* de la orden de trabajo y si se debe estar en cola o se puede asignar el tiempo de inicio *i+1,* etc.

3. Refinamiento del modelo

Con respecto al tiempo entre llegada de las órdenes es posible que éstas respondan a una distribución aleatoria (por ejemplo a una distribución Erlang (mu, sigma) o de Poisson), cuyas características estadísticas se deben de estudiar.

Con respecto a las tareas de una orden de trabajo, pueden darse varios casos que complican enormemente la simulación. Si la tarea es de procesamiento en una máquina, puede que no coincida la capacidad de la máquina con la orden de trabajo, asimismo se debe contemplar la posibilidad de parada de la máquina o la bifurcación en caso de que la calidad de salida no sea la esperada y se necesite un reprocesamiento o bien traslado a otra secuencia de tareas.

En el caso de que la tarea no sea de procesamiento, sino sólo de reposo o enfriamiento, se debe establecer si estos eventos pueden realizarse off-line (fuera del horario de trabajo) pero dentro del tiempo de simulación, lo cual implica diferenciar en los eventos de revisión, de qué tipo de tarea se trata. Si el tiempo de trabajo es inferior diariamente al tiempo simulado se debe tomar el módulo 24 del tiempo simulado en horas, y sólo llamar a los eventos de lanzamiento de órdenes o de revisión en el intervalo de trabajo.

Por último, existen diferencias en los resultados en función de las reglas de *scheduling* de las tareas en cola, de las que puede haber múltiples opciones, así como diferentes reglas para resolver los empates. En el ejemplo que tomamos a continuación, se crean colas FIFO (primero en llegar, primero en ser procesado) y desempates (si hubiera) por orden de trabajo.

4. Ejemplo de Input

Tomado del manual de programación de SIMSCRIPT II.5 (Ver CACI Products Co.):

```
0 0 00
PRESS 0.5 SAW 1.0 SHAPER 0.5 WELDER 2.0 LATHE 1.0
0 0 00
SAW 1.0 LATHE 4.00 GRINDER 0.5 SHAPER 2.0
0 0 30
DRILL 1.0 SHAPER 1.00 LATHE 2.0 SHAPER 2.0 MILL 1.00
0 2 30
SAW 1.0 WELDER 1.00 DRILL 0.5 LATHE 1.5 MILL 2.00
```

Con una disponibilidad de 3 máquinas SAW, 2 LATHE y DRILL, y 1 el resto (Figura 1).

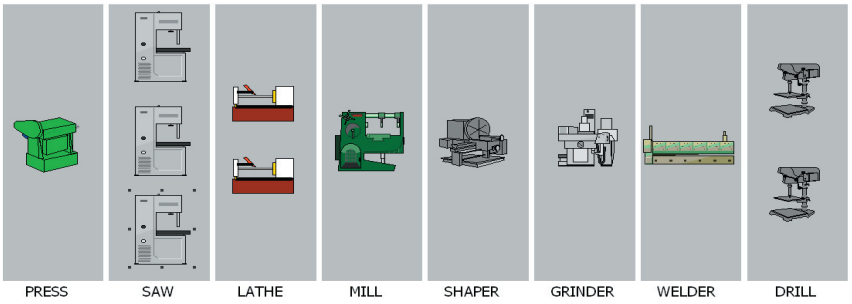

Figura 1. Disponibilidad de máquinas para el ejemplo desarrollado.

5. Código en Perl

En este apartado se puede ver el código Perl creado para el ejemplo, además del flujograma representado en la Figura 2.

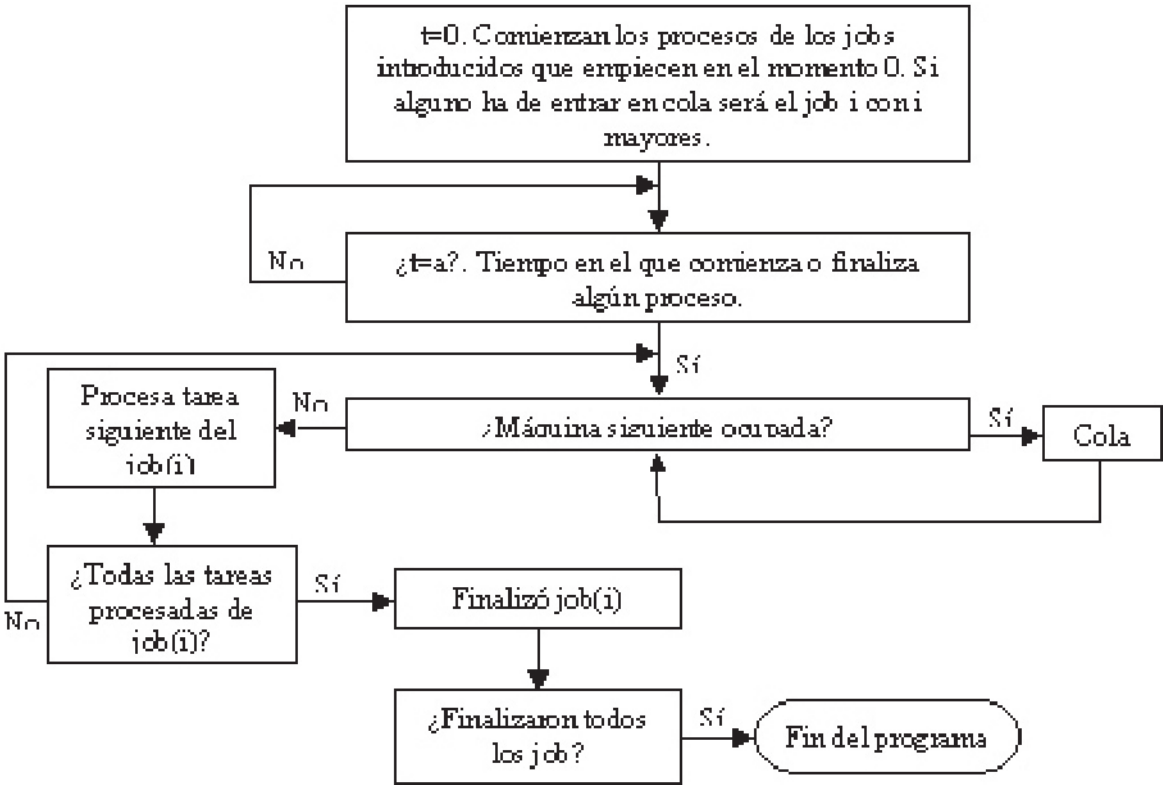

Figura 2. Flujograma del código Perl creado.

Código Perl

```
#!/usr/local/bin/perl -w

$DEBUG = 1;

if ($DEBUG){
        open LOG, ">"."js1.log" or die "Cannot open js1.log";
}

%name = (          # number of machines per type
    "PRESS" => 1,
```

```
    "SAW"  =>  3,
    "LATHE" => 2,
    "MILL"  => 1,
    "SHAPER" => 1,
    "GRINDER" => 1,
    "WELDER" => 1,
    "DRILL" => 2,
);
$N=8;
%busy = %name;
@an = ([0, 8, 0], [1, 0, 0]);
open IN , "<".$ARGV[0] or die "Cannot open $ARGV[0]:$!";
while (<IN>){
    chomp;
    $count++;
    ($d[$count], $h[$count], $m[$count])= split /\s+/, $_;
    $total[$count]=$d[$count]*24+$h[$count]+$m[$count]/60;
    chomp ($s = <IN>);
    @s = split /\s+/, $s;
    $c=0;
    while (@s){
      $c++;
      $seq[$count][$c]=shift @s;
      $seq[$count][++$c]=shift @s;
      $total[$count] += $seq[$count][$c];
    }
    print "Minimum time for completion job $count is $total[$count]\n";
}
close IN;
$jobs=0;
$time=0;
$ok=0;
$timeunit=1/(60*60);
for $day (0..30){
    for $hour (0..23){
        for $minu (0..59){
            for $seco (0..59){
            for (1..$count){
                if (defined $job[$_]) {
                    if ($job[$_]==0) {
                       event($_);
                    }
                }elsif ($day==$d[$_] && $hour==$h[$_] && $minu==$m[$_] && $seco==0){
                    event($_);
                }
    }
    if ($jobs==$count && $ok==0) { $totaltime=$time; $ok=1; }
    for (0..$#an){
                if ($day==$an[$_][0] && $hour==$an[$_][1] && $minu==$an[$_][2] && $seco==0){
                report($_);
     }
    }
            $time += $timeunit;
            }
        }
    }
}
if ($DEBUG){ close LOG; }

sub event{
    my $i = shift;
    if (!defined $startevent[$i]){ # start a job
      $j[$i] = 1;
      $job[$i]=0;
        $machine[$i]=$seq[$i][1];
        $startevent[$i]=$time;
        if ($DEBUG){
                print LOG "$time: Job $i started\n";
        }
    }
    if (!defined $stopevent[$i][$j[$i]]){  # waiting for a machine
        if ($busy{$machine[$i]}<=0){
            cola($i);
        }elsif ( @{ $machine[$i] } ){
```

```
          $z = shift @{$machine[$i]};
          procesa($z);
      }else{
          procesa($i);
    }
}
    if (defined $stopevent[$i][$j[$i]]){ # start a new subprocess
        if ($time >= $stopevent[$i][$j[$i]]){

          if ($DEBUG){
              print LOG "$time: Job $i stopped being processed in $machine[$i]\n";
          }

          $busy{$machine[$i]}++;
        if ($busy{$machine[$i]}>$name{$machine[$i]}){
           $busy{$machine[$i]}=$name{$machine[$i]};
           }
          $j[$i]+=2;
        $machine[$i]=$seq[$i][$j[$i]];
          if (!defined $machine[$i] && $job[$i]==0){
          $timejob[$i]=$time-$startevent[$i];
          $job[$i]=1;
          $jobs++;
              if ($DEBUG) {
                      print LOG "$time: Job $i finished\n";
              }
          return ;
         }
      }else{ # processing
      }
    }
}

sub cola{
    my $i = shift;
    $busy{$machine[$i]}=0;
    $queue{$machine[$i]}+=$timeunit;
    $totalqueue+=$timeunit;
    unless (grep $i, @{ $machine[$i] } ){
       push @{ $machine[$i] }, $i;

       $len{$machine[$i]}++;
        if ($DEBUG){
                print LOG "$time: Job $i enters in queue $machine[$i]\n";
                print LOG "       The queue is now: @{ $machine[$i] }\n";
        }
    }
}

sub procesa{
    my $i = shift;
    $busy{$machine[$i]}--;
    $stopevent[$i][$j[$i]]=$time+$seq[$i][$j[$i]+1];
    if ($DEBUG){
        if (!defined $queue{$machine[$i]}) { $queue{$machine[$i]}=0; }
        print LOG "$time: Job $i after $queue{$machine[$i]} queue, is being processed by
$machine[$i]\n";
    }
}

sub report{
    print "Elapsed time is $time\n";
    print "Number of jobs completed is $jobs\n";
    if (defined $totaltime){
        print "All jobs completed in $totaltime\n";
    }
    $mean=0;
    for (1..$count){
        if ($job[$_]==1){
            $mean += $timejob[$_];
        }
    }
    if ($jobs>0){
       $mean /= $jobs;
```

```
       print "Average time for completion is $mean\n";
    }
    print "QUEUE TIME STATUS:\n";
    for my $i (keys %name){
      if (!defined $queue{$i}) { $queue{$i}=0; }
        print "average queue $i ", $queue{$i}/$name{$i}, "\n";
    }
    print "Total queue time is ",$totalqueue,"\n";
    print "MEAN QUEUE LENGTH:\n";
    $cc=0;
    $overall=0;
    for my $i (keys %name){

        if (!defined $len{$i}){ $len{$i}=0; }
        $cc++;
        print "length queue $i ", $len{$i}/$N , "\n";
        $overall+= $len{$i}/$N;
    }
    print "Overall queue length ", $overall/$N, "\n";
    print "JOB STATUS:\n";
    for (1 .. $count){
      if (!defined $job[$_]){
          print "Job $_ not started\n";
      if (!defined $job[$_]){
      }elsif ($job[$_]==1){
          print "Job $_ completed in $timejob[$_]\n";
      }elsif ($job[$_]==0){
          print "Job $_ started, not completed\n";
      }else{
          print "Error in job $_\n";
}
    }
    print "\n";
}

__END__
```

6. Output de SIMSCRIPT II.5

Tomado de CACI Products Co., página 245:

E X A M P L E J O B S H O P S I M U L A T I O N

REPORTING PERIOD 0.0 HRS. TO 8.0 HRS.

JOBS COMPLETED DURING PERIOD : 2

AVERAGE COMPLETION TIME : 6.50 HRS.

AVERAGE NUMBER OF JOBS WAITING FOR EACH PRODUCTION CENTER :

MACHINE CENTER AVERAGE QUEUE

PRESS 0.0

SAW 0.0

LATHE 0.0

MILL 0.0

SHAPER .25

GRINDER 0.0

WELDER .19

DRILL 0.0

OVERALL AVERAGE QUEUE LENGTH : .05

7. El problema de la solución doble

Examinando el archivo LOG, se observa que la adición de unidades de tiempo de 1/(60*60) horas (al segundo) en cada iteración provoca un error de coma flotante (a pesar de usar un intérprete Perl compilado con *long doubles*) que hace que donde debería haber un empate a la hora de decidir cuál trabajo es procesado por SHAPER en el instante 1,5 h., éste se resuelva a favor del tercer trabajo y no del primero. Ello es provocado por el error de no hacer redondeos, de tal forma que ambos trabajos empaten y se deba decidir arbitrariamente, pero tal como habíamos mencionado previamente, que será procesado el primer trabajo frente al tercero.

Hemos comprobado cómo esta diferencia de decisión en el momento 1,5 provoca un cambio en la totalidad del resultado de colas subsecuentes. Así, por ejemplo, el programa ARENA 8.0 (Ver ARENA simulation software) proporciona una solución con más colas, pero que responde al orden de preferencia. Podemos observar las diferencias en la Tabla 1 realizada a mano:

Tabla 1. Diferencias entre los dos métodos utilizados

ARENA 8.0

Job 1	press	saw	saw	shaper	welder	welder	welder	welder	**queue**	**queue**	lathe	lathe							
Job 2	saw	saw	lathe	lathe	lathe	lathe	lathe	lathe	lathe	lathe	grinder	**queue**	**queue**	**queue**	shaper	shaper	shaper	shaper		
Job 3		drill	drill	**queue**	shaper	shaper	lathe	lathe	lathe	lathe	shaper	shaper	shaper	shaper	mill	mill				
Job 4						saw	saw	**queue**	welder	welder	drill	lathe	lathe	lathe	**queue**	**queue**	mill	mill	mill	mill
Time	0,5	1	1,5	2	2,5	3	3,5	4	4,5	5	5,5	6	6,5	7	7,5	8	8,5	9	9,5	10
Queues				shaper				welder	lathe	lathe		shaper	shaper	shaper	mill	mill				

SIMSCRIPT II.5

Job 1	press	saw	saw	**queue**	**queue**	shaper	welder	welder	welder	welder	lathe	lathe								
Job 2	saw	saw	lathe	lathe	lathe	lathe	lathe	lathe	lathe	lathe	grinder	**queue**	**queue**	shaper	shaper	shaper	shaper			
Job 3		drill	drill	shaper	shaper	lathe	lathe	lathe	lathe	shaper	shaper	shaper	shaper	mill	mill					
Job 4						saw	saw	**queue**	**queue**	**queue**	welder	welder	drill	lathe	lathe	lathe	mill	mill	mill	mill
Time	0,5	1	1,5	2	2,5	3	3,5	4	4,5	5	5,5	6	6,5	7	7,5	8	8,5	9	9,5	10
Queues				shaper	shaper			welder	welder	welder		shaper	shaper							

8. La solución

Hemos visto cómo el no efectuar un redondeo, produce desempates arbitrarios. En el caso del código Perl (Perl documentation) propuesto, así como en el SIMSCRIPT esto lleva a una solución válida, con menos colas, más optimista en este caso. Para reproducir el comportamiento del empate instantáneo, con preferencia según el orden de los trabajos deberíamos introducir en el bucle principal del código Perl el siguiente redondeo:

```
$timer+=$timeunit;
$time = sprintf ("%3.3f" ,$timer);
```

Lo cual produce los siguientes resultados (no se han redondeado las colas):

```
Minimum time for completion job 1 is 5
Minimum time for completion job 2 is 7.5
Minimum time for completion job 3 is 7.5
Minimum time for completion job 4 is 8.5
Elapsed time is 8.000
Number of jobs completed is 2
Average time for completion is 6.75

QUEUE TIME STATUS:
average queue GRINDER 0
average queue SAW 0
average queue SHAPER 1.99972222222222242
average queue MILL 0.999722222222222221
average queue LATHE 0.5
average queue PRESS 0
average queue DRILL 0
average queue WELDER 0.499722222222222222
Total queue time is 4.49916666666666735

MEAN QUEUE LENGTH:
length queue GRINDER 0
length queue SAW 0
length queue SHAPER 0.25
length queue MILL 0.125
length queue LATHE 0.125
length queue PRESS 0
length queue DRILL 0
length queue WELDER 0.125
Overall queue length 0.078125

JOB STATUS:
Job 1 completed in 6
Job 2 started, not completed
Job 3 completed in 7.5
Job 4 started, not completed
Elapsed time is 24.000
Number of jobs completed is 4
All jobs completed in 10.000
Average time for completion is 7.5

QUEUE TIME STATUS:
average queue GRINDER 0
average queue SAW 0
average queue SHAPER 1.99972222222222242
average queue MILL 0.999722222222222221
average queue LATHE 0.5
average queue PRESS 0
average queue DRILL 0
average queue WELDER 0.499722222222222222
Total queue time is 4.49916666666666735

MEAN QUEUE LENGTH:
length queue GRINDER 0
length queue SAW 0
length queue SHAPER 0.25
length queue MILL 0.125
length queue LATHE 0.125
length queue PRESS 0
length queue DRILL 0
length queue WELDER 0.125
Overall queue length 0.078125

JOB STATUS:
Job 1 completed in 6
Job 2 completed in 9
Job 3 completed in 7.5
Job 4 completed in 7.5
```

A continuación se adjunta el fichero LOG. Adviértase que al tomar la resolución en segundos, esto es, 1/(60*60) horas, queda afectada la precisión de los resultados. Se tomó esta unidad de tiempo como compromiso entre precisión y velocidad de ejecución.

```
0: Job 1 started
0: Job 1 after 0 queue, is being processed by PRESS
0: Job 2 started
0: Job 2 after 0 queue, is being processed by SAW
0.500: Job 1 stopped being processed in PRESS
0.500: Job 1 after 0 queue, is being processed by SAW
0.500: Job 3 started
0.500: Job 3 after 0 queue, is being processed by DRILL
1.000: Job 2 stopped being processed in SAW
1.000: Job 2 after 0 queue, is being processed by LATHE
1.500: Job 1 stopped being processed in SAW
1.500: Job 3 stopped being processed in DRILL
1.500: Job 1 after 0 queue, is being processed by SHAPER
1.500: Job 3 enters in queue SHAPER
       The queue is now: 3
2.000: Job 1 stopped being processed in SHAPER
2.000: Job 3 after 0.499722222222222222 queue, is being processed by SHAPER
2.000: Job 1 after 0 queue, is being processed by WELDER
2.500: Job 4 started
2.500: Job 4 after 0 queue, is being processed by SAW
3.000: Job 3 stopped being processed in SHAPER
3.000: Job 3 after 0 queue, is being processed by LATHE
3.500: Job 4 stopped being processed in SAW
3.500: Job 4 enters in queue WELDER
       The queue is now: 4
4.000: Job 1 stopped being processed in WELDER
4.000: Job 4 after 0.499722222222222222 queue, is being processed by WELDER
4.000: Job 1 enters in queue LATHE
       The queue is now: 1
5.000: Job 2 stopped being processed in LATHE
5.000: Job 3 stopped being processed in LATHE
5.000: Job 4 stopped being processed in WELDER
5.000: Job 1 after 0.999999999999999999 queue, is being processed by LATHE
5.000: Job 2 after 0 queue, is being processed by GRINDER
5.000: Job 3 after 0.499722222222222222 queue, is being processed by SHAPER
5.000: Job 4 after 0 queue, is being processed by DRILL
5.500: Job 2 stopped being processed in GRINDER
5.500: Job 4 stopped being processed in DRILL
5.500: Job 2 enters in queue SHAPER
       The queue is now: 2
5.500: Job 4 after 0.999999999999999999 queue, is being processed by LATHE
6.000: Job 1 stopped being processed in LATHE
6.000: Job 1 finished
7.000: Job 3 stopped being processed in SHAPER
7.000: Job 4 stopped being processed in LATHE
7.000: Job 2 after 1.999722222222222242 queue, is being processed by SHAPER
7.000: Job 3 after 0 queue, is being processed by MILL
7.000: Job 4 enters in queue MILL
       The queue is now: 4
8.000: Job 3 stopped being processed in MILL
8.000: Job 3 finished
8.000: Job 4 after 0.999722222222222221 queue, is being processed by MILL
9.000: Job 2 stopped being processed in SHAPER
9.000: Job 2 finished
10.000: Job 4 stopped being processed in MILL
10.000: Job 4 finished
```

9. Mejoras propuestas

Si los trabajos son de carácter repetitivo (desarrollado en E. C. Russell. (1999)), se hace necesario utilizar matrices de tres dimensiones para acumular los tiempos, colas, y demás estadísticas. La estructura podría describirse del siguiente modo:

QUEUE [tipo de trabajo i] [máquina j[i][c]] [contador j[i][c]];

(tarea tipo i, contador de ejecución de trabajo c)

10. Conclusión

Es posible desarrollar una simulación de eventos discretos en un lenguaje de alto nivel como

el Perl. Las pequeñas discrepancias obtenidas con un lenguaje específico de simulación de eventos discretos pueden estar debidas a la elección de la granularidad del tiempo, y, sobre todo, a la forma en que se realicen los redondeos para evitar los problemas inherentes al uso de aritmética de coma flotante.

Referencias

CACI Products Co. SIMSCRIPT II.5 Programming Language.

E. C. Russell. (1999). Building Simulation Models with Simscript II.5, CACI Products Co.

Disponibles on-line en: http://www.simprocess.com/products/simscript_documentation.cfm

ARENA simulation software. http://www.arenasimulation.com/

Perl documentation. http://perldoc.perl.org

Perl language. http://cpan.org, “Source code” o “Binary ports”