

Universidad de Costa Rica

Escuela de Ciencias de la Computación e Informática

Investigación de Operaciones

PROYECTO DE SIMULACIÓN

Profesora:

Ileana Alpizar

Hecho por:

Esteban Rodríguez Betancourt

B15512

12 de octubre del 2013

II Semestre

CONTENIDO

Manual de Usuario	3
Iniciar la aplicación	3
Uso de la Simulación	4
Descripción del sistema a simular	6
Diseño del Simulador	7
Eventos	8
Estadísticas	10
El simulador	11
Demostraciones sobre la lógica esperada del programa	12
Estadísticas y análisis del sistema	14
Rendimiento del sistema.....	15
Mensajes rechazados	16
Mensajes enviados	17
Todos los mensajes	18
Análisis del sistema	18
Problemas encontrados	19
Rendimiento	19
Ejecución del programa.....	19

INICIAR LA APLICACIÓN

La simulación fue implementada utilizando HTML y Javascript. Para abrir el sitio web es necesario utilizar un servidor web, de lo contrario abrir directamente el HTML con un navegador puede dar errores de presentación.

Si se tiene instalado Python 3 se puede crear fácilmente un servidor usando los siguientes comandos desde una terminal de Linux:

```
cd directorioProyecto
python3 -m http.server
```

o bien en una terminal de Windows (como PowerShell):

```
cd directorioProyecto
python.exe -m http.server
```

Luego se puede abrir un navegador con la dirección que indicó el script anterior, normalmente <http://localhost:8000/>.

Además el proyecto se encuentra disponible en las siguientes direcciones:

- <http://proyectoio2013.estebarb.tk>
- <http://pacific-lowlands-9599.herokuapp.com>

El programa descarga Angular.js, jQuery y Twitter Bootstrap, por lo que requiere conexión a internet. Si no se utiliza un servidor para proveer el HTML al navegador entonces se requiere utilizar Firefox, ya ni Chrome ni Internet Explorer permiten que los scripts accedan a archivos en file://.

El funcionamiento del programa (utilizando las URLs anteriores) ha sido comprobado los siguientes navegadores:

1. Google Chrome (Ubuntu 13.04 y Windows 8)
2. Mozilla Firefox (Ubuntu 13.04 y Windows 8)
3. Internet Explorer 10 (Windows 8)

El código fuente también está disponible en <https://github.com/estebarb/pio13>.

USO DE LA SIMULACIÓN

El programa permite configurar la cantidad de veces que se ejecutará la simulación, la cantidad de minutos a simular y la pausa (en milisegundos) entre cada paso de tiempo. También permite especificar si se desea que se actualice la GUI en tiempo real o bien si se actualiza hasta el final (esto a veces es útil por motivos de rendimiento).

Para iniciar la simulación hay dos opciones:

- **Iniciar Simulación:** En este modo la simulación será ejecutaba paso a paso, con una pausa entre cada paso de la simulación. El estado de la simulación y las estadísticas acumuladas serán actualizadas instantáneamente en la pantalla.
- **Ejecución Rápida:** En este modo el sistema no hará pausas entre cada paso de la simulación, por lo que es el modo ideal para correr simulaciones que en el otro modo tardarían demasiado. Sin embargo, las estadísticas y los datos de la simulación actual solo se actualizan al final de cada simulación.

Para borrar los datos acumulados o detener una simulación se puede presionar el botón **Reiniciar**.

Una vez iniciada la simulación se pueden ver los datos de la simulación actual:

- El reloj actual
- Nombre y descripción del evento que se está procesando
- Cantidad de mensajes enviados
- Cantidad de mensajes rechazados
- Estado de ocupación de cada servidor y cantidad de mensajes en cola

Estos datos son actualizados en de forma instantánea según se va dando la simulación (pero no en el modo de ejecución rápida).

Configuración

- Cantidad de repeticiones
- Minutos a simular
- Pausa entre eventos (milisegundos)

No mostrar los resultados en tiempo real

Porcentaje de confianza

Simulación Actual

- Reloj: 25.045180767510328
- Evento: C atendió un mensaje:C atendió un mensaje
- Mensajes enviados: 118
- Mensajes rechazados: 217

Servidor	Estado	Mensajes en cola
Servidor 1	Ocupado	1
Servidor 2.1	Ocupado	0
Servidor 2.2	Ocupado	0
Servidor 3	Ocupado	0

Simulación Actual

- Reloj: 61.23769948930322
- Evento: C recibe un mensaje:Se ha creado un mensaje nuevo para que sea recibido por C
- Mensajes enviados: 311
- Mensajes rechazados: 517

Servidor	Estado	Mensajes en cola
Servidor 1	Libre	0
Servidor 2.1	Ocupado	0
Servidor 2.2	Ocupado	0
Servidor 3	Ocupado	0

Simulación Actual

- Reloj: 95.8423365019832
- Evento: A atendió un mensaje:A atendió un mensaje
- Mensajes enviados: 482
- Mensajes rechazados: 810

Servidor	Estado	Mensajes en cola
Servidor 1	Ocupado	2
Servidor 2.1	Ocupado	1
Servidor 2.2	Ocupado	0
Servidor 3	Ocupado	4

Simulación Actual

- Reloj: 758.9083833693967
- Evento: B recibe un mensaje:Se ha creado un mensaje nuevo para que sea recibido por B
- Mensajes enviados: 3817
- Mensajes rechazados: 6534

Servidor	Estado	Mensajes en cola
Servidor 1	Ocupado	3
Servidor 2.1	Ocupado	1
Servidor 2.2	Ocupado	0
Servidor 3	Ocupado	1

Además, de forma simultánea a la simulación también se actualizan las estadísticas acumuladas del sistema. Estas estadísticas incluyen:

- Reloj del último evento simulado (en minutos)
- Cantidad de mensajes enviados, rechazados y procesados en total
- Porcentaje de ocupación en cada CPU
- Porcentaje de ocupación con mensajes que al final son rechazados
- Porcentaje de mensajes rechazados

Rendimiento del sistema

Simulación	Reloj (minutos)	Mensajes			% ocupación				% ocupación rechazados		% mensajes rechazados
		Enviados	Rechazados	Total	PC 1	PC 2.1	PC 2.2	PC 3	PC 1	PC 3	
1	9.999.9702	50682	85694	136376	71.9873%	81.6525%	73.1746%	70.9930%	9.5763%	63.0339%	62.8366
2	9.999.9347	50759	85708	136467	71.7270%	81.3173%	73.1020%	71.5351%	9.7915%	63.5554%	62.8049
3	9.999.9899	50904	85547	136451	71.3770%	80.9930%	72.9104%	71.3285%	9.6278%	63.3294%	62.6943
4	9.999.9890	50805	85590	136395	71.7771%	81.3700%	73.0604%	70.9849%	9.6724%	62.9605%	62.7516
5	9.999.9764	50659	85756	136415	71.5458%	81.5167%	73.3719%	71.7620%	9.4155%	63.7668%	62.8641
6	9.999.9775	50719	85731	136450	72.0452%	81.5290%	73.3386%	71.5265%	9.4292%	63.4789%	62.8296
7	9.999.9722	50591	85818	136409	71.8971%	81.3965%	73.3200%	71.3215%	9.6737%	63.5141%	62.9123
8	9.999.9979	50795	85631	136426	71.1482%	81.1511%	72.9665%	71.5718%	9.6326%	63.5748%	62.7674
9	9.999.9991	50781	85480	136261	71.1154%	81.2327%	73.0712%	71.1826%	9.5296%	63.1860%	62.7326
10	9.999.9822	50625	85796	136421	70.9925%	81.0320%	72.6251%	71.4349%	9.3073%	63.6041%	62.8906
Promedio total	9.999.9789	50.732.0000	85.675.1000	136.407.1000	71.5613%	81.3191%	73.0941%	71.3641%	9.5656%	63.4004%	62.8084
Intervalo de confianza:	[9.999.9657, 9.999.9922]	[50.663.9977, 50.800.0023]	[85.596.0893, 85.754.1107]	[136.365.5062, 136.448.6938]	[71.2863%, 71.8362%]	[81.1632%, 81.4749%]	[72.9313%, 73.2569%]	[71.1826%, 71.5456%]	[9.4612%, 9.6700%]	[63.2114%, 63.5894%]	[62.7575, 62.8593]

Además, se incluye las siguientes estadísticas, para los mensajes rechazados, aceptados y ambos:

- Promedio de devoluciones desde la computadora 1 hasta la 2 o la 3.
- Tiempo promedio por mensaje en el sistema, en colas o en transmisión
- El porcentaje de tiempo en procesamiento por mensaje (no en colas o en transmisión).

Mensajes rechazados						
Simulación	Media rebotes en		T. promedio por mensaje en			% tiempo cómputo
	PC1	Sistema	Colas	Trans.		
1	0.11133	13.7849s	6.1405s	2.5605s		63.8259%
2	0.11308	14.0073s	6.2718s	2.6009s		63.1016%
3	0.11300	13.8345s	6.1185s	2.5991s		63.5453%
4	0.11050	13.7379s	6.1047s	2.5416s		64.0068%
5	0.11001	13.8566s	6.2061s	2.5302s		63.3026%
6	0.11099	13.9195s	6.2642s	2.5527s		63.3319%
7	0.11089	13.8022s	6.1349s	2.5504s		63.6337%
8	0.11136	13.8273s	6.1365s	2.5613s		63.3904%
9	0.11210	13.8804s	6.1982s	2.5782s		63.4604%
10	0.10918	13.8685s	6.2585s	2.5111s		63.4153%
Promedio total	0.11124	13.8519s	6.1834s	2.5586s		63.5014%
Intervalo de confianza:	[0.11036, 0.11213]	[13.7982s, 13.9056s]	[6.1372s, 6.2296s]	[2.5383s, 2.5789s]		[63.3122%, 63.6906%]

Mensajes enviados						
Simulación	Media rebotes en PC1		T. promedio por mensaje en			% tiempo cómputo
	Sistema	Colas	Trans.			
1	0.22588	67.3839s	15.5309s	25.1952s		40.0497%
2	0.22024	66.6693s	15.0888s	25.0654s		40.1731%
3	0.21818	65.9061s	14.5280s	25.0180s		40.3423%
4	0.22081	66.7491s	15.1535s	25.0785s		40.1868%
5	0.22551	66.5713s	14.7364s	25.1867s		40.3493%
6	0.22502	67.0326s	15.1783s	25.1756s		40.2058%
7	0.22583	66.4551s	14.6125s	25.1941s		40.3443%
8	0.21866	65.4931s	14.0497s	25.0293s		40.5487%
9	0.22107	65.7390s	14.2026s	25.0845s		40.4188%
10	0.21963	66.1513s	14.6520s	25.0516s		40.3308%
Promedio total	0.22208	66.4151s	14.7733s	25.1079s		40.2949%
Intervalo de confianza:	[0.21985, 0.22432]	[65.9914s, 66.8388s]	[14.4417s, 15.1048s]	[25.0565s, 25.1593s]		[40.1929%, 40.3970%]

Todos los mensajes						
Simulación	Media rebotes en PC1		T. promedio por mensaje en			% tiempo cómputo
	Sistema	Colas	Trans.			
1	0.15390	33.7041s	9.6303s	10.9723s		54.9898%
2	0.15294	33.5950s	9.5513s	10.9566s		54.5733%
3	0.15224	33.2602s	9.2557s	10.9626s		54.8893%
4	0.15159	33.4838s	9.4752s	10.9362s		55.1342%
5	0.15290	33.4327s	9.3739s	10.9439s		54.7787%
6	0.15337	33.6618s	9.5776s	10.9617s		54.7359%
7	0.15352	33.3300s	9.2791s	10.9484s		54.9962%
8	0.15131	33.0639s	9.0828s	10.9267s		54.8858%
9	0.15271	33.2068s	9.1812s	10.9657s		54.8734%
10	0.15017	33.2703s	9.3733s	10.8757s		54.8488%
Promedio total	0.15246	33.4009s	9.3780s	10.9450s		54.8705%
Intervalo de confianza:	[0.15164, 0.15329]	[33.2503s, 33.5514s]	[9.2488s, 9.5072s]	[10.9249s, 10.9651s]		[54.7598%, 54.9812%]

En todas las tablas es posible modificar el porcentaje de confianza con que se desea calcular el intervalo de confianza. Basta con seleccionarlo de la lista y hacer clic en "Actualizar Estadísticas".

Reiniciar
Iniciar Simulación
Ejecuci

No mostrar los resultados en tiempo real

Porcentaje de confianza 95% ▼ Actualizar estadísticas

DESCRIPCIÓN DEL SISTEMA A SIMULAR

El enunciado del proyecto, incluidas las modificaciones, es el siguiente:

Suponga que hay una red de 3 computadoras cuyo único trabajo es el de encargarse de la recepción, distribución y envío de los mensajes electrónicos en determinada empresa. Esta red funciona así:

- La computadora No.1 es la única encargada de enviar los mensajes a su destino, pero ésta no los recibe directamente de los usuarios sino desde las otras 2 computadoras.
- La computadora No.2, que tiene 2 procesadores, recibe mensajes con una distribución para el tiempo entre arribos normal con una media de 15 segundos y una varianza de 1 segundo cuadrado. Esta computadora analiza y prepara cada uno de estos mensajes con alguno de sus dos procesadores (cada procesador tarda en promedio un tiempo con distribución uniforme entre 12 y 25 segundos, una vez analizado y preparado un mensaje, éste lo envía siempre a la computadora No.1. La computadora No. 1 usualmente le devuelve a esta computadora (la Núm 2) el 20% de los mensajes que recibe de ella para que los analice y prepare de nuevo pues llegaron con algún error. (Esto ocurre igual con los mensajes que llegan por primera vez a la computadora 1 desde la 2 o con los que llegan por n-ésima vez).
- La computadora No. 3 recibe mensajes con distribución de tiempo entre arribos con densidad igual a $f(x) = x/24$ con x entre 4 y 8 segundos. Esta máquina solo cuenta con 1 procesador y tiene el mismo trabajo que se describe para la computadora 2: analiza y prepara cada mensaje para ser

enviado a la computadora 1, sin embargo se da el caso especial acá de que en promedio, el 80% de todos los mensajes que aquí llegan son rechazados totalmente como resultado del análisis por no cumplir con alguna especificación obligatoria. Los que no son rechazados son enviados a la computadora No.1. La computadora No. 1 usualmente le devuelve a esta computadora el 50% de todos los mensajes que recibe de ella (igual probabilidad para los mensajes nuevos y para lo que ya habían sido devueltos 1 o más veces). Estos mensajes devueltos se vuelven a analizar y preparar en la computadora No. 3, y corren igual suerte como si fueran nuevos: puede ser rechazado con la misma probabilidad. El procesador de esta máquina puede analizar y preparar (o rechazar) un mensaje en un promedio de 4 segundos, tiempo exponencial.

- La computadora No. 1, con un procesador, puede procesar un promedio de 10 mensajes por minuto (t. exponencial), ya sea enviándolos a su destino, o devolviéndolos a su computadora de origen por llegar con error. Cuando se devuelve un mensaje, ya sea a la computadora 2 o a la 3 el tiempo que tarda en llegar tiene distribución constante de 3 segundos.
- El tiempo que tarda en llegar un mensaje a la computadora 1 cuando es enviado desde la computadora 2 o desde la 3 siempre son 20 segundos (Suponga que no se usan las mismas líneas de transmisión y que las colisiones de arribos no son problema).
- En cualquier computadora el tiempo que se tarda en colocar un mensaje que llega en la cola, o en asignárselo a un procesador si éste está libre es 0 para efectos prácticos.

DISEÑO DEL SIMULADOR

El simulador representa la simulación por medio de eventos que modifican el estado del sistema. El siguiente diagrama representa los tipos de objetos que representan la simulación:



Los conceptos fundamentales del simulador son los siguientes:

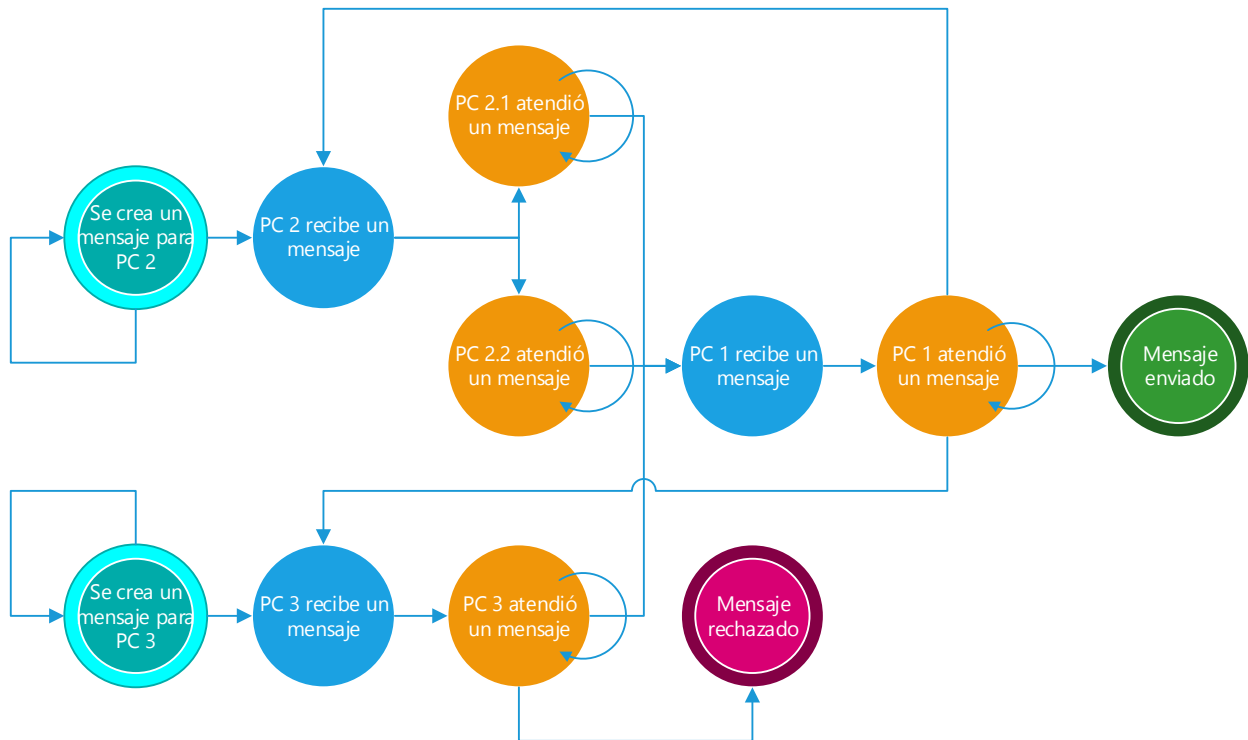
1. El estado del sistema está totalmente contenido en un objeto Estado.
2. Los eventos son representados por un objeto Evento que contiene los siguientes datos:
 - a. Información sobre la instancia actual del evento (como la hora de ocurrencia o el mensaje asociado al evento).
 - b. Una función lambda que transforma un Estado en el siguiente estado. Debido a que es una función pura (en el ámbito del simulador) se facilita enormemente ejecutar la simulación, pues cada evento puede ejecutar cualquier tipo de lógica, sin afectar la lógica del simulador.
3. Los eventos a ejecutar están en una lista, ordenada por el tiempo de ocurrencia del evento. Esto ofrece las siguientes ventajas:
 - a. Como la lista está ordenada el próximo evento siempre será la cabeza de la lista.
 - b. Programar nuevos eventos es trivial: simplemente se insertan a la lista de eventos. Como la lista se puede expandir para contener la cantidad de eventos que sea nunca van a suceder problemas como que no se pueda programar un evento porque hay otro del mismo tipo programado (o que le "caiga encima").
 - c. Los eventos pueden programar nuevos eventos de una forma funcionalmente pura. Nunca modifican el estado del simulador. En particular, nunca hay mutación del estado de eventos previamente programados.
4. El simulador básicamente ejecuta pasos de la simulación hasta que la simulación se termine (*reloj* \geq *tiempo máximo*).

EVENTOS

El sistema fue simulado utilizando los siguientes eventos:

1. Se crea un mensaje para PC 2
2. Se crea un mensaje para PC 3
3. PC 1 recibe un mensaje (de cualquier computadora)
4. PC 2 recibe un mensaje (de cualquier computadora)
5. PC 3 recibe un mensaje (de cualquier computadora)
6. Se atendió un mensaje en PC 1
7. Se atendió un mensaje en PC 2.1
8. Se atendió un mensaje en PC 2.2.
9. Se atendió un mensaje en PC 3
10. Metaeventos
 - a. Se envió un mensaje
 - b. Se rechazó un mensaje

La programación/creación de los eventos se da de la siguiente forma:



La lógica de los eventos es la siguiente¹:

1. **Se crea un mensaje para PC2/PC3:** Este evento corresponde a la llegada/creación de un mensaje desde afuera del sistema.
 - a. Se crea un nuevo mensaje (usando el reloj actual, etc).
 - b. Se programa la recepción inmediata del mensaje en la PC correspondiente
 - c. Se programa la creación de un nuevo mensaje en la PC correspondiente a futuro, según la distribución del tiempo entre arribos correspondiente por PC.
2. **Se recibe un mensaje en PC 1, PC2 o PC3:** Este evento corresponde a la llegada de un mensaje a una PC. El mensaje puede provenir de cualquier computadora (cada mensaje tiene información sobre su origen).
 - a. Se actualiza el tiempo de transmisión del mensaje
 - b. Se actualiza la hora de evento del mensaje
 - c. Si hay algún CPU libre:
 - i. Se marca el CPU como ocupado
 - ii. Se actualiza la hora de ocupación del CPU
 - iii. Se programa un nuevo evento "Mensaje atendido en PC#-#" a futuro, según la distribución del tiempo de atención correspondiente a cada PC/CPU.
 - d. Sino:
 - i. Se encola el mensaje en la PC correspondiente. Nótese que hay una cola por PC, no por CPU.

¹ La lógica de los eventos está en el archivo eventos.js

3. **Se atendió un mensaje en PC1, PC2.1, PC2.2 o PC3:** Este evento corresponde al momento en que se terminó de atender un mensaje en un determinado CPU. Los pasos "e" y "f" NO aplican para el PC 2, pues este nunca rechaza mensajes.
 - a. Se actualiza el tiempo de procesamiento del mensaje y otras estadísticas
 - b. Se actualiza la hora actual en el mensaje
 - c. Se actualiza la PC anterior en el mensaje
 - d. Se calcula la probabilidad de que el mensaje sea rechazado, usando la distribución de probabilidad de rechazo correspondiente a cada PC/CPU.
 - e. Si debe ser rechazado entonces:
 - i. Se programa un nuevo evento, según el PC, usando la distribución del tiempo de transmisión correspondiente para dicha ruta:
 1. PC 1: Se programa el evento "PC# recibe mensaje", según el mensaje venga de la PC 2 o la 3
 2. PC 3: Se programa el metaevento "Mensaje Rechazado"
 - f. Sino:
 - i. Se programa un nuevo evento, según el PC, usando la distribución de tiempo de transmisión correspondiente para dicha ruta:
 1. PC 1: Se programa el evento "Mensaje Enviado"
 2. PC 3: Se programa el evento "PC 1 recibe mensaje"
 - g. Se libera el CPU
 - h. Si hay mensajes en la cola del PC actual:
 - i. Se marca el CPU como ocupado
 - ii. Se actualiza la hora de ocupación del CPU
 - iii. Se borra el mensaje de la cola
 - iv. Se actualizan las estadísticas del tiempo en cola del mensaje y la hora actual.
 - v. Se programa un nuevo evento "Se atendió mensaje en PC#", según la distribución del tiempo de servicio para el PC actual.
4. **Metaeventos:** No forman parte del sistema simulado, pero son convenientes porque representan momentos donde se desea recolectar estadísticas. Se tiene un metaevento para los mensajes enviados y otro para los mensajes rechazados. Su única función es actualizar las estadísticas acumuladas con las estadísticas del mensaje que está saliendo del sistema.

ESTADÍSTICAS

Las estadísticas del sistema son recolectadas tanto en acumuladores globales (separados en mensajes rechazados y aceptados), los cuales son actualizados a partir de las estadísticas que carga cada mensaje al momento de salida del sistema (o por los CPU al terminar la simulación).

Los promedios entre simulaciones y los intervalos de confianza son calculados por la parte del programa que controla la GUI, no por el código de la simulación.

EL SIMULADOR

Cada simulación es realizada en pasos discretos. El programa ejecuta un paso de la simulación hasta que el reloj del siguiente evento supere el tiempo máximo que se desea simular. A continuación se describe la forma en que se realizan las “mutaciones” al estado de una simulación.

Como anteriormente se había mencionado, cada evento contiene una función capaz de tomar un Estado del sistema y devolver el estado siguiente. Esto aprovecha la naturaleza funcional de Javascript, pues abstrae y encapsula la lógica de los eventos. A continuación se muestra el código que ejecuta un paso de la simulación:

```
// Archivo simulacion.js
// SimulationStep(estado)
// Ejecuta un paso de la simulación y devuelve el nuevo
// estado.
// Recibe:
// - estado: un estado de una simulación
// Retorna:
// El siguiente estado de la simulación
function SimulationStep(estado){
    // Toma el evento más reciente
    var evento = PeekEvent(estado);
    // Y lo borra de la lista de eventos:
    estado = PopEvent(estado);

    // Actualiza el reloj:
    estado.Reloj = evento.Tiempo;

    // Y escribe cual evento se está
    // ejecutando ahora, para poder
    // hacer binding en la GUI.
    estado.Ejecutando = evento;

    // Ejecuta el evento actual y retorna
    // el nuevo estado del sistema.
    estado = evento.Lambda(estado, evento.Data);
    return estado;
}
```

Su funcionamiento es el siguiente:

1. PeekEvent retorna el siguiente evento programado en la simulación actual.
2. PopEvento borra el evento obtenido anteriormente de la lista de eventos.
3. Se actualiza el reloj de la simulación actual. El nuevo reloj corresponde a la hora del siguiente evento.
4. Se guarda en una variable cual es el evento que se está procesando. Esto es solamente para facilitar la interacción con la GUI
5. Se retorna el nuevo estado del sistema, que es evento.Lambda(estado, evento.Data)

- a. La función evento.Lambda toma el estado actual del sistema y los datos del evento y retorna el siguiente estado del sistema.

El código de la interfaz gráfica hace otros cálculos, como manejar una lista con todas las simulaciones y calcular estadísticas, pero en general la única "mutación" al estado de la simulación es la función SimulationStep.

El uso de programación funcional y listas de eventos nos da los siguientes beneficios:

1. Implementar un evento es simple. Solamente hay que definir una función lambda que modifique el estado del sistema según la lógica que deseemos implementar.
2. No hay efectos secundarios dentro del ámbito de la simulación.
 - a. Esto facilita probar y depurar el programa. Si cada función de evento es correcta y el estado inicial del sistema es correcto entonces, por inducción, el simulador solamente puede mostrar estados correctos del sistema, sin importar la cantidad de veces que se "mute" el estado del sistema.
 - b. Usar una lista de eventos evita posibles errores a la hora de programar:
 - i. Es imposible sobrescribir un evento. Las únicas operaciones válidas son insertar un evento (de forma ordenada) o sacar un evento para ejecutarlo.
 - c. Es sencillo demostrar el funcionamiento correcto del programa. Se puede demostrar la correctitud de la composición de eventos dado que cumplan ciertos invariantes. Se podría demostrar la correctitud del código si se comprueba que este cumple los invariantes especificados anteriormente.
3. Permite hacer razonamientos y demostraciones sobre el funcionamiento esperado del código. Facilita comprobar que el código es acorde a la especificación.

DEMOSTRACIONES SOBRE LA LÓGICA ESPERADA DEL PROGRAMA

A continuación se especifican condiciones esperadas del código del programa, y cómo estas condiciones bastan para garantizar ciertas invariantes sobre el sistema.

Supuestos:

1. La interfaz gráfica obtiene el siguiente Estado de la simulación llamando varias veces a SimulationStep. Nunca modifica el estado.
2. Las estadísticas del lado de la interfaz gráfica están implementadas de forma correcta.
3. La lógica de cada evento está programada de forma correcta.

En particular deseamos garantizar lo siguiente:

1. El estado de la simulación en la interfaz gráfica es correcto
2. El reloj de la simulación NUNCA se devuelve
3. Nunca se sobrescribe ningún evento
4. Los eventos serán ejecutados según su hora programada.

Sea $E = \{e_0, e_1, \dots, e_n\}$ el conjunto con todos los eventos posibles, y sea S una sucesión de estados del sistema, donde s_i representa el i -ésimo estado del sistema.

Defínase la función $e_i. \lambda: S \rightarrow S$, donde $e_i. \lambda(s_n) = s_{n+1}$.

Cada estado s_i de la simulación tiene una lista de eventos $s_i. L$. Solamente es "modificada" por las siguientes funciones:

PushEvent(evento, s_i): Inserta el evento en $s_i. L$ y retorna s_i , manteniendo el orden de programación de eventos en $s_i. L$.

PopEvent(s_i): Borra de $s_i. L$ el primer evento.

PeekEvent(s_i): Retorna $s_i. L_0$.

Supóngase que S_0 es un estado correcto del sistema. Para efectos de nuestra simulación es necesario que S_0 contenga programados dos eventos: la llegada de mensajes a PC 2 y PC 3.

Dado S_n el estado S_{n+1} se obtiene de la siguiente manera (SimulationStep):

1. $S_{n+1} =$
 - a. $S := S_n$,
 - b. $e := PeekEvent(S)$,
 - c. $S.Reloj := e.Hora$,
 - d. $e. \lambda(S)$ En el programa lambda también recibe los datos extra del evento.

Si la función lambda está bien implementada entonces, por inducción, en este punto se puede garantizar que la simulación estaría correcta. Esto de ningún modo demuestra que el programa está implementado de forma correcta, pero demuestra que no es necesario comprobar que la composición de lambdas de eventos resulta en una simulación correcta, sino que basta con verificar y validar la función lambda y el estado inicial.

Para comprobar que nunca se sobrescriben o "pierden" eventos es suficiente comprobar que la función PushEvent solamente inserta eventos de forma ordenada, pero no sobrescribe o borra los que ya están. Esto garantiza que se puedan tener programados cualquier cantidad de eventos y de cualquier tipo. Esto permite, por ejemplo, tener un único evento "PC recibe mensaje", en lugar de "PC recibe de B1", "PC recibe de B2", "PC recibe de C". Tener menos eventos significa menos código y probablemente menos errores.

Como los eventos son insertados en la lista de eventos en orden (la lista luego de la inserción está ordenada por hora de ocurrencia del evento) entonces el primer elemento de la lista necesariamente será el siguiente evento (tiene el tiempo de ocurrencia menor de todos los eventos). Por lo tanto los eventos siempre son ejecutados en orden, sin importar en qué orden fueron insertados a la lista.

Finalmente se comprueba que el reloj nunca se devuelve. Para garantizar esto es necesario que se cumpla el siguiente supuesto: TODOS los eventos programados (por otro evento) ocurrirán a la hora $Reloj + r$, con $r \geq 0$. En el caso de nuestra simulación esto es fácil de comprobar, pues r es un valor aleatorio generado con funciones de distribución de probabilidad que NUNCA retornan valores negativos:

Sea r un valor real aleatorio en el intervalo $[0,1[$:

1. Distribución uniforme entre a y b ($0 \leq a \leq b$):
 - a. Se cumple que:
 - b. $a \geq 0$
 - c. $b - a \geq 0$
 - d. $r \geq 0$
 - e. $\therefore a + (b - a) * r \geq a \geq 0$ ■
2. Distribución exponencial con parámetro $\lambda, \lambda \geq 0$:
 - a. $1 > r \geq 0$
 - b. $0 < 1 - r \leq 1$
 - c. $\ln 1 - r \leq 0$
 - d. $\frac{\ln 1 - r}{-\lambda} \geq 0$
3. Distribución $X/24$ y $X/600$:
 - a. Tienen la forma $c\sqrt{3r + 1}$, con $c > 0$. Siempre es positivo.
4. Distribución normal con media 15 segundos y varianza de 15 al cuadrado:
 - a. Se calculó con software CAS (Wolfram Alpha) la integral desde menos infinito hasta o de la función acumulada de dicha distribución. El resultado es ≈ 0 .
 - b. La implementación de la distribución normal estándar usada utiliza 12 valores random que se suman y luego se les resta 6.
 - c. La Distribución Normal(u, v) = $N v + u$, con N un valor aleatorio que sigue una distribución normal estándar.
 - d. $DistribucionNormal(15,1) = N * 1 + 15 < 0 \Leftrightarrow N < -15$. Pero, por la forma en que se calcula se cumple que $-6 \leq N < 6$. Por lo tanto, $N > -15$, lo que implica que la distribución normal con media 15 segundos y varianza 15 al cuadrado en este programa siempre da un número positivo.
 - i. De hecho, solamente retorna valores entre 9 y 21.

Como los eventos siempre son ejecutados en orden y los eventos siempre son programados ahora o en el futuro es imposible que el reloj en algún punto de la simulación "se devuelva".

ESTADÍSTICAS Y ANÁLISIS DEL SISTEMA

A continuación se muestran los resultados de ejecutar la simulación diez veces durante 1000 minutos, usando un porcentaje de confianza del 95%.

RENDIMIENTO DEL SISTEMA

Simulación	Reloj (minutos)	Mensajes			% ocupación				% ocupación rechazados		% mensajes rechazados
		Enviados	Rechazados	Total	PC1	PC2.1	PC2.2	PC3	PC1	PC3	
1	999.9 703	5038	8623	13661	70.980 1%	81.558 7%	73.168 6%	71.640 9%	9.416 2%	63.884 0%	63.12 13
2	999.9 898	5031	8591	13622	73.150 5%	81.314 4%	73.653 0%	71.673 4%	9.801 8%	63.710 8%	63.06 71
3	999.9 875	5056	8596	13652	68.59 93%	81.286 2%	73.229 9%	70.926 5%	8.432 9%	63.163 5%	62.9 651
4	999.9 593	5101	8527	13628	71.972 8%	81.559 8%	73.498 6%	71.519 6%	9.548 9%	63.777 8%	62.5 697
5	999.9 950	5047	8588	13635	71.709 4%	81.512 6%	73.042 9%	71.808 2%	10.171 3%	64.06 95%	62.9 850
6	999.9 559	4992	8655	13647	71.631 8%	81.929 0%	73.968 1%	70.938 0%	9.638 1%	63.807 8%	63.42 05
7	999.9 733	5016	8656	13672	70.120 9%	81.381 2%	72.428 4%	72.752 1%	9.845 5%	65.388 1%	63.31 19
8	999.9 964	5044	8615	13659	71.285 1%	80.427 6%	72.969 5%	72.844 4%	9.308 9%	64.745 1%	63.07 20
9	999.9 969	5038	8612	13650	70.690 1%	82.028 4%	73.878 7%	71.279 8%	9.379 1%	63.338 4%	63.09 16
10	999.9 701	5058	8589	13647	72.207 1%	80.747 6%	73.065 2%	70.571 0%	9.7471 %	62.547 5%	62.93 69
\bar{X}	999.9 794	5,042.1 000	8,605.2 000	13,647.3 000	71.234 7%	81.374 6%	73.290 3%	71.595 4%	9.529 0%	63.843 3%	63.05 41
I. Confianza	[999.9683, 999.9906]	[5,021.7407, 5,062.4593]	[8,578.5638, 8,631.8362]	[13,636.3006, 13,658.2994]	[70.3396%, 72.1298%]	[81.0271%, 81.7220%]	[72.9568%, 73.6238%]	[71.0630%, 72.1278%]	[9.1976%, 9.8604%]	[63.2754%, 64.4111%]	[62.8910, 63.2173]

MENSAJES RECHAZADOS

Simulación	Media rebotes en PC1	T. promedio por mensaje en			% tiempo cómputo
		Sistema	Colas	Trans.	
1	0.10843	13.5158s	5.9217s	2.4939s	63.9107%
2	0.11675	14.3981s	6.5787s	2.6853s	62.7924%
3	0.09981	13.3081s	6.0150s	2.2957s	64.5208%
4	0.10930	14.0094s	6.3361s	2.5139s	63.5875%
5	0.11667	14.2520s	6.3816s	2.6835s	62.7714%
6	0.11473	14.0283s	6.2981s	2.6388s	63.8058%
7	0.11229	14.4022s	6.6047s	2.5827s	61.9804%
8	0.11120	14.1960s	6.4808s	2.5576s	62.1034%
9	0.10996	13.7611s	6.1657s	2.5291s	63.5780%
10	0.11259	13.8883s	6.2487s	2.5895s	64.1780%
Promedio total	0.11117	13.9759s	6.3031s	2.5570s	63.3228%
Intervalo de confianza	[0.10765, 0.11470]	[13.7147s, 14.2371s]	[6.1421s, 6.4641s]	[2.4759s, 2.6381s]	[62.7031%, 63.9426%]

MENSAJES ENVIADOS

Simulación	Media rebotes en PC1	T. promedio por mensaje en			% tiempo cómputo
		Sistema	Colas	Trans.	
1	0.22787	66.3629s	14.4468s	25.2410s	40.5315%
2	0.22739	68.0751s	15.8714s	25.2300s	40.3132%
3	0.21479	64.5414s	13.2127s	24.9403s	41.0811%
4	0.21819	66.2477s	14.7602s	25.0184s	40.3519%
5	0.21914	66.4889s	14.8530s	25.0402s	40.2901%
6	0.23137	68.3179s	15.9733s	25.3215s	40.1874%
7	0.21910	64.7784s	13.2666s	25.0393s	41.0255%
8	0.21372	65.2064s	13.7564s	24.9155s	40.9286%
9	0.23065	66.3063s	14.1952s	25.3049s	40.5080%
10	0.21945	67.3621s	15.7197s	25.0474s	40.1095%
Promedio total	0.22217	66.3687s	14.6055s	25.1099s	40.5327%
Intervalo de confianza:	de [0.21750, 0.22684]	[65.4468s, 67.2906s]	[13.8727s, 15.3384s]	[25.0024s, 25.2173s]	[40.2782%, 40.7872%]

TODOS LOS MENSAJES

Simulación	Media rebotes en PC1	T. promedio por mensaje en			% tiempo cómputo
		Sistema	Colas	Trans.	
1	0.15248	33.0051s	9.0657s	10.8827s	55.2888%
2	0.15761	34.2226s	10.0108s	11.0117s	54.4902%
3	0.14240	32.2823s	8.6806s	10.6821s	55.8399%
4	0.15006	33.5624s	9.4893s	10.9374s	54.8904%
5	0.15460	33.5875s	9.5173s	10.9589s	54.4499%
6	0.15740	33.8871s	9.8372s	10.9360s	55.1663%
7	0.15148	32.8843s	9.0488s	10.8216s	54.2924%
8	0.14906	33.0331s	9.1676s	10.8140s	54.2840%
9	0.15451	33.1547s	9.1293s	10.9353s	55.0632%
10	0.15219	33.7073s	9.7589s	10.9131s	55.2575%
Promedio total	0.15218	33.3326s	9.3706s	10.8893s	54.9023%
Intervalo de confianza:	[0.14899, 0.15536]	[32.9273s, 33.7380s]	[9.0703s, 9.6708s]	[10.8217s, 10.9568s]	[54.5345%, 55.2700%]

ANÁLISIS DEL SISTEMA

El sistema en general está en equilibrio. Sin embargo, se puede observar que mientras que los mensajes rechazados son rechazados muy rápidamente los mensajes que al final son enviados tardan mucho más.

La computadora 3 atiende los mensajes más rápido que los CPU de la computadora 2. Pero la computadora 3 pasa ocupada generalmente con mensajes que al final son rechazados. Si se desea reducir el tiempo que tardan los mensajes enviados en el sistema quizá sea mejor intercambiar los roles de la PC 2 y la PC3.

También es necesario mejorar la velocidad de la transferencia de datos. Los mensajes enviados tardan en promedio 25,1099 segundos en transporte. Eso es muchísimo tiempo, considerando que el ping entre mi computadora y servidores alejados en Japón o Rusia es menos de 0,5s. Aunque es posible que sean mensajes muy grandes.

PROBLEMAS ENCONTRADOS

Se encontraron los siguientes problemas:

RENDIMIENTO

Debido a la forma en que los navegadores implementan los motores de Javascript y la interacción con la interfaz gráfica hay que escoger entre actualizar la pantalla regularmente o hacer la simulación lo más rápido posible.

Para solucionar esta disyuntiva se optó por ofrecer dos modos de simulación: uno que actualiza la pantalla a cada paso y se espera cierta cantidad de milisegundos (pero es muy lento) y otro que actualiza la interfaz gráfica luego de terminar cada simulación. Estas dos opciones ofrecen un buen balance entre velocidad y visualización de la simulación.

EJECUCIÓN DEL PROGRAMA

Debido a las políticas de seguridad de los navegadores no es posible ejecutar cualquier código de forma local. Para solucionar esto se tomaron las siguientes medidas:

- Durante el desarrollo se utilizó un servidor local, específicamente el paquete `http.server` de Python 3.
- Para facilitar la ejecución por otras personas se desplegó la aplicación en un servidor público. La aplicación funciona sin restricciones en este entorno.
- Si ninguna de las opciones anteriores es posible de usar la alternativa sería correr el programa usando Firefox.