UNIVERSIDAD NACIONAL DE CÓRDOBA

FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES

CÁTEDRA DE PROGRAMACIÓN CONCURRENTE

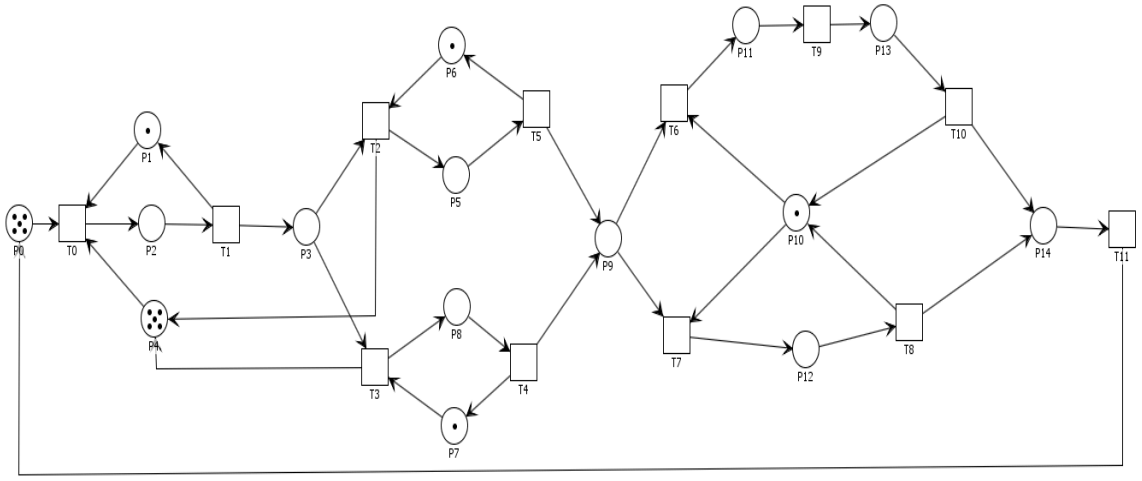
TRABAJO PRÁCTICO Nº 2

“Sistema de reservas de vuelos”

Grupo Nº 8

Integrantes		
Nombre	DNI	email
Piñera Nicolas	44195541	nicolas.pinera@mi.unc.edu.ar
Ñañez Josias Tomas	41886293	josias.nanez@mi.unc.edu.ar
Fatu Javier Alejandro	44741112	javier.fatu@mi.unc.edu.ar

1. Análisis red de petri



Esta red de Petri modela un sistema de agencia de viajes. Para realizar el análisis de la red de petri decidimos utilizar el simulador petrinator

- Las plazas $\{P_1, P_4, P_6, P_7, P_{10}\}$ representan recursos compartidos en el sistema.
- La plaza $\{P_0\}$ es una plaza idle que corresponde al buffer de entrada de clientes de la agencia.
- La plaza $\{P_2\}$ representa el ingreso a la agencia.
- La plaza $\{P_3\}$ representa la sala de espera de la agencia.
- Las plazas $\{P_5, P_8\}$ representan los estados en los cuales se realiza la gestión de las reservas.
- La plaza $\{P_9\}$ representa la espera de los clientes para pasar por el agente que aprueba o rechaza la reserva.
- En la plaza $\{P_{12}\}$ se modela la cancelación de la reserva.
- En las plazas $\{P_{11}, P_{13}\}$ se modela la confirmación y el pago respectivamente de la reserva.
- En la plaza $\{P_{14}\}$ se modela la instancia previa a que el cliente se retire.

1.1.1 Matriz de post-incidencia W^+:

Esta matriz nos muestra cuantos token se colocan a la salida de una transición, en una plaza determinada

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
P0	0	0	0	0	0	0	0	0	0	0	0	1
P1	0	1	0	0	0	0	0	0	0	0	0	0
P2	1	0	0	0	0	0	0	0	0	0	0	0
P3	0	1	0	0	0	0	0	0	0	0	0	0
P4	0	0	1	1	0	0	0	0	0	0	0	0
P5	0	0	1	0	0	0	0	0	0	0	0	0
P6	0	0	0	0	0	1	0	0	0	0	0	0
P7	0	0	0	0	1	0	0	0	0	0	0	0
P8	0	0	0	1	0	0	0	0	0	0	0	0
P9	0	0	0	0	1	1	0	0	0	0	0	0
P10	0	0	0	0	0	0	0	0	1	0	1	0
P11	0	0	0	0	0	0	1	0	0	0	0	0
P12	0	0	0	0	0	0	0	1	0	0	0	0
P13	0	0	0	0	0	0	0	0	0	1	0	0
P14	0	0	0	0	0	0	0	0	1	0	1	0

1.1.2 Matriz de pre-incidencia W^-:

Esta matriz nos muestra cuantos tokens se sacan de una plaza al momento de disparar una determinada transición.

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
P0	1	0	0	0	0	0	0	0	0	0	0	0
P1	1	0	0	0	0	0	0	0	0	0	0	0
P2	0	1	0	0	0	0	0	0	0	0	0	0
P3	0	0	1	1	0	0	0	0	0	0	0	0
P4	1	0	0	0	0	0	0	0	0	0	0	0
P5	0	0	0	0	0	1	0	0	0	0	0	0
P6	0	0	1	0	0	0	0	0	0	0	0	0
P7	0	0	0	1	0	0	0	0	0	0	0	0
P8	0	0	0	0	1	0	0	0	0	0	0	0
P9	0	0	0	0	0	0	1	1	0	0	0	0
P10	0	0	0	0	0	0	1	1	0	0	0	0
P11	0	0	0	0	0	0	0	0	0	1	0	0
P12	0	0	0	0	0	0	0	0	1	0	0	0
P13	0	0	0	0	0	0	0	0	0	0	1	0
P14	0	0	0	0	0	0	0	0	0	0	0	1

1.1.3 Matriz de incidencia $W = W^+ - W^-$:

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
P0	-1	0	0	0	0	0	0	0	0	0	0	1
P1	-1	1	0	0	0	0	0	0	0	0	0	0
P2	1	-1	0	0	0	0	0	0	0	0	0	0
P3	0	1	-1	-1	0	0	0	0	0	0	0	0
P4	-1	0	1	1	0	0	0	0	0	0	0	0
P5	0	0	1	0	0	-1	0	0	0	0	0	0
P6	0	0	-1	0	0	1	0	0	0	0	0	0
P7	0	0	0	-1	1	0	0	0	0	0	0	0
P8	0	0	0	1	-1	0	0	0	0	0	0	0
P9	0	0	0	0	1	1	-1	-1	0	0	0	0
P10	0	0	0	0	0	0	-1	-1	1	0	1	0
P11	0	0	0	0	0	0	1	0	0	-1	0	0
P12	0	0	0	0	0	0	0	1	-1	0	0	0
P13	0	0	0	0	0	0	0	0	0	1	-1	0
P14	0	0	0	0	0	0	0	0	1	0	1	-1

1.2 Invariantes de transición:

T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
1	1	0	1	1	0	0	1	1	0	0	1
1	1	0	1	1	0	1	0	0	1	1	1
1	1	1	0	0	1	0	1	1	0	0	1
1	1	1	0	0	1	1	0	0	1	1	1

Los invariantes de transición son aquellas secuencias de disparo de transiciones, la cual nos devuelve el marcado inicial de la red. Podemos ver que para las siguientes secuencias de disparo, la red mantiene el mismo marcado.

$$T_0 \rightarrow T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_7 \rightarrow T_8 \rightarrow T_{11}$$

$$T_0 \rightarrow T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_6 \rightarrow T_9 \rightarrow T_{10} \rightarrow T_{11}$$

$$T_0 \rightarrow T_1 \rightarrow T_2 \rightarrow T_5 \rightarrow T_7 \rightarrow T_8 \rightarrow T_{11}$$

$$T_0 \rightarrow T_1 \rightarrow T_2 \rightarrow T_5 \rightarrow T_6 \rightarrow T_9 \rightarrow T_{10} \rightarrow T_{11}$$

1.3 Invariantes de Plaza:

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
1	0	1	1	0	1	0	0	1	1	0	1	1	1	1

Los invariantes de plaza son un conjunto de plazas cuyo número total de tokens se mantiene constante a lo largo del tiempo, independientemente de las transiciones que se disparen. Podemos observar los siguientes invariantes de plaza

$$m(P_1) + m(P_2) = 1$$

$$m(P_2) + m(P_3) + m(P_4) = 5$$

$$m(P_5) + m(P_6) = 1$$

$$m(P_7) + m(P_8) = 1$$

$$m(P_{10}) + m(P_{11}) + m(P_{12}) + m(P_{13}) = 1$$

$$m(P_0) + m(P_2) + m(P_3) + m(P_5) + m(P_8) + m(P_9) + m(P_{11}) + m(P_{12}) + m(P_{13}) + m(P_{14}) = 5$$

En la primera ecuación podemos ver que la cantidad de personas que van a pasar es una por vez. En la segunda vemos que la cantidad máxima de personas que puede haber en la sala de espera es de cinco.

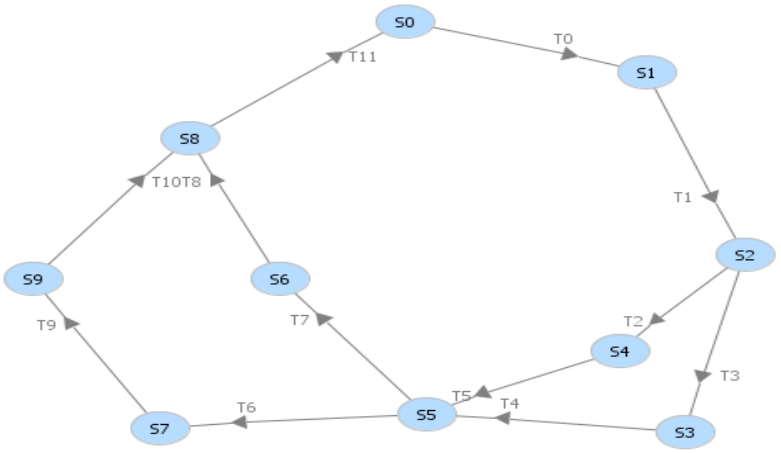
En la tercera y cuarta ecuación vemos que cada agente de reserva atenderá a una sola persona por vez. En la quinta vemos que se verificará el pago o cancelación de una persona a la vez.

En este caso para poder estudiar las propiedades de la red, los clientes que salen vuelven a entrar en la agencia eso es lo que observamos en la ecuación 6), la cantidad de clientes es de cinco.

1.4 Grafo de alcanzabilidad y deadlock:

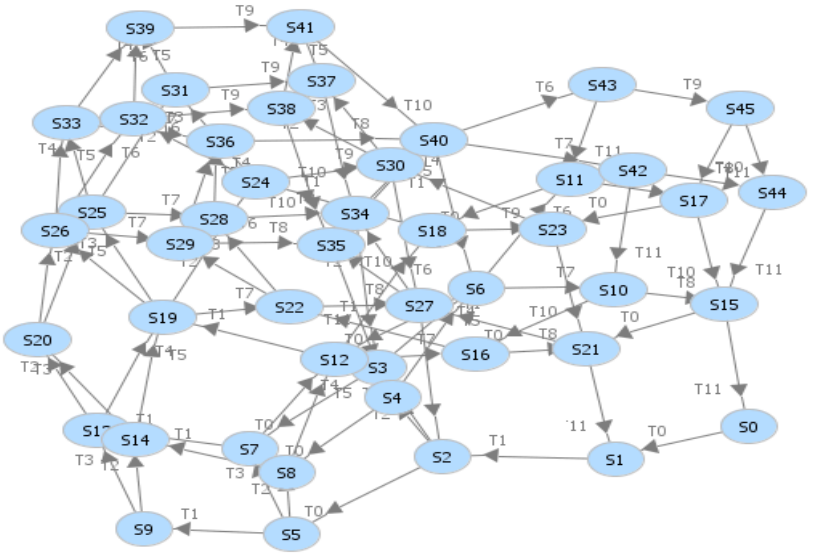
Marcado inicial: $(1, 1, 0, 0, 5, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0)$

En el grafo de alcanzabilidad podemos notar que no se produce deadlock, también vemos los invariantes de transición, por ejemplo, si disparamos la secuencia $T_0 \rightarrow T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_7 \rightarrow T_8 \rightarrow T_{11}$ regresamos al marcado inicial.



.

Grafo de alcanzabilidad con marcado inicial $(2, 1, 0, 0, 5, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0)$



Vemos que al aumentar el marcado de la plaza P_0 El número de combinaciones de transición crece exponencialmente pero seguimos viendo que las propiedades se mantienen.

1.5 Seguridad:

Una red de petri se denomina segura cuando el marcado máximo de cada una de sus plazas es uno. Podemos ver que en la plazas que representan salas de espera como por ejemplo P_3 pueden esperar más de un cliente por lo tanto esta red de petri no se considera segura.

1.6 Tabla de estados y eventos:

Estado inicial	Evento	Estado final
No hay clientes	Arriba una persona	Una persona está entrando
Una persona está entrando	Termina de pasar	El cliente está en la sala de espera
El cliente está en la sala de espera	El cliente pasa a ser atendido por el agente superior	El cliente está siendo atendido por agente superior
El cliente está en la sala de espera	El cliente pasa a ser atendido por el agente inferior	El cliente está siendo atendido por agente inferior
El cliente está siendo atendido por el agente superior	El cliente termina de ser atendido	El cliente espera a ser aprobado o rechazado
El cliente está siendo atendido por el agente inferior	El cliente termina de ser atendido	El cliente espera a ser aprobado o rechazado
El cliente espera a ser aprobado o rechazado	Se acepta la reserva	La reserva está aceptada
El cliente espera a ser aprobado o rechazado	Se rechaza la reserva	La reserva está rechaza
La reserva está aceptada	Se efectúa el pago	La reserva está pagada
La reserva está rechaza	Se despide al cliente	El cliente se va
La reserva está pagada	Se despide al cliente	El cliente se va

Tabla de estados			
Tabla de estados del sistema		**Tabla de eventos del sistema**	
P0	Entrada de clientes de la agencia.	T0	Entra una persona
P1	Limita entrada clientes		
P2	Cliente entrando	T1	Pasa a la sala de espera
P3	Sala de espera para hacer reserva		
P4	Libera lugar en sala de espera	T2	Pasa a ser atendido por el agente superior
P5	Cliente atendido por agente superior		
P6	Agente superior	T3	Pasa a ser atendido por el agente inferior
P7	Agente inferior	T4	Termina de ser atendido por el agente inferior
P8	Cliente atendido por agente inferior	T5	Termina de ser atendido por el agente superior
P9	Sala de espera para ser aprobado o rechazado	T6	Se acepta la reserva
P10	Agente que aprueba, cobra y rechaza	T7	Se cancela la reserva
P11	Aceptación de reserva	T8	Termina de ser atendido luego de que su reserva se cancele
P12	Cancelación de reserva	T9	Se paga su reserva
P13	Pago de reserva	T10	Termina de ser atendido luego de que su reserva fuera confirmada y pagada
P14	Salida de la agencia	T11	Cliente se retira

2. Determinación de hilos máximos activos simultáneos

2.1 IT de la red:

$$IT_1 = \{T_0, T_1, T_3, T_4, T_7, T_8, T_{11}\}$$

$$IT_2 = \{T_0, T_1, T_3, T_4, T_6, T_9, T_{10}, T_{11}\}$$

$$IT_3 = \{T_0, T_1, T_2, T_5, T_7, T_8, T_{11}\}$$

$$IT_4 = \{T_0, T_1, T_2, T_5, T_6, T_9, T_{10}, T_{11}\}$$

2.2 Obtenemos el conjunto de plazas PI de cada IT:

$$PI_1 = \{P_0, P_1, P_2, P_3, P_4, P_7, P_8, P_9, P_{12}, P_{10}, P_{14}\}$$

$$PI_2 = \{P_0, P_1, P_2, P_3, P_4, P_7, P_8, P_9, P_{11}, P_{13}, P_{10}, P_{14}\}$$

$$PI_3 = \{P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_9, P_{12}, P_{10}, P_{14}\}$$

$$PI_4 = \{P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_9, P_{11}, P_{13}, P_{10}, P_{14}\}$$

2.3 Obtenemos el conjuntos de plazas de acción PA de cada IT

$$PA_1 = \{P_2, P_3, P_8, P_9, P_{12}, P_{14}\}$$

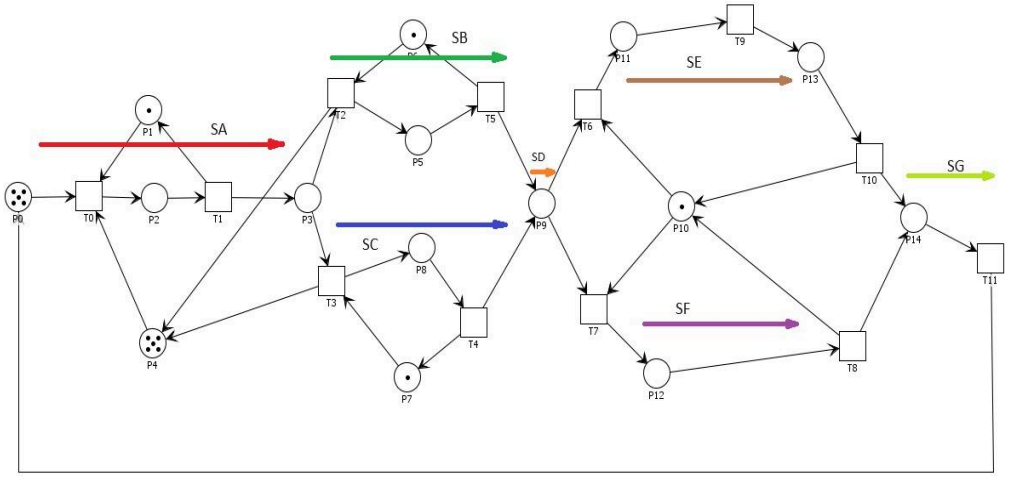
$$PA_2 = \{P_2, P_3, P_8, P_9, P_{11}, P_{13}, P_{14}\}$$

$$PA_3 = \{P_2, P_3, P_5, P_9, P_{12}, P_{14}\}$$

$$PA_4 = \{P_2, P_3, P_5, P_9, P_{11}, P_{13}, P_{14}\}$$

$$PA = \{P_2, P_3, P_5, P_8, P_9, P_{11}, P_{12}, P_{13}, P_{14}\}$$

1) De todos los marcados posibles de PA Se busca el marcado máximo, este valor determina la máxima cantidad de hilos activos simultáneos. En nuestro caso son **5 hilos activos en simultáneo**.



La red cuenta con 7 segmentos: el segmento S_A previo al primer conflicto (fork), el segmento superior S_B e inferior S_C, luego el segmento S_D que cumple con la condición de unión (join) y de conflicto (fork), luego sigue el segmento S_E y S_F y por último el segmento de unión S_G. Una vez determinados los segmentos, calculamos los hilos máximos necesarios por segmento de ejecución:

$$PS_A = \{P_2, P_3\} \;\; Max(MS_A) = 5$$

$$PS_B = \{P_5\} \;\; Max(MS_B) = 1$$

$$PS_C = \{P_8\} \;\; Max(MS_C) = 1$$

$$PS_D = \{P_9\} \;\; Max(MS_D) = 5$$

$$PS_E = \{P_{11}, P_{13}\} \;\; Max(MS_E) = 1$$

$$PS_F = \{P_{12}\} \;\; Max(MS_F) = 1$$

$$PS_G = \{P_{14}\} \;\; Max(MS_G) = 5$$

Debido que la plaza P_9 es un fork y join por lo tanto las siguientes transiciones cuentan con un conflicto, además por la lógica de la red de petri, en cada segmento $S_E y\, S_F$ no va a haber más de un cliente, decidimos no utilizar los 5 hilos de ese segmento y utilizar los hilos de los segmentos siguientes al conflicto. Como el simulado de los clientes termina con las transiciones T_{10} y T_8 decidimos no utilizar la transición T_{11} y por lo tanto no usar el segmento S_G.

Una vez determinados los hilos máximos por segmento, la suma de todos estos determina la máxima cantidad de hilos necesarios del sistema. En nuestro caso la cantidad de hilos necesarios es de **9 hilos**.

---

3. Analisis de politicas e Invariantes

En nuestro sistema hay dos políticas, las cuales son:

Una política balanceada: La cantidad de clientes atendidos por el agente de reservas superior (plaza P6), debe ser equitativo a la cantidad de clientes atendidos por el agente de reservas inferior (plaza P7). La cantidad de reservas canceladas debe ser equitativa a la cantidad de reservas confirmadas.

Una política de procesamiento priorizada: Se debe priorizar al agente de reservas superior (plaza P6). El 75% de las reservas deben ser creadas por dicho agente. Se debe priorizar la confirmación de reservas, obteniendo al finalizar, un 80% del total de reservas confirmadas.

Para realizar un análisis sobre la implementación de las políticas utilizamos una expresión regular en el lenguaje python la cual toma la secuencia completa de los disparos de transiciones y analiza cuántas veces se ejecutó cada invariante.

Tenemos los siguientes invariantes de transición que cada uno representa el camino que realizo un cliente en la agencia:

- **Invariante de Transición 1: T0T1T3T4T7T8T11** Representa a un cliente que ingresa a la agencia, es atendido por el agente inferior, y luego su reserva es cancelada.
- **Invariante de Transición 2: T0T1T3T4T6T9T10T11**: Representa a un cliente que ingresa a la agencia, es atendido por el agente inferior y luego su reserva es confirmada y pagada.
- **Invariante de Transición 3: T0T1T2T5T7T8T11**: Representa a un cliente que ingresa a la agencia, es atendido por el agente superior y luego su reserva es cancelada.
- **Invariante de Transición 4: T0T1T2T5T6T9T10T11**: Representa a un cliente que ingresa a la agencia, es atendido por el agente superior y luego su reserva es confirmada y pagada.

Analisis de politicas e Invariantes									
Política	**Ejec**	**IT 1**	**%**	**IT 2**	**%**	**IT3**	**%**	**IT 4**	**%**
1	1	55	30%	38	20%	41	22%	52	28%
	2	52	28%	41	22%	42	23%	51	27%
	3	59	32%	36	19%	55	30%	36	19%
	4	48	26%	45	24%	47	25%	46	25%
	5	35	19%	62	33%	37	20%	52	28%
	6	41	22%	52	28%	45	24%	48	26%
	7	45	24%	48	26%	48	26%	45	24%
	8	51	27%	42	23%	43	23%	50	27%
	9	53	28%	45	24%	41	22%	47	25%
	10	52	28%	47	25%	42	23%	45	24%
	Total	491	26%	456	25%	441	24%	472	25%
2	1	37	20%	38	20%	56	30%	55	30%
	2	29	16%	54	29%	41	22%	62	33%
	3	27	15%	41	22%	57	31%	61	33%
	4	33	18%	51	27%	39	21%	63	34%
	5	36	19%	40	22%	37	20%	73	39%
	6	34	18%	55	30%	27	15%	70	38%
	7	37	20%	35	19%	57	31%	57	31%
	8	38	20%	38	20%	50	27%	60	32%
	9	17	9%	51	27%	40	22%	78	42%
	10	17	9%	51	27%	26	14%	92	49%
	Total	305	16%	454	24%	430	23%	671	36%

En esta tabla podemos ver un total de 20 ejecuciones de nuestro programa, 10 utilizando la política balanceada y 10 con la política priorizada. En cada ejecución tenemos 186 invariantes de transición disparados, ya que son los 186 clientes que ingresan a nuestra agencia y cada uno recorre su camino hasta finalizar el proceso

Luego de este análisis podemos ver el correcto funcionamiento de las políticas, ya que en la política balanceada (1) si realizamos la suma de los porcentajes de los invariantes IT1 e IT2 sabemos que el agente inferior atiende el 50% de los clientes, luego sumando los

porcentajes de los invariantes IT3 e IT 4 sabemos que el agente superior atiende la otra mitad de clientes. Por otro lado, sumando los IT1 e IT3 vemos que la mitad de las reservaciones son canceladas y sumando IT2 y IT4 la otra mitad son confirmadas y pagadas.

Al ejecutar la política de procesamiento priorizado (2), vemos que si sumamos los porcentajes de los invariantes 3 y 4 obtenemos que el 60% de los clientes son atendidos por el agente superior y si sumamos los invariantes 2 y 4 obtenemos que un 50% de las reservas son confirmadas y pagadas. Hay que considerar que la política funciona de este modo para estos tiempos de simulación y los intervalos de alfa y beta establecidos para las transiciones temporales ya que se generan conflictos efectivos por lo cual es más impredecible el comportamiento , siendo que a veces elige la política y a veces no.

---

4. Análisis de tiempos

A continuación se presenta el análisis analítico de los tiempos, partiendo de un tiempo fijo definido por el grupo para cada transición o proceso. Luego, dejando fijo los tiempos de las demás transiciones, y variando la transición a analizar, sacamos el tiempo que demoró la ejecución.

Primero, veremos cómo varía el tiempo de ejecución si hacemos cambios en el tiempo de T1 siendo esta la que modela la llegada de los clientes a la agencia de vuelo.

	T1 [s]	T4 [s]	T5 [s]	T8 [s]	T9 [s]	T10 [s]	Tiempo Total si varia T1[s]
1	0,01	0,2	0,2	0,07	0,06	0,09	27,249
2	0,02	0,2	0,2	0,07	0,06	0,09	27,330
3	0,03	0,2	0,2	0,07	0,06	0,09	27,440
Tiempo Inicial	0,04	0,2	0,2	0,07	0,06	0,09	27,442
5	0,05	0,2	0,2	0,07	0,06	0,09	27,446
6	0,06	0,2	0,2	0,07	0,06	0,09	27,440
7	0,07	0,2	0,2	0,07	0,06	0,09	27,267

Seguimos haciendo variar el tiempo de las transiciones T4 y T5 las cuales representan el tiempo que demora un cliente en ser atendido por cualquiera de los agentes.

	T1 [s]	T4 [s]	T5 [s]	T8 [s]	T9 [s]	T10 [s]	Tiempo Total si varía T4 y T5[s]
1	0,04	0,14	0,14	0,07	0,06	0,09	25,541
2	0,04	0,16	0,16	0,07	0,06	0,09	26,443
3	0,04	0,18	0,18	0,07	0,06	0,09	26,815
Tiempo Inicial	0,04	0,2	0,2	0,07	0,06	0,09	27,442
5	0,04	0,22	0,22	0,07	0,06	0,09	26,644
6	0,04	0,24	0,24	0,07	0,06	0,09	26,333
7	0,04	0,26	0,26	0,07	0,06	0,09	27,154

Se hace variar el tiempo de la transición T8 la que modela a un cliente que termina de ser atendido luego de que su reserva se cancele

	T1 [s]	T4 [s]	T5 [s]	T8 [s]	T9 [s]	T10 [s]	Tiempo Total si varía T8[s]
1	0,04	0,2	0,2	0,01	0,06	0,09	21,295
2	0,04	0,2	0,2	0,03	0,06	0,09	23,628
3	0,04	0,2	0,2	0,05	0,06	0,09	25,364
Tiempo Inicial	0,04	0,2	0,2	0,07	0,06	0,09	27,442
5	0,04	0,2	0,2	0,09	0,06	0,09	28,549
6	0,04	0,2	0,2	0,11	0,06	0,09	29,627
7	0,04	0,2	0,2	0,13	0,06	0,09	32,87

Hacemos variar el tiempo de la transición T9 que modela a los clientes a los cuales su reserva fue pagada.

	T1 [s]	T4 [s]	T5 [s]	T8 [s]	T9 [s]	T10 [s]	Tiempo Total si varía T9[s]
1	0,04	0,2	0,2	0,07	0,01	0,09	21,286
2	0,04	0,2	0,2	0,07	0,02	0,09	22,982
3	0,04	0,2	0,2	0,07	0,04	0,09	24,337
Tiempo Inicial	0,04	0,2	0,2	0,07	0,06	0,09	27,442
5	0,04	0,2	0,2	0,07	0,08	0,09	26,907
6	0,04	0,2	0,2	0,07	0,1	0,09	29,964
7	0,04	0,2	0,2	0,07	0,11	0,09	29,559

Por último hacemos variar a la transición T10 que modela a un cliente que termina de ser atendido luego de que su reserva fuera confirmada y pagada.

	T1 [s]	T4 [s]	T5 [s]	T8 [s]	T9 [s]	T10 [s]	Tiempo Total si varía T10[s]
1	0,04	0,2	0,2	0,07	0,06	0,03	21,291
2	0,04	0,2	0,2	0,07	0,06	0,05	23,779
3	0,04	0,2	0,2	0,07	0,06	0,07	25,053
Tiempo Inicial	0,04	0,2	0,2	0,07	0,06	0,09	27,442
5	0,04	0,2	0,2	0,07	0,06	0,11	28,153
6	0,04	0,2	0,2	0,07	0,06	0,13	31,221
7	0,04	0,2	0,2	0,07	0,06	0,15	31,464

Podemos apreciar que la transición T1, que simula la entrada de clientes, es la única que no varía significativamente el tiempo de ejecución esto se puede deber a que el tiempo de entrada es menor al tiempo de atención de los agentes por lo tanto la plaza que simula la sala de espera siempre esta llena y los agentes no deben esperar al cliente.

También se observa que la mayor variación de tiempo se da en las transiciones T8, T9 y T10 esto también se puede deber al bajo tiempo que tardan las transición anteriores, generando que los procesos de cancelación, confirmación y pago sean el cuello de botella del sistema que ralentizan la ejecución.

4.1 Intervalos de disparo $[\alpha_i; \beta_i]$

En nuestra lógica modelada por una red de petri, tenemos algunas transiciones temporales, en las cuales el tiempo se interpreta como un parámetro que determina el tiempo que ha de transcurrir desde que una transición queda sensibilizada hasta que se dispara, la colocación y extracción de token se produce de forma atómica.

Se especifica el tiempo de disparo de una transición mediante la asociación de un intervalo que abarque todas las posibilidades de duración de la actividad.

Esta semántica es llamada de tiempo débil, ya que la transición se debe disparar en un intervalo determinado. Este intervalo tiene un α llamado intervalo de disparo más cercano y un β o intervalo de disparo más lejano.

Decidimos implementar los siguientes intervalos de tiempo

T1 = [8 ; 12]

T4 = [24 ; 29]

T5 = [24 ; 29]

T8 = [16 ; 21]

T9 = [20 ; 25]

T10 = [24 ; 29]

Consideramos T1, que simula el ingreso de un cliente a la sala de espera, como el proceso más rápido por eso decidimos implementar un alfa menor. Mientras que los procesos de atención los consideramos los más tardados. Una vez los clientes atendidos se pasa a los procesos de confirmación o cancelación que consideramos que no deben ser tan tardados.

Por otro lado al Beta decidimos modelarlos de manera no bloqueante a diferencia del alfa para no detener la ejecución del programa y comprobar su buen funcionamiento para cualquier Red de Petri que se quiera modelar.

5. Informe de Código

A continuación se detallan todas las clases de nuestro trabajo y cada uno de sus variables y métodos. Dentro del proyecto tenemos diferentes clases, de las cuales Atencion Agente, cancelación, confirmación y pago, entrada de clientes son nuestras clases vivas ya que hay hilos que ejecutan su método run, además tenemos el hilo principal que ejecuta nuestra clase main y un hilo adicional que ejecuta la clase LOG.

5.1 Clase Alfa Exception

La clase Alfa Exception es una excepción personalizada que se lanza cuando el tiempo transcurrido no cumple con el límite mínimo establecido “alfa” o intervalo de disparo más cercano. Esta excepción se utiliza en el contexto de la clase Alfa Y Beta para indicar que el tiempo registrado es inferior al valor mínimo permitido. Tiene dos constructores, uno en el cual no se le envía un mensaje personalizado y otro en el que si.

5.2 Clase Alfa y Beta

La clase Alfa Y Beta gestiona un sistema de control de tiempos basado en dos límites: un tiempo mínimo alfa o intervalo de disparo más cercano y un tiempo máximo beta o intervalo de disparo más lejano. La clase permite comprobar si un tiempo transcurrido cumple con los valores alfa y beta, lanzando excepciones específicas en caso de incumplimiento.

Esta clase posee como atributos el tiempo actual en milisegundos desde un momento inicial registrado, el alfa y beta del intervalo de disparo, dos valores booleanos que nos dice el estado de habilitación para comprobaciones de tiempo y otro para saber si no tiene alfa y beta una transición.

Tenemos dos métodos constructores, uno en donde si se establecen los alfa y beta, otro donde se establecen como 0. Un método setter para establecer el tiempo actual, métodos getter para obtener el tiempo alfa, beta, actual y el booleano para saber si está habilitado. Por último tenemos el método que verifica si el tiempo transcurrido desde que se sensibilizó una transición entra dentro del intervalo alfa y beta.

5.3 Clase: Atencion Agente

En esta clase se modela la atención de un cliente por un agente de viajes para realizar la gestión del viaje, se implementa la interfaz Runnable. En la misma tenemos el número de agente para saber si es el agente número 1 (superior) o el número 2 (inferior), y por último tenemos acceso a la única instancia de nuestro monitor.

5.3.1 Método Run

Dentro de este método nos fijamos que agente es el que está por atender, según esta información vamos a disparar la transición 2 si el que atiende es el agente 1, una vez finalizada la atención dispara la transición 5. Si el agente que atiende es el 2 se dispara la transición 3, una vez finalizada la atención se dispara la transición 4. Este proceso tiene una duración de 200 milisegundos el cual lo simulamos con un sleep.

5.4 Clase Beta Exception

La clase Beta Exception es una excepción personalizada que se lanza cuando el tiempo transcurrido supera el límite máximo permitido beta o intervalo de disparo más lejano. Esta excepción se utiliza en el contexto de la clase “Alfa Y Beta” para indicar que el tiempo registrado excede el valor máximo permitido.

5.5 Clase: Cancelación

En esta clase se modela el proceso de cancelación de la reserva de viaje, la misma implementa la interfaz Runnable, y tiene acceso a la única instancia de nuestro monitor. El método constructor asigna la instancia del monitor.

5.5.1 Método Run

Dentro del método se dispara en bucle hasta que termine todo el proceso la transición 7 la cual hace que los clientes entren al proceso de cancelación de viaje, este proceso dura 70 milisegundos, luego se dispara la transición 8 la cual hace que los clientes pasen a la etapa previa a que se retiren de la agencia y por último la transición 11 para que se retiren.

5.6 Clase: Confirmación y Pago

En esta clase se modela tanto el proceso de confirmación como el pago de dicha reserva, la misma implementa la interfaz Runnable, y tiene acceso a la única instancia de nuestro monitor. El método constructor asigna la instancia del monitor.

5.6.1 Método Run

Dentro del método se dispara en bucle hasta que termine todo el proceso la transición 6 la cual hace que los clientes entren el proceso de confirmación del viaje, este proceso dura 60 milisegundos, luego se dispara la transición 9 pasando a realizar el proceso de pago que dura 90 milisegundos y finalizando con la transición 10, lo cual lleva al cliente a la plaza 14 de clientes salientes y por último se dispara la transición 11 para que el cliente se retire de la agencia.

5.7 Clase: Entrada de clientes

Esta clase modela la entrada de los clientes a nuestra agencia de viajes, la misma implementa la interfaz runnable y tiene acceso a la única instancia de nuestro monitor. El método constructor asigna la instancia del monitor.

5.7.1 Método Run

Dentro de este método, se dispara la transición 0 la cual hace que los clientes ingresen a la agencia, este proceso demora 40 milisegundos y luego se dispara la transición 1 para que los clientes pasen a la sala de espera de la agencia.

5.8 Clase: Main

Es nuestra clase principal que gestiona la ejecución del programa. Primero se pide por consola un número entero entre 1 y 2 el cual representa la política que implementará el monitor sobre los conflictos estructurales que se generen, siendo 1 la política balanceada donde la cantidad de clientes atendidos por el agente de reservas 1, debe ser equitativo a la cantidad de clientes atendidos por el agente de reservas 2. La cantidad de reservas canceladas debe ser equitativa a la cantidad de reservas confirmadas.

Y 2 una política de procesamiento priorizada en la cual Se debe priorizar al agente de reservas 1 con el 75% de las reservas deben ser creadas por dicho agente y se debe priorizar la confirmación de reservas, obteniendo al finalizar, un 80% del total de reservas confirmadas.

Se crea la matriz de incidencia, la cual nos muestra cual es la relación entre plaza, transiciones, entrada y salida de token. Y se inicializa el marcado inicial de la red de petri. Creamos la instancia de nuestro monitor al cual le pasamos el marcado inicial, la matriz de incidencia, la política y los alfa y betas.

Se inicializa tanto la fábrica de hilos y un ArrayList de hilos y se agregan hilos en los cuales se le asignan diferentes procesos, los cuales son: uno por cada agente (1 y 2), uno proceso de cancelación, uno para la confirmación y pago, 5 para la generación y entrada de clientes, y uno para el log. Y luego se inicializan todos los hilos.

5.9 Clase: Monitor

Esta clase está encargada de la centralización de la concurrencia. Los hilos ingresan al monitor, corroboran si las condiciones para realizar su proceso están dadas y salen del mismo para hacer su trabajo o se duermen e ingresan a una cola de espera para que no se produzca un deadlock. No es una clase viva, los que trabajan, duermen y señalan son los mismos hilos.

Dentro de las variables del monitor encontramos: la instancia del monitor ya que usamos un patrón de diseño singleton para que nuestras variables compartidas sean únicas y se use una sola instancia de esta clase, un semáforo para la exclusión mutua a la hora de ingresar al monitor, un arraylist de los diferentes locks, la matriz incidencia de la red, el marcado actual, la política, la secuencia de disparo, un booleano para saber si termino, un arraylist con los alfa y beta, y el número máximo de clientes por atender.

En el constructor recibimos el marcado inicial, la matriz de incidencia, la política y la lista con los alfa y beta. Se realizan comprobaciones para ver si lo que recibimos es correcto, y luego se asignan los valores recibidos a nuestras variables locales.

5.9.1 Método GetInstance

Este método público y estático nos devuelve la única instancia de nuestro monitor, es necesaria por el patrón de diseño utilizado en este sistema.

5.9.2 Método Generar Llaves

Este método privado agrega a nuestro arraylist de llaves, objetos de la clase Locks, cada uno con un nombre específico.

5.9.3 Métodos Getters

Dentro del listado de métodos para obtener información sobre variables de nuestro monitor tenemos: Obtener marcado (devuelve el marcado actual), obtener secuencia (devuelve la secuencia de disparo), obtener beta errores (devuelve los errores en los cuales se sobre paso el intervalo de disparo más lejano), obtener matriz de incidencia (devuelve la matriz de incidencia), Término (devuelve el booleano para saber si finalizó la ejecución)

5.9.4 Método Fire Transition

Método público que recibe como parámetro el número de transición a disparar. El hilo que ejecute este método,va a ejecutar en bucle hasta que termine el programa lo siguiente: toma el mutex para poder ingresar al monitor, luego verifica si la transición que quiere disparar está sensibilizada, si lo está se verifica si es una transición temporal, luego se comprueba si el tiempo de sensibilizado es el correcto, si el tiempo es menor a alfa, se tira una excepción y se devuelve el mutex para que pueda ingresar otro hilo.

Si las comprobaciones de sensibilizado y de tiempo son positivas, se continúa llamando al método para disparar transición, se devuelve el mutex y se sale del bucle.

5.9.5 Metodo Dormir Hilo

En este método privado entran todos aquellos hilos que quisieron disparar una transición que no estaba sensibilizada o no se cumplió el tiempo de sensibilizado. Primero se devuelve el mutex y vuelve a la cola de entrada siguiendo con nuestra política de señalizado (signal and continue), se lo hace dormir al hilo con un wait.

5.9.6 Metodo Disparar

Este método privado recibe por parámetro el número de transición que vamos a disparar. Primero se simula la salida de los clientes si la transición que se quiere disparar es la T11. Luego se genera el nuevo marcado de la red, se despiertan todos los hilos que estaban dormidos y se comprueba si terminó la ejecución.

5.9.7 Método Comprobar término

En este método privado se comprueba que no queden transiciones para disparar, y se notifica todos los hilos.

5.9.8 Método Despertar Hilo

En este método privado se genera una lista con las transiciones sensibilizadas. Luego se verifica si dos transiciones que estén sensibilizadas están en conflicto, se llama a la política para saber cual se tiene que disparar y se despierta a ese hilo.

5.9.9 Método Comparten Lugares de entrada

En este método privado se comprueba si dos transiciones comparten lugares de entrada viendo si en la matriz de incidencia ambas transiciones le sacan un token a una misma plaza, si esto sucede devuelve true, caso contrario false.

5.9.10 Método Nuevo Marcado

En este método privado recibe por parámetro el número de la transición que queremos disparar, se va a generar el nuevo marcado de la red que representa el nuevo estado de la misma, se va a calcular utilizando la ecuación fundamental, se crea un vector de disparo poniendo un 1 en la posición de la transición que se quiere disparar, se hace el producto matricial entre la matriz de incidencia y el vector de disparo, por último se suma el marcado actual al resultado del producto y obtenemos el nuevo marcado.

5.9.11 Método Sensibilizado

En este método privado recibe por parámetro el número de transición que queremos disparar, se llama al nuevo marcado que se generaría con el disparo de esa transición y se verifica que en todo el nuevo marcado no encuentra un número negativo, si sucede ese la transición no esta sensibilizada y se devuelve un false, caso contrario si lo está y se devuelve un true.

5.10 Interface: Monitor Interface

Es una interface que nos da mayor capacidad de abstracción, posee el método abstracto fire Transition() el cual implementamos en nuestro monitor siendo este el único método público de nuestro monitor teniendo un único punto de acceso, centralizando nuestros recursos compartidos, la concurrencia y la toma de decisiones viendo cual hilo se dispara dependiendo de su sensibilizado.

5.11 Enumerado: Número de Agente

Este enumerado es una variable en la clase Atencion de Agente, la cual nos permite definir el comportamiento de los agentes según se defina como agente uno (superior) o agente dos (Inferior). Usar enumerados tiene muchas ventajas, como la facilidad de mantención cuando crece nuestro software.

5.12 Our Thread Factory

Este patrón de diseño sirve para centralizar la creación de hilos en nuestro código y lleva registros y estadísticas de estos

5.13 Clase: Política

Interfaz que define el comportamiento de una política para decidir entre transiciones en un modelo de red de Petri. Las clases que implementen esta interfaz deberán proporcionar una implementación específica para seleccionar transiciones basadas en diferentes criterios.

Los métodos que deberán sobre escribir las clases que implementen esta interfaz son: set política el cual establece la política a utilizar según el número de política especificado y llamada a política el cual selecciona una transición entre dos opciones basándose en la política definida.

5.14 Clase Política Agencia Vuelo

Clase que implementa la interfaz Política para manejar la selección de transiciones en un modelo de red de Petri, con dos políticas posibles: balanceada y priorizada. Primero se define el enumerado de las políticas posibles.

Luego, dentro de nuestra clase, tenemos el identificador de la política a emplear y contadores para saber cuántas veces se ejecutan las transiciones en conflicto dentro de la lógica de nuestra red de petri.

En nuestro constructor pedimos el número de política a implementar y se restablecen los contadores. En el método set política, que se sobre escribe de la interfaz política, se verifica que el código de la política es válido, y se establece. Dentro del método llamado política, se actualizan los contadores y se llama a los métodos que correspondan según la política implementada.

Si la política a implementar es la priorizada, se corrobora en qué conflicto se llamó, si es en T2 y T3, se calcula el porcentaje de veces que se disparó T2, si es menor que el 75% se le da prioridad a esa transición, sino a T3. Por otro lado si se la llama en el conflicto entre T6 y T7, se calcula el porcentaje de veces que se disparó T6, si es menor al 80% se le da prioridad a esa transición, sino a T7.

Dentro del método política balanceada, solo se comprueba cual es la transición de los dos conflictos, con menor cantidad de disparos y esa se ejecuta, ya que tenemos que disparar 50% ambas transiciones en conflicto.

5.15 Clase Política Inexistente Exception

Excepción personalizada que se lanza cuando se intenta utilizar una política que no existe. Extiende la clase Exception, lo que la convierte en una excepción verificable. Esta excepción se utiliza para manejar casos en los que se ingresa un valor inválido para una política, indicando que la política solicitada no está definida o no es válida.

6. Conclusión

En este trabajo pudimos comprobar las ventajas de modelar con redes de Petri, ya que nos permite tener herramientas matemáticas que nos aseguran que si implementamos bien estas redes, no tendremos problemas por condiciones de carrera. Por otra parte el uso del monitor, nos ayuda a centralizar todos nuestros problemas en una sola clase. Y a través de un lenguaje poder comprobar de manera sencilla si existió algún error inesperado con el uso de expresiones regulares.