UNIVERSIDAD NACIONAL DE CÓRDOBA

FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES

CÁTEDRA DE PROGRAMACIÓN CONCURRENTE

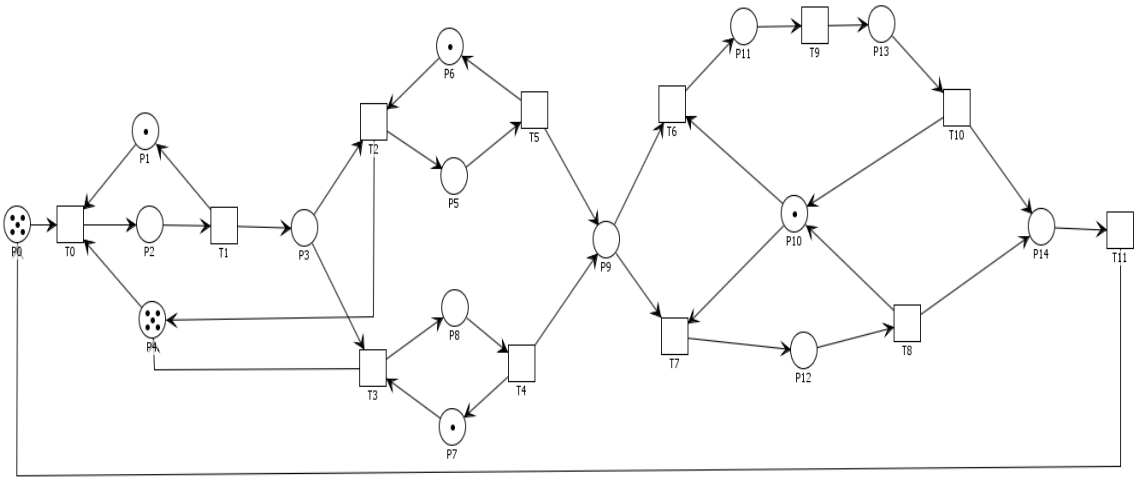
TRABAJO PRÁCTICO Nº 2

“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. La interpretación de la red de petri es la siguiente:

- Las plazas $\{P_1, P_4, P_6, P_7, P_{10}\}$ representan recursos compartidos en el sistema. La puerta de entrada (P_1), los lugares disponibles (P_4), los agentes superior e inferior (P_6, P_7), y el agente para la última parte del proceso de la agencia (P_{10}).
- 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 cuántos tokens 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.

- $T_0 \rightarrow T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_7 \rightarrow T_8 \rightarrow T_{11}$: Representa a un cliente que ingresa a la agencia, es atendido por el agente inferior, su reserva se cancela y sale de la agencia.
- $T_0 \rightarrow T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_6 \rightarrow T_9 \rightarrow T_{10} \rightarrow T_{11}$: Representa a un cliente que ingresa a la agencia, es atendido por el agente inferior, su reserva se confirma y paga, y sale de la agencia.
- $T_0 \rightarrow T_1 \rightarrow T_2 \rightarrow T_5 \rightarrow T_7 \rightarrow T_8 \rightarrow T_{11}$: Representa a un cliente que ingresa a la agencia, es atendido por el agente superior, su reserva es cancelada y sale de la agencia.
- $T_0 \rightarrow T_1 \rightarrow T_2 \rightarrow T_5 \rightarrow T_6 \rightarrow T_9 \rightarrow T_{10} \rightarrow T_{11}$: Representa a un cliente que ingresa a la agencia, es atendido por el agente superior, su reserva es confirmada y pagada, y sale de la agencia.

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

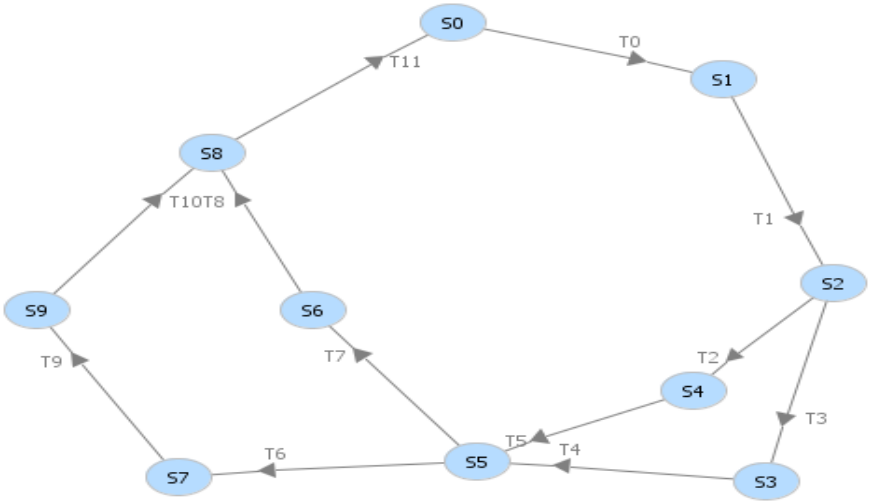
- $m(P_1) + m(P_2) = 1$: Nos indica la cantidad de gente que ingresa a la vez a la agencia.
- $m(P_2) + m(P_3) + m(P_4) = 5$: Representa que solo pueden haber 5 clientes como máximo, esperando por ser atendidos
- $m(P_5) + m(P_6) = 1$: Representa que el agente superior puede estar solo en dos estados, disponible para atender o atendiendo
- $m(P_7) + m(P_8) = 1$: Representa que el agente inferior puede estar solo en dos estados, disponible para atender o atendiendo

- $m(P_{10}) + m(P_{11}) + m(P_{12}) + m(P_{13}) = 1$: Representa que no puede haber clientes confirmando y pagando o cancelando al mismo tiempo.
- $m(P_0) + m(P_2) + m(P_3) + m(P_5) + m(P_8) + m(P_9) + m(P_{11}) + m(P_{12}) + m(P_{13}) + m(P_{14}) = N$ Nos indica el número máximo de clientes que vamos a atender es “N”

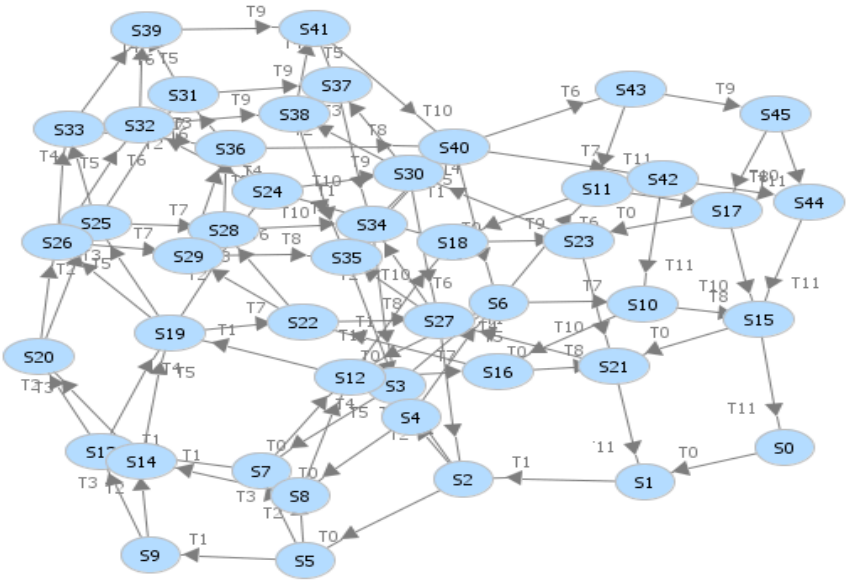
1.4 Grafo de alcanzabilidad y deadlock:

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. Al momento de ejecutar nuestro sistema la plaza P_0 tendrá un valor de 186.

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:

- $IT_1 = \{T_0, T_1, T_3, T_4, T_7, T_8, T_{11}\}$
- $IT_2 = \{T_0, T_1, T_3, T_4, T_6, T_9, T_{10}, T_{11}\}$
- $IT_3 = \{T_0, T_1, T_2, T_5, T_7, T_8, T_{11}\}$
- $IT_4 = \{T_0, T_1, T_2, T_5, T_6, T_9, T_{10}, T_{11}\}$

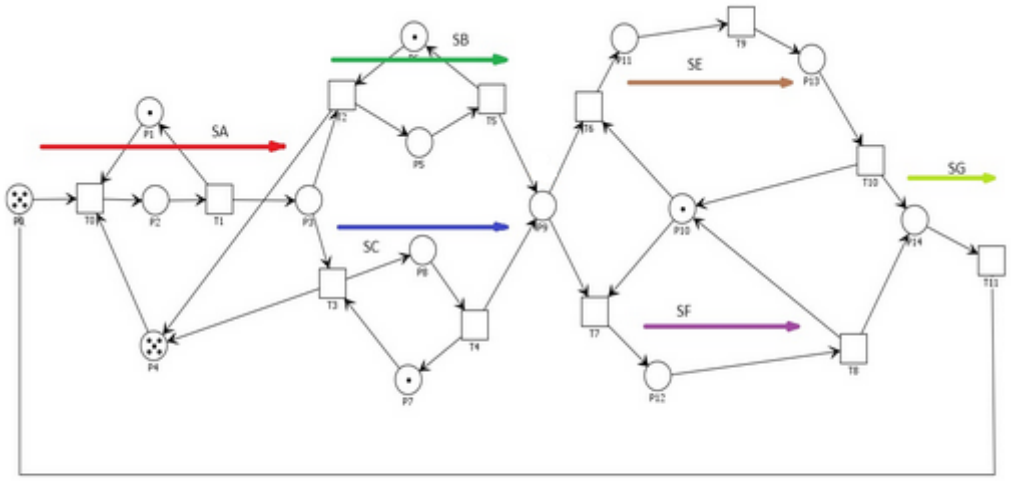
2.2 Obtenemos el conjunto de plazas PI de cada IT:

- $PI_1 = \{P_0, P_1, P_2, P_3, P_4, P_7, P_8, P_9, P_{12}, P_{10}, P_{14}\}$
- $PI_2 = \{P_0, P_1, P_2, P_3, P_4, P_7, P_8, P_9, P_{11}, P_{13}, P_{10}, P_{14}\}$
- $PI_3 = \{P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_9, P_{12}, P_{10}, P_{14}\}$
- $PI_4 = \{P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_9, P_{11}, P_{13}, P_{10}, P_{14}\}$

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 o de clientes siendo atendidos al mismo tiempo. 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 siguen los 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:

- $PS_A = \{P_2, P_3\}\ Max(MS_A) = 5$
- $PS_B = \{P_5\}\ Max(MS_B) = 1$
- $PS_C = \{P_8\}\ Max(MS_C) = 1$
- $PS_E = \{P_{11}, P_{13}\}\ Max(MS_E) = 1$

- $PS_F = \{P_{12}\}\ \ Max(MS_F) = 1$
- $PS_G = \{P_{14}\}\ \ Max(MS_G) = 5$

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									
Politica	**Ejec**	**IT 1**	**%**	**IT 2**	**%**	**IT3**	**%**	**IT 4**	**%**
1	1	43	23%	35	19%	51	27%	57	31%
	2	43	23%	57	31%	45	24%	41	22%

	3	49	26%	40	22%	47	25%	50	27%
	4	46	25%	37	20%	54	29%	49	26%
	5	38	20%	59	32%	41	22%	48	26%
	6	45	24%	52	28%	47	25%	42	23%
	7	40	22%	51	27%	54	29%	41	22%
	8	51	27%	47	25%	42	23%	46	25%
	9	43	23%	47	25%	45	24%	51	27%
	10	48	26%	43	23%	43	23%	52	28%
	Total	446	24,0%	468	25,2%	469	25,2%	477	25,6%
2	1	12	6%	42	23%	30	16%	102	55%
	2	8	4%	36	19%	28	15%	114	61%
	3	8	4%	37	20%	24	13%	117	63%
	4	6	3%	37	20%	34	18%	109	59%
	5	4	2%	39	21%	38	20%	105	56%
	6	5	3%	39	21%	22	12%	120	65%
	7	5	3%	39	21%	40	22%	102	55%
	8	10	5%	41	22%	31	17%	104	56%
	9	16	9%	37	20%	30	16%	103	55%
	10	6	3%	37	20%	22	12%	121	65%
	Total	80	4,3%	384	20,6%	299	16,1%	1097	59,0%

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 49,2% de los clientes, luego sumando los porcentajes de los invariantes IT3 e IT4 sabemos que el agente superior atiende al 50,8% de los clientes. Por otro lado, sumando los IT1 e IT3 vemos que el 49,2% de las reservaciones son canceladas y sumando IT2 y IT4 el 50,8% son confirmadas y pagadas.

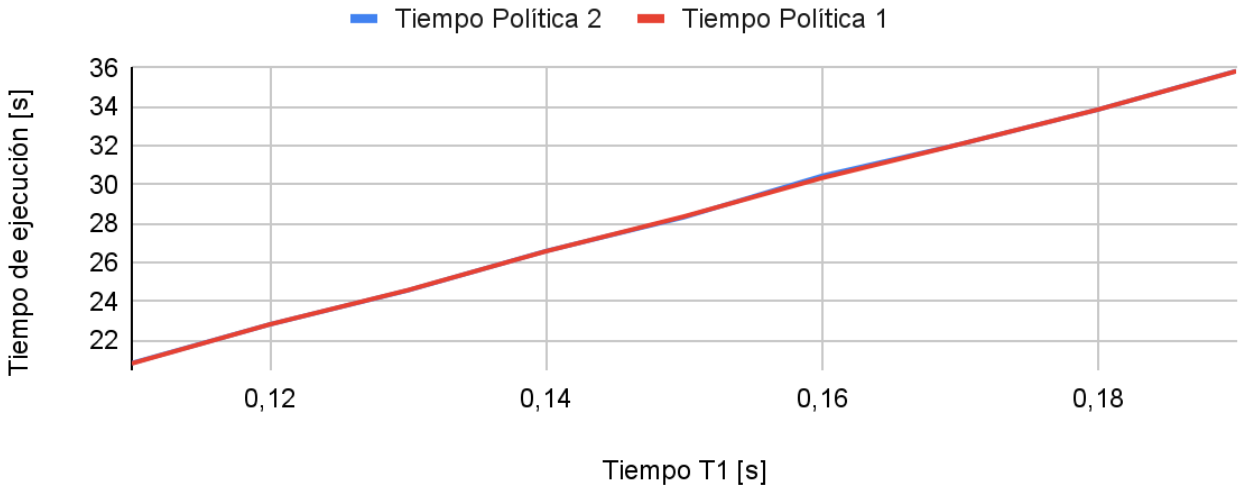
Al ejecutar la política de procesamiento priorizado (2), vemos que si sumamos los porcentajes de los invariantes 3 y 4 obtenemos que el 75,1% de los clientes son atendidos por el agente superior y si sumamos los invariantes 2 y 4 obtenemos que un 79,6% de las reservas son confirmadas y pagadas. Hay que 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.

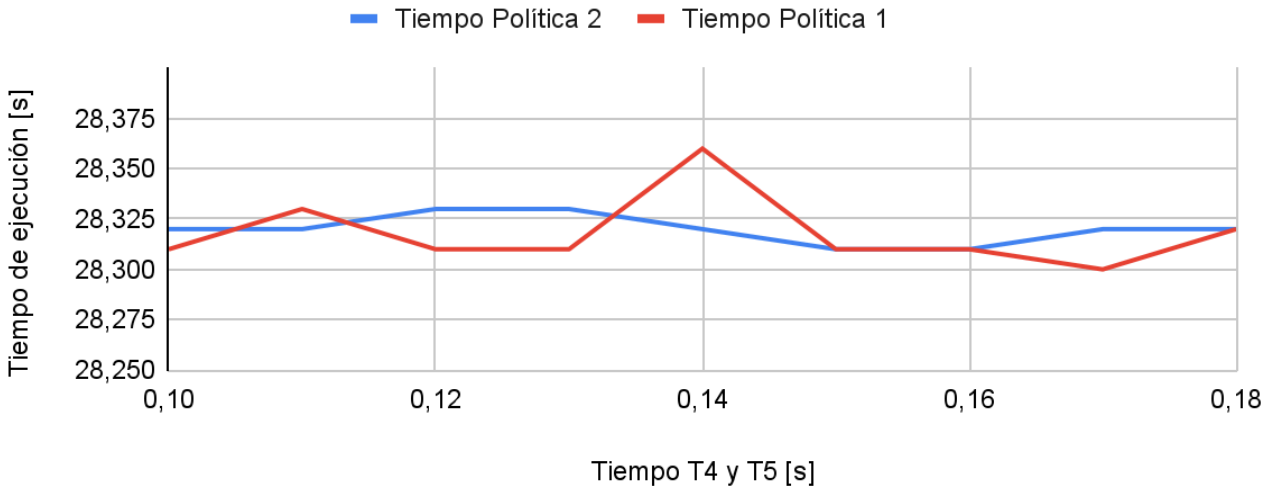
	T1 [s]	T4 [s]	T5 [s]	T8 [s]	T9 [s]	T10 [s]	Tiempo Política 2	Tiempo Política 1
1	0,11	0,14	0,14	0,09	0,04	0,07	20,82	20,8
2	0,12	0,14	0,14	0,09	0,04	0,07	22,81	22,81
3	0,13	0,14	0,14	0,09	0,04	0,07	24,56	24,57
4	0,14	0,14	0,14	0,09	0,04	0,07	26,58	26,57
T inicial	0,15	0,14	0,14	0,09	0,04	0,07	28,320	28,360
5	0,16	0,14	0,14	0,09	0,04	0,07	30,45	30,34
6	0,17	0,14	0,14	0,09	0,04	0,07	32,08	32,08
7	0,18	0,14	0,14	0,09	0,04	0,07	33,86	33,86
8	0,19	0,14	0,14	0,09	0,04	0,07	35,84	35,83



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.

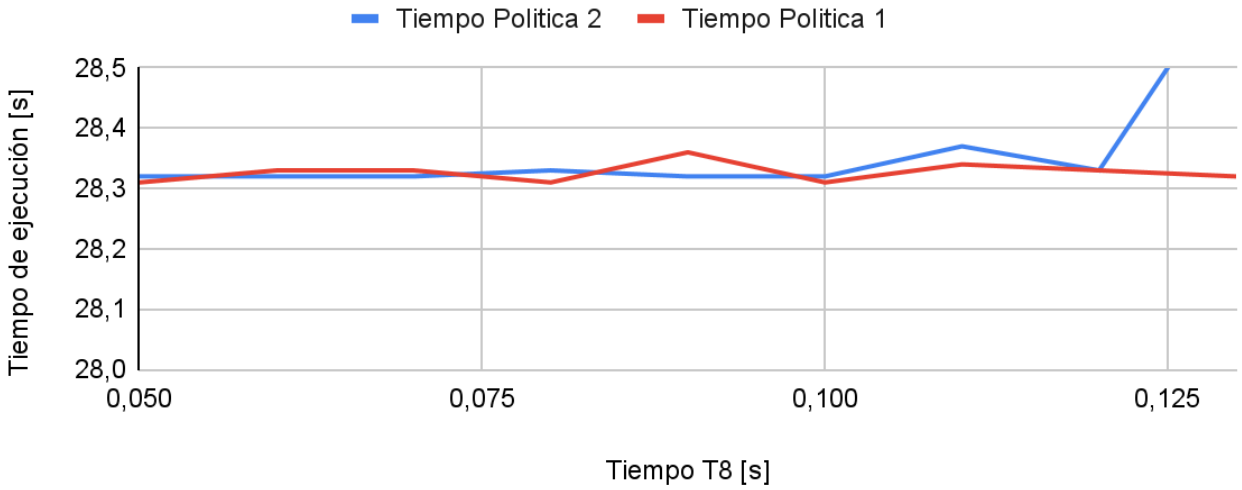
	T1 [s]	T4 [s]	T5 [s]	T8 [s]	T9 [s]	T10 [s]	Tiempo Política 2	Tiempo Política 1
1	0,15	0,1	0,1	0,09	0,04	0,07	28,32	28,31
2	0,15	0,11	0,11	0,09	0,04	0,07	28,32	28,33
3	0,15	0,12	0,12	0,09	0,04	0,07	28,33	28,31
4	0,15	0,13	0,13	0,09	0,04	0,07	28,33	28,31

T inicial	0,15	0,14	0,14	0,09	0,04	0,07	28,32	28,36
5	0,15	0,15	0,15	0,09	0,04	0,07	28,31	28,31
6	0,15	0,16	0,16	0,09	0,04	0,07	28,31	28,31
7	0,15	0,17	0,17	0,09	0,04	0,07	28,32	28,3
8	0,15	0,18	0,18	0,09	0,04	0,07	28,32	28,32



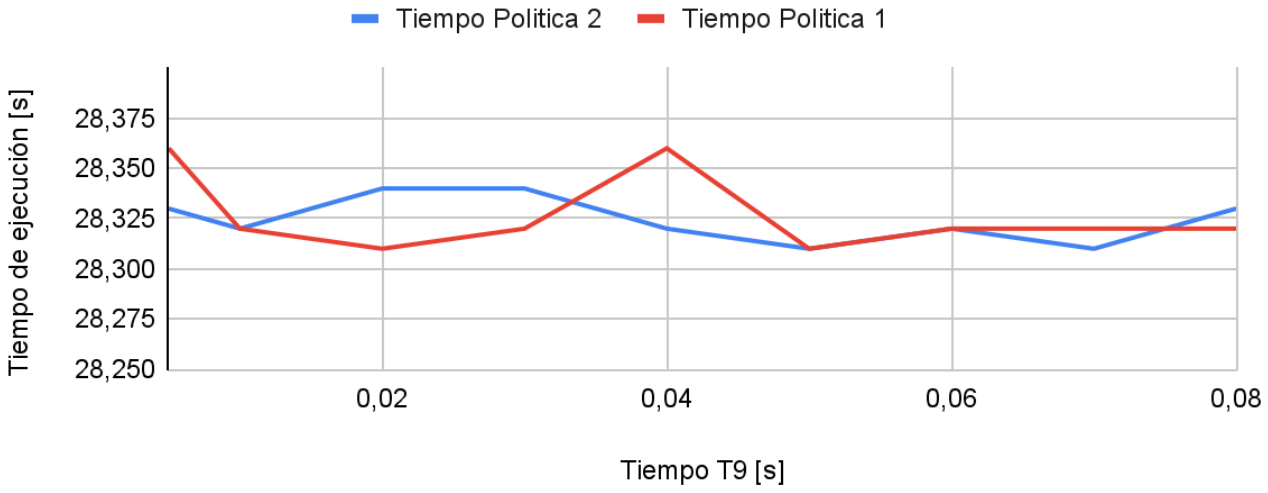
Se hace variar el tiempo de la transición T8 la que modela a un cliente que termina de ser atendido luego de que su reserva se cancele:

	T1 [s]	T4 [s]	T5 [s]	T8 [s]	T9 [s]	T10 [s]	Tiempo Politica 2	Tiempo Politica 1
1	0,15	0,14	0,14	0,05	0,04	0,07	28,32	28,31
2	0,15	0,14	0,14	0,06	0,04	0,07	28,32	28,33
3	0,15	0,14	0,14	0,07	0,04	0,07	28,32	28,33
4	0,15	0,14	0,14	0,08	0,04	0,07	28,33	28,31
T inicial	0,15	0,14	0,14	0,09	0,04	0,07	28,320	28,360
5	0,15	0,14	0,14	0,1	0,04	0,07	28,32	28,31
6	0,15	0,14	0,14	0,11	0,04	0,07	28,37	28,34
7	0,15	0,14	0,14	0,12	0,04	0,07	28,33	28,33
8	0,15	0,14	0,14	0,13	0,04	0,07	28,67	28,32



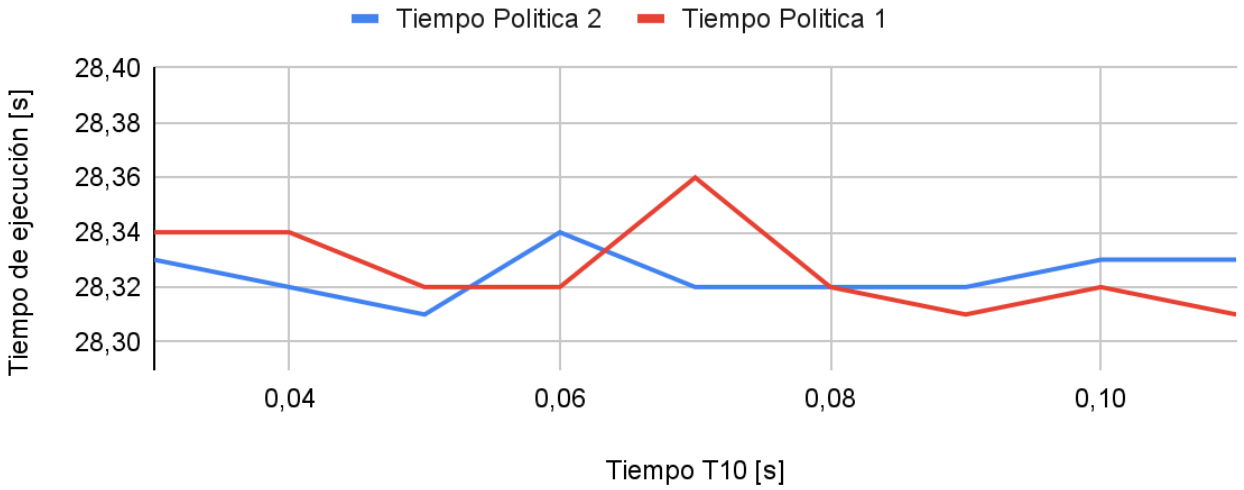
Hacemos variar el tiempo de la transición T9 que modela a los clientes a los cuales su reserva fue pagada:

	T1 [s]	T4 [s]	T5 [s]	T8 [s]	T9 [s]	T10 [s]	Tiempo Politica 2	Tiempo Politica 1
1	0,15	0,14	0,14	0,09	0,005	0,07	28,33	28,36
2	0,15	0,14	0,14	0,09	0,01	0,07	28,32	28,32
3	0,15	0,14	0,14	0,09	0,02	0,07	28,34	28,31
4	0,15	0,14	0,14	0,09	0,03	0,07	28,34	28,32
T inicial	0,15	0,14	0,14	0,09	0,04	0,07	28,320	28,360
5	0,15	0,14	0,14	0,09	0,05	0,07	28,31	28,31
6	0,15	0,14	0,14	0,09	0,06	0,07	28,32	28,32
7	0,15	0,14	0,14	0,09	0,07	0,07	28,31	28,32
8	0,15	0,14	0,14	0,09	0,08	0,07	28,33	28,32



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 Politica 2	Tiempo Politica 1
1	0,15	0,14	0,14	0,09	0,04	0,03	28,33	28,34
2	0,15	0,14	0,14	0,09	0,04	0,04	28,32	28,34
3	0,15	0,14	0,14	0,09	0,04	0,05	28,31	28,32
4	0,15	0,14	0,14	0,09	0,04	0,06	28,34	28,32
T inicial	0,15	0,14	0,14	0,09	0,04	0,07	28,320	28,360
5	0,15	0,14	0,14	0,09	0,04	0,08	28,32	28,32
6	0,15	0,14	0,14	0,09	0,04	0,09	28,32	28,31
7	0,15	0,14	0,14	0,09	0,04	0,1	28,33	28,32
8	0,15	0,14	0,14	0,09	0,04	0,11	28,33	28,31



Del análisis temporal se concluye que el tiempo total de ejecución del sistema es altamente sensible a variaciones en la transición T1, la cual se ejecuta en todos los invariantes de transición. En contraste, las transiciones T4, T5, T8, T9 y T10 no presentan impacto significativo en el tiempo total, lo que indica que su ejecución se encuentra solapada por concurrencia o restringida a ramas no críticas del modelo. Esto demuestra que el sistema es robusto frente a variaciones en los tiempos de atención y confirmación, pero depende fuertemente de la transición.

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 ; 151]
- T4 = [24 ; 141]
- T5 = [24 ; 141]
- T8 = [16 ; 90]
- T9 = [20 ; 40]
- T10 = [24 ; 70]

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.

4.2 Análisis secuencial

Considerando el peor caso temporal y suponiendo que las transiciones se disparan tan pronto como están habilitadas, inicialmente se disparan las transiciones T0 y T1, siendo el tiempo de ejecución de T1 igual a 151 unidades de tiempo.

A continuación se dispara T2, lo que vuelve a habilitar T0 y T1. Luego de 141 unidades de tiempo se dispara la transición T5, mientras que a T1 aún le restan 10 unidades de tiempo para completar su ejecución.

Durante este intervalo se ejecuta el bloque formado por las transiciones T6, T9 y T10, cuyo tiempo total es de 130 unidades de tiempo. Mientras dicho bloque se encuentra en ejecución, se completan los 10 segundos restantes de T1, se dispara la transición T3 y se cumplen 120 unidades de tiempo correspondientes a la espera de T4.

Una vez transcurridas las 21 unidades de tiempo restantes necesarias para completar la ejecución de T4, se vuelve a ejecutar el bloque T6, T9 y T10, con una duración adicional de 130 unidades de tiempo.

Por lo tanto, el tiempo total de ejecución del sistema en este escenario resulta: 151+141+130+21+130=573

En un segundo escenario Inicialmente se disparan las transiciones T0 y T1, siendo el tiempo de ejecución de T1 igual a 151 unidades de tiempo. A continuación se dispara la transición T2, lo que vuelve a habilitar T0 y T1. En este punto, el sistema debe esperar 24 unidades de tiempo correspondientes a la transición T5 y 151 unidades de tiempo asociadas a T1.

Transcurridas las 24 unidades de tiempo, se dispara la transición T5, quedando aún 127 unidades de tiempo para que finalice la ejecución de T1. Posteriormente se dispara la transición

T6, la cual requiere la finalización del bloque compuesto por T9 y T10, cuyo tiempo total es de 130 unidades de tiempo.

Durante este intervalo, luego de 10 unidades de tiempo, finaliza la ejecución de T1 y se dispara la transición T3. En ese instante, de las 127 unidades de tiempo restantes, solo 3 corresponden al bloque T6–T9–T10; sin embargo, el sistema debe esperar 141 unidades de tiempo para que finalice la ejecución de la transición T4.

Una vez cumplidas dichas 141 unidades de tiempo, las transiciones T9 y T10 ya se habían ejecutado previamente, por lo que deben ejecutarse nuevamente. En consecuencia, el tiempo total de ejecución del sistema en este escenario resulta: 151+151+141+130=573.

A partir de ambos escenarios se observa que, aun cuando la transición T5 se ejecuta en su tiempo mínimo (α), el tiempo total de ejecución del sistema coincide con el obtenido cuando todas las transiciones operan en su peor caso.

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 Monitor

La clase Monitor representa el componente central del sistema de control de concurrencia, implementando el patrón Singleton para garantizar una única instancia que coordine la ejecución de todas las transiciones de la red de Petri. Esta clase es responsable de gestionar la exclusión mutua mediante un semáforo que controla el acceso al recurso compartido, garantizando que solamente un hilo pueda evaluar y disparar transiciones en un momento determinado. El Monitor mantiene una referencia a la red de Petri que supervisa y almacena un mapa de llaves de sincronización, donde cada transición tiene asociado un objeto que funciona como variable de condición para implementar mecanismos de espera y notificación entre hilos.

El método principal fireTransition implementa toda la lógica de disparo de transiciones, donde primero se adquiere el mutex para entrar en la sección crítica. Una vez dentro, verifica las restricciones temporales alfa y beta asociadas a cada transición mediante el sistema de AlfaYBeta. Si una transición no ha alcanzado su tiempo mínimo alfa, el hilo libera el mutex y se duerme en su variable de condición específica hasta que transcurra el tiempo necesario. Cuando se excede el tiempo máximo de beta, el sistema registra el error pero permite que la transición continúe para evitar deadlocks.

Después de disparar exitosamente una transición, el Monitor actualiza los tiempos alfa y beta de todas las transiciones que acaban de sensibilizarse como consecuencia del disparo, iniciando sus ventanas temporales. Luego ejecuta la lógica de despertar hilos, consultando primero a la red de Petri si existe alguna situación de conflicto estructural entre transiciones sensibilizadas. Si existe conflicto, utiliza la política configurada para decidir cuál transición debe tener prioridad y despierta únicamente al hilo correspondiente. Si no hay conflictos, despierta a todos los hilos asociados a transiciones sensibilizadas mediante sus respectivas llaves de sincronización. El método dormir Hilo encapsula la lógica de liberar el mutex antes de esperar y el método get Llave automatiza la creación lazy de objetos de sincronización para cada transición, asegurando que cada una tenga su propia cola de espera.

5.2 Red De Petri

La clase Red De Petri encapsula la representación y el comportamiento dinámico de una red de Petri, actuando como el modelo matemático que define el sistema de estados y transiciones del proceso de atención de la agencia de vuelos. Esta clase mantiene el estado actual del sistema mediante un vector de marcado que representa la cantidad de tokens en cada plaza de la red, y una matriz de incidencia que define las relaciones estructurales entre plazas y transiciones, especificando cuántos tokens consume o produce cada transición en cada plaza. La red mantiene también una secuencia de registro donde se acumulan todas las transiciones disparadas durante la ejecución, permitiendo un análisis posterior del flujo del sistema.

La clase gestiona las restricciones temporales mediante una lista de objetos Alfa y Beta, uno por cada transición, que controlan los límites de tiempo mínimo y máximo en los que una transición puede ser disparada después de habilitarse. El método sensibilizado implementa la ecuación fundamental de las redes de Petri, calculando si una transición específica puede dispararse según el marcado actual. Para ello, aplica la columna correspondiente de la matriz de incidencia al marcado actual y verifica que el resultado no contenga valores negativos, lo que indicaría insuficiencia de tokens en alguna plaza de entrada. El método nuevoMarcado realiza el cálculo matemático del próximo estado aplicando la ecuación de estado M' = M + I * S, donde M es el marcado actual, I es la matriz de incidencia y S es el vector de disparo con un único 1 en la posición de la transición a disparar.

El método dispararTransicion ejecuta el cambio de estado efectivo, primero verificando que la transición esté sensibilizada, luego actualizando el marcado con el nuevo estado calculado y registrando la transición en la secuencia. Implementa un tratamiento especial para la transición T11 que representa la salida de clientes del sistema, donde en lugar de modificar el marcado, simplemente incrementa un contador interno que permite determinar cuándo se ha completado el proceso con todos los clientes. Este mecanismo de terminación verifica si se alcanzó el número máximo de clientes procesados o si no quedan transiciones sensibilizadas en el sistema.

La clase proporciona métodos para detectar conflictos estructurales, donde el método compartenLugaresDeEntrada determina si dos transiciones compiten por los mismos recursos al tener arcos de entrada desde las mismas plazas. El método verificarConflicto analiza el conjunto de transiciones sensibilizadas en busca de pares que comparten lugares de entrada, y cuando detecta un conflicto, delega en la política configurada la decisión de cuál transición debe tener prioridad de disparo. Esta arquitectura permite que el sistema maneje situaciones de competencia por recursos mediante estrategias configurables.

5.3 AlfaYBeta

La clase AlfaYBeta implementa el mecanismo de restricciones temporales que controla las ventanas de tiempo en las que una transición puede ser disparada después de habilitarse. Cada instancia de esta clase representa las restricciones temporales asociadas a una transición específica, definiendo un tiempo mínimo alfa que debe transcurrir antes de que la transición pueda dispararse, y un tiempo máximo beta que no debe excederse sin que la transición se dispare. Esta clase utiliza una marca temporal de inicio que se registra en milisegundos cuando la transición se sensibiliza por primera vez, permitiendo calcular el tiempo transcurrido en verificaciones posteriores.

El método verificar implementa la lógica de evaluación de ventanas temporales, retornando un estado ALFA cuando no ha transcurrido suficiente tiempo desde la sensibilización, lo que indica que el hilo solicitante debe bloquearse. Retorna BETA cuando se ha excedido el tiempo máximo, situación que se registra pero que permite el disparo para evitar interbloqueos. Retorna OK cuando el tiempo transcurrido está dentro de la ventana válida entre alfa y beta. La clase soporta también transiciones sin restricciones temporales mediante una bandera sinRestriccion que hace que verificar siempre retorna OK, permitiendo que algunas transiciones se disparan inmediatamente cuando están sensibilizadas. El método iniciar establece el punto de partida temporal cuando una transición se sensibiliza, y getTiempoExcedido calcula cuánto tiempo se superó el límite beta para propósitos de logging y análisis de rendimiento.

5.4 Log

La clase Log implementa la funcionalidad de auditoría y registro del sistema, ejecutándose como un hilo independiente que monitorea periódicamente el estado del sistema y registra información estadística sobre la ejecución. Esta clase mantiene una referencia al tiempo de inicio del proceso y tiene acceso a la red de Petri para consultar la secuencia de transiciones disparadas. Utiliza un FileWriter estático inicializado en un bloque static para crear y mantener abierto el archivo de log durante toda la ejecución del programa, con autoflush activado para garantizar que los datos se escriban inmediatamente al disco.

El método run implementa un ciclo continuo que monitorea constantemente si el proceso ha terminado, y cuando detecta la finalización, calcula el tiempo total transcurrido y registra estadísticas completas en el archivo de log. El método contarTransiciones utiliza expresiones regulares para analizar la secuencia de transiciones y contar cuántas veces se disparó cada transición específica. Esta clase permite generar reportes post-ejecución que incluyen la secuencia completa de transiciones, tiempos de ejecución y métricas sobre el comportamiento del sistema, proporcionando datos valiosos para análisis de rendimiento y validación del comportamiento del sistema.

5.5 Main

La clase Main es el punto de entrada del programa y orquesta la inicialización y ejecución de todo el sistema concurrente. Define las cantidades que determinan el grado de paralelismo del sistema, especificando cuántos hilos se asignan a cada tipo de tarea: generación de clientes, atención por cada agente, cancelaciones y confirmaciones. Crea una instancia de Configuración Inicial que configura la matriz de incidencia, el marcado inicial, las políticas de resolución de conflictos y las restricciones temporales del sistema. Instancia una pantalla de carga gráfica que proporciona retroalimentación visual del progreso de ejecución.

Utiliza una fábrica de hilos personalizada OurThreadFactory para crear todos los hilos del sistema con nombres descriptivos y capacidades de registro. Crea múltiples hilos para la entrada de clientes, permitiendo que varios clientes lleguen concurrentemente al sistema. Instancia un hilo por cada agente definido en el sistema, diferenciando entre agentes de nivel superior e inferior. Crea hilos dedicados para procesos de cancelación y confirmación-pago, además de un hilo específico para el sistema de logging. Después de inicializar todos los hilos, los arranca y utiliza join para esperar a que todos completen su ejecución antes de finalizar el programa, garantizando una terminación ordenada.

5.6 EntradaDeclientes

La clase EntradaDeClientes modela el proceso de llegada de clientes a la agencia, implementando la interfaz Runnable para ejecutarse en hilos concurrentes. Cada instancia representa un generador de clientes que dispara secuencialmente las transiciones T0 y T1 de la red de Petri, donde T0 representa la entrada del cliente al sistema y T1 representa algún proceso inicial posterior a la entrada. Entre ambos disparos, el hilo se duerme durante 150 milisegundos para simular la duración del proceso de entrada, modelando el tiempo que toma que un cliente ingrese completamente al sistema. El método run ejecuta un ciclo infinito que genera clientes continuamente, disparando el par de transiciones T0-T1 en cada iteración. El ciclo se interrumpe

solamente cuando el monitor indica mediante el retorno false que el sistema ha terminado, ya sea porque se alcanzó el número máximo de clientes o porque no quedan transiciones sensibilizadas.

5.7 AtencionAgente

La clase AtencionAgente modela el comportamiento de los agentes que atienden a los clientes en la agencia, diferenciando entre dos tipos de agentes mediante la enumeración NumeroDeAgente. Cada instancia representa un hilo que ejecuta repetidamente el proceso de atención según el tipo de agente que representa. Los agentes de tipo AGENTE1 (superior) disparan las transiciones T2 y T5, mientras que los agentes de tipo AGENTE2 (inferior) disparan las transiciones T3 y T4, modelando diferentes flujos de trabajo o niveles de autoridad en el proceso de atención. Entre los disparos de transiciones, cada hilo se duerme durante 140 milisegundos para simular la duración del proceso de atención, representando el tiempo que un agente dedica a atender a un cliente. El ciclo de atención se repite continuamente hasta que el monitor señale la terminación del sistema.

5.8 Cancelación

La clase Cancelación modela el flujo de trabajo cuando un cliente decide cancelar su trámite en la agencia. Implementa una secuencia de tres transiciones: T7, T8 y T11, donde T7 representa la decisión de cancelación, T8 representa el proceso administrativo de la cancelación, y T11 representa la salida del cliente del sistema. Entre T7 y T8, el hilo se duerme 90 milisegundos para simular el tiempo que toma procesar una cancelación. Este proceso se ejecuta en un ciclo continuo en un hilo dedicado, compitiendo con el proceso de confirmación y pago por los clientes que han completado la atención con un agente.

5.9 Confirmación Y Pago

La clase Confirmación y Pago representa el flujo alternativo a la cancelación, donde el cliente decide proceder con la confirmación de su reserva y realizar el pago correspondiente. Dispara una secuencia de cuatro transiciones: T6 para la decisión de confirmar, T9 para el proceso de confirmación, T10 para el proceso de pago, y T11 para la salida del cliente del sistema. Introduce dos períodos de espera que modelan las duraciones de los subprocesos: 40 milisegundos para la confirmación y 70 milisegundos para el pago. Esta diferenciación temporal refleja que el proceso de pago generalmente toma más tiempo que la simple confirmación.

5.10 Salida

La clase Salida modela el proceso de egreso de los clientes del sistema una vez finalizado su trámite, ya sea por confirmación o cancelación. Implementa la interfaz Runnable, permitiendo su ejecución concurrente en un hilo independiente que interactúa con el monitor de la Red de Petri.

El comportamiento del proceso consiste en el disparo de la transición T11, la cual representa la salida del cliente del sistema y constituye el cierre de todos los invariantes de transición definidos en el modelo. El hilo permanece activo mientras la transición esté habilitada y se detiene cuando el monitor indica que no pueden realizarse más disparos.

5.11 Configuración Inicial

La clase ConfiguracionInicial encapsula toda la lógica de inicialización del sistema, proporcionando un punto centralizado para definir la configuración de la red de Petri, las políticas y las restricciones temporales. En el método setupMatriz define el marcado inicial con 186 clientes por procesar y la disponibilidad inicial de recursos como agentes y capacidades de procesamiento. Define una matriz de incidencia de 15 plazas por 12 transiciones que codifica la estructura completa de la red de Petri del sistema de atención de la agencia. El método setupAlfaYBeta inicializa las restricciones temporales, creando inicialmente instancias sin restricciones para todas las transiciones, y luego configurando límites específicos para seis transiciones críticas que requieren control temporal. El método setupPolitica selecciona la política número 2, que es la política priorizada. Esta clase coordina la creación de todos los componentes principales del sistema, asegurando que la red de Petri, el monitor y todas las restricciones estén correctamente inicializadas antes de comenzar la simulación.

5.12 Política Agencia Vuelo

La clase Política Agencia Vuelo implementa las estrategias de resolución de conflictos cuando múltiples transiciones sensibilizadas compiten por recursos compartidos. Define dos políticas mediante una enumeración interna: POLITICA_1 (balanceada) y POLITICA_2 (priorizada), cada una con un código identificador único. El método llamadaApolitica recibe dos transiciones en conflicto y delega la decisión a la política configurada. La política priorizada implementa lógica probabilística que favorece ciertas transiciones sobre otras.

5.13 Política

La interfaz Politica define el contrato que deben cumplir todas las implementaciones de políticas de resolución de conflictos en el sistema. Declara el método llamadaApolitica que recibe dos identificadores de transiciones en conflicto y retorna el identificador de la transición seleccionada según los criterios de la política. Incluye también el método setPolitica que permite cambiar dinámicamente la política activa mediante un código numérico, lanzando PoliticaInexistenteException si el código no corresponde a ninguna política válida.

5.14 MonitorInterface

La interfaz MonitorInterface define el contrato básico que expone el Monitor hacia los hilos trabajadores del sistema. Declara únicamente el método fireTransition que recibe el identificador de una transición y retorna un booleano indicando si el disparo fue exitoso. Este valor de retorno permite a los hilos detectar cuando el sistema ha terminado y deben finalizar su ejecución.

5.15 OurThreadFactory

La clase OurThreadFactory implementa la interfaz ThreadFactory del framework de concurrencia de Java, proporcionando un mecanismo personalizado para crear hilos con capacidades extendidas de identificación y auditoría. Mantiene un contador atómico que asigna identificadores únicos a cada hilo creado, garantizando unicidad incluso en creaciones

concurrentes. El método newThread construye hilos con nombres descriptivos que incluyen el identificador único y una representación del Runnable que ejecutarán, facilitando la depuración y el monitoreo.

Cada vez que se crea un hilo, la fábrica registra en una lista sincronizada la información del hilo junto con la marca temporal de creación, permitiendo auditorías posteriores sobre cuándo y en qué orden se crearon los hilos del sistema. Esta capacidad de trazabilidad es valiosa para análisis de rendimiento y depuración de problemas relacionados con la concurrencia, proporcionando un registro completo del ciclo de vida de los hilos desde su creación.

5.16 NumeroDeAgente

La enumeración NumeroDeAgente proporciona una abstracción tipo-segura para representar los diferentes tipos de agentes en el sistema. Define dos constantes AGENTE1 y AGENTE2, cada una asociada con un número entero que identifica el tipo de agente. El método getnumeroDeAgente permite obtener el identificador numérico asociado a cada tipo de agente.

Esta enumeración evita el uso de números mágicos o strings en el código, proporcionando un mecanismo robusto y compilable para diferenciar entre tipos de agentes. Facilita la extensión del sistema a más tipos de agentes y previene errores de asignación de valores inválidos, ya que el compilador verifica que solo se usen valores de la enumeración.

5.17 PoliticaInexistenteException

La clase PoliticaInexistenteException es una excepción personalizada verificada que se lanza cuando se intenta configurar o utilizar una política que no está definida en el sistema. Al extender la clase Exception en lugar de RuntimeException, obliga a los métodos que puedan lanzarla a declararla en su firma o manejarla mediante try-catch, promoviendo un manejo explícito de situaciones de configuración inválida.

Esta excepción se utiliza como mecanismo de validación durante la configuración del sistema, particularmente en el método setPolitica de las implementaciones de Politica, donde se verifica que el código numérico proporcionado corresponda a una política válida. Proporciona un mecanismo robusto para comunicar errores de configuración al código cliente, permitiendo que el sistema rechace configuraciones inválidas en tiempo de inicialización antes de comenzar la simulación.

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.