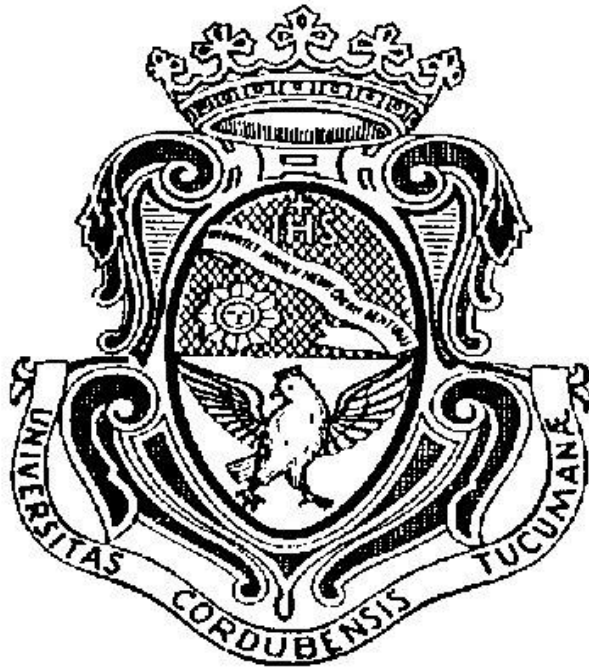


# Universidad Nacional de Córdoba

Facultad de Cs. Exactas, Físicas y Naturales



## Programación Concurrente

Trabajo Práctico Final

### Profesores:

- Ing. Luis Orlando Ventre
- Ing. Mauricio Ludemann

### Alumnos:

- |                             |          |
|-----------------------------|----------|
| - Emanuel Nicolás Rodríguez | 42259187 |
| - Brian Joel Gerard         | 40989936 |
| - Franco Viotti             | 42051491 |

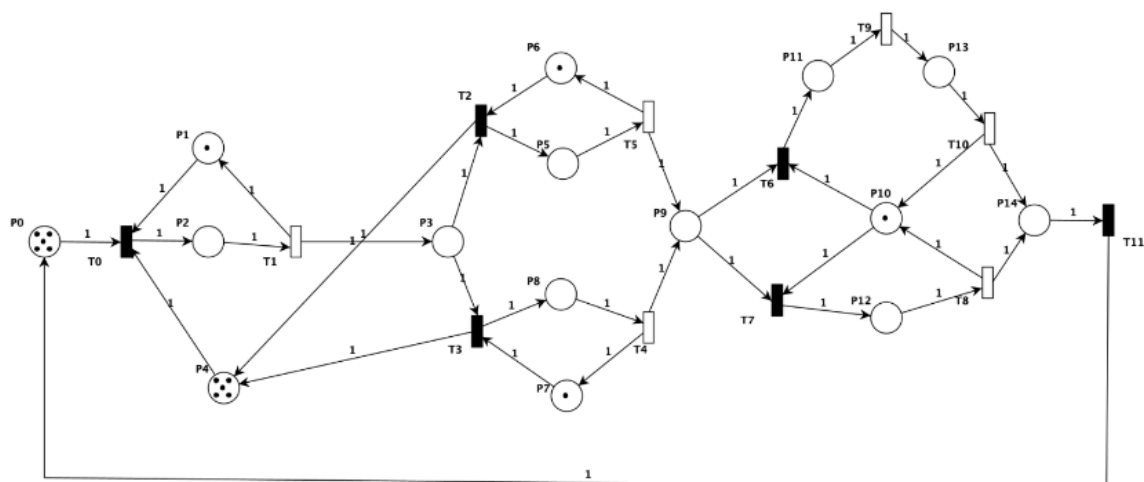
# Índice

<b>Índice</b>	<b>1</b>
<b>Introducción</b>	<b>2</b>
<b>Descripción de la Red de Petri</b>	<b>3</b>
<b>Análisis de propiedades de la red</b>	<b>4</b>
Tipos de Redes de Petri	4
Propiedades matemáticas	4
Red Simple y Extendida	4
Limitación	4
Seguridad	5
Ausencia de Deadlock	5
Vivacidad	5
<b>Invariantes de la Red.</b>	<b>5</b>
Matriz de Incidencia	5
Invariantes de Plaza	6
Invariantes de Transición	6
Análisis de Matriz de Incidencia	7
Obtención de las matrices post, pre e incidencia utilizando Petrinator	7
Análisis de Invariantes	8
Invariantes de Plaza	8
Invariantes de Transición	8
<b>Implementación</b>	<b>9</b>
Tabla de Estados del Sistema	9
Tabla de Eventos del Sistema	10
Determinación de la cantidad de hilos para la ejecución del sistema con el mayor paralelismo posible	10
Políticas	13
Tiempo	14
Demora en uno de los agentes que toman las reservas	14
Demora en la última etapa del sistema (cancelación/aceptación y pago)	16
Demora en la etapa inicial del sistema	16
<b>Desarrollo en Java</b>	<b>18</b>
GitHub repo	18
Diagrama de clases	18
Diagramas de Secuencia	21
Expresiones Regulares (REGEX)	24
<b>Conclusión</b>	<b>25</b>
Efectividad del Modelado con Redes de Petri	25
Ventajas del Monitor sobre Otras Técnicas de Sincronización	25
Verificación Automatizada de Invariantes	25
Análisis de Políticas y Rendimiento	25
Resultado Final	25
<b>Bibliografía</b>	<b>26</b>

# Introducción

Este informe presenta un análisis detallado, modelado y simulación de una red de Petri aplicada a un sistema de agencia de viajes. El trabajo se llevó a cabo en el marco de la asignatura Programación Concurrente, con el objetivo de estudiar el comportamiento de sistemas concurrentes utilizando redes de Petri, un modelo gráfico y matemático ampliamente empleado en la modelización de sistemas distribuidos. Para este propósito, se utilizó la herramienta Petrinator, que permitió identificar y analizar las propiedades clave de la red de Petri. Posteriormente, se implementó un monitor de concurrencia en Java, siguiendo las consignas del trabajo práctico.

## Descripción de la Red de Petri



● Fig. 1.0, Red de Petri

La red de Petri utilizada en este trabajo modela el flujo de clientes y la gestión de reservas en una agencia de viajes. Cada lugar y transición dentro de la red representa diferentes estados y eventos en el sistema. A continuación, se describen las principales plazas y transiciones:

- **P0 (Idle):** Representa el buffer de entrada de clientes a la agencia.
- **P1, P4, P6, P7, P10 (Recursos Compartidos):** Plazas que modelan recursos compartidos en el sistema, como agentes de reservas y áreas de gestión.
- **P2 (Ingreso a la Agencia):** Representa el ingreso de un cliente a la agencia de viajes.
- **P3 (Sala de Espera):** Lugar donde los clientes esperan antes de ser atendidos por un agente.
- **P5, P8 (Gestión de Reservas):** Estados del sistema en los cuales se gestionan las reservas de los clientes.
- **P9 (Espera para Aprobación/Rechazo de Reservas):** Modela la espera de los clientes mientras su reserva es procesada por un agente.
- **P11 (Confirmación de Reserva):** Plazas que representan la confirmación de una reserva por parte del agente.
- **P12 (Cancelación de Reserva):** Lugar donde se modela la cancelación de una reserva por parte del cliente o del agente.
- **P13 (Pago de la Reserva):** Representa el momento en que el cliente realiza el pago de la reserva confirmada.
- **P14 (Salida del Cliente):** Instancia previa a que el cliente se retire de la agencia.

La estructura de esta red de Petri permite capturar de manera efectiva las interacciones y dependencias entre los diferentes estados y procesos del sistema de la agencia de viajes.

## Análisis de propiedades de la red

Se hizo uso de la herramienta Pipe para clasificación de la red, de donde se obtuvo el siguiente resultado:

### Tipos de Redes de Petri

<b>State Machine</b>	false
<b>Marked Graph</b>	false
<b>Free Choice Net</b>	false
<b>Extended Free Choice Net</b>	false
<b>Simple Net</b>	false
<b>Extended Simple Net</b>	true

- Fig. 2.0, Tipos de Redes

### Propiedades matemáticas

<b>Bounded</b>	true
<b>Safe</b>	false
<b>Deadlock</b>	false

- Fig. 2.1, Propiedades de la Red

### Red Simple y Extendida

La red de Petri es **simple y extendida**, lo que significa que las transiciones reciben entradas de una sola plaza, o bien, en aquellos casos donde reciben más de una, el conjunto de salida de una de estas últimas está contenido o es igual al de las otras. Esta característica asegura una estructura regular en la red, lo que facilita su análisis y garantiza la coherencia en el flujo de tokens a través del sistema.

### Limitación

La red de Petri es **limitada**, lo que significa que la cantidad de tokens en cada lugar nunca excederá un valor máximo predefinido. Esta propiedad es fundamental para evitar un crecimiento ilimitado en cualquier parte del sistema, lo que podría llevar a estados de desbordamiento o comportamiento incontrolado. La limitación previene la acumulación excesiva de tokens en cualquier lugar lo que nos asegura un modelo más acorde a la realidad.

## Seguridad

A pesar de ser limitada, la red no es **segura**. Esto se debe a que, en algunos estados alcanzables desde la marca inicial, uno o más lugares pueden contener más de un token. Esta condición podría representar la ocurrencia simultánea de eventos que deberían ser mutuamente excluyentes, lo que podría tener implicaciones en la concurrencia y sincronización de eventos dentro del sistema. En el caso analizado, esto puede reflejar situaciones en las que múltiples clientes están esperando ya sea a ser atendidos o a recibir la confirmación/rechazo de su reserva, lo cual es aceptable pero requiere supervisión para evitar sobrecargas.

## Ausencia de Deadlock

Un deadlock es una situación en la que ninguna transición del sistema puede ser disparada, provocando que el sistema se quede bloqueado en un estado inactivo. La ausencia de deadlocks asegura que desde cualquier estado alcanzable siempre existe al menos una transición que puede ser disparada, garantizando que el sistema nunca se quedará sin posibilidad de avanzar. Esta propiedad es esencial para mantener la continuidad operativa del sistema. En conclusión, la red de Petri **no presenta deadlock**.

## Vivacidad

El hecho que una red de Petri sea viva significa que todas las transiciones tienen la posibilidad de ser disparadas en algún momento del ciclo de vida. Esto garantiza que ninguna parte quedará bloqueada indefinidamente y que todos los procesos podrán completarse eventualmente. En el caso analizado la red de Petri **es viva**. Esta es una propiedad clave para asegurar que este sistema pueda gestionar de manera eficiente todas las reservas, cancelaciones y otros eventos sin interrupciones.

## Invariantes de la Red.

El análisis de invariantes es crucial para comprender la conservación y repetitividad dentro de la red de Petri. Las invariantes de una red son propiedades independientes tanto del marcado inicial como de la secuencia de disparos, y pueden asociarse a ciertos subconjuntos de plazas o de transiciones.

Para llevar a cabo este análisis, se ha utilizado la matriz de incidencia, que juega un papel fundamental en la identificación tanto de los invariantes de plaza como de los invariantes de transición.

## Matriz de Incidencia

La matriz de incidencia de la red de Petri es una representación matemática que describe cómo las transiciones afectan las plazas en la red. Esta matriz se define como la diferencia entre la **matriz de incidencia de salida**  $I^+$  y la **matriz de incidencia de entrada**  $I^-$

$$I = I^+ - I^-$$

- $I^+$  = Indica cómo las transiciones añaden tokens a las plazas.
- $I^-$  = Indica cómo las transiciones remueven tokens de las plazas.

Las matrices  $I^+$  e  $I^-$  son también conocidas como *post* y *pre*, respectivamente. Para el caso de la matriz *post* se tiene que cada elemento  $I^+(P_i, T_j)$  contiene el peso asociado al arco que va desde  $T_j$  hasta  $P_i$ , es decir, la cantidad de tokens que se *generan* en la plaza  $P_i$  cuando la transición  $T_j$  es disparada. El caso contrario ocurre para los elementos de  $I^-(P_i, T_j)$ , que indica los tokens que se *retiran* en la plaza  $P_i$  cuando la transición  $T_j$  es disparada.

Las filas de las matrices representan las plazas mientras que las columnas las transiciones.

## Invariantes de Plaza

Una invariante de plaza o **p-invariante** es un conjunto de plazas cuya suma de tokens no se modifica con una secuencia de disparos arbitrarios. Estos invariantes son esenciales para analizar la **conservación de recursos** dentro del sistema, asegurando que ciertas cantidades o recursos se mantengan estables a lo largo del tiempo

Para su cálculo se hace uso de la ecuación:

$$I \cdot x = 0$$

siendo  $I$  la matriz de incidencia y  $x$  un vector característico constituido por las plazas que forman parte de la invariante.

## Invariantes de Transición

Una invariante de transición o **t-invariante** es el conjunto de transiciones que deben dispararse para que la red retorne a su estado inicial. Estos invariantes son clave para entender los **ciclos operacionales** en la red, asegurando que el sistema puede regresar a un estado de referencia después de completar un ciclo de operaciones.

Para el cálculo de los vectores que forman parte de las t-invariantes, la ecuación asociada es:

$$I^T \cdot x = 0$$

donde  $I^T$  es la transpuesta de la matriz de incidencia y  $x$  un vector característico constituido por las transiciones necesarias para que el sistema vuelva a su estado inicial.

## Análisis de Matriz de Incidencia

Obtención de las matrices post, pre e incidencia utilizando Petrinator

Forwards incidence matrix  $I^+$

	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	1	0	1	0	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	1	0	1	0	0

Backwards incidence matrix  $I^-$

	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	1	0	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	1	0	0
P14	0	0	0	0	0	0	0	0	0	0	0	1

- Fig. 3.0, Matriz de incidencia entrada/salida

Combined incidence matrix  $I$

	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

- Fig. 3.1, Matriz de incidencia combinada



## Análisis de Invariantes

### 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

• Fig. 4.0, Invariantes de plaza

Un uno en una posición indica que esa plaza es parte de la invariante y un cero indica lo contrario.

### Ecuaciones de las invariantes de plaza

$$M(P1) + M(P2) = 1$$

$$M(P5) + M(P6) = 1$$

$$M(P7) + M(P8) = 1$$

$$M(P2) + M(P3) + M(P4) = 5$$

$$M(P10) + M(P11) + M(P12) + M(P13) = 1$$

$$M(P0) + M(P2) + M(P3) + M(P5) + M(P8) + M(P9) + M(P11) + M(P12) + M(P13) + M(P14) = 5$$

### 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

• Fig. 4.1, Invariantes de transición

Un uno en una posición indica que esa transición es parte de la invariante y un cero indica lo contrario.

# Implementación

Tabla de Estados del Sistema

Plaza	Estado
P0	Buffer entrada clientes a la agencia
P1	Recurso de ingreso a la agencia
P2	Ingreso a la agencia
P3	Sala de espera de la agencia
P4	Recursos de ingreso a sala de espera
P5	Estado donde se realiza la reserva
P6	Recurso para realizar reserva
P7	Recurso para realizar reserva
P8	Estado donde se realiza la reserva
P9	Espera de quien aprueba/rechaza reservas
P10	Recurso para confirmación y pago o cancelación
P11	Confirmación de la reserva
P12	Cancelación de la reserva
P13	Pago de la reserva
P14	Instancia previa a la salida del cliente

## Tabla de Eventos del Sistema

Transición	Evento
T0	Ingreso de cliente a la agencia
T1	Ingreso de cliente a la sala de espera
T2	Inicio de reserva
T3	Inicio de reserva
T4	Espera de realización de reserva
T5	Espera de realización de reserva
T6	Ingreso a confirmación de la reserva
T7	Cancelación de la reserva
T8	Ingreso al buffer de salida de la agencia
T9	Ingreso a pago de la reserva
T10	Ingreso al buffer de salida de la agencia
T11	Salida del cliente

## Determinación de la cantidad de hilos para la ejecución del sistema con el mayor paralelismo posible

El número máximo de hilos que pueden correr en paralelo es igual al número máximo de secuencias de transiciones que pueden ejecutarse concurrentemente sin conflictos. Para determinar la cantidad de hilos necesarios para la ejecución del sistema con el mayor paralelismo posible, es necesario:

1. Obtener los invariantes de transición de la red:

$$IT_1 = \{T0, T1, T3, T4, T7, T8, T11\}$$

$$IT_2 = \{T0, T1, T3, T4, T6, T9, T10, T11\}$$

$$IT_3 = \{T0, T1, T2, T5, T7, T8, T11\}$$

$$IT_4 = \{T0, T1, T2, T5, T6, T9, T10, T11\}$$

2. Obtener el conjunto de plazas  $PI$  para cada  $IT$

$$PI_1 = \{P0, P1, P2, P3, P4, P7, P8, P9, P10, P12, P14\}$$

$$PI_2 = \{P0, P1, P2, P3, P4, P7, P8, P9, P10, P11, P13, P14\}$$

$$PI_3 = \{P0, P1, P2, P3, P4, P5, P6, P9, P10, P12, P14\}$$

$$PI_4 = \{P0, P1, P2, P3, P4, P5, P6, P9, P10, P11, P13, P14\}$$

3. Obtenemos el conjunto de plazas de acción ( $PA$ ) de cada  $IT$ :

$$PA_1 = \{P2, P8, P12, P14\}$$

$$PA_2 = \{P2, P8, P11, P13, P14\}$$

$$PA_3 = \{P2, P5, P12, P14\}$$

$$PA_4 = \{P2, P5, P11, P13, P14\}$$

4. Obtenemos el conjunto de estados  $MA$  del conjunto de plazas  $PA$  donde

$$PA = \{P2, P5, P8, P11, P12, P13, P14\}$$

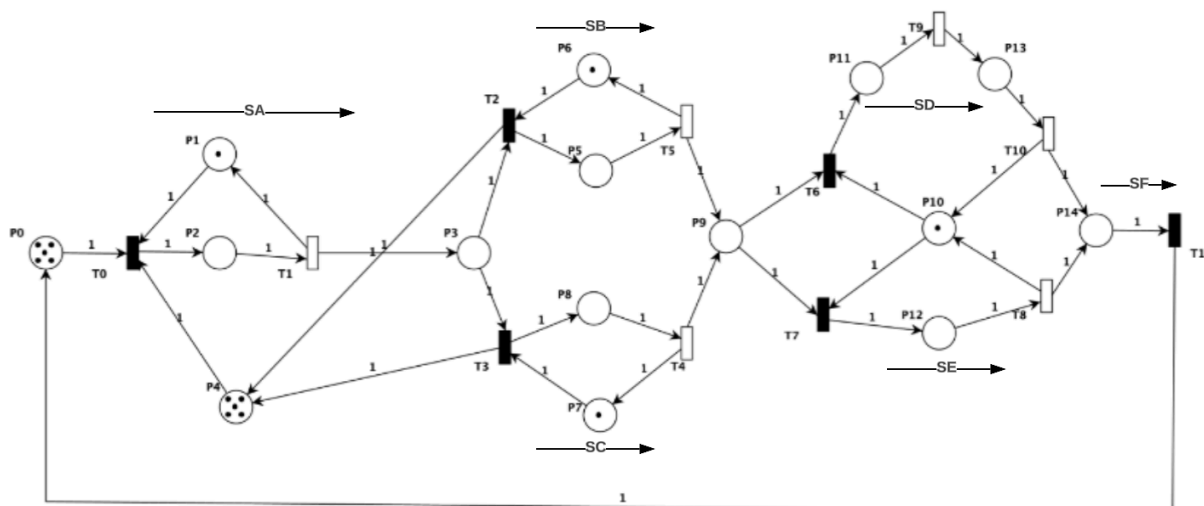
Luego, se busca el marcado máximo entre todos los marcados posibles con la tabla  $MA$ :

P2	P5	P8	P11	P12	P13	P14	SUMA
1	0	0	0	0	0	0	1
0	1	0	0	0	0	0	1
1	1	0	0	0	0	0	2
0	1	1	0	0	0	0	2
1	1	1	0	0	0	0	3
1	0	1	1	0	0	0	3
0	1	1	1	0	0	0	3
1	1	1	0	0	1	0	4
1	1	1	0	0	0	1	4
1	1	0	0	1	0	1	4
1	1	1	0	1	0	1	5

**Tabla:** la tabla muestra el número de tokens en las plazas pertenecientes al conjunto  $PA$  en un momento dado, al que se puede alcanzar mediante el disparo de una serie definida de transiciones.

Con el resultado obtenido de la tabla *MA*, se puede observar que la cantidad máxima de hilos simultáneos en el sistema será *cinco*.

Se aplica el algoritmo para determinar la responsabilidad de los hilos:



• Fig. 5.0, Segmentos de la rdp

Tenemos 6 segmentos:

$$PA = \{P2, P5, P8, P11, P12, P13, P14\}$$

$$PS_A = \{P2\}$$

$$PS_B = \{P5\}$$

$$PS_C = \{P8\}$$

$$PS_D = \{P11, P13\}$$

$$PS_E = \{P12\}$$

$$PS_F = \{P14\}$$

$$MS_A = \{0\} \mid \{1\}$$

$$MS_B = \{0\} \mid \{1\}$$

$$MS_C = \{0\} \mid \{1\}$$

$$MS_D = \{0\} \mid \{1\}$$

$$MS_E = \{0\} \mid \{1\}$$

$$MS_F = \{0\} \mid \{1\}$$

¿Cuál es la responsabilidad de cada hilo?

$H_A$ : El hilo A se encarga del proceso de ingreso y espera por parte de los clientes a la agencia.

$H_B$ : El hilo B se encarga del proceso de gestión de las reservas.

$H_C$ : El hilo C se encarga del proceso de gestión de las reservas.

$H_D$ : El hilo D se encarga del proceso de confirmación y pago de las reservas.

$H_E$ : El hilo E se encarga del proceso de rechazo de las reservas.

$H_F$ : El hilo F se encarga del proceso de salida de los clientes de la agencia.

Los hilos D y E se podrían unificar, ya que la plaza P10 tiene un solo recurso y no es posible la ejecución en paralelo de los segmentos D y E. Decidimos dejar un hilo para cada segmento como criterio de diseño, teniendo en cuenta que se podría aumentar la cantidad de recursos en P10 y así permitir la ejecución en paralelo.

### Cambios en el diseño:

Se agregaron cambios solicitados y se pudo apreciar durante las pruebas realizadas sobre funcionamiento del monitor, que, utilizando la cantidad de hilos según el algoritmo del paper “Algoritmo para la determinación de hilos máximos activos simultáneos” (6 hilos), nuestra red no está teniendo en cuenta la política en los segmentos donde se encuentran los conflictos, por lo que se reestructuró el sistema asignando 1 hilo más a cada transición en conflicto, llegando así a un total de 10 hilos asumiendo la pérdida de performance.

Se realizaron pruebas utilizando un diseño donde por cada transición le asignamos un hilo, llegando a tener 12 hilos, pero el análisis de los tiempos empeoraba, por lo que se lograba una peor performance con más hilos.

## Políticas

Se desarrollaron tres políticas para poder cambiar el comportamiento del sistema.

Se comprueba el funcionamiento de las mismas utilizando una expresiones regulares y un script de *python*.

Para poder ejecutar el programa con alguna política determinada hay dos opciones:

1. Pasar el número de la política como argumento al ejecutar el programa, por ejemplo “./gradlew run --args=<1/2/3>” donde 1 corresponde a la política balanceada, 2 a la que tiene prioridades y 3 a la FCFS.
2. Correr el programa y seleccionar la opción deseada desde la consola:

```
2025-02-16T15:03:51.419851871 INFO: Application starting...

Available policies:
1: Balanced Policy (50/50 and 50/50 distributions)
2: Prioritized Policy (75/25 and 80/20 distributions)
3: FCFS Policy (First-Come-First-Served)
Enter your choice (1, 2 or 3):
```

• Fig. 6.0, Políticas disponibles

## Resultados de ejecuciones para cada política

Balanceada:

```
Percentages calculated from invariants:
T2: 50.00%↑      T6: 50.00%↑
T3: 50.00%↓      T7: 50.00%↓
```

• Fig. 6.1, Política balanceada resultado

Priorizada:

```
Percentages calculated from invariants:
T2: 74.73%↑      T6: 79.57%↑
T3: 25.27%↓      T7: 20.43%↓
```

- Fig. 6.2, Política Priorizada resultado

FCFS:

```
Percentages calculated from invariants:
T2: 48.92%↑      T6: 33.87%↑
T3: 51.08%↓      T7: 66.13%↓
```

- Fig. 6.3, Política FCFS resultado

## Tiempo

Se eligieron los siguientes valores para las transiciones temporales:

$T1 = 2 \text{ minutos}$   
 $T4 = 15 \text{ minutos}$   
 $T5 = 15 \text{ minutos}$   
 $T8 = 5 \text{ minutos}$   
 $T9 = 5 \text{ minutos}$   
 $T10 = 15 \text{ minutos}$

En el código se utilizó la escala:

$1 \text{ milisegundo} = 1 \text{ minuto}$

para poder representar el tiempo en minutos sin hacer que la simulación dure demasiado.

Para la primera transición se pensó en dos minutos ya que ésta representa el ingreso del cliente a la sala de espera por lo que no debería ser mucho tiempo. Luego para T4 y T5 se pensó en quince minutos asumiendo que ese tiempo sería suficiente para poder realizar la reserva. Para T8 se asume que cinco minutos sería correcto para recibir la cancelación de la misma. Por último cinco minutos para confirmar la reserva y controlar que todo esté bien, y quince para hacer el pago de la misma.

Con un script de *bash* se automatizó la ejecución de diez ejecuciones del programa, alcanzando un total de 1860 invariantes de plaza. Esto con el objetivo de tener valores representativos de comportamiento del sistema. Luego se analizaron los resultados con un script de *python* y expresiones *regex*.

### Demora en uno de los agentes que toman las reservas

Primero realizamos 10 ejecuciones del programa con cada política para tomar tiempos y para así poder obtener conclusiones en análisis posteriores, esto lo podemos realizar utilizando el script de *bash* y seleccionando la política deseada:

Política Balanceada: Tiempo total 41,531s

```
Number of times each transition invariant was found:
T0 T1 T2 T5 T6 T9 T10 T11: 452 times
T0 T1 T2 T5 T7 T8 T11: 478 times
T0 T1 T3 T4 T6 T9 T10 T11: 477 times
T0 T1 T3 T4 T7 T8 T11: 453 times

Percentages calculated from invariants:
T2: 50.00%↑      T6: 49.95%↑
T3: 50.00%↓      T7: 50.05%↓
```

- Fig. 6.4, Política Balanceada 10 Ejecuciones

Política Priorizada: Tiempo total 50,500s

```
Number of times each transition invariant was found:
T0 T1 T2 T5 T6 T9 T10 T11: 1131 times
T0 T1 T2 T5 T7 T8 T11: 259 times
T0 T1 T3 T4 T6 T9 T10 T11: 353 times
T0 T1 T3 T4 T7 T8 T11: 117 times

Percentages calculated from invariants:
T2: 74.73%↑      T6: 79.78%↑
T3: 25.27%↓      T7: 20.22%↓
```

- Fig. 6.5, Política Priorizada 10 Ejecuciones

Luego procedemos a aumentar el tiempo de demora de uno de los agentes que toman las reservas:

$$T4 = 45 \text{ minutos}$$

Ejecutamos el programa diez veces y los resultados fueron los siguientes:

Política Balanceada: La ejecución demoró 41,819s, lo cual no parece haber afectado mucho en performance, pero lo que sí podemos observar es que una mayor cantidad de clientes pasan por la plaza P6, con una relación aproximada de 3-1 a pesar de que estemos utilizando la política de 50-50. Esto tiene sentido ya que aunque la política intente equilibrar la carga, la demora inherente de T4 (45 min) limita su capacidad, desplazando más clientes a T5 que se encuentra disponible.

```
Percentages calculated from invariants:
T2: 71.77%↑      T6: 50.00%↑
T3: 28.23%↓      T7: 50.00%↓
```

- Fig. 6.6, Política Balanceada 10 Ejecuciones T4=45

Política Priorizada: Ahora tenemos que la ejecución demoró 50,425s, tampoco parece haber afectado en performance, y en este caso si se cumplió el comportamiento esperado de la política.

```
Percentages calculated from invariants:
T2: 74.78%↑      T6: 79.89%↑
T3: 25.22%↓      T7: 20.11%↓
```



- Fig. 6.7, Política Priorizada 10 Ejecuciones T4=45

### Demora en la última etapa del sistema (cancelación/aceptación y pago)

En una de las pruebas, se aumentó el tiempo en caso de que haya muchas demoras en la recepción de la cancelación y pago de la reserva con los siguientes tiempos:

$$T8 = 30 \text{ minutos}$$

$$T9 = 10 \text{ minutos}$$

$$T10 = 50 \text{ minutos}$$

Política Balanceada: La duración obtenida es de 1 m 42,769s, y si bien se sigue cumpliendo con la política, el aumento de tiempo se debe a que en la última etapa en la plaza (P10) ocurre un cuello de botella debido a que posee un único recurso compartido entre la cancelación de reserva y la confirmación y posterior pago de la reserva, por lo tanto con estos cambios solo estamos empeorando performance.

```
Percentages calculated from invariants:
T2: 50.00%↑      T6: 50.00%↑
T3: 50.00%↓      T7: 50.00%↓
```

- Fig. 6.8, Política Balanceada 10 Ejecuciones T8=30, T9=10, T10=50

Política Priorizada: Como es lo esperado la duración aumentó a 1 m 59,610s, lo cual nos muestra un comportamiento parecido al observado en el caso anterior, es decir, cumple las políticas pero en tiempos es mayor al original por las mismas razones.

```
Percentages calculated from invariants:
T2: 74.73%↑      T6: 79.78%↑
T3: 25.27%↓      T7: 20.22%↓
```

- Fig. 6.9, Política Priorizada 10 Ejecuciones T8=30, T9=10, T10=50

### Demora en la etapa inicial del sistema

La transición T1 es la encargada de enviar a los clientes a la plaza P3, donde esperan para ser recibidos por algunos de los agentes que tomarán su reserva. Se agregó un aumento en el tiempo de demora del disparo de esta transición:

$$T1 = 60 \text{ minutos}$$

Política Balanceada: En este caso se vio un aumento en el tiempo de ejecución del programa a un tiempo de 2m11,558s, y vemos un claro cuello de botella en el análisis del remanente de transiciones, ya que una vez finalizado el programa las transiciones disparadas que no podían ser asociadas a un invariante de transición eran correspondientes a las dos primeras del sistema (T0, T1)

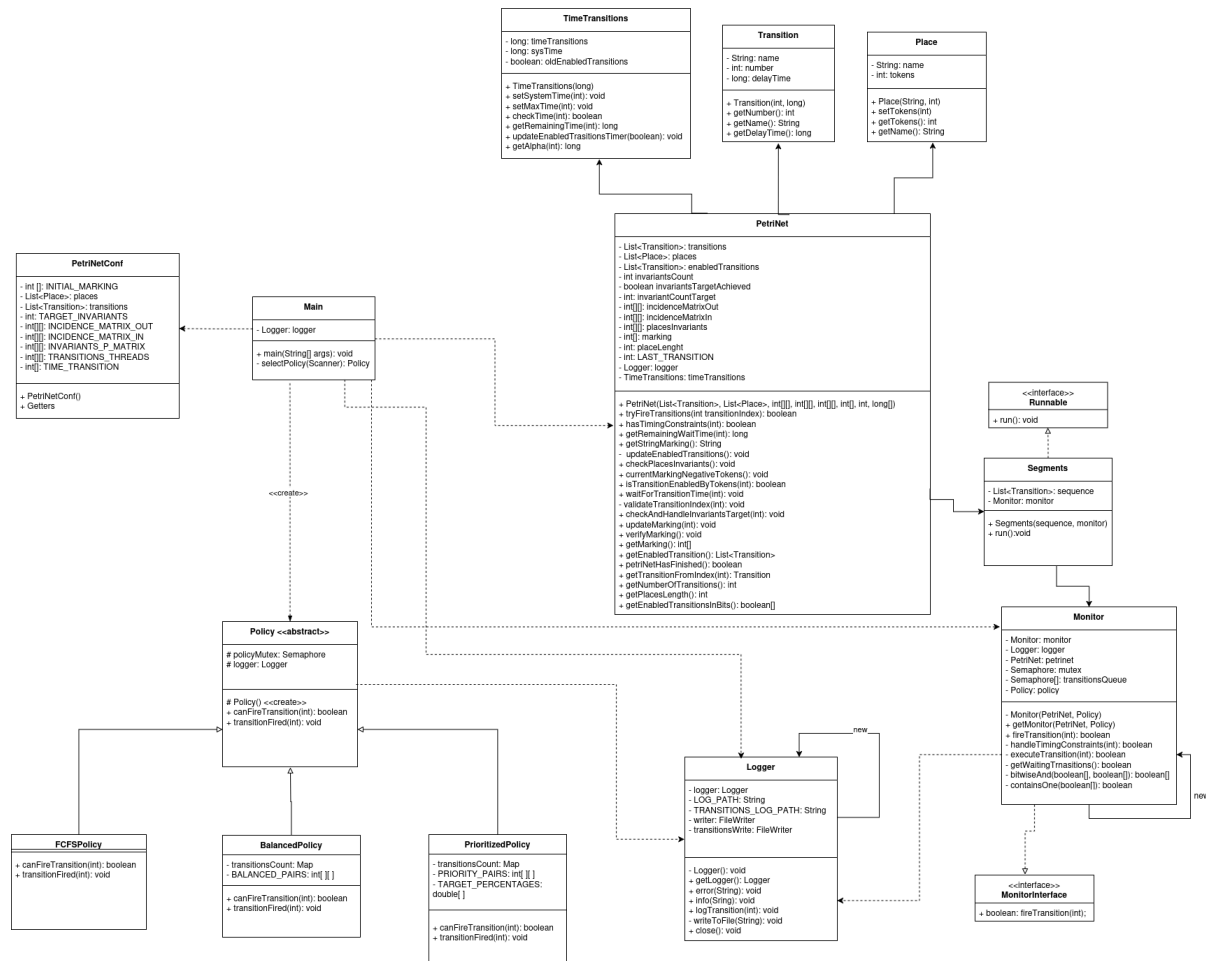
- Fig. 6.10, Política Balanceada 10 Ejecuciones T1=60

- Fig. 6.11, Política Priorizada 10 Ejecuciones T1=60

## Desarrollo en Java

[GitHub repo](#)

## Diagrama de clases



• Fig. 8.0, [Diagrama de clases](#).

El proyecto está compuesto por las siguientes clases:

- **Main**: Inicializa el sistema, permite seleccionar la política a utilizar y maneja la ejecución de los hilos.
- **Place**: Representa las plazas en la red de Petri, conteniendo tokens y su respectivo nombre.
- **Transition**: Modela las transiciones, incluyendo tiempos de retardo para transiciones temporales.
- **TimeTransitions**: Gestiona los tiempos de espera (alpha) para transiciones temporizadas en la red
- **PetriNet**: El núcleo del sistema que implementa la lógica de la red de Petri, maneja el disparo de transiciones y verifica invariantes de plaza.

- **Policy (abstracta):** Define la interfaz base para implementar diferentes políticas de control.
- **BalancedPolicy:** Implementa una política equilibrada 50/50 para ciertos pares de transiciones.
- **PrioritizedPolicy:** Implementa una política priorizada (75/25 y 80/20) para pares específicos.
- **PetriNetConf:** Contiene toda la configuración del sistema, incluyendo matrices de incidencia y marcado inicial.
- **Logger:** Sistema de registro que guarda eventos y errores en archivos de log.
- **Segments:** La clase Segments implementa la interfaz Runnable, permitiendo la ejecución concurrente de diferentes secuencias de transiciones en la red de Petri.

Cada instancia de Segments se encarga de ejecutar una secuencia específica de transiciones asignadas.

La clase recibe en su constructor una lista de transiciones que debe ejecutar y una referencia al Monitor que controla la red. Durante su ejecución, cada hilo Segments intenta disparar sus transiciones asignadas en orden, respetando las restricciones impuestas por el Monitor y las políticas de control.

El ciclo de ejecución continúa hasta que la red de Petri alcanza su objetivo (en este caso, completar 186 invariantes). Durante este proceso, múltiples instancias de Segments pueden estar ejecutándose simultáneamente, cada una manejando su propia secuencia de transiciones, lo que permite aprovechar el paralelismo dentro de la estructura de la red.

- **Monitor:** Su principal responsabilidad es controlar y sincronizar el acceso concurrente a la red, asegurando que las transiciones se disparen de manera segura y consistente. Implementa el patrón Singleton para garantizar que existe una única instancia que controla toda la red. Esto es crucial porque se necesita un punto único de control para mantener la consistencia del sistema.

El Monitor gestiona tres aspectos fundamentales:

1. La sincronización entre hilos mediante un semáforo, que garantiza el acceso exclusivo a las secciones críticas de la red. Utiliza un semáforo principal (`mutex`) para proteger el estado de la red y un array de semáforos (`transitionsQueue`) para gestionar los hilos que esperan por transiciones específicas.
2. La aplicación de políticas de control, consultando si una transición puede dispararse según la política actual y notificando cuando se realiza el disparo. Esto permite implementar diferentes estrategias como la política balanceada o priorizada. El Monitor delega en la política configurada la decisión sobre qué transición debe ejecutarse cuando múltiples están habilitadas y en espera.
3. El manejo de transiciones temporales, gestionando los tiempos de espera necesarios y asegurando que se respeten las restricciones temporales de cada transición. Coordina con el componente `TimeTransitions` para determinar si una transición temporizada ha cumplido su tiempo de espera requerido.

El Monitor expone una interfaz simple con un único método para disparar transiciones, pero internamente maneja la sincronización, el control de políticas y el manejo de errores. También se encarga del registro de eventos y errores para facilitar el seguimiento y depuración del sistema.

### Funcionamiento interno

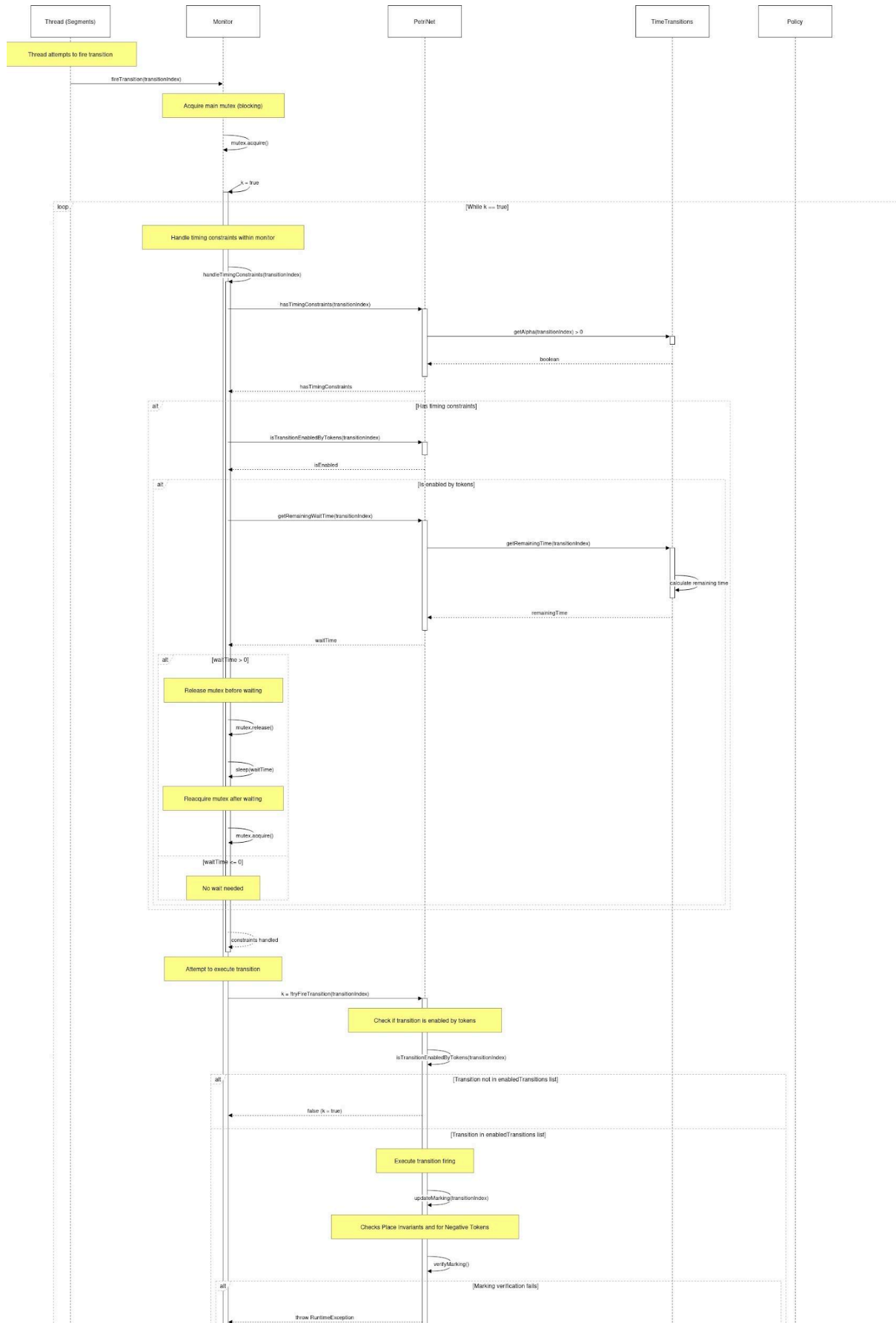
El Monitor implementa un mecanismo de tipo **Signal and Exit**, donde un hilo que completa su operación despierta a otro hilo en espera y luego sale del monitor. Este enfoque permite una ejecución más eficiente al evitar cambios de contexto innecesarios.

Proceso de disparo de transiciones:

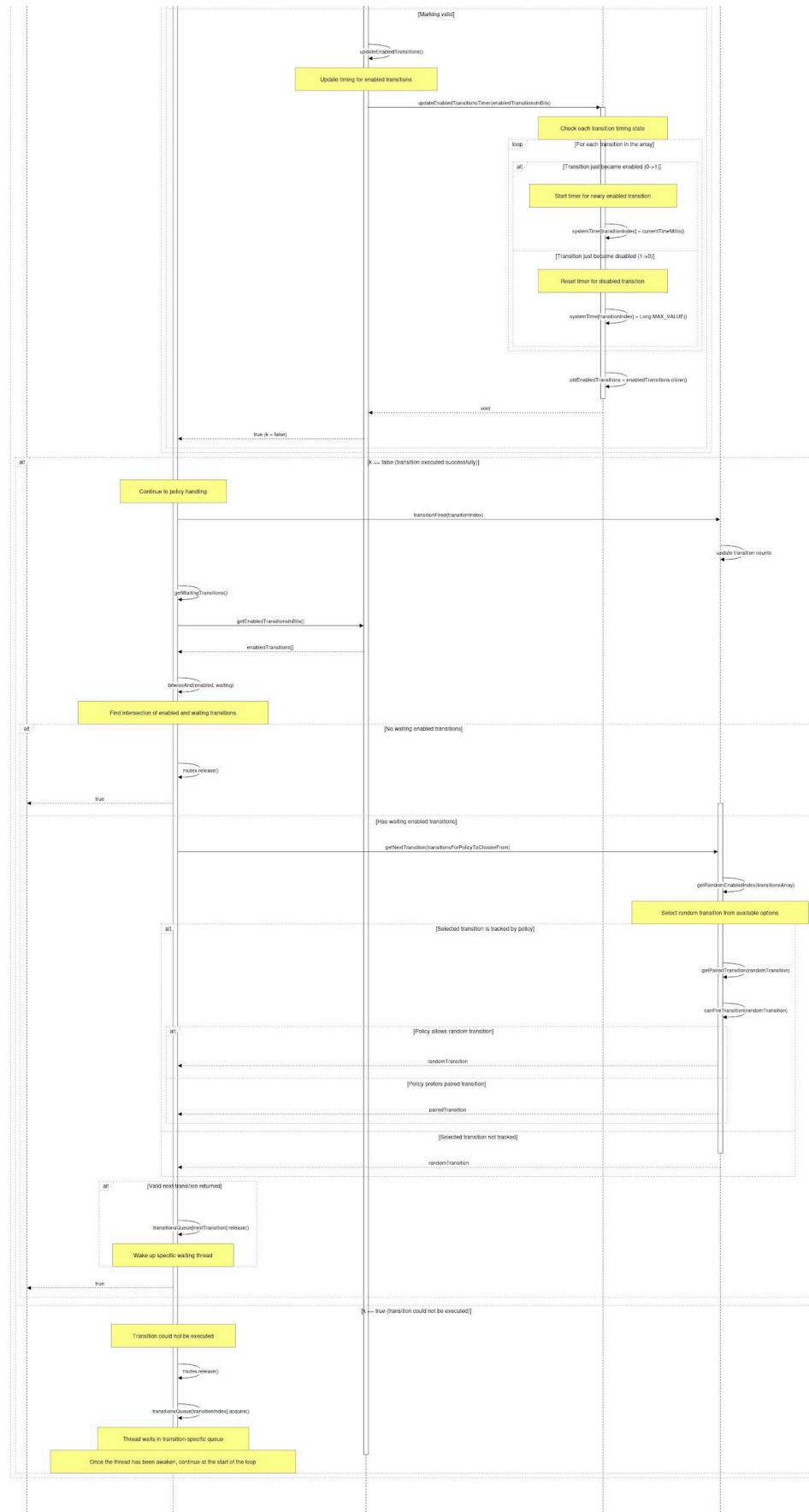
1. Adquisición de exclusión mutua: Cuando un hilo intenta disparar una transición, primero adquiere el `mutex` para garantizar acceso exclusivo al estado de la red.
2. Verificación de habilitación: Comprueba si la transición está habilitada según el marcado actual y las restricciones temporales.
3. Ejecución de la transición: Si está habilitada, actualiza el marcado de la red y registra el disparo.
4. Selección del siguiente hilo: Determina qué hilo debe ejecutarse a continuación mediante:
  - 1. Identificación de transiciones habilitadas en la red
  - 2. Identificación de hilos en espera por transiciones
  - 3. Aplicación de la política configurada para seleccionar entre las transiciones habilitadas con hilos en espera
5. Notificación y salida: Despierta al hilo seleccionado y sale del monitor, permitiendo que el sistema continúe su ejecución.

## Diagramas de Secuencia

- Fig. 8.1, [Diagrama de secuencia inicialización](#)



● Fig. 8.2, [Diagrama de secuencia hilos parte 1](#)



• Fig. 8.3, [Diagrama de secuencia hilos parte 2](#)



## Expresiones Regulares (REGEX)

El script [regex.py](#) analiza los logs generados por la ejecución de la red de Petri. Sus funciones principales son:

1. Leer secuencias de transiciones desde el archivo `"tmp/transitionsSequence.txt"`
2. Utiliza una expresión regular para identificar 4 invariantes de transición válidos:
  - T0 T1 T2 T5 T6 T9 T10 T11
  - T0 T1 T2 T5 T7 T8 T11
  - T0 T1 T3 T4 T6 T9 T10 T11
  - T0 T1 T3 T4 T7 T8 T11
3. Procesar iterativamente el archivo:
  - Busca coincidencias de invariantes
  - Las cuenta
  - Las elimina del texto
  - Continúa hasta procesar todas las transiciones
4. Calcula estadísticas importantes:
  - Cantidad de veces que aparece cada invariante
  - Porcentajes de uso de las transiciones T2/T3 y T6/T7 (útil para verificar políticas)
5. Al finalizar:
  - Muestra los resultados del análisis

# Conclusión

## Efectividad del Modelado con Redes de Petri

El uso combinado de redes de Petri y un monitor resultó altamente eficaz para modelar, analizar e implementar un sistema concurrente. A través de la herramienta PIPE se pudieron verificar propiedades fundamentales como la ausencia de deadlocks, la vivacidad del sistema y existencia de invariantes, lo que garantizó un diseño correcto y consistente desde la fase teórica.

## Ventajas del Monitor sobre Otras Técnicas de Sincronización

En cuanto a la implementación, el uso del monitor permitió centralizar la lógica de sincronización, garantizando un acceso seguro y coherente a la red de Petri. A diferencia de los semáforos, que dispersan el control de concurrencia en distintas partes del código, el monitor concentró toda la lógica en un único lugar, lo que facilitó el mantenimiento, mejoró la legibilidad y redujo significativamente la probabilidad de errores de sincronización.

## Verificación Automatizada de Invariantes

Se implementó un script basado en expresiones regulares (REGEX) para verificar automáticamente los invariantes de transición a partir de los logs de ejecución. Esto permitió validar de forma sistemática que el sistema ejecuta correctamente los ciclos esperados, asegurando que las secuencias de transiciones se comportan de acuerdo al modelo teórico y que las políticas implementadas funcionan según lo diseñado.

## Análisis de Políticas y Rendimiento

Las dos políticas desarrolladas (balanceada y priorizada) demostraron comportamientos diferenciados y medibles. Los experimentos temporales revelaron que ciertos recursos únicos como la plaza P10 se convierten en cuellos de botella inevitables.

## Resultado Final

En conjunto, este enfoque integró herramientas formales, técnicas de sincronización robustas y verificación automatizada, logrando un sistema concurrente confiable, eficiente y fácil de mantener.

## Bibliografía

- Ventre, O. L; Micolini, O. (2021). *Algoritmos para determinar cantidad y responsabilidad de hilos en sistemas embebidos modelados con Redes de Petri S<sup>3</sup>PR*. ResearchGate. <https://www.researchgate.net/publication/358104149>
- Diaz, M; (2009) *Petri Nets fundamental Models, Verification and Applications*: John Wiley & Sons - ISTE: USA.
- Gallardo Melgarejo, M. M. (2002). *Comunicación y sincronización con monitores*. Notas de clase, Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga.