

# Programación Concurrente

## Trabajo Práctico Final

### *Alumnos:*

Bernaus, Julieta - email: [julieta.bernaus@mi.unc.edu.ar](mailto:julieta.bernaus@mi.unc.edu.ar)

Cirrincione, Franco - email: [francocirrincione@mi.unc.edu.ar](mailto:francocirrincione@mi.unc.edu.ar)

Rodriguez, Luciano Ariel - email: [lucianorodriguez131@mi.unc.edu.ar](mailto:lucianorodriguez131@mi.unc.edu.ar)

Vignolo, Gabriel Enrique - email: [gabriel.vignolo@mi.unc.edu.ar](mailto:gabriel.vignolo@mi.unc.edu.ar)

Zelaya Alanis, Darío Marcelo - email [dario.m.zelaya@unc.edu.ar](mailto:dario.m.zelaya@unc.edu.ar)

### *Docentes:*

Ing. Luis Orlando Ventre.

Ing. Mauricio Ludemann.

## Consigna

### Enunciado

En la *Figura 1* se observa una **red de Petri** que modela un sistema doble de procesamiento de imágenes.

Las plazas  $\{P1, P3, P5, P7, P9, P11, P15\}$  representan **recursos compartidos** en el sistema.

La plaza  $\{P0\}$  es una plaza idle que corresponde al **buffer de entrada** de imágenes al sistema.

En las plazas  $\{P2, P4\}$  se realiza la **carga de imágenes en el contenedor** para procesamiento.

La plaza  $\{P6\}$  representa el **contenedor de imágenes a procesar**.

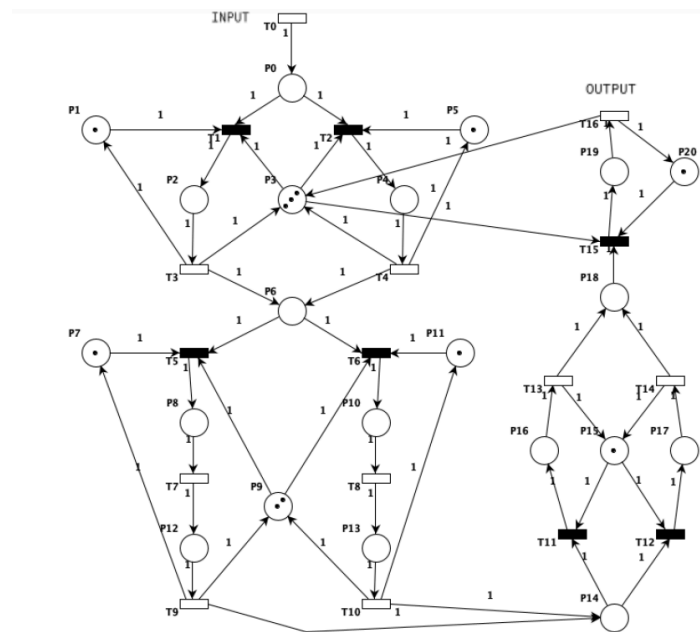
Las plazas  $\{P8, P10, P12, P13\}$  representan los estados en los cuales se realiza un **ajuste de la calidad** de las imágenes. Este ajuste debe hacerse en **dos etapas secuenciales**.

Con la plaza  $\{P14\}$  se modela el **contenedor de imágenes mejoradas en calidad, listas para ser recortadas** a su tamaño definitivo.

En las plazas  $\{P16, P17\}$  se realiza el **recorte**.

En la plaza  $\{P18\}$  se depositan las **imágenes en estado final**.

La plaza  $\{P19\}$  representa el proceso por el cual **las imágenes son exportadas** fuera del sistema.



*Figura 1: Red a analizar.*

### Propiedades de la red

Es necesario determinar con una herramienta (simulador Ej: Petrinator, Pipe), la cual deberá justificar, las propiedades de la red (deadlock, vivacidad, seguridad). Cada

propiedad debe ser interpretada para esta red particular.

- Indicar cual o cuales son los invariantes de plaza y los invariantes de transición de la red. Realizar una breve descripción de lo que representan en el modelo.

- NOTA IMPORTANTE 1: Para realizar el análisis de propiedades estructurales en la herramienta PIPE, es necesario unir {T16} con {P0} y eliminar {T0}.

- NOTA IMPORTANTE 2: Para realizar el análisis de invariantes en la herramienta PIPE marcar todas las transiciones como inmediatas (no temporizadas).

### **Tareas a implementar**

Es necesario implementar un monitor de concurrencia para la simulación y ejecución del modelo:

- Realizar una tabla, con los estados del sistema.
- Realizar una tabla, con los eventos del sistema.
- Determinar la cantidad de hilos necesarios para la ejecución del sistema con el mayor paralelismo posible (ver Referencias \*1):
  - Caso 1: si el invariante de transición tiene un conflicto, con otro invariante, debe haber un hilo encargado de la ejecución de la/s transición/es anterior/es al conflicto y luego un hilo por invariante.
  - Caso 2: si el invariante de transición presenta un join, con otro invariante de transición, luego del join debe haber tantos hilos, como token simultáneos en la plaza, encargados de las transiciones restantes dado que hay un solo camino.
  - Realizar un gráfico donde pueda observarse las responsabilidades de cada hilo con diferentes colores. A modo de ejemplo se observa en la figura 2 una red y cómo presentar lo solicitado, las flechas coloreadas representan cada tipo y cantidad de hilos.

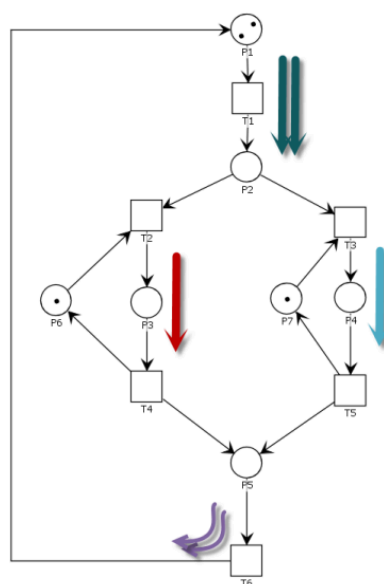


Figura 2.

## **Tiempo**

Una vez implementado el monitor de concurrencia, con el sistema funcionando, se deberá **implementar la semántica temporal**. Las transiciones  $\{T0, T3, T4, T7, T8, T11, T12, T14\}$  son transiciones temporales. **Implementarlas y asignarles un tiempo** (a elección del grupo) en milisegundos. Se debe hacer un **análisis temporal** tanto analítico como práctico (ejecutando el proyecto múltiples veces), **justificando** los resultados obtenidos. Además, es necesario también **variar los tiempos** elegidos, analizar los resultados y obtener **conclusiones**.

## **Políticas**

Es necesario para el modelado del sistema **implementar políticas** que resuelvan los conflictos. Se requiere considerar dos casos (ejecutados y analizados por separado e independientes uno de otro):

1. Una política de procesamiento de imágenes balanceada. La cantidad de imágenes procesadas por cada segmento de la red al finalizar la ejecución, debe ser equitativa (izquierda vs derecha). Esto se debe corroborar al finalizar la ejecución mostrando la cantidad de imágenes procesadas por cada segmento. Para ello, se debe mostrar la cantidad de veces que se ejecutaron cada una de las transiciones pertenecientes a cada segmento.
2. Una política de procesamiento que priorice el segmento izquierdo en la etapa 3. Considere que el segmento izquierdo reciba el 80% de la carga  $\{T11, T13\}$ . Esto se debe corroborar de la misma manera que se indicó en el punto anterior.

## **Requerimientos**

1. Implementar la red de Petri de la Figura 1 haciendo uso de una herramienta, ej: PIPE. Verificar todas sus propiedades.
2. El proyecto debe ser modelado con objetos en Java, haciendo uso de un monitor de concurrencia para guiar la ejecución de la red de Petri.
3. Implementar un objeto Política que cumpla con los objetivos establecidos en el apartado Políticas.
4. Hacer el diagrama de clases que modele el sistema.
5. Hacer el diagrama de secuencia que muestre el disparo exitoso de una transición que esté sensibilizada, mostrando el uso de la política.
6. Indicar la cantidad de hilos necesarios para la ejecución y justificar de acuerdo a lo mencionado en el apartado Implementación (ver Referencias \*1).
7. Realizar múltiples ejecuciones con 200 invariantes completados (para cada ejecución), y demostrar con los resultados obtenidos:
  - a. Cuán equitativa es la política implementada en el balance de carga en los invariantes.

- b. La cantidad de cada tipo de invariante, justificando el resultado.
8. Registrar los resultados del punto 7, haciendo uso de un archivo de log para su posterior análisis.
  9. Hacer un análisis de tiempos, de acuerdo a lo mencionado en el apartado Tiempo.
  10. Mostrar e interpretar los invariantes de plazas y transiciones que posee la red.
  11. Verificar el cumplimiento de los invariantes de plazas luego de cada disparo de la red.
  12. Verificar el cumplimiento de los invariantes de transiciones mediante el análisis de un archivo log de las transiciones disparadas al finalizar la ejecución. El análisis de los invariantes debe hacerse mediante expresiones regulares. Tip: [www.regex.com](http://www.regex.com) - [www.debuggex.com](http://www.debuggex.com).
  13. El programa debe poseer una clase Main que al correrla, inicie el programa.

### **Modificaciones a la consigna**

Siguiendo las indicaciones de los profesores, se realizan las siguientes modificaciones a la red:

- se eliminan las plazas 12 y 13;
- se eliminan las transiciones 7 y 8.

Además, se adquieren tablas de invariantes ya previamente calculadas.

### **Consideraciones adicionales**

Para facilitar la comprensión de la red, se decidió renombrar las plazas y transiciones para que sigan un orden lógico. De esta manera, esta será la red final a utilizar:

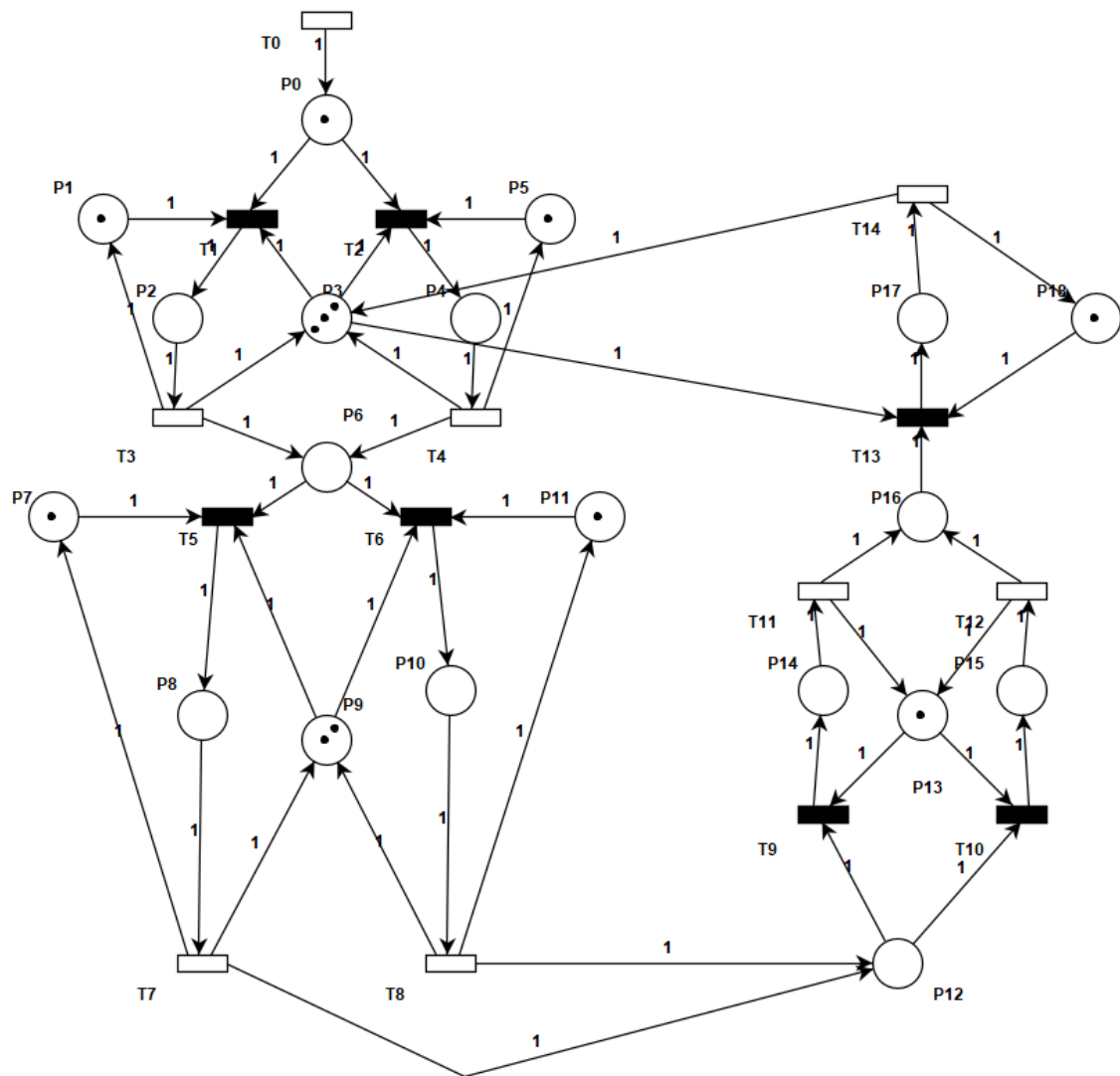


Figura 3: Diagrama de la red de Petri final.

## Marco Teórico

### Redes de Petri

Una red de Petri es una representación gráfica y matemática de un sistema distribuido, concurrente o paralelo, creada por su homónimo Carl Adam Petri. Está compuesta por *plazas y transiciones*. Las plazas y transiciones están comunicadas por *arcos*. Es una *red-bipartita*, esto quiere decir que los arcos van de las plazas a las transiciones o de las transiciones a la plazas. Osea, va desde un tipo de elemento a otro tipo de elemento.

Las plazas (círculos, gráficamente) representan a *estados o condiciones* del sistema, y pueden contener *marcas o tokens* que representan la presencia o *cantidad de recursos*.

Las transiciones (rectángulos, gráficamente) representan *acciones o eventos* en el sistema. Para dispararse, necesitan que sus condiciones estén satisfechas. Como mínimo, necesitan que haya suficientes marcas en la entrada (si se disparan solo con eso, son *instantáneas*) pero también pueden tener tiempos mínimos como condición necesaria (*temporizadas*).

Los arcos (flechas, gráficamente) representan la *dirección del movimiento de marcas*. El *disparo* consume, de las plazas que entran a la transición, el *número de tokens del peso del arco* que une la plaza con la transición. Genera en las plazas de salida el número de tokens del peso del arco que une la transición con la plaza. En el caso de este trabajo, se considerará que ***todos los arcos son de peso uno***, es decir, se trata de una red de petri ***ordinaria***.

### Propiedades de las Redes de Petri

**Deadlock o estado sumidero:** se produce un deadlock cuando los tokens caen en un sifón y éste no permite que los tokens salgan. Es una marca en la que no se puede disparar ninguna transición.

**Liveness o Vivacidad:** una transición  $T_i$  va a estar viva para una marca inicial  $m_0$  si para cada marca alcanzable  $m_i \in M(m_0)$  existe una secuencia de disparo  $S$  de  $m_i$  que contenga a  $T_i$ . Es decir, en algún momento la transición va a quedar sensibilizada y va a existir la posibilidad de disparar a  $T_i$ .

**Seguridad:** se dice que una red de Petri es segura para una marca inicial  $m_0$ , si para cada marca alcanzable cada plaza contiene cero o un único token.

## **Conflictos en las Redes de Petri**

En las redes de Petri, se pueden habitualmente encontrar *tres tipos* de conflictos.

En primer lugar, existe el *conflicto estructural*, que depende plenamente de la construcción y estructura de la red de Petri, sin importar su marcado. Corresponde a un conjunto de al menos 2 transiciones T1 y T2 que tienen un lugar de entrada común. Se escribe como:

$$K = \langle Pi, \{T1, T2, \dots\} \rangle$$

Por otro lado, se encuentra el de *tipo efectivo*. Se trata de aquel que aparece por las marcas en el momento en el que se efectúa algún disparo. Este conflicto se da cuando existe un conflicto estructural "K" y una marca "m", tal que el número de tokens en una plaza "Pi" es menor que el número de transiciones de salida de "Pi" que son habilitadas por "m". Se representa por  $K^E = \langle Pi, \{T1, T2, \dots\}, m \rangle$ .

Finalmente, existe el *conflicto general*, en el cual se pueden disparar las N transiciones que dependan de una misma plaza Pi, pero no se pueden disparar en simultáneo porque la cantidad de marcas en Pi es menor a la suma de todas las que se requerirían para eso. Es decir, el número de tokens en Pi no es suficiente para disparar todas las transiciones de salida de Pi según sus grados habilitantes.

## **Invariantes en las Redes de Petri**

Un *invariante* es una propiedad de la red de Petri que caracteriza el marcado de las plazas o las transiciones. Los *invariantes de plaza* son un conjunto de plazas tal que la suma de sus marcas se mantiene constante. Los *invariantes de transición* son un conjunto de transiciones que forman una secuencia para retornar al estado inicial. Se buscan solamente las mínimas.



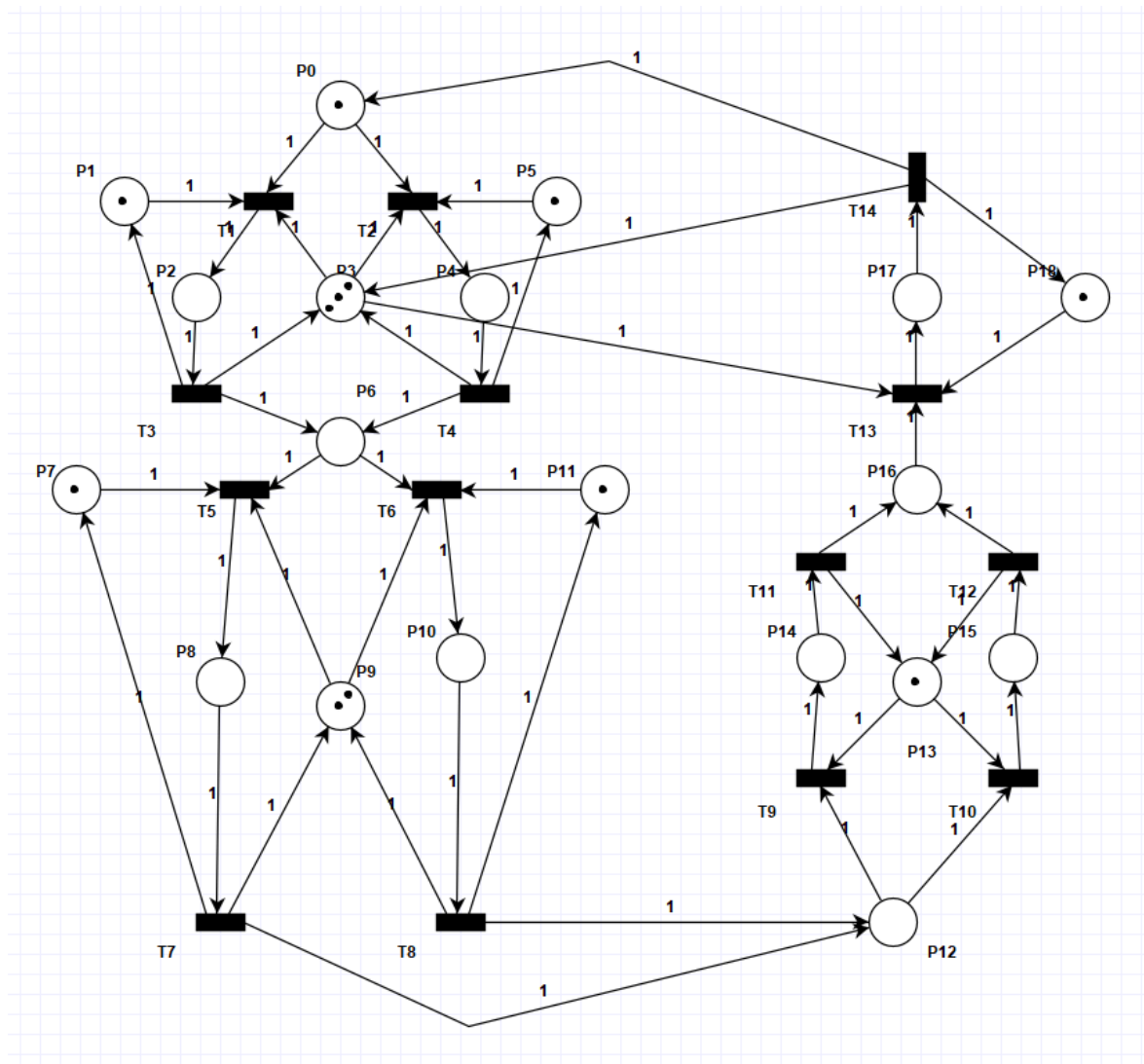


Figura 4: Red usada para el análisis

## Propiedades de la Red

**Deadlock:** la red de Petri no presenta deadlock debido a que siempre es posible sensibilizar al menos una transición que pueda dispararse.

**Vivacidad:** como no presenta deadlock, se deduce que es una red de Petri viva.

**Seguridad:** no es una red de Petri segura porque presenta plazas con más de un token.

## Tabla de estados del sistema

Plaza	Estado
P0	Buffer de entrada de imágenes a procesar

P1	Recurso compartido para cargar la imagen a procesar
P2	Carga de imagen a procesar
P3	Recurso compartido para cargar las imágenes a procesar
P4	Carga de imagen a procesar
P5	Recurso compartido para cargar las imágenes a procesar
P6	Buffer con las imágenes a ser procesadas
P7	Recurso compartido para procesar las imágenes
P8	Ajuste de calidad de las imágenes
P9	Recurso compartido para procesar las imágenes
P10	Ajuste de calidad de las imágenes
P11	Recurso compartido para procesar las imágenes
P12	Buffer con las imágenes procesadas
P13	Recurso compartido para recortar las imágenes
P14	Recorte de las imágenes
P15	Recorte de las imágenes
P16	Buffer con imagenes en estado final
P17	Exportado de imagen recortada
P18	Recurso compartido para el exportado de imágenes

**Tabla de eventos del sistema**

Transición	Evento
T0	Llegada de una nueva imagen a ser procesada
T1	Toma imagen a cargar en el contenedor
T2	Toma imagen a cargar en el contenedor
T3	Carga de imagen a procesar en el contenedor de procesamiento
T4	Carga de imagen a procesar en el contenedor de procesamiento

T5	Toma imagen a procesar
T6	Toma imagen a procesar
T7	Carga de imagen procesada en el contenedor de recorte de imagen
T8	Carga de imagen procesada en el contenedor de recorte de imagen
T9	Toma imagen a recortar
T10	Toma imagen a recortar
T11	Carga de imagen recortada en el contenedor de imágenes a exportar
T12	Carga de imagen recortada en el contenedor de imágenes a exportar
T13	Toma imagen a exportar
T14	Exporta imagen recortada

## Invariantes

### Invariantes de transición

Se obtienen los siguientes invariantes de transición:

T-Invariants														
	T7	T8	T6	T3	T2	T4	T9	T10	T11	T12	T13	T14	T1	T5
It8	0	1	1	0	1	1	0	1	0	1	1	1	0	0
It7	0	1	1	0	1	1	1	0	1	0	1	1	0	0
It6	1	0	0	0	1	1	0	1	0	1	1	1	0	1
It5	1	0	0	0	1	1	1	0	1	0	1	1	0	1
It4	0	1	1	1	0	0	0	1	0	1	1	1	1	0
It3	0	1	1	1	0	0	1	0	1	0	1	1	1	0
It2	1	0	0	1	0	0	0	1	0	1	1	1	1	1
It1	1	0	0	1	0	0	1	0	1	0	1	1	1	1

The net is covered by positive T-Invariants, therefore it might be bounded and live.

Figura 5: tabla de invariantes de transición.

De esta tabla y el análisis visual de la red, se puede desprender el análisis de estas ocho secuencias de disparos, donde  $m$  es una marca que pasa por las T transiciones.

$It1: m \rightarrow T1T3T5T7T9T11T13T14 \rightarrow m$

$It2: m \rightarrow T1T3T5T7T10T12T13T14 \rightarrow m$

$It3: m \rightarrow T1T3T6T8T9T11T13T14 \rightarrow m$

$It4: m \rightarrow T1T3T6T8T10T12T13T14 \rightarrow m$

$It5: m \rightarrow T2T4T5T7T9T11T13T14 \rightarrow m$

$It6: m \rightarrow T2T4T5T7T10T12T13T14 \rightarrow m$

$It7: m \rightarrow T2T4T6T8T9T11T13T14 \rightarrow m$

$It8: m \rightarrow T2T4T6T8T10T12T13T14 \rightarrow m$

Las secuencias deben tener uno de los conjuntos T1/T3 o T2/T4 (camino izquierdo y derecho respectivamente), uno de T5/T7 o T6/T8 (ídem), uno de T9/T11 o T10/T12 (ídem) y T13/T14, que es un único camino. Cualquiera de esas secuencias permite volver al comienzo del sistema.

## Invariantes de plaza

Se obtienen los siguientes invariantes de plaza:

P-Invariants																			
P6	P12	P16	P4	P1	P0	P5	P13	P7	P11	P9	P18	P3	P10	P8	P14	P15	P17	P2	
1	1	1	1	0	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1

The net is covered by positive P-Invariants, therefore it is bounded.

Figura 6: tabla de invariantes de plaza.

**P-Invariant equations**

$$\begin{aligned}
 M(P6) + M(P12) + M(P16) + M(P4) + M(P0) + M(P10) + M(P8) + M(P14) + M(P15) + M(P17) + M(P2) &= 1 \\
 M(P1) + M(P2) &= 1 \\
 M(P4) + M(P5) &= 1 \\
 M(P13) + M(P14) + M(P15) &= 1 \\
 M(P7) + M(P8) &= 1 \\
 M(P11) + M(P10) &= 1 \\
 M(P9) + M(P10) + M(P8) &= 2 \\
 M(P18) + M(P17) &= 1 \\
 M(P4) + M(P3) + M(P17) + M(P2) &= 3
 \end{aligned}$$

Figura 7: ecuaciones de invariantes de plaza.

Las ecuaciones representan diferentes casos:

$$M(P0) + M(P2) + M(P4) + M(P6) + M(P8) + M(P10) + M(P12) + M(P14) + M(P15) + M(P16) + M(P17) = 1$$

Esta ecuación representa como solo puede haber una imagen en circulación por el sistema en un momento dado.

$M(P1) + M(P2) = 1$  y  $M(P4) + M(P5) = 1$  representan como una imagen solo puede estar o en el recurso compartido de la carga de imágenes (P1, P4) o en la carga propiamente dicha (P2, P5), ya que se están acotando las plazas a un token.

$M(P13) + M(P14) + M(P15) = 1$  significa que en el sector de recorte está acotado a una imagen por vez.

$M(P7) + M(P8) = 1$  y  $M(P10) + M(P11) = 1$  muestran como una imagen sólo puede estar circulando por el ajustador de calidad (P8, P10) o en el recurso compartido correspondiente (P9, P11).

$M(P8) + M(P9) + M(P10) = 2$  hace referencia a como la etapa de ajuste de calidad está limitada a dos recursos en simultáneo.

$M(P17) + M(P18) = 1$  representa como no se puede tener una imagen en el exportador (P17) y su recurso compartido al unísono.

$M(P2) + M(P3) + M(P4) + M(P17) = 3$  muestra como la carga de imágenes está acotada a tres.

## Cálculo de hilos

Para calcular la cantidad de hilos, primero notamos las plazas asociadas a cada invariante de transición.

$$P(It1) = \{P0, P1, P2, P3, P6, P7, P8, P9, P12, P13, P14, P16, P17, P18\}$$

$$P(It2) = \{P0, P1, P2, P3, P6, P7, P8, P9, P12, P13, P15, P16, P17, P18\}$$

$$P(It3) = \{P0, P1, P2, P3, P6, P9, P10, P11, P12, P13, P14, P16, P17, P18\}$$

$$P(It4) = \{P0, P1, P2, P3, P6, P9, P10, P11, P12, P13, P15, P16, P17, P18\}$$

$$P(It5) = \{P0, P3, P4, P5, P6, P7, P8, P9, P12, P13, P14, P16, P17, P18\}$$

$$P(It6) = \{P0, P3, P4, P5, P6, P7, P8, P9, P12, P13, P15, P16, P17, P18\}$$

$$P(It7) = \{P0, P3, P4, P5, P6, P9, P10, P11, P12, P13, P14, P16, P17, P18\}$$

$$P(It8) = \{P0, P3, P4, P5, P6, P9, P10, P11, P12, P13, P15, P16, P17, P18\}$$

A continuación, se quitan las plazas:

- de restricción:  $PIt_{restricciones} = \{P1, P5, P7, P11, P18\}$
- de recursos:  $PIt_{recursos} = \{P3, P9, P13\}$
- idle:  $PIt_{idle} = \{P0, P6, P12, P16\}$

A partir de eso, las ecuaciones de las plazas de acción son como siguen:

$$P(A1) = \{P2, P8, P14, P17\}$$

$$P(A2) = \{P2, P8, P15, P17\}$$

$$P(A3) = \{P2, P10, P14, P17\}$$

$$P(A4) = \{P2, P10, P15, P17\}$$

$$P(A5) = \{P4, P8, P14, P17\}$$

$$P(A6) = \{P4, P8, P15, P17\}$$

$$P(A7) = \{P4, P10, P14, P17\}$$

$$P(A8) = \{P4, P10, P15, P17\}$$

El conjunto de plazas a utilizar es entonces:

$$P(A1) \cup P(A2) \cup P(A3) \cup P(A4) \cup P(A5) \cup P(A6) \cup P(A7) \cup P(A8) = \{P2, P4, P8, P10, P14, P15, P17\}$$

Debido a que, por el tamaño de la RdP, el software no nos proporcionaba el árbol de alcanzabilidad decidimos realizar una simulación de la misma. El resultado fue un marcado máximo de 6, por lo tanto, la cantidad máxima de hilos activos en simultáneo que puede tener nuestra RdP es de 6. Esto tiene sentido ya que la plaza P13 tiene un sólo token y por lo tanto podremos tener un solo hilo en P14 o en P15, y en las restantes plazas de acción vamos a poder tener un hilo de manera simultánea en cada una.

A continuación, se utiliza el algoritmo para determinar la responsabilidad de los hilos del sistema.

Se obtienen los siguientes segmentos de ejecución y las plazas de acción asociadas:

$$P(S_A) = \{P2\} \quad \text{Max}(M(S_A)) = 1$$

$$P(S_B) = \{P4\} \quad \text{Max}(M(S_B)) = 1$$

$$P(S_C) = \{P8\} \quad \text{Max}(M(S_C)) = 1$$

$$P(S_D) = \{P10\} \quad \text{Max}(M(S_D)) = 1$$

$$P(S_E) = \{P14\} \quad \text{Max}(M(S_E)) = 1$$

$$P(S_F) = \{P15\} \quad \text{Max}(M(S_F)) = 1$$

$$P(S_G) = \{P17\} \quad \text{Max}(M(S_G)) = 1$$





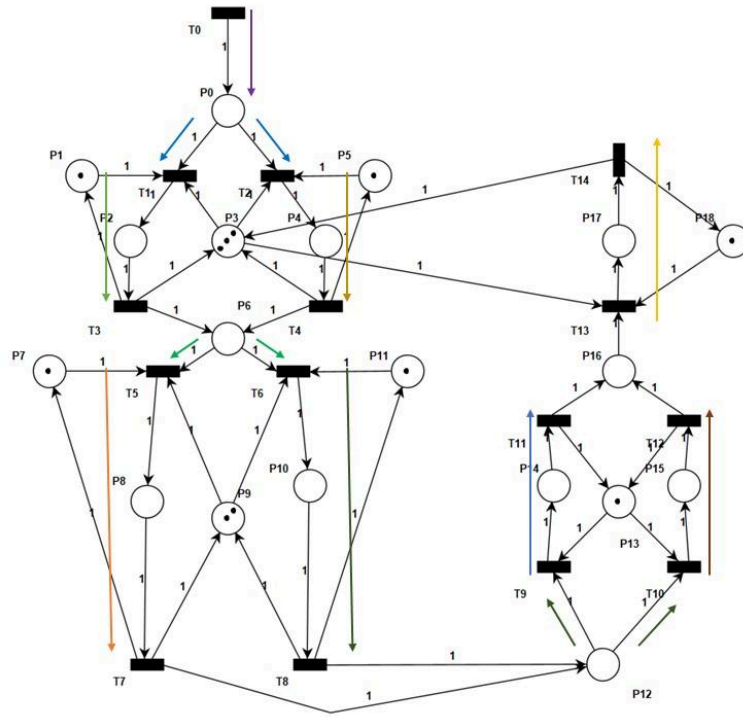


Figura 9a: Hilos del sistema según la determinación teórica.

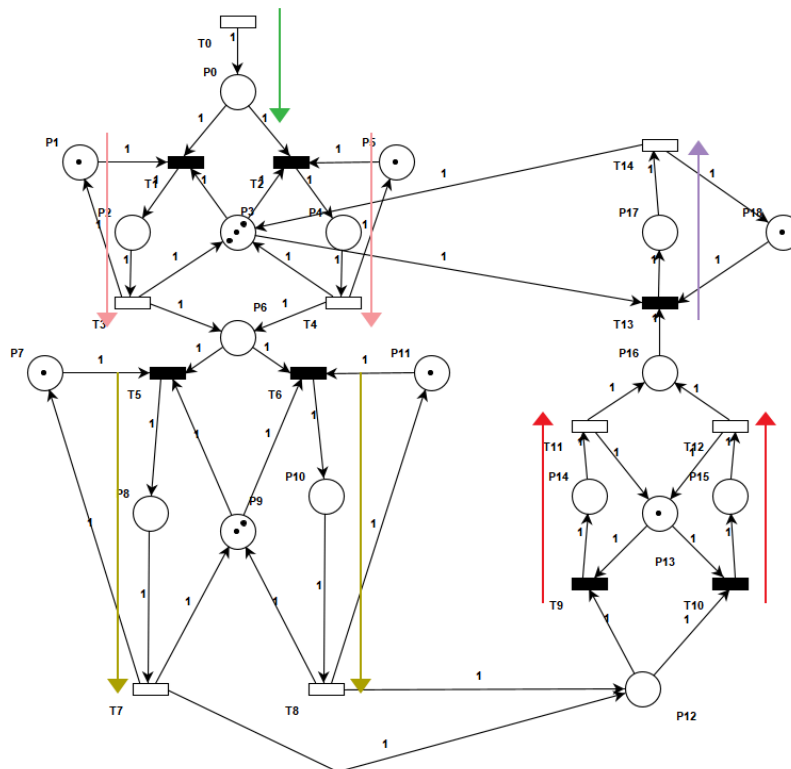
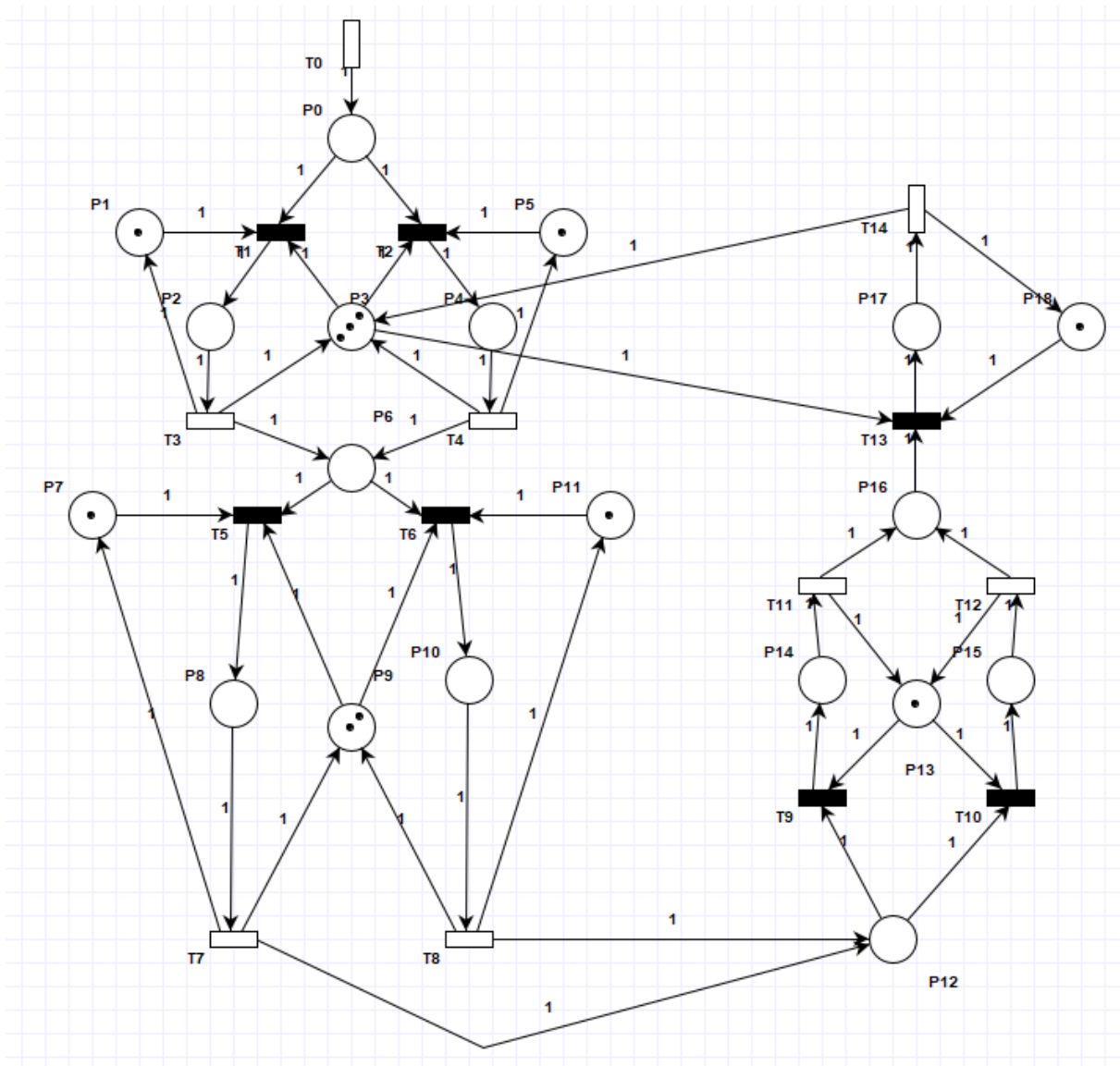


Figura 9b: Hilos del sistema implementados en el proyecto

## Determinación de los tiempos



*Figura 10: red de Petri temporizada.*

### Determinación teórica

Para la determinación del tiempo mínimo tomamos el tiempo máximo entre las transiciones temporizadas, dado que asumimos paralelismo puro donde el factor limitante es únicamente esa transición, Por lo tanto, para 200 invariantes tenemos:

$$t_{total\_min} = 200 * \max(t_{T_i})$$

En cambio, para calcular el tiempo máximo, tomamos el tiempo de cada transición en un invariante (donde las transiciones espejadas tienen tiempos iguales) será:

$$t_{total\_max} = 200 * t_{T0} + 200 * t_{max(T3,T4)} + 200 * t_{max(T7,T8)} + 200 * t_{max(T11,T12)} + 200 * t_{T14}$$

En la fórmula anterior, se tiene en cuenta que cuando se trabaja con una política balanceada existe lo siguiente:

$$t_{max(T3,T4)} = t_{T3} = t_{T4}$$

$$t_{max(T7,T8)} = t_{T7} = t_{T8}$$

$$t_{max(T11,T12)} = t_{T11} = t_{T12}$$

Por el contrario, cuando trabajamos con una política de izquierda favorecida tendremos lo siguiente:

$$t_{max(T3,T4)} = t_{T3} = t_{T4}$$

$$t_{max(T7,T8)} = t_{T7} = t_{T8}$$

$$t_{max(T11,T12)} = t_{T12}$$

Esto es debido a que necesitamos que el segmento izquierdo sea más rápido.

A continuación, se presentan los resultados de los tiempos de ejecución aplicando la **política balanceada** en cinco computadoras diferentes, y se detallan las características de cada una:

IDE: Eclipse, versión: 2024-06 (4.32.0)

- P1: CPU i7 6700 3.4 GHz (4.0 GHz turbo) cache 8mb, 8 Threads, 16gb ram ddr4 3200 mHz
- P2: CPU i5 11600KF 3.9 GHz (4.9 GHz turbo) cache 12mb, 12 Threads, 32gb ram ddr4 3200 mHz
- P3: CPU i7 10700K 3.8 GHz (4.80 GHz turbo) cache 16mb, 16 Threads, 32gb ram ddr4 3200 mHz
- P4: CPU i5-10210U 1.6 GHz (4.2 GHz turbo) cache 6mb, 8 Threads, 16gb ddr4 2667 mHz
- P5: CPU i5-7200U 2.5 GHz (3.1 GHz turbo) cache 4mb, 4 Threads, 8gb ddr4 2133 mHz

	Alfa[ms]	Beta [ms]	Tmin [s]	Tmax [s]	P1[s]	P2[s]	P3[s]	P4[s]	P5[s]
1	10, 0, 0, 10, 10, 0, 0, 20, 20, 0, 0, 20, 20, 0, 10	500,0,0,500,500 ,0,0,500,500,0,0 ,500,500,0,500	4	14	4,45	6,31	6,35	6,87	6,75
2	10, 0, 0, 10, 10, 0, 0, 20, 20, 0, 0, 20, 20, 0, 10	5000,0,0,5000,5 000,0,0,5000,50 00,0,0,5000,500 0,0,5000	4	14	4,43	6,26	6,32	7,08	6,74
3	100, 0, 0, 100, 100, 0, 0, 200, 200, 0, 0, 200, 200, 0, 100	8000,0,0,8000,8 000,0,0,8000,80 00,0,0,8000,800 0,0,8000	40	140	40,79	41,34	41,26	43,3 2	42,03
4	5, 0, 0, 5, 5, 0, 0, 10, 10, 0, 0, 10, 10, 0, 5	5000,0,0,5000,5 000,0,0,5000,50 00,0,0,5000,500 0,0,5000	2	7	2,3	2,73	2,87	2,81	2,92

Figura 11: tabla de tiempos para cada computadora con política balanceada.

En la siguiente tabla, se muestran los resultados para la política de **izquierda favorecida**:

	Alfa[ms]	Beta [ms]	Tmin [s]	Tmax [s]	P1[s]	P2[s]	P3[s]	P4[s]	P5[s]
1	10, 0, 0, 10, 10, 0, 0, 20, 20, 0, 0, 5, 20, 0, 10	500,0,0,500,500 ,0,0,500,500,0,0 ,500,500,0,500	4	14	4,30	5,73	5,27	5,89	5,57
2	10, 0, 0, 10, 10, 0, 0, 20, 20, 0, 0, 5, 20, 0, 10	5000,0,0,5000,5 000,0,0,5000,50 00,0,0,5000,500 0,0,5000	4	14	4,31	5,63	5,37	5,50	5,47
3	100, 0, 0, 100, 100, 0, 0, 200, 200, 0, 0, 50, 200, 0, 100	8000,0,0,8000,8 000,0,0,8000,80 00,0,0,8000,800 0,0,8000	40	140	40,65	41,36	41,23	41,5 3	41,96
4	5, 0, 0, 5, 5, 0, 0, 10, 10, 0, 0, 2, 10, 0, 5	5000,0,0,5000,5 000,0,0,5000,50 00,0,0,5000,500 0,0,5000	2	7	2,29	2,57	2,70	2,85	2,65

Figura 12: tabla de tiempos para cada computadora con política izquierda favorecida.

Podemos ver que se cumple, para ambas políticas, que los tiempos obtenidos se encuentran dentro de los tiempos mínimos y máximos calculados.

Por otro lado, se observa que la política de izquierda favorecida es ligeramente mas rapida que la balanceada, esto se debe a que para mantener los segmentos balanceados no era suficiente realizar la elección ante el conflicto dado que con tiempos iguales en T11 y T12 este no se presentaba lo suficiente, por lo que se hizo una disminución del alfa en la transición T11, agilizando los invariantes que la contienen y por tanto también la ejecución del programa.

## Diagramas UML

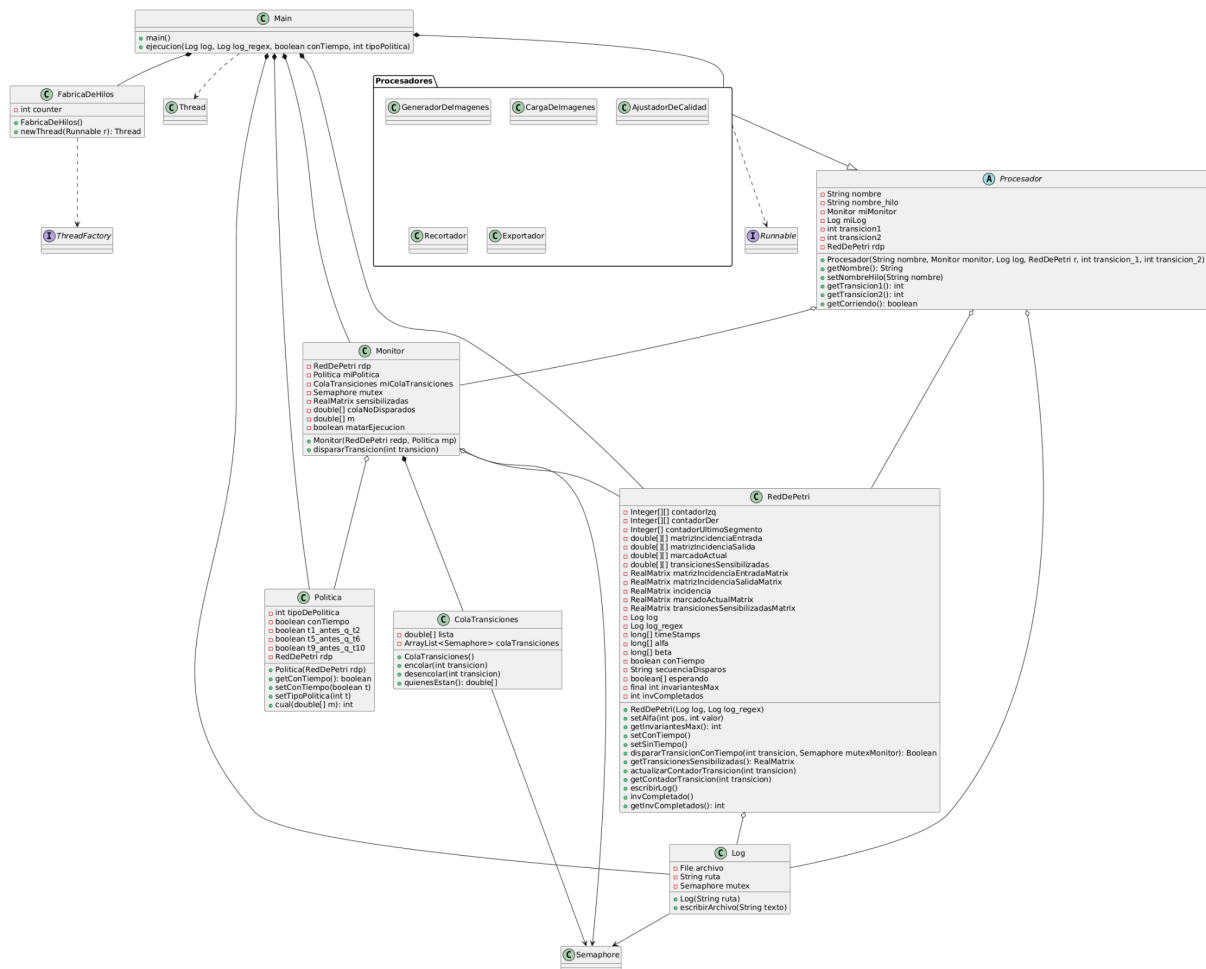


Figura 12: diagrama de clases.

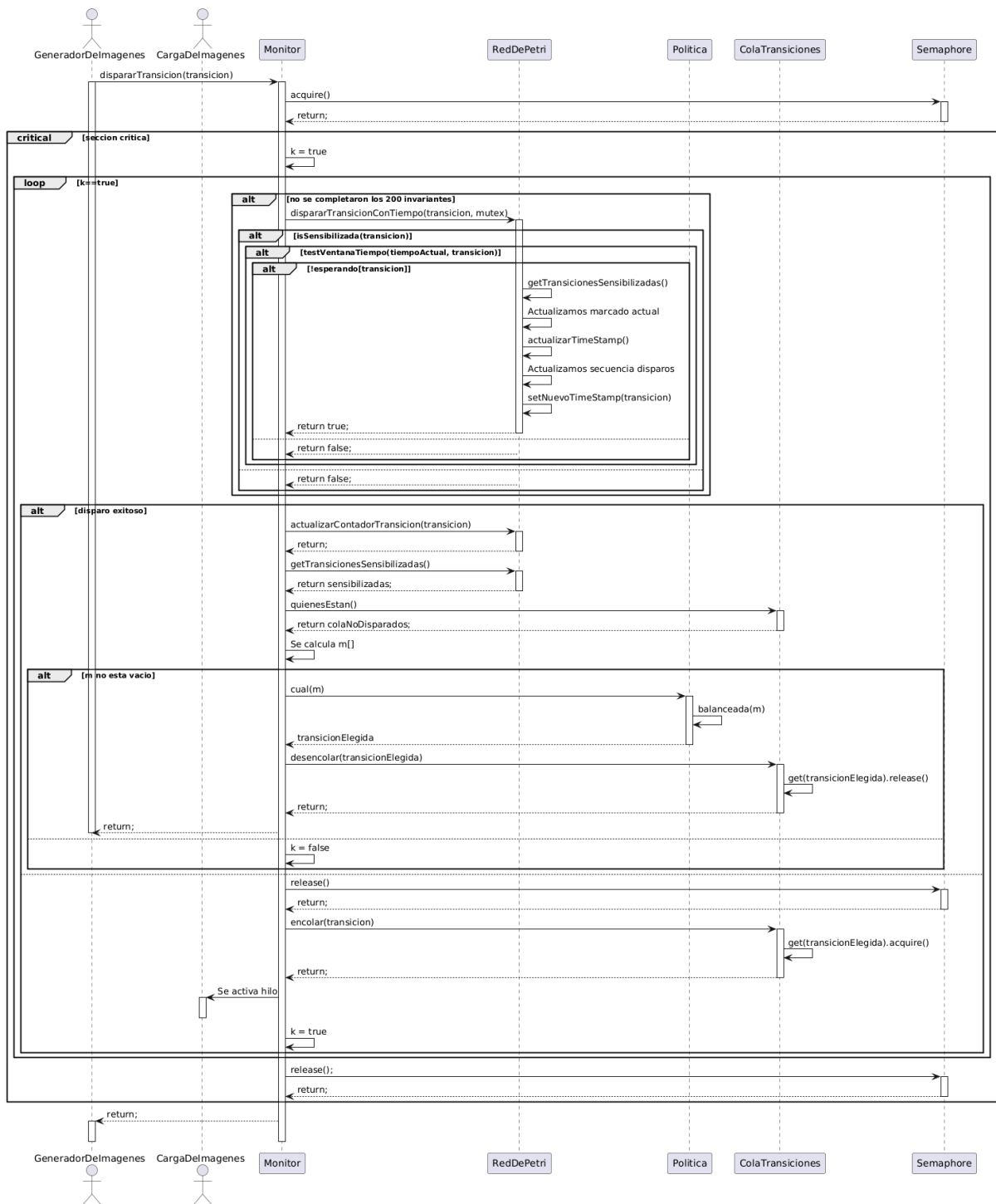


Figura 13: diagrama de secuencias.



### Análisis de tiempos de transiciones por segmento

Previo a las correcciones en la defensa del trabajo, realizamos un análisis a partir de ciertas características que observamos sobre cómo afectan los tiempos de determinadas transiciones sobre el tiempo total y el balance de segmentos con respecto a la política. Para dicho análisis se deja de lado la configuración principal de los tiempos, estableciendo el mismo valor de  $\alpha$  (10 milisegundos) y un  $\beta \gg 0$  lo suficientemente grande para que no influya sobre el tiempo a estudiar.

Con esta base, se variaron los valores de alfa para los distintos grupos de transiciones temporizadas que contiene nuestra red, tales grupos afectan a un mismo segmento.

$$\{ [T3, T4], [T7, T8], [T11, T12], T14 \}$$

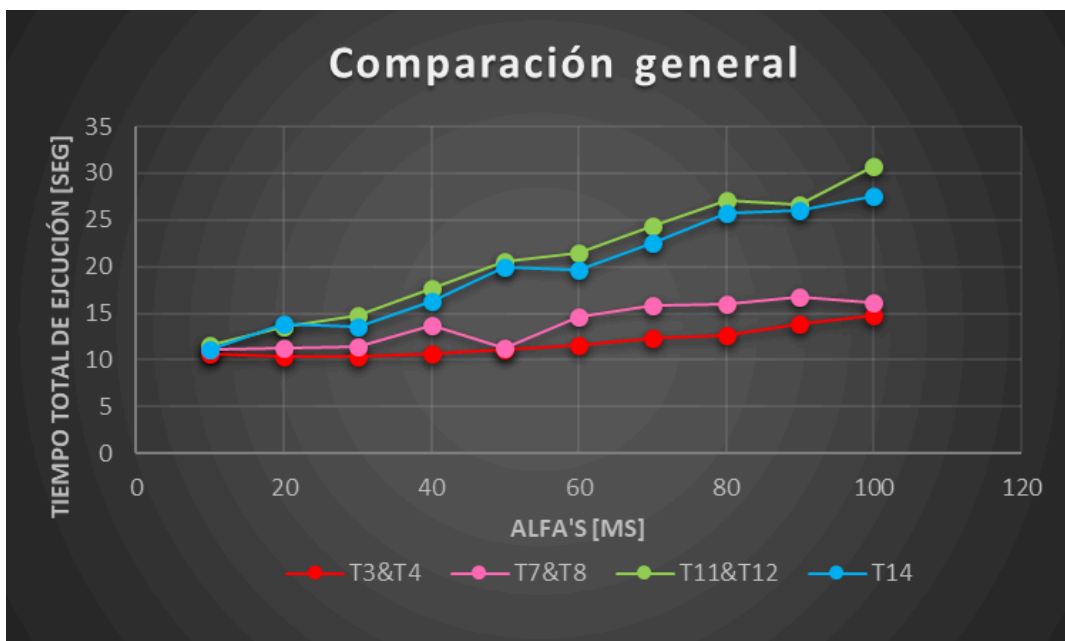


Figura 14: comparación de tiempo total de ejecución para cada par de segmentos.

De aquí resultaba claramente que la configuración temporal y nuestra implementación anterior, hacía que la variación de T11 y T12 impacte fuertemente sobre el tiempo total a comparación de las demás. También T14, lo que tenía un poco más de sentido ya que es la última transición de la red y si su tiempo es mayor, lo será directamente el tiempo total. Esto nos daba una guía para ajustar los tiempos de estas transiciones para que las políticas, tiempos y carga de segmentos, trabajen adecuadamente.

Luego de las correcciones volvimos a realizar este análisis, y podemos observar que el monitor administra mucho mejor las cargas en relación al tiempo y las políticas (si bien ya teníamos valores más bajos en las ejecuciones de prueba). El gráfico refleja que el grupo de  $\{ [T3, T4], [T7, T8] \}$  actúa muy similar. Mientras que ahora en el grupo de T11 y T12 el impacto sobre el tiempo de ejecución total no es tan fuerte. Si lo sigue siendo T14 que como

dijimos, es la última transición de la red por lo que su aumento impacta directamente en el final.

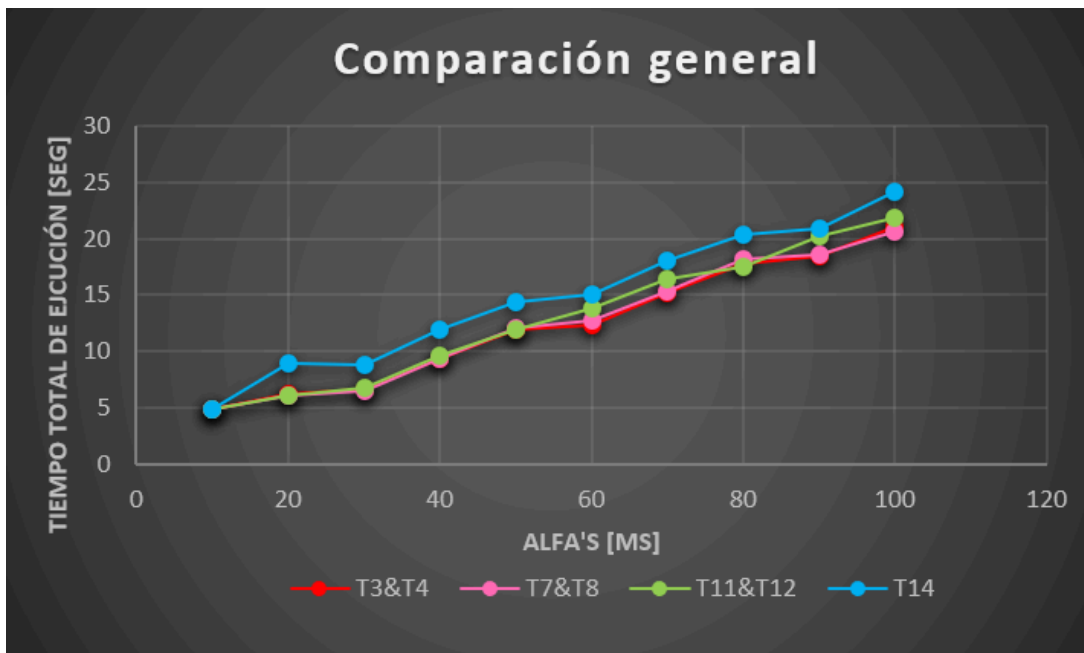


Figura 15: comparación de tiempo total post correcciones.

## Análisis de invariantes de transición

Se realizó un script en lenguaje Python para revisar que todos los disparos realizados por el monitor formen parte de algún invariante de transición mediante la siguiente expresión regular:

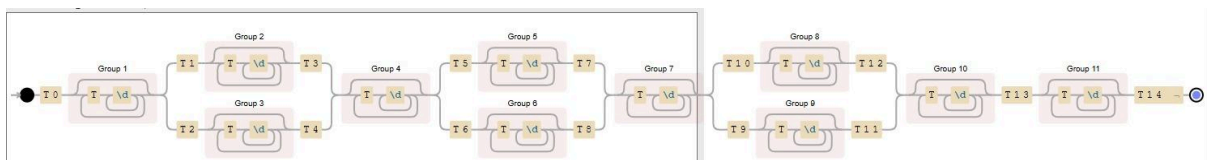


Figura 16: Regex visualizada en Debuggex.

$T0((?:T(?:\d+))*)(?:T1(?:\d+))*T3|T2(?:\d+))*T4)((?:T(?:\d+))*)(?:T5(?:\d+))*T7|T6(?:\d+))*T8)((?:T(?:\d+))*)(?:T10(?:\d+))*T12|T9(?:\d+))*T11)((?:T(?:\d+))*T13(?:\d+))*T14$

Mediante el script realizado en Python pudimos obtener que los invariantes de transición, para una política balanceada, se completaron de la siguiente forma:

```

Contador de invariantes:
El invariante 1 se completo: 35 veces.
El invariante 2 se completo: 21 veces.
El invariante 3 se completo: 21 veces.
El invariante 4 se completo: 23 veces.
El invariante 5 se completo: 19 veces.
El invariante 6 se completo: 25 veces.
El invariante 7 se completo: 25 veces.
El invariante 8 se completo: 31 veces.

```

Figura 17: Log de invariantes en una ejecución

Y podemos ver que el script nos indica que todos los disparos forman parte de algún invariante de transición:

```

LAS TRANSICIONES DE LOS INVARIANTES CORRECTAS PARA: log_regex1.txt
[Done] exited with code=0 in 0.574 seconds

```

Figura 18: Salida correcta en script de Python.

### Análisis de carga de invariantes

#### Política equitativa

Esta política asegura que para cada par de segmentos se cumpla la condición del 50% en la elección del camino, siempre y cuando estén encoladas ambas transiciones de conflicto. En caso de no existir casos de decisión, se toma un valor al azar de las transiciones que están listas para poder dispararse. Por ejemplo, en el segmento A y B se chequea la decisión según T1 y T2.

Los invariantes pares se corresponden a segmentos de la derecha en el tramo 9/10/11/12 y los invariantes impares, de la izquierda, por lo que se calcula de esta manera el balance derecha/izquierda de la ejecución.

Ej / Inv	1	2	3	4	5	6	7	8	Izq %
1	35	21	21	23	19	25	25	31	50
2	23	33	24	19	24	20	26	31	48,5
3	20	27	29	25	21	31	30	17	50
4	23	31	27	18	21	25	28	27	49,5
5	19	21	32	28	23	39	25	13	49,5

*Figura 19: tabla de invariantes de transición en 5 ejecuciones de política equitativa.*

En promedio, se obtuvo 49,5% por izquierda y 50,5% por derecha, lo cual representa una desviación del 0,5% de una distribución 50/50.

### **Izquierda favorecida**

Esta política asegura que para cada par de segmentos se cumpla la condición del 50% en la elección del camino, siempre y cuando estén encoladas ambas transiciones de conflicto y con la excepción del tramo T9/10/11/12, en el cual se ejecutará con 80% por la izquierda (T9/T11) y 20% por la derecha (T10/T12).

Los invariantes pares se corresponden a segmentos de la derecha en el tramo 9/10/11/12 y los invariantes impares, de la izquierda, por lo que se calcula de esta manera el balance derecha/izquierda de la ejecución.

Ej / Inv	1	2	3	4	5	6	7	8	Izq %
1	38	12	37	12	40	9	39	13	77
2	38	11	38	13	44	8	37	11	78,5
3	41	11	38	10	36	13	41	10	78
4	36	17	41	6	42	6	38	14	78,5
5	45	9	38	9	35	13	40	11	79

*Figura 20: tabla de invariantes de transición en 5 ejecuciones de política izquierda favorecida.*

En promedio, se obtuvo 78,2% por izquierda y 21,8% por derecha, lo cual representa una desviación menor al 2% de una distribución 80/20.

## Conclusión

Con este trabajo fue posible aplicar los conocimientos aprendidos en la materia para el modelado y manejo de concurrencia y recursos compartidos. En base a los mismos, se implementó un monitor de concurrencia, con una política asociada que permitió separar las decisiones de concurrencia de las acciones realizadas por los hilos.

Como conclusión principal, pudimos observar teóricamente como el paralelismo permite tener tiempos 250% más rápidos que una ejecución secuencial ( $\frac{T_{max}}{T_{min}} \times 100 - 100$ ), en todos los casos analizados. Pudimos ver que los resultados obtenidos, utilizando concurrencia, se encontraron más próximos al tiempo de ejecución en paralelo que al tiempo de ejecución secuencial. Esto demuestra que es posible obtener tiempos mucho más bajos que el secuencializado utilizando concurrencia, incluso cuando hay tiempos que no se tienen en cuenta en el cálculo teórico como son las demoras causadas por el tiempo perdido en tareas auxiliares.

Durante las ejecuciones del sistema, con respecto a las dos políticas implementadas, “balanceada” e “izquierda favorecida”, se pudo observar en ambos casos que la carga de los segmentos tiene solo un desvío menor al 1% aprox. del 50% que se esperaba para los segmentos izquierdo y derecho. Por este motivo, se puede deducir que funcionan correctamente, con cada segmento procesando aproximadamente el porcentaje requerido. En cuanto al caso de la política “izquierda favorecida”, la desviación que se obtuvo es menor al 2%, de las distribuciones esperadas de 20% de carga para el segmento derecho y 80% del segmento izquierdo.

Para la ejecución del sistema de procesamiento de imágenes, limitamos los disparos de transiciones al cumplimiento de 200 invariantes de transición. Con la expresión regular realizada, tras aplicarla al registro (en el archivo log) de todas las transiciones disparadas por el monitor, se verificó que todos los disparos de transición forman parte de algún invariante de transición. Además, se deduce que todos los tokens ingresados a la Red de Petri, fueron consumidos por el disparo de alguna transición perteneciente a algún invariante de transición.

## **Bibliografía**

Apuntes de la materia en aula virtual.

[https://www.researchgate.net/publication/358104149\\_Algoritmos\\_para\\_determinar\\_cantidad\\_y\\_responsabilidad\\_de\\_hilos\\_en\\_sistemas\\_embebidos\\_modelados\\_con\\_Redes\\_de\\_Petri\\_S\\_3\\_PR?enrichId=rgreq-60283978208390559e3c61c7c14c0c45-XXX&enrichSource=Y292ZXJQYWdIOzM1ODEwNDE0OTtBUzoxMTE2MTQ3MjU2NzA1MDI0QDE2NDMxMjE2NTE4MTQ%3D&el=1\\_x\\_2&\\_esc=publicationCoverPdf](https://www.researchgate.net/publication/358104149_Algoritmos_para_determinar_cantidad_y_responsabilidad_de_hilos_en_sistemas_embebidos_modelados_con_Redes_de_Petri_S_3_PR?enrichId=rgreq-60283978208390559e3c61c7c14c0c45-XXX&enrichSource=Y292ZXJQYWdIOzM1ODEwNDE0OTtBUzoxMTE2MTQ3MjU2NzA1MDI0QDE2NDMxMjE2NTE4MTQ%3D&el=1_x_2&_esc=publicationCoverPdf)

<https://docs.oracle.com/javase/8/docs/api/>

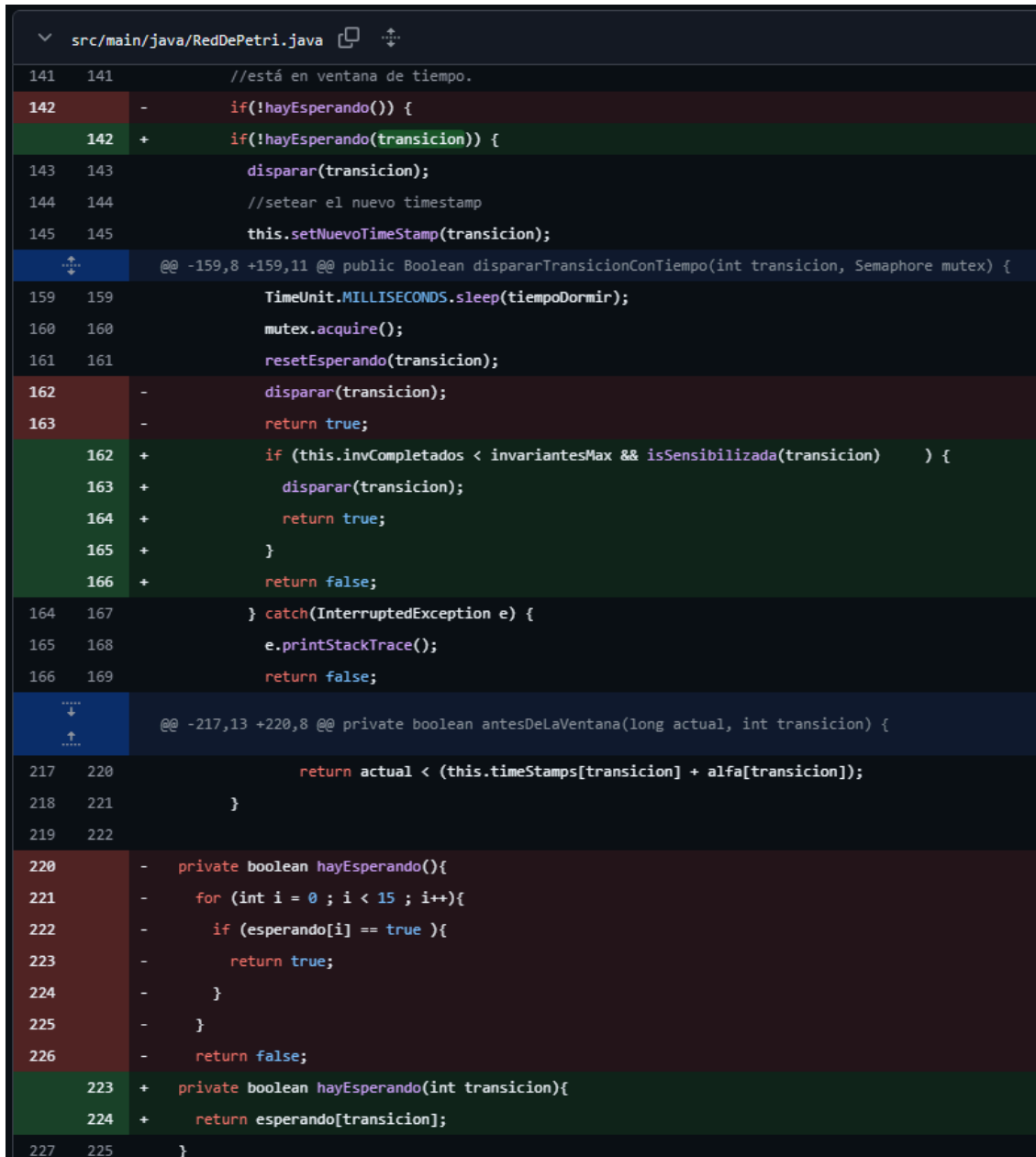
## **Repositorio de GitHub**

[Repositorio de Github](#)

# Anexo

## Correcciones post presentación 03/02/25

### Cambios realizados en el código



```
src/main/java/RedDePetri.java
141 141 //está en ventana de tiempo.
142 - if(!hayEsperando()) {
142 + if(!hayEsperando(transicion)) {
143 143 disparar(transicion);
144 144 //setear el nuevo timestamp
145 145 this.setNuevoTimeStamp(transicion);
@@ -159,8 +159,11 @@ public Boolean dispararTransicionConTiempo(int transicion, Semaphore mutex) {
159 159 TimeUnit.MILLISECONDS.sleep(tiempoDormir);
160 160 mutex.acquire();
161 161 resetEsperando(transicion);
162 - disparar(transicion);
163 - return true;
162 + if (this.invCompletados < invariantesMax && isSensibilizada(transicion) ) {
163 + disparar(transicion);
164 + return true;
165 + }
166 + return false;
164 167 } catch (InterruptedException e) {
165 168 e.printStackTrace();
166 169 return false;
@@ -217,13 +220,8 @@ private boolean antesDeLaVentana(long actual, int transicion) {
217 220 return actual < (this.timeStamps[transicion] + alfa[transicion]);
218 221 }
219 222
220 - private boolean hayEsperando(){
221 - for (int i = 0 ; i < 15 ; i++){
222 - if (esperando[i] == true ){
223 - return true;
224 - }
225 - }
226 - return false;
223 + private boolean hayEsperando(int transicion){
224 + return esperando[transicion];
227 225 }
```

Figura A1: Correcciones aplicadas post presentación sobre la clase RedDePetri

En primer lugar, se modificó la función “hayEsperando()” por “hayEsperando(transicion)” ya que, anteriormente, se estaba revisando si cualquier transición estaba esperando en lugar de preguntar por la transición que se va a disparar. Esto no permitía el avance de la red mientras una transición estaba durmiendo. Ahora, se revisa si efectivamente esa transición tiene algún hilo esperando, y si es ese el caso, no podría disparar y devolvería false.

Se supone que existe una transición A sensibilizada que llega al disparo, pero se encuentra antes de la ventana de tiempo. Anteriormente, se seteaba el *esperando[]* y luego al hacer esto, cualquier otra transición B que intentase disparar no iba a poder hacerlo, porque el chequeo era para cualquier transición en espera. Ahora, como el chequeo del *esperando[]* revisa la transición individualmente, podría suceder que otra transición B llegue mientras la transición A estaba dormida, se dispare y luego haya un cambio en la red tal que la transición A deje de estar sensibilizada. Por esto mismo, se agregó un chequeo para revisar nuevamente si las transiciones continúan sensibilizadas tras pasar tiempo dormidas.

```

Elige un numero de politica:

1. Balanceada

2. Izquierda favorecida

Ejecución 1:
Completada con éxito.
Tiempo de ejecución: 4399 milisegundos.
Ejecución 2:
Completada con éxito.
Tiempo de ejecución: 4355 milisegundos.
Ejecución 3:
Completada con éxito.
Tiempo de ejecución: 4378 milisegundos.
Ejecución 4:
Completada con éxito.
Tiempo de ejecución: 4351 milisegundos.
Ejecución 5:
Completada con éxito.
Tiempo de ejecución: 4350 milisegundos.
Tiempo promedio de ejecución: 4367 milisegundos.

```

Figura A2: Pool de ejecuciones.

```

LAS TRANSICIONES DE LOS INVARIANTES CORRECTAS PARA: log_regex1.txt

Contador de invariantes:
El invariante 1 se completó: 26 veces.
El invariante 2 se completó: 23 veces.
El invariante 3 se completó: 25 veces.
El invariante 4 se completó: 26 veces.
El invariante 5 se completó: 28 veces.
El invariante 6 se completó: 23 veces.
El invariante 7 se completó: 21 veces.
El invariante 8 se completó: 28 veces.

Segmento A(T1-T3): 100
Segmento B(T2-T4): 100
Segmento C(T5-T7): 100
Segmento D(T6-T8): 100
Segmento E(T9-T11): 100
Segmento F(T10-T12): 100
Segmento G(T13-T14): 200

```

Figura A3. Salida de script con regex para una ejecución.