UNIVERSIDAD NACIONAL DE CÓRDOBA
Facultad ciencias exactas, físicas y naturales

TRABAJO FINAL PROGRAMACIÓN CONCURRENTE 2025

INTEGRANTES:

Benavides, María Candela - 45559700

Cechich. Federico - 43372534

Fariñas Gómez, Rafael - 52096795

Salinas, Joaquín Alejandro - 45211874

PROFESORES:

Ing. Luis Orlando Ventre

Ing. Mauricio Ludemann

Aspirante Adscripto: Ing. Agustin Carranza.

Introducción

En este trabajo se desarrolla el modelado, análisis e implementación de un sistema concurrente basado en una red de Petri que representa un sistema de procesamiento de datos. El objetivo central es estudiar el funcionamiento del modelo, identificar sus propiedades y reproducir su comportamiento mediante una simulación en Java que haga uso de un monitor de concurrencia, garantizando la correcta coordinación entre los distintos hilos que intervienen en el sistema.

Como primer paso, se realiza un análisis teórico de la red utilizando la herramienta PIPE, evaluando propiedades como deadlock, vivacidad y seguridad, e identificando los invariantes de plaza y de transición. Estos invariantes permiten comprender cómo se conserva el flujo de tokens dentro del sistema y resultan fundamentales para la verificación posterior durante la simulación.

A continuación, se desarrolla una implementación orientada a objetos en Java, donde el comportamiento de la red se reproduce mediante un monitor genérico capaz de gestionar el disparo de transiciones sin depender de una red específica. Se construyen tablas de estados y eventos, y se determina la cantidad adecuada de hilos que deben participar en la ejecución, siguiendo los criterios establecidos para maximizar el paralelismo. Además, cada hilo asume responsabilidades concretas de acuerdo con los invariantes de transición presentes en la red, acompañándose con un gráfico que muestra visualmente esta distribución.

El trabajo también incorpora la semántica temporal requerida: las transiciones temporizadas se implementan con tiempos asignados, lo que permite realizar un análisis tanto analítico como experimental sobre el impacto de estas demoras en la ejecución total del sistema. Paralelamente, se implementan dos políticas de resolución de conflictos —una aleatoria y otra priorizando el modo simple de procesamiento— para estudiar cómo influyen en la distribución de carga y en el uso de los distintos invariantes durante múltiples ejecuciones.

Finalmente, se registran los resultados mediante archivos de log y se verifican los invariantes de plaza y de transición a lo largo de la ejecución, analizando el cumplimiento de las propiedades del modelo. Con esto, el trabajo integra de manera completa el análisis formal de la red con su implementación concurrente, permitiendo observar en detalle cómo se comporta el sistema bajo distintas configuraciones, políticas y condiciones temporales.

Marco Teórico

Redes de petri

Es una abstracción formal matemática que permite modelar y visualizar comportamientos con paralelismo, concurrencia, sincronización e intercambios de recursos. Es un grafo bipartito, es decir, los lugares y las transiciones se alternan en una ruta formada por arcos consecutivos.

Componentes de una Red de Petri

- Arco: es la unión entre dos nodos y solo pueden tener dos direcciones, de una plaza a una transición o viceversa.

- Plaza: es uno de los dos tipos de nodos que conforman una red de petri. Estos representan condiciones, estados o recursos del sistema.

- Transición: es uno de los dos tipos de nodos y representa un evento, acción o cambio de estado. Actúa como detector de condiciones y transformador estado: cuando las condiciones se cumplen, se dispara y el estado de la red de petri cambia.

- Marcado: es un vector que describe la distribución de tokens en las plazas y por lo tanto define el estado del sistema modelado por la red de petri.

Evolución del Sistema

El comportamiento del sistema se expresa a través de la evolución del marcado, la cual se produce cuando una transición se dispara. Para que esto ocurra, es necesario verificar que cada plaza de entrada de la transición tenga al menos la misma cantidad de tokens (que representan la disponibilidad del recurso) que el peso del arco que las conecta.
Si esta condición se cumple, la transición puede dispararse: se consumen de las plazas de entrada los tokens correspondientes y se generan tokens en las plazas de salida según los pesos de sus arcos. Este mecanismo determina cómo cambia el estado del sistema y permite modelar la secuencia de eventos posibles.

Propiedades de una Red de Petri

Vivacidad (Liveness)

Una transición Tj está 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 contiene la transición Tj .
La vivacidad depende tanto del marcado como de la estructura de la red.

Una red está viva si ninguna transición queda permanentemente bloqueada

Deadlock

Es una marca tal que no se puede disparar ninguna transición.
Se dice que una red de petri está libre de interbloqueo para una marca inicial m_0 si ninguna marca alcanzable $m_i \in M(m_0)$ es un Deadlock

Acotación (Boundedness)

Es cuando una plaza P_i está acotada respecto del marcado inicial m_0 si existe un entero natural k tal que, para todas las marcas alcanzables desde m_0, la cantidad de tokens en P_i nunca es mayor que k .
Por lo tanto, una RdP limitada es acotada si, dado un marcado inicial m_0 , todas sus plazas están acotadas por k

Segura (Safeness)

Es cuando para cada marca alcanzable, cada lugar contiene cero o un token. Es un caso particular de una RdP acotada para la cual todos los lugares tienen un límite de 1

Reversibilidad

Una red es reversible si, desde cualquier marcado alcanzable, es posible volver al marcado inicial

Persistencia

Garantiza que el disparo de una transición habilitada no deshabilite a otra transición que también estaba habilitada de manera independientemente. Es decir, una transición sólo puede deshabilitarse por su propio disparo y no por el disparo de otra transición

Representación matemática

Para analizar de manera formal el comportamiento de una red de petri, es útil representar mediante matrices que describen cómo interactúan las plazas y las transiciones. Estas matrices permiten expresar de forma algebraica el consumo y la producción de tokens, así como el cambio de marcada cuando una transición se dispara. A continuación, se presentan las principales matrices utilizadas en la representación matemática de una red de Petri

Matriz de Pre-incidencia (W^-)

La matriz de pre-incidencia W^- indica cuántos tokens debe consumir una transición de cada plaza para poder dispararse. Cada elemento $W^-(p,t)$ representa el peso del arco que va desde la plaza p hacia la transición t

Matriz de Pos-incidencia (W^+)

La matriz de pos-incidencia W^+ indica cuántos tokens produce una transición en cada plaza al dispararse. Cada elemento $W^+(p,t)$ representa el peso del arco que va desde la transición t hacia la plaza p

Matriz de Incidencia (W)

La matriz de incidencia W describe el cambio neto de tokens que ocurre en una plaza cuando una transición se dispara. Se define como:

$$W = W^+ - W^-$$

Cada elemento $W\ (p,t)$ expresa la variación total en la plaza p tras el disparo de la transición t

$W\ (p,t)$ > 0 : la transición agrega tokens

$W\ (p,t)$ < 0 : la transición consume tokens

$W\ (p,t)$ = 0 : no afecta esa plaza

A partir de estas matrices, es posible expresar formalmente como evoluciona el marcado de la red utilizando la llamada ecuación fundamental. Esta ecuación permite calcular el nuevo marcado después del disparo de una transición, considerando el consumo y la producción de tokens en cada plaza. De esta manera, el comportamiento dinámico de la red puede describirse de forma algebraica, vinculando las matrices definidas anteriormente con la variación del estado del sistema

$$Ecuación\ fundamental \quad \rightarrow \quad m_k = m_i + Ws$$

Sea S una secuencia de disparo que lleva a la red desde un estado m_i al m_k.

El vector característico de la secuencia S, escrito como s, es el vector de componente m cuyo número de componente corresponde al número de disparos de una transición de la secuencia S

donde:

m_i = marcado actual

m_k = marcado resultante

W = matriz de incidencia

s = vector de disparo

Redes de Petri Temporizadas

Las Redes de Petri Temporizadas (TPN, Time Petri Nets) constituyen una variante de las redes de Petri clásicas en la que el tiempo pasa a ser un componente fundamental del modelo. En este tipo de redes, el comportamiento de las transiciones no solo depende de la posibilidad de tokens, sino también de restricciones temporales asociadas a su habilitación.

Este enfoque resulta especialmente útil para representar sistemas donde la sincronización, los plazos y la respuesta temporal son críticos, como ocurre en aplicaciones de control industrial, sistemas embebidos en tiempo real, telecomunicaciones o procesos productivos

Incorporación del tiempo

En una TPN, cada transición tiene asociado un intervalo temporal que determina en qué momento puede ejecutarse después de haber sido habilitada. Este intervalo está definido por dos parámetros fundamentales :

- Tiempo mínimo (α): representa la cantidad mínima de tiempo que debe transcurrir desde que la transición queda habilitada hasta que esté autorizada a dispararse.
- Tiempo máximo (β): establece el límite superior del tiempo durante el cual la transición puede permanecer habilitada sin dispararse. Si se supera este valor, la transición pierde su habilitación

Principios de funcionamiento

El comportamiento operativo de las TPN puede resumirse en las siguientes reglas:

- Habilitación: una transición queda habilitada cuando las plazas de entrada poseen la cantidad de tokens requerida por los arcos
- Espera mínima (α): aunque la transición esté habilitada, sólo podrá dispararse una vez que haya transcurrido su tiempo mínimo (α)
- Ventana temporal (β): la transición debe ejecutarse antes de que termine el tiempo β desde su habilitación, de lo contrario, deja de estar habilitada
- Disparo: al dispararse, se consume los tokens de las plazas de entrada y produce tokens en las de salida, igual que en una Red de Petri clásica, pero respetando las restricciones temporales

Monitor

Un monitor es un mecanismo de software (de alto nivel) para el control de concurrencia que contiene los datos y los procedimientos necesarios para realizar la asignación de un determinado recurso o grupo de recursos compartidos reutilizables en serie.

Componentes

- Variables compartidas: son los datos internos del monitor y representan el estado del recurso que se protege. Solo pueden ser accedidas por los procedimientos del monitor

- Procedimientos del monitor: son las operaciones mediante las cuales los procesos interactúan con el recurso. Cada procedimiento se ejecuta bajo exclusión mutua y es el único medio para consultar o modificar las variables compartidas

- Exclusión mutua implícita: garantiza que solo un proceso puede ejecutar uno de sus procedimientos a la vez, evitando accesos concurrentes al recurso

- Variables de condición: permiten que un proceso espere hasta que se cumpla una condición asociada al estado del recurso. Utilizan las operaciones $wait()$ y $signal()$ para bloquear y despertar procesos dentro del monitor

- Cola de entrada: es la cola donde esperan los procesos que quieren ingresar al monitor cuando ya hay otro proceso ejecutando un procedimiento .

Partes lógicas de un monitor

- Algoritmo para la manipulación del recurso y sincronización
- Mecanismo para la asignación del orden en el cual los procesos asociados pueden compartir el recurso y/o son sincronizados

Prioridad y equidad

Para evitar situaciones de inanición (starvation), un monitor debe asegurar que los procesos que llevan más tiempo esperando tengan prioridad por sobre aquellos que recién intentan ingresar. Si esto no se respeta, los procesos nuevos podrían adelantar continuamente a los ya bloqueados,impidiendoles acceder al recurso.
Un esquema de equidad permite garantizar que todos lo procesos obtengan eventualmente acceso al monitor

Gestión del acceso mediante colas

El ingreso al monitor se controla utilizando una cola de espera. Su funcionamiento puede resumirse de la siguiente manera:

- Cuando un proceso intenta entrar al monitor y hay otro ejecutando un procedimiento, el proceso que llega se bloquea y se coloca en la cola correspondiente
- Una vez que el proceso activo finaliza su ejecución dentro del monitor, se habilita al primer proceso de la cola para que ingrese
- Si no existen procesos esperando, el monitor queda disponible para cualquier proceso que solicite acceso

Este mecanismo garantiza un orden definido de entrada y evita que múltiples procesos compiten simultáneamente por el recurso

Políticas de concurrencia

Las políticas del monitor establecen cómo se determina el orden en que los procesos acceden a los recursos compartidos. Estas políticas combinan mecanismos de exclusión mutua con criterios de prioridad o sincronización, permitiendo decidir qué proceso debe avanzar primero.

De esta forma, procesos con mayor urgencia o tareas más sensibles pueden recibir un acceso preferencial, mientras se mantiene un comportamiento justo y coherente en la gestión de concurrencia

Desarrollo

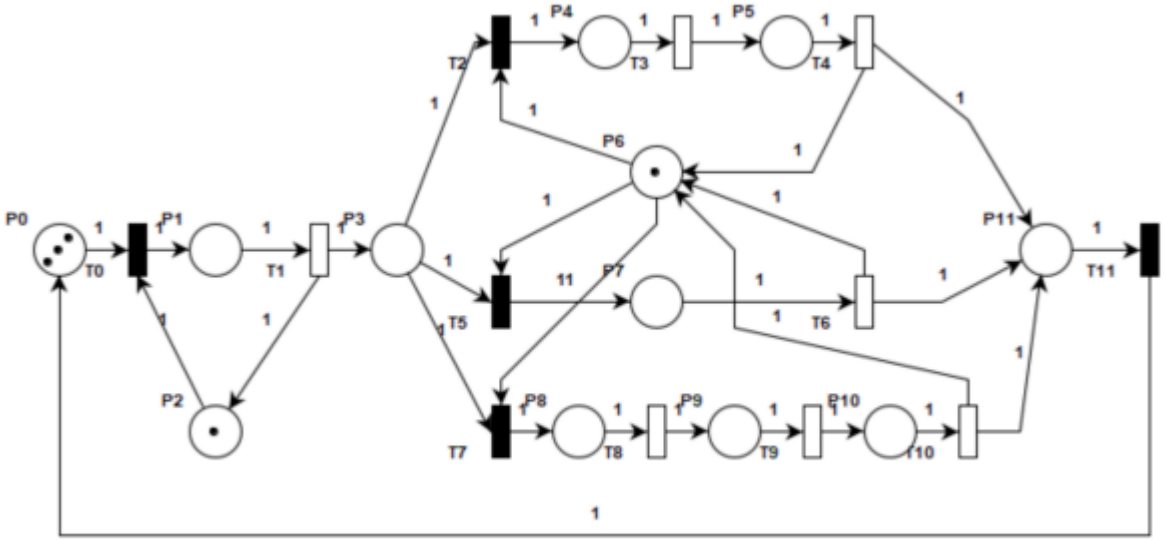


Figura 1

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

Las plazas {P2, P6} representan recursos compartidos en el sistema:

- La plaza {P2} representa el bus de acceso al buffer.
- La plaza {P6} representa la unidad de procesamiento.

Las plazas {P3, P11} representan buffers:

- La plaza {P3} representa el buffer de los datos a procesar.
- La plaza {P11} representa el buffer de salida de los datos.

La plaza {P0} es una plaza idle que corresponde a la cola de arribo de datos del sistema.

La plaza {P1} representa un dato accediendo al buffer a través del bus.
Cada dato puede procesarse por uno de los siguientes tres modos de procesamiento:

- Modo de complejidad simple: una sola etapa llevada a cabo en la plaza {P7}
- Modo de complejidad media: dos etapas llevadas a cabo en las plazas {P4, P5}
- Modo de complejidad alta: tres etapas llevadas a cabo en las plazas {P8, P9, P10}

Análisis de las propiedades de la red y sus invariantes

Se modeló la red de Petri de la Figura 1 utilizando la herramienta de simulación PIPE (Platform Independent Petri net Editor), con el objetivo de analizar y verificar las propiedades de la red, como deadlock, vivacidad y seguridad, así también como los invariantes de plaza y de transición.

Invariantes de transición

Un invariante de transición es una secuencia de disparo de transiciones de una Red de Petri que, cuando se ejecuta completa, devuelve a la red al mismo marcado en el que estaba antes de empezar.
En este caso, se identificaron 3 invariantes de transición (Fig. 2), cada uno representando un posible ciclo completo de ejecución del sistema.

1. T-invariante: {T0,T1,T2,T3,T4,T11}
2. T-invariante: {T0,T1,T5,T6,T11}
3. T-invariante: {T0,T1,T7,T8,T9,T10,T11}

Para el caso del primer invariante de transición, esta invariante corresponde al modo de procesamiento superior del sistema.
El token sube por la rama que involucra las plazas P4 y P5, avanza hacia P11 y finalmente vuelve al inicio mediante T11. Representa el ciclo completo donde un dato atraviesa la parte alta del modelo
La segunda invariante de transición representa el modo central de procesamiento.
El token pasa por el segundo modo de procesamiento (P7), llega a P11 y mediante T11 retorna al punto inicial. Corresponde al flujo más directo entre el comienzo y el final del proceso.
El tercer invariante describe el modelo inferior de procesamiento. El token recorre la parte baja del modelo (P8→ P9 → P10), luego sigue por P11 y finalmente retorna al inicio mediante T11.

T-Invariants

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

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

Fig. 2: Invariantes de transición de la Red de Petri.

Invariantes de plaza

Un invariante de plaza es una combinación lineal de plazas de una Red de Petri cuya suma ponderada de tokens permanece constante durante toda la evolución del sistema, sin importar qué transiciones se disparen.

En la red analizada se identificaron 3 invariantes de plaza (Fig.3):

1. P-invariante: M(P0) + M(P1) + M(P3) + M(P4) + M(P5) + M(P7) + M(P8) + M(P9) + M(P10) + M(P11) = 3
2. P-invariante: M(P1) + M(P2) = 1
3. P-invariante: M(P4) + M(P5) + M(P6) + M(P7) + M(P8) + M(P9) + M(P10) = 1

En el caso del primer invariante de plaza corresponde a los tokens que representan unidades del proceso principal, a través de cualquiera de las ramas. La suma total siempre es 3, lo que indica que el sistema procesa 3 elementos simultáneamente independientemente de por dónde circulan.
La segunda invariante de plaza representa un recurso exclusivo, donde solo puede haber un token activo entre las plazas P1 y P2.
Por último, el tercer invariante de plaza agrupa las plazas que pertenecen a los 3 modos alternativos de procesamiento. La suma siempre es 1, lo que implica que sólo un elemento a la vez puede encontrarse dentro de las ramas internas del proceso.

P-Invariants

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

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

P-Invariant equations

M(P0) + M(P1) + M(P10) + M(P11) + M(P3) + M(P4) + M(P5) + M(P7) + M(P8) + M(P9) = 3
M(P1) + M(P2) = 1
M(P10) + M(P4) + M(P5) + M(P6) + M(P7) + M(P8) + M(P9) = 1

Fig. 3: Invariantes de plaza de la Red de Petri.

Propiedades de la Red de Petri

Podemos observar, gracias a las herramientas de análisis de PIPE, que la red es acotada, no es segura y que está libre de deadlocks.

- La red es acotada ya que en todos los lugares, la cantidad de tokens nunca crece sin límite. Esto se debe a la existencia de invariantes de plaza positivos, los cuales garantizan que ciertas combinaciones de lugares mantienen su suma de tokens constante a lo largo de toda la ejecución.
- La red no es segura porque, aunque es acotada, hay lugares de la red donde puede haber más de un token simultáneamente. Por lo tanto, la red no cumple con la propiedad de redes de petri seguras.

- La red está libre de deadlocks debido a que siempre existe al menos una transición sensibilizada capaz de dispararse desde cualquier marcado alcanzable. Por ende, la red siempre puede seguir ejecutándose y no alcanza estados de estancamiento.

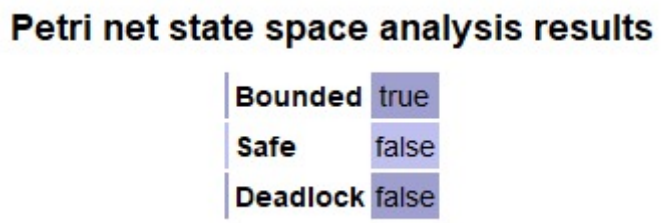


Fig. 4: Propiedades de la red de Petri

Cálculo de la cantidad de hilos.

Se empleó el algoritmo propuesto en *[Referencia 1]* con el objetivo de determinar la cantidad óptima de hilos necesarios para lograr la mayor ejecución en paralelo posible dentro del sistema.

Determinación de los hilos máximos activos simultáneos

1) Se obtuvieron los invariantes de transición de la red:
   - IT1: {T0,T1,T2,T3,T4,T11}
   - IT2: {T0,T1,T5,T6,T11}
   - IT3: {T0,T1,T7,T8,T9,T10,T11}

2) Se obtuvieron las plazas asociadas a cada invariante:
   - IT1: {P0, P1, P2, P3, P4, P5, P6, P11}
   - IT2: {P0, P1, P2, P3, P6, P7, P11}
   - IT3: {P0, P1, P2, P3, P6, P8, P9, P10, P11}

3) De las plazas asociadas a cada invariante se dejaron solos las plazas de acción:
   - IT1: {P1, P3, P4, P5, P11}
   - IT2: {P1, P3, P7, P11}
   - IT3: {P1, P3, P8, P9, P10, P11}

4) Del árbol de alcanzabilidad de la RdP, se obtuvo el conjunto de todos los marcados posibles de todos los conjuntos de plazas de acción.

5) Se determina la cantidad máxima de hilos activos simultáneos que corresponde al mayor valor de las sumas de todas las marcas para cada marcado posible, siendo en este caso igual a 3.

Determinación de la responsabilidad de los hilos

El análisis de la estructura de los invariantes muestra que éstos comparten transiciones entre sí:

- T0,T1: Representa un punto de bifurcación.
- T11: Representa un punto de unión.

Dado que los IT1, IT2 e IT3 presentan fork en T1 y join en T11, el algoritmo determina que cada IT debe dividirse al menos en tres zonas:

1. Un segmento previo, común a los tres IT.
2. Tres segmentos intermedios, uno por IT, posteriores al fork.
3. Un segmento final, común tras el join.

Como resultado, la asignación de hilos se realiza por segmentos y no por IT completos, lo cual evita que cada hilo deba resolver internamente los conflictos; la decisión queda completamente delegada a la política responsable.

Cálculo de hilos máximos por segmento

Para cada uno de los segmentos definidos se identificó el conjunto de plazas de acción que lo componen:

- Segmento 1: {P1}
- Segmento 2: {P4, P5}
- Segmento 3: {P7}
- Segmento 4: {P8, P9, P10}
- Segmento 5: {P11}

Con el mismo método utilizado previamente, se determinaron los máximos marcados posibles para cada conjunto de plazas. Los valores obtenidos fueron:

- MaxS1 = 1
- MaxS2 = 1
- MaxS3 = 1
- MaxS4 = 1
- MaxS5 = 2

Estos resultados indican que los cuatro primeros segmentos nunca requieren más de un hilo activo simultáneamente, mientras que el Segmento 5 puede alcanzar un máximo de dos hilos concurrentes, representando el punto de mayor paralelismo dentro del sistema.

Funcionamiento General del Programa

Este programa se basa en la colaboración entre clases para simular un sistema concurrente para el procesamiento de datos. Es una implementación clásica de Sistemas Concurrentes simulando una Red de Petri con monitores en Java, donde se modelan procesos concurrentes, sincronización y restricciones temporales. La simulación se ejecuta hasta

completar 200 invariantes (disparos de la transición T11), y luego un script en Python analiza los logs generados para contar patrones específicos de invariantes de transición.

Main: Es punto de entrada al programa y la clase que configura todo. Define la estructura de la red (la matriz matemática de incidencia), los tiempos de sensibilización, el marcado inicial, crea las instancias de las otras clases (RedDePetri, Politica, Monitor) y arranca los hilos asignándoles sus tareas a través de objetos Task. Agrupa las transiciones en "segmentos" para distribuir la carga entre hilos, asegurando concurrencia controlada.

RedDePetri: Es el "cerebro matemático". Guarda cuántos tokens hay en cada plaza (marcado), calcula si una transición puede dispararse (sensibilizada, verificando que no haya tokens negativos), gestiona los cronómetros de tiempo (time stamps y ventanas de sensibilización) y verifica que no haya errores lógicos (invariantes, como sumas constantes de tokens). No sabe de hilos, solo de matemáticas; actualiza el estado al disparar transiciones y registra eventos en un log.

Monitor: Es la clase que maneja la concurrencia. Protege a la Red De Petri usando un cerrojo (ReentrantLock) para evitar condiciones de carrera. Decide cuándo un hilo debe bloquearse (esperar en colas de condición) y cuándo despertar, basándose en sensibilización, tiempos y hilos esperando. Es el puente entre los Hilos y la Red, implementando el patrón Monitor para sincronización.

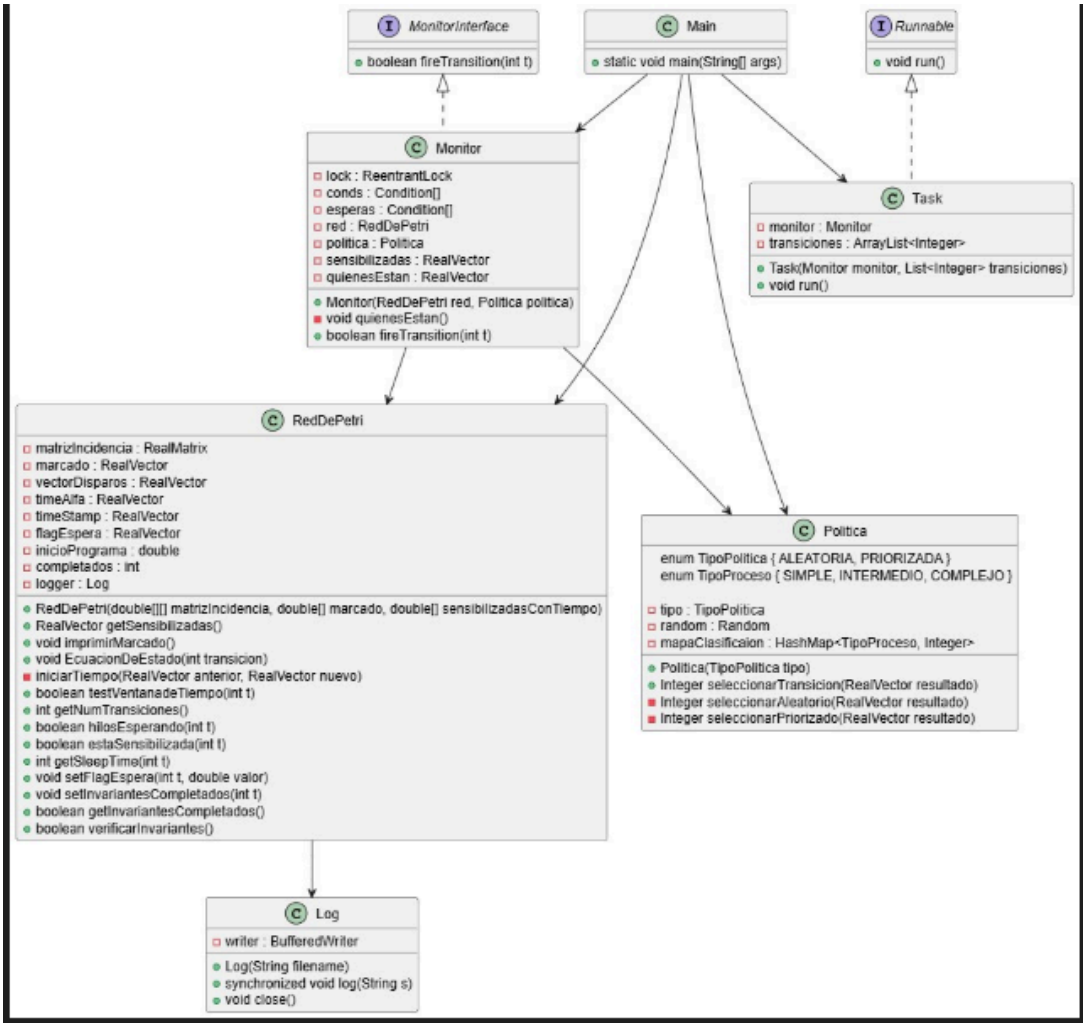
Política: Esta clase ayuda al Monitor a tomar decisiones. Cuando se libera el monitor y hay varios hilos esperando que podrían avanzar, la Política decide quién va primero, ya sea de forma aleatoria o priorizada (dando preferencia a transiciones de tipo "simple" si están disponibles). Usa un mapa de clasificación para asociar tipos de procesos a índices de transición.

Task: Es la parte del código que ejecutan los hilos. Su funcionamiento se basa en un bucle infinito intentando disparar las transiciones que le asignaron una y otra vez, en orden cíclico. Cada hilo ejecuta una instancia de Task, interactuando con el Monitor para solicitar disparos y terminando cuando la simulación ha finalizado (los invariantes han sido completados).

MonitorInterface: Una interfaz simple que define que el monitor debe tener un método fireTransition. Sirve para desacoplar el código, permitiendo que otras implementaciones de monitor puedan usarse sin cambiar el resto del sistema.

Log: Es una utilidad auxiliar para el registro de eventos. Escribe de forma sincronizada en un archivo de texto, como "transiciones.txt", cada disparo de transición, tiempos de ejecución y errores.

Script en Python: No es parte del código Java, pero complementa la simulación. Lee el archivo de logs generado ("transiciones.txt") y usa expresiones regulares para identificar y contar patrones de invariantes de transición (como secuencias que incluyen T2, T5 o T7 en contextos específicos). Además realiza los reemplazos iterativos para simplificar la línea de log y calcula frecuencias, imprimiendo resultados finales.



Análisis temporal con política aleatoria

Para asignar tiempos a las transiciones temporales de la red se realizó primero un análisis teórico considerando la política aleatoria de resolución del conflicto. Bajo esta política, cada invariante puede ser completado siguiendo uno de los tres caminos posibles (segmento 2, 3 o 4), elegidos de manera equiprobable.

Para cada invariante se ejecutan siempre las siguientes transiciones temporales:

- T1 se ejecuta siempre
- T3, T4 si se elige el segmento 2
- T6 si se elige el segmento 3
- T8, T9, T10 si se elige el segmento 4

Dado que los caminos son tres y cada uno contiene un conjunto diferente de transiciones temporales, el número promedio de transiciones temporales por invariante se obtiene como:

Tpromedio = (T3 + T4 + T6 + T8 + T9 + T10) / 3 = 2T

Por lo tanto, por cada invariante completado, se ejecutan en promedio 2 transiciones temporales. T1 no se incluye en el cálculo ya que se ejecuta en paralelo con las transiciones de los segmentos después del conflicto.

El programa se debe ejecutar entre 40 y 20 segundos, completando 200 invariantes. Esto implica que cada transición temporal debe durar:

- Si el sistema se ejecuta en 40 s: 40 s / 200×2 = 0.1 s por transición
- Si el sistema se ejecuta en 20 s: 20 s / 200×2 = 0.05 s por transición

Se realizaron múltiples ejecuciones bajo la política aleatoria para distintos valores asignados a las transiciones temporales. La siguiente tabla resume los tiempos obtenidos:

T1	T3	T4	T6	T8	T9	T10	Ejecución 1	Ejecución 2	Ejecución 3
50 ms	50 ms	50 ms	50 ms	50 ms	50 ms	50 ms	19,91 s	20,32 s	20,58 s
100 ms	50 ms	50 ms	50 ms	50 ms	50 ms	50 ms	21,8 s	22,15 s	21,7 s
50 ms	50 ms	50 ms	100 ms	50 ms	100 ms	50 ms	30,19 s	30,2 s	29,8 s
100 ms	100 ms	100 ms	100 ms	100 ms	100 ms	100 ms	39,2 s	41,15 s	40,68 s

Podemos ver que los tiempos asignados de 50 ms generan tiempos totales cercanos a los 20 segundos. Al aumentar los tiempos a 100 ms la duración del programa se acerca a los 40 segundos.

Análisis temporal con política priorizada

Para el análisis con la política priorizada, el camino elegido la mayoría de las veces es el tipo de procesamiento simple. En este camino, las únicas transiciones temporales involucradas en el camino principal son T1 y T6. Ambas se ejecutan en paralelo, por lo que el tiempo efectivo asociado a cada invariante queda determinado por la transición de mayor duración entre ellas. La siguiente tabla refleja este comportamiento:

T1	T3	T4	T6	T8	T9	T10	Ejecución 1	Ejecución 2	Ejecución 3
50 ms	50 ms	100 ms	50 ms	50 ms	150 ms	50 ms	23,27 s	23,64 s	26,1 s
50 ms	50 ms	100 ms	200 ms	50 ms	150 ms	50 ms	39,89 s	39,07 s	41,5 s
50 ms	200 ms	100 ms	50 ms	50 ms	150 ms	50 ms	31,41 s	30,21 s	31,22 s
50 ms	50 ms	100 ms	50 ms	50 ms	150 ms	200 ms	28,43 s	30,16 s	32,42 s

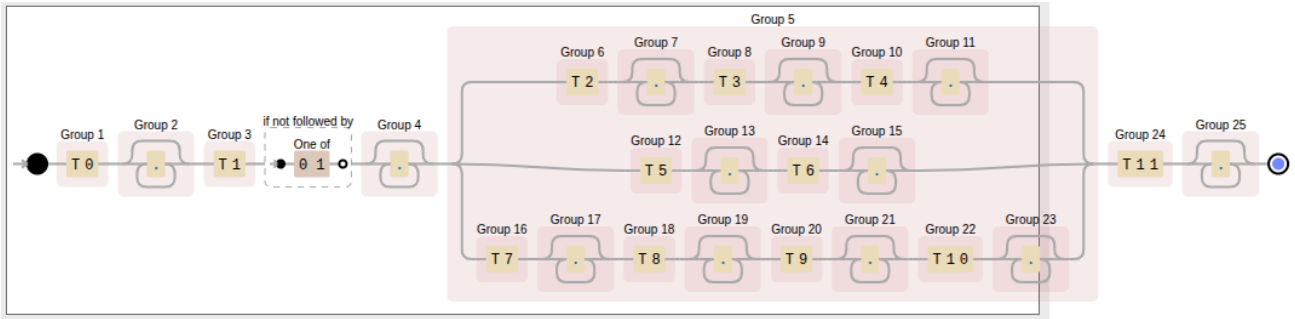
Podemos observar que en la ejecución 2, el tiempo de ejecución total del programa queda determinado por los 200 ms en T6. En la ejecución 3 y 4, al aumentar T3 y T10 respectivamente, también aumentan el tiempo total del programa ya que el camino de ejecución por esos segmentos también es tomado pero en menor medida.

Verificación del cumplimiento de los invariantes de transición

Para verificar de manera formal que se cumplen los invariantes de transición de la red luego de cada ejecución se hace análisis de un archivo log de las transiciones disparadas al durante la ejecución del programa.

El análisis se hace mediante expresiones regulares, buscando en la traza secuencias de transiciones que correspondan con uno de los invariantes de transición. La expresión regular está diseñada de la siguiente manera:

- Primero se detecta las transiciones T0, T1, previas al conflicto
- Luego se identifica uno de los tres caminos alternativos:
    - T2, T3, T4
    - T5, T6
    - T7, T8, T9, T10
- Finalmente se verifica la aparición de la transición T11



La regex permite localizar cada aparición completa de un invariante dentro de la traza, independientemente del interleaving entre ejecuciones de invariantes, gracias al uso de cuantificadores no codiciosos (.*?) y a una restricción con lookahead negativo (?![01]) que evita capturas incorrectas.
En cada iteración:

1. Se buscan coincidencias completas del patrón del invariante.
2. Se reemplaza la coincidencia por el contenido interno capturado, reduciendo progresivamente la traza. Este paso permite descubrir correctamente otros invariantes que quedaron entrelazados.
3. Se contabiliza cuál de los tres invariantes posibles fue reconocido, según qué grupo de captura resulte no vacío (G6, G12, G16).

Este proceso continúa hasta que no se encuentran más coincidencias, garantizando así la detección completa de todos los invariantes efectivamente ejecutados, incluso cuando sus transiciones se encuentran intercaladas entre sí.

Conclusión

En conclusión, el trabajo permitió integrar de forma completa el análisis formal de la red de Petri con una implementación concurrente realista en Java. A partir del modelo inicial se estudiaron sus propiedades estructurales con PIPE, identificando invariantes de plaza y de transición, y verificando que la red es acotada, libre de deadlocks y con un comportamiento adecuado para representar un sistema de procesamiento de datos. Sobre esa base se diseñó un monitor genérico que coordina el disparo de transiciones, asigna responsabilidades a los hilos según los segmentos de la red y aplica distintas políticas de resolución de conflictos, lo que hizo posible explotar el paralelismo sin perder corrección.

La incorporación de transiciones temporizadas y el análisis comparativo entre la política aleatoria y la priorizada permitieron estudiar cómo las decisiones de planificación afectan el tiempo total de ejecución y el uso de los distintos caminos de procesamiento. Finalmente, el uso de logs y del script en Python para detectar secuencias de disparo mediante expresiones regulares brindó una validación práctica de los invariantes de transición observados en el modelo teórico. En conjunto, el proyecto no solo confirmó las propiedades esperadas de la red, sino que también mostró cómo estas herramientas teóricas se traducen en un sistema concurrente controlado, medible y configurable bajo diferentes escenarios de ejecución.