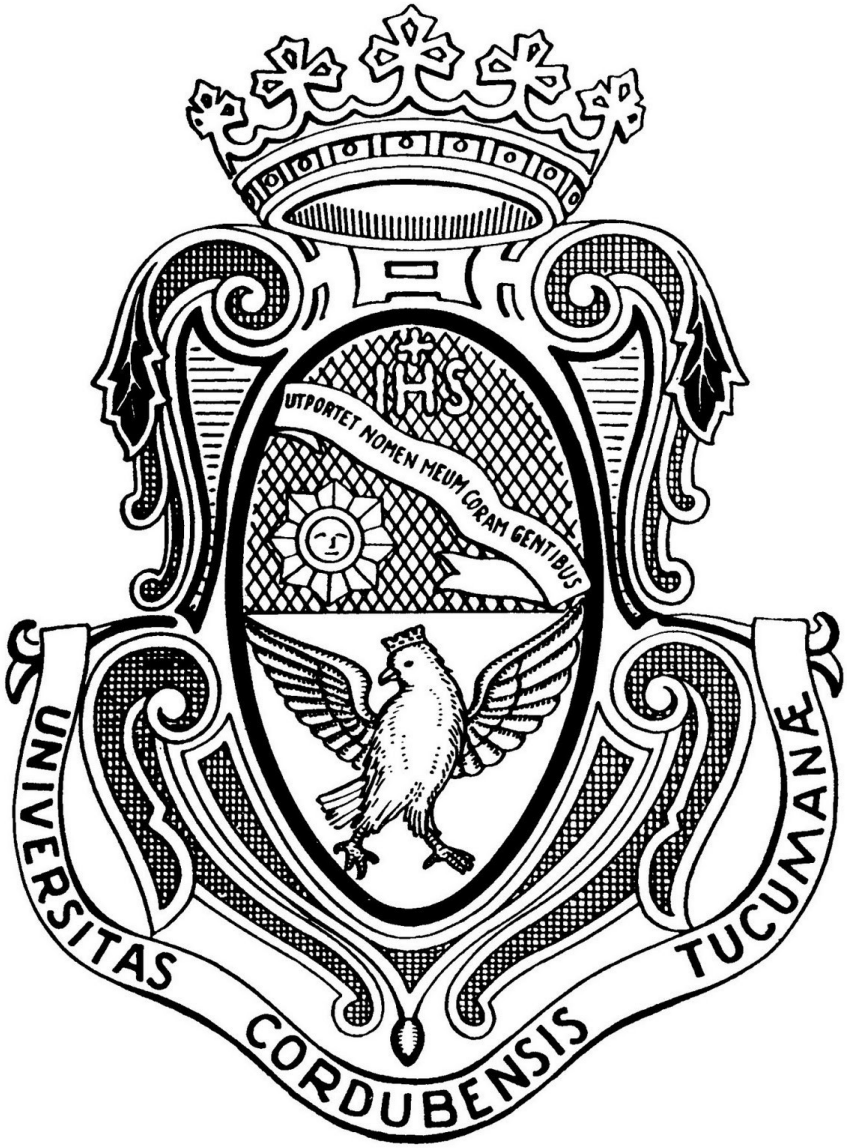
UNIVERSIDT1D NACIONAL DE CORDOBA

Facultad de Ciencias Exactas Físicas Y Naturales



Trabajo Práctico Integrador

Programación Concurrente

Docentes:
Dr. Ing. Micolini, Orlando
Ing. Ventre, Luis Orlando

Alumnos:

Collante, Gerardo Alexis	39022782	geracollante95@gmail.com
Oliva Arias, Carlos Agustín	35915867	aolivaarias@gmail.com
Rueda, Horacio	33565119	horange88@gmail.com

Índice

Enunciado

En este práctico se debe resolver el control de acceso a una playa de estacionamiento con 3 entradas (calles) diferentes. En esta playa hay 2 pisos, y en cada piso pueden estacionar 30 autos. La playa cuenta con 2 salidas diferentes y una única estación de pago (caja). En los accesos a la playa y en los egresos existen barreras que deben modelarse.

La playa cuenta con lugares (3) donde los vehículos se detienen cuando quieren entrar (barrera), una vez que ingresaron se les indica un piso y estacionan (puede ser piso 1 o piso 2). Se debe cuidar que no se permita el ingreso (superar barrera) a más vehículos de los espacios disponibles totales.

Los autos que se retiran de la playa deben liberar un espacio del piso en que se encontraban (diferenciar estacionamiento en cada piso). Cuando un vehículo se va a retirar puede optar por salida a calle 1 o salida a calle 2. Luego debe abonar la estadía. El cobro de la estadía le lleva a un empleado promedio al menos 3 minutos. (Existe una sola caja)

En caso de que la playa esté llena, se debe encender un cartel luminoso externo que indica tal situación.

El sistema controlador debe estar conformado por distintos hilos, los cuales deben ser asignados a cada conjunto de responsabilidades afines en particular. Por ejemplo: Ingreso de vehículos, manejo de barreras, etcétera.

Debe realizar:

- La red de Petri que modela el sistema.
- Agregar las restricciones necesarias para evitar interbloqueos ni accesos cuando no hay lugar, mostrarlo con la herramienta elegida y justificarlo.
- Simular la solución en un proyecto desarrollado con la herramienta adecuada (explique porque eligió la herramienta usada).
- Colocar tiempo en las estación de pago caja (en la/s transición/es correspondiente/s).
- Determinar la cantidad de hilos necesarios (justificarlo).
- Tabla de estados
- Tabla de eventos
- Implementar dos casos de Políticas para:
    - Prioridad llenar de vehículos planta baja (piso 1) y luego habilitar el piso superior. Prioridad salida indistinta (caja).
    - Prioridad llenado indistinta. Prioridad salida a calle 2.
- Hacer el diagrama de clases.
- Hacer los diagramas de secuencias.
- Hacer el código.
- Hacer el *testing*.

Desarrollo

Requerimientos funcionales del sistema

- El sistema sólo debe disparar transiciones sensibilizadas.
- El sistema no debe disparar transiciones no sensibilizadas o inhibidas.
- El sistema debe conocer las transiciones sensibilizadas.
- El sistema debe tener la capacidad para disparar una transición sensibilizada por tiempo.
- El sistema no debe disparar una transición no sensibilizada por tiempo.
- El sistema debe tener prioridades/políticas de disparo entre transiciones.
- El sistema no debe permitir el acceso al monitor a más de un hilo.

Parámetros del Sistema

- El diseño de la Red de Petri fue realizado en el software PIPE, a partir de los requerimientos indicados en el enunciado. La simulación del modelo y la implementación de las políticas fue realizado en lenguaje Java.
- El software PIPE permite exportar el diseño de la red en formato XML, que puede ser luego incorporado como input en el programa desarrollado en java
- Para interpretar el archivo exportado de PIPE, se creó un módulo de JAVA denominado PipeParser cuya función es leer el archivo XML y definir las matrices correspondientes a la Red de Petri dentro del programa. Las matrices que caracterizan el sistema son:

- `marcado[][]:` marcado inicial. (mx1)
- `Iplus[][]:` incidencia+. (mxn)
- `Iminus[][]:` incidencia-. (mxn)
- `Icombined[][]:` incidencia. (mxn)
- `Inhibition[][]:` inhibición. (mxn)
- `ArmReader[][]:` brazo lector. (mxn)

  Donde m es la cantidad de plazas y n es la cantidad de transiciones en la Red de Petri.

Tabla de estados

Numero	Estado
P0	Generador de autos
P1	Limitador clientes calle 1
P2	Autos esperando ingresar calle 1
P3	Limitador clientes calle 2
P4	Autos esperando ingresar calle 2
P5	Limitador clientes calle 3
P6	Autos esperando ingresar calle 3
P7	Cartel - Hay lugares disponibles
P8	Cartel - No hay lugares disponibles
P9	Auto pasando barrera
P10	Limitador de autos planta alta
P11	Estacionamiento planta alta
P12	Limitador de autos planta baja
P13	Estacionamiento planta baja
P14	Limitador de autos total
P15	Autos esperando por cobrar
P16	Limitador de clientes para salida calle A
P17	Autos esperando egreso calle A
P18	Autos esperando egreso calle B
P19	Limitador de clientes para salida calle B
P20	Autos decidiendo salida
P21	Clientes que salieron por calle A
P22	Clientes que salieron por calle B

Cuadro 1: Tabla de estados.

Tabla de eventos

Numero	Evento
T0	Entrar calle 1
T1	Entrar calle 2
T2	Entrar calle 3
T3	Prender cartel estacionamiento lleno
T4	Levantar barrera 1
T5	Levantar barrera 2
T6	Levantar barrera 3
T7	Apagar cartel estacionamiento lleno
T8	Ir a planta baja
T9	Ir a planta alta
T10	Cliente saliendo de planta baja
T11	Cliente saliendo de planta alta
T12	Eligiendo calle A
T13	Eligiendo calle B
T14	Levanta barrera salida calle B
T15	Levanta barrera salida calle A
T16	Cobro de clientes

Cuadro 2: Tabla de eventos.

Hilos

La simulación del problema se re za aprovechando la posibilidad de emplear hilos para la ejecución de los disparos de la red. La gestión de los recursos compartidos se realiza mediante la implementación de un monitor que gestiona la concurrencia durante la ejecución de los disparos dentro de cada hilo.

Cada hilo se encarga de ejecutar un determinado número de transiciones relacionadas entre sí y que, a su vez, no dependen de la ejecución de las transiciones que son disparadas por otros hilos. De esta forma, mientras algún hilo se encuentre en espera o dormido, el programa puede continuar funcionando a la vez que otros hilos continúan con su ejecución. Según la configuración adoptada en el modelo, se definieron nueve hilos, además del hilo Main, que ejecutan las siguientes transiciones:

- Entrada1: T0 – T4.
- Entrada2: T1 - T5.
- Entrada3: T2 - T6.
- Cartel: T3 - T7
- Ingreso PB: T8
- Ingreso PA: T9
- Egreso PB: T10
- Egreso PA: T11

- Caja: T16
- Salida1: T12 - T14
- Salida2: T13 - T15

Fundamentación:

- **Hilos entrada**: Los 3 hilos iniciales en las entradas son de existencia trivial, donde cada calle (o entrada) es independiente una de la otra, y cada una cuenta con sus propios recursos. Estos hilos representan a los autos pasando las barreras. Cada hilo administra el ingreso de vehículos y el accionamiento de la barrera correspondiente a cada entrada.
- **Ingreso PB - PA:** Son los encargados de distribuir los vehículos dentro de la playa y determinan la ocupación de los lugares de estacionamiento. Estos hilos están permanentemente esperando por vehículos que ingresan a la playa.
- **Egresos PB – PA:** son los encargados de ejecutar las transiciones que determinan la salida de los vahículos de las plazas de estacionamiento. Las transiciones disparadas por estos hilos están temporizadas, para simular el tiempo en que un auto permanece estacionado dentro de la playa.
- **Caja:** es el encargado del cobro a los vehículos que abandonan la playa. Este hilo permanece esperando por vehículos que hayan abandonado las plazas de estacionamiento. La transición que representa el cobre del estacionamiento es también temporizada
- **Salida 1 y 2:** Cada hilo se encarga de gestionar la salida de vehiculos de la playa por cada una de las calles. Las transiciones disparadas por cada hilo representan la elección de la salida y el abandono de la playa.
- **Cartel:** Se encarga del manejo del cartel de la playa. El hilo gestiona el encendido y apagado del cartel que indica cuándo la playa se encuentra llena o tiene al menos un lugar disponible.

Discusiones

Las políticas fueron implementadas a partir de interfaz *politicas* cuyo método *cual()* devuelve la próxima transición a disparar. De acuerdo a los requerimientos indicados en el enunciado del problema, se definen dos clases que implementan dicha interfaz: PoliticaA y PoliticaB.
Para determinar la prioridad en el disparo de las transiciones, se definió un array de dimensión 1xn donde n es la cantidad de transiciones, en el que se ordenan por prioridad la totalidad de transiciones que componen el sistema. Luego, el método cual() devuelve la primera transición del array que se encuentra sensibilizada y cuyo hilo asociado se encuentra en cola, listo para disparar.

Ecuación generalizada de estado

Para determinar si una transición se encuentra sensibilizada se emplea la ecuación generalizada de estado. Con esta ecuación, además de verificar que las plazas desde

donde parte una transición tenga los tokens necesarios para disparar la misma, se verifica que no exista inhibición por arco inhibidor, por arco lector, ni por tiempo. Para ello se definen los vectores Q, W y H y se hacen uso de las matrices H, R y E. El vector de transiciones sensibilizadas generalizado indica cuáles transiciones se pueden disparar.

El vector Q (mx1) está conformado por i componentes que derivan de la siguiente relación:

$q_i = cero(M(p_i))$.

El vector W (mx1) está conformado por i componentes que derivan de la siguiente relación:

$w_i = uno(M(p_i))$.

La matriz H (mxn) relaciona las plazas que conectan las transiciones con un brazo inhibidor.

La matriz R (mxn) relaciona las plazas que conectan las transiciones con un brazo lector.

El vector E (nx1) es el vector de sensibilizado.

Vector de transiciones des-sensibilizadas por arco lector L:

$L = R'W$

Vector de transiciones des-sensibilizadas por arco inhibidor B:

$B = H'Q$

Vector de transiciones des-sensibilizadas por tiempo Z:

$Z = Tim(q(E, B, L, clk), intervalos)$

- E (nx1) es un vector de valores binarios que indica con un cero cuales transiciones están inhibidas porque no se ha alcanzado o se ha superado el intervalo de tiempo transcurrido desde de que la transición fue sensibilizada.
- La relación $Tim(q, intervalos)$ es un vector binario (nx1), en donde el valor de la componente i es uno si q_i , que es un contador, tiene valor dentro del intervalo indicado por la componente fila i-ésima de la matriz de intervalos $intervalos = [\alpha_i, \beta_i]$. De otra forma es cero.
- Para que el contador q_i arranque, la ecuación $e_i \wedge b_i \wedge l_i$ debe ser uno, de otra forma el contados se pone a cero. Este contador se incrementa en una unidad de tiempo por cada pulso de reloj de la base de tiempo (clk).
- La matriz $intervalos = [\alpha_i, \beta_i]$ (nx2) contiene el límite inferior y superior de la ventana de tiempo asociada a cada transición. En los casos en que las transiciones no están temporizadas, el límite inferior es 0 y el superior es una constante que expresa el máximo valor asignable a un entero.

Así el vector de sensibilizado extendido Ex (nx1) se obtiene de la conjunción lógica de todos los vectores anteriores.

$$E_x = E \wedge B \wedge L \wedge Z$$

Red de Petri

Al tratarse de una red que está acotada, se tiene una secuencia de disparos infinita, por lo tanto la red es monótona.

Semántica utilizada para las transiciones temporizadas

Se tomo la primera de las semánticas especificadas en *"Redes de Petri Temporales 2017.pdf"– Hoja 11*, por lo tanto el disparo se realiza en dos etapas:

- Transcurre un determinado tiempo desde que una transición se sensibiliza.
- Se retiran y colocan las marcas de forma atómica.

Cabe mencionar que esto se usa para ambos tipos de transiciones: inmediatas y temporales, ya que en el caso de la inmediata, el paso 1 implica un intervalo de tiempo nulo.

Al tener un arco lector, no se utiliza la matriz de incidencia combinada para realizar el cálculo del estado siguiente de la red, ya que este tipo de arco puede definirse como un arco común que entra y sale de una misma transición, por lo que se verá reflejado como un 0 en la matriz de incidencia combinada. Debido a esto se realiza el cálculo del estado siguiente en dos etapas: primero se realiza la toma de tokens utilizando la matriz de incidencia negativa y luego se realiza la inserción de tokens utilizando la matriz de incidencia positiva.

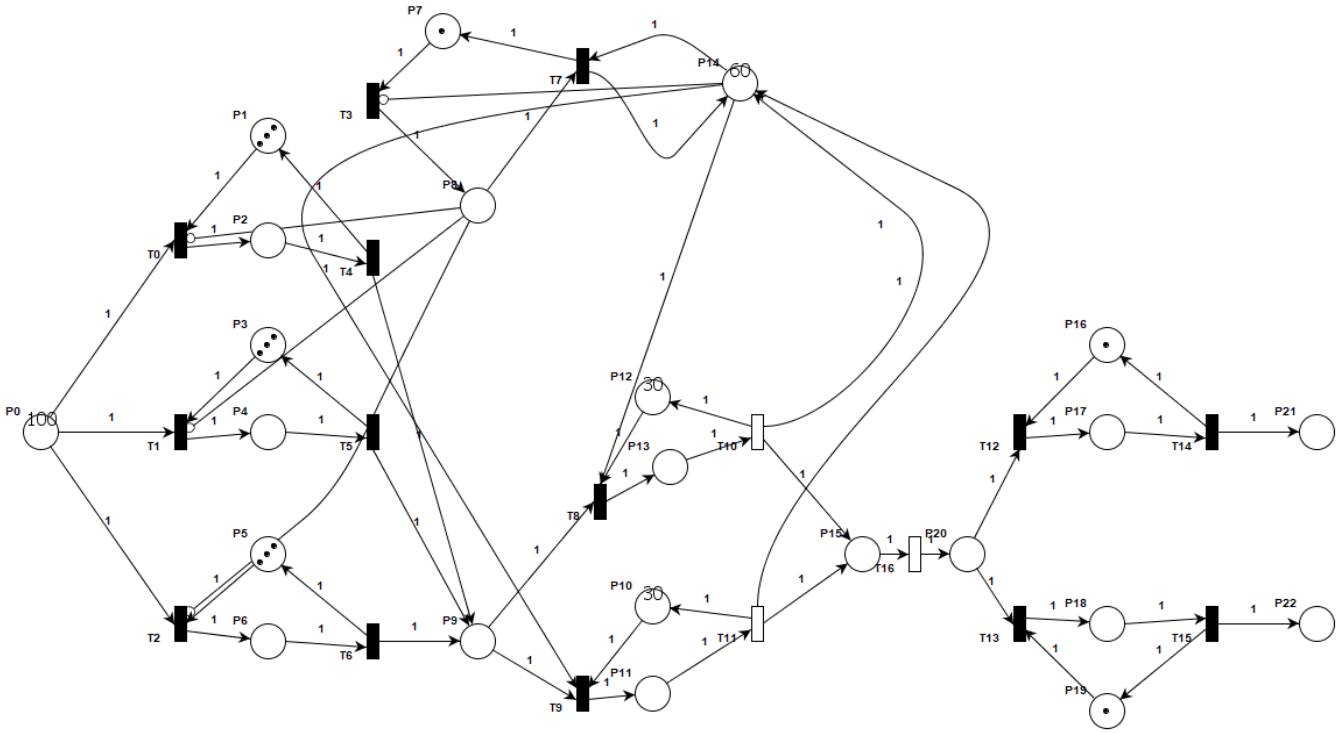


Grafico de la Red de Petri.

Tiempos

Para las transiciones temporizadas se utilizan dos arreglos. Una transición con tiempo se "cronometra" desde el inicio de su sensibilización, y en el momento en el que

se la quiere disparar, se debe verificar que el tiempo dicho esté dentro de su intervalo de tiempo $[\alpha_i, \beta_i]$. Si el disparo se quiere realizar antes del límite inferior (α_i), el hilo debe dormir $\alpha_i - x$ segundos, donde x es el tiempo transcurrido desde que se sensibilizó la transición.

Una transición no temporizada o inmediata se parametriza con $\alpha_i = 0$ y $\beta_i = 0$. En el caso de esta red en particular *T0, T1* y *T2* representan a los tiempos de ingreso al la playa de estacionamiento.

T10 y T11 corresponden al tiempo que permanece en planta alta y baja respectivamente. T16 es el tiempo necesario para cobrarle al cliente.

- $T0 = [20, MAXVALUE]$
- $T1 = [20, MAXVALUE]$
- $T2 = [20, MAXVALUE]$
- $T10 = [150, MAXVALUE]$
- $T11 = [150, MAXVALUE]$
- $T16 = [25, MAXVALUE]$

Resulta importante la asignación de estos tiempos. Un alfa muy próximo a 0 es rápidamente sensibilizada, pero si es acompañada de un beta cercano, la ventana de disparo de la transición será excedido con mucha certeza, y en consecuencia pocas veces o incluso nunca será disparada, acarreando problemas de fluidez y un eventual interbloqueo.

Análisis de Invariantes

El análisis de invariantes es utilizado para la realización del *testing* del código. Para que éste sea exitoso ambos invariantes (de plaza y transiciones) deben cumplirse, ya que el hecho de que no se cumpla alguno implica que el sistema está funcionando de manera incorrecta. Si no se cumplen los invariantes de plaza significa que no se está respetando la cantidad de recursos existentes en la red, mientras que si no se cumplen los invariantes de transiciones se verifica que los bucles posibles que posee la red no se están realizando, por lo que los disparos están ocurriendo en manera incorrecta.

P-Invariantes

Un P-invariante indica que el número de tokens en todas las marcas alcanzables satisface una invariante lineal. Las invariantes asociadas a la red de Petri del modelo, generadas por PIPE son las siguientes:

Limitador de clientes totales que pueden encontrarse en el sistema

$$P0 + P2 + P4 + P6 + P9 + P11 + P13 + P15 + P17 + P18 + P20 + P21 + P22 = 100$$

Limitador de autos en la primer entrada.

$$P1 + P2 = 3$$

Limitador de autos en la segunda entrada.

$$P3 + P4 = 3$$

Limitador de autos en la tercera entrada.

$$P5 + P6 = 3$$

Cartel de playa

$P7+P8=1$

Limitador de autos en planta baja.
$P10+P11=30$

Limitador de autos en planta alta.
$P12+P13=30$

Cantidad total de plazas disponibles en el estacionamiento
$P11+P13+P14=60$

Salida por calle 1
$P16+P17=1$

Salida por calle 2
$P18+P19=1$

Las P-invariantes se verifican en tiempo de ejecución. Luego de un disparo, y en consecuencia, luego de cada evolución de la red, se verifica el cumpliento de cada una de las 14 condiciones de P-Invariantes arriba listadas. Una bandera booleana inicializada en *true* permanece en ese estado siempre que el resultado del análisis sea satisfactorio, pero en caso de haber una inconsistencia en la red que viole el análisis, la bandera pasará al estado *false* y permanecerá en ese estado hasta el fin de la ejecución, indicando que como mínimo una P-Invariante no se cumplió.

T-Invariantes

Estas indican posibles bucles (*loops*) en la red. Es decir, una secuencia de disparos que que genera el mismo marcado desde el que partió.

La verificación de las T-invariantes se realiza una vez finalizado el programa, mediante el análisis de un log, generado durante la ejecución del mismo. Para generar el log, el programa escribe en pantalla una secuencia de caracteres, donde cada uno representa el disparo de una transición: $T0=a$, $T1=b$, …, $T16=q$

Las siguientes son todas las T-invariantes de la red.

T1	**T5**	**T9**	**T11**	**T17**	**T13**	**T15**
T1	**T5**	**T9**	**T11**	**T17**	**T14**	**T16**
T1	**T5**	**T10**	**T12**	**T17**	**T13**	**T15**
T1	**T5**	**T10**	**T12**	**T17**	**T14**	**T16**
T2	**T6**	**T9**	**T11**	**T17**	**T13**	**T15**
T2	**T6**	**T9**	**T11**	**T17**	**T14**	**T16**
T2	**T6**	**T10**	**T12**	**T17**	**T13**	**T15**
T2	**T6**	**T10**	**T12**	**T17**	**T14**	**T16**
T3	**T7**	**T9**	**T11**	**T17**	**T13**	**T15**
T3	**T7**	**T9**	**T11**	**T17**	**T14**	**T16**
T3	**T7**	**T10**	**T12**	**T17**	**T13**	**T15**
T3	**T7**	**T10**	**T12**	**T17**	**T14**	**T16**
T4	**T8**					

La secuencia de caracteres asociados a las transiciones son:

a	**e**	**i**	**k**	**q**	**m**	**o**
a	**e**	**i**	**k**	**q**	**n**	**p**

a	**e**	**j**	**l**	**q**	**m**	**o**
a	**e**	**j**	**l**	**q**	**n**	**p**
b	**f**	**i**	**k**	**q**	**m**	**o**
b	**f**	**i**	**k**	**q**	**n**	**p**
b	**f**	**j**	**l**	**q**	**m**	**o**
b	**f**	**j**	**l**	**q**	**n**	**p**
c	**g**	**i**	**k**	**q**	**m**	**o**
c	**g**	**i**	**k**	**q**	**n**	**p**
c	**g**	**j**	**l**	**q**	**m**	**o**
c	**g**	**j**	**l**	**q**	**n**	**p**
d	**h**					

La verificación de las T-invariantes en el log se realiza mediante el uso de expresiones regulares, que detectan coincidencia de los patrones de caracteres asociados a las T-invariantes. Una vez encontrado un conjunto de caracteres que coincide con un patrón, éstos son extraídos del log. Al finalizar el análisis, si no hubieron inconsistencias en el disparo de las transiciones, el log debe quedar vacío. Esto indica que todos los vehículos que ingresaron a la playa, ya han salido.

A continuación se indica el ejemplo de una expresión regular empleada para la verificación de la primera T-invariante

(?:a)([a-df-q]?)(?:e)([a-hj-q]?)(?:i)([a-jl-q]?)(?:k)([a-p]?)(?:q)([a-ln-q]?)(?:m)([a-np-q]?)(?:o)

Comentarios sobre el análisis de T-invariantes

Dado que alguna de las T-invariantes resultan combinaciones lineales de otras, se generaron inconvenientes al extraer los caracteres del log usando expresiones regulares. Sin embargo para poder verificar el cumplimiento de las T-invariantes en el sistema, se emplearon políticas especiales para forzar sólo el disparo de transiciones correspondientes a una invariante en particular. En este caso se verifico el complimiento en el análisis del log.

2.9. Diagrama de clases

Diagrama de clases

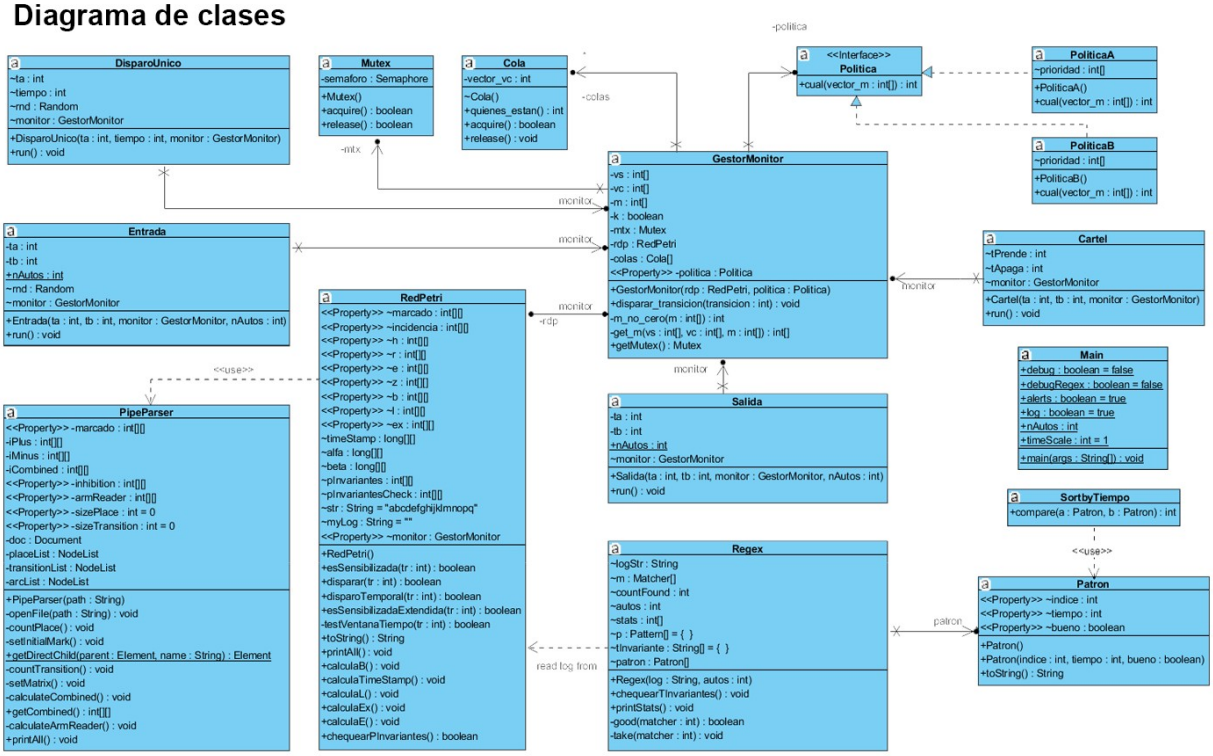


Figura 2: Diagrama de clases del sistema.

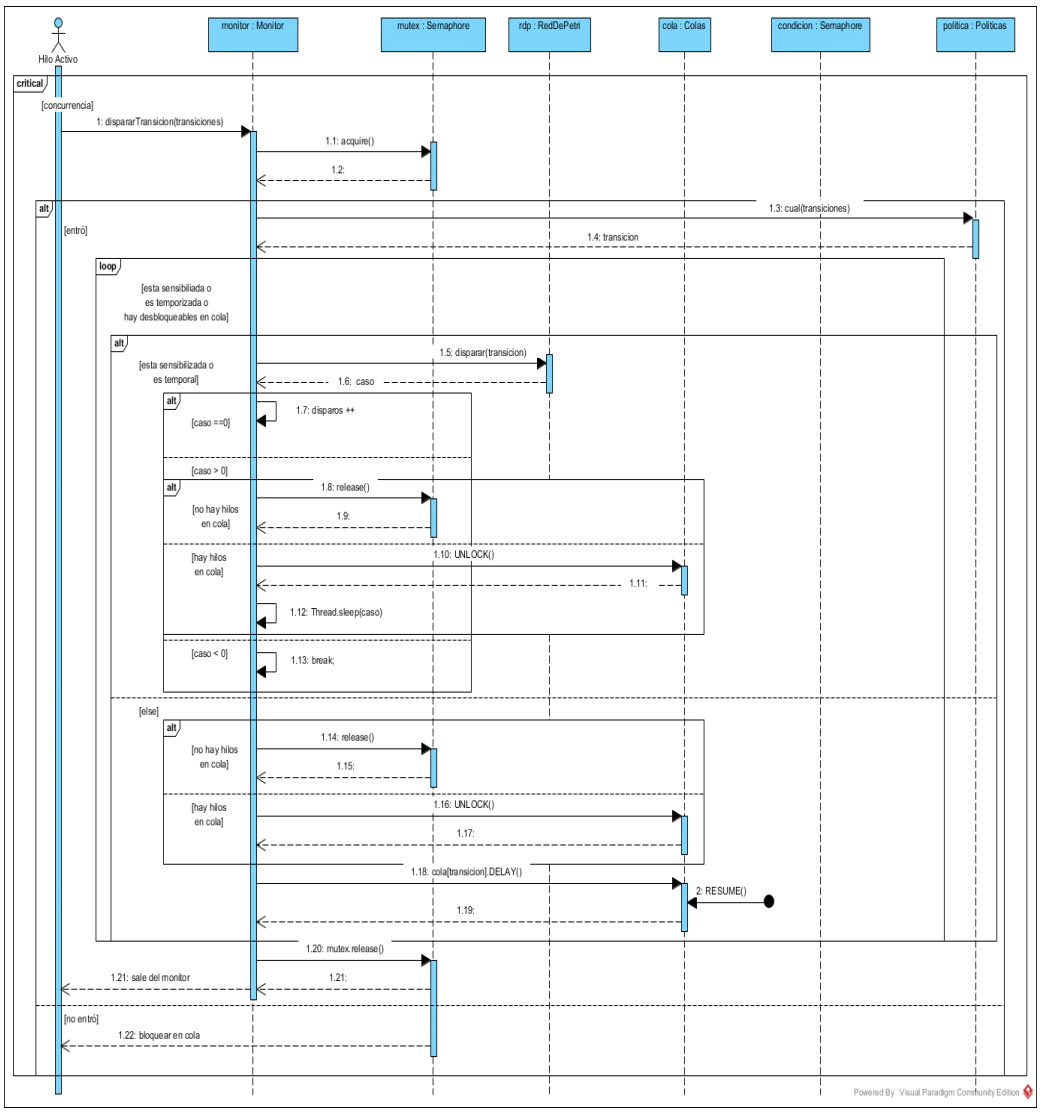


Figura 3: Diagrama del método ‘dispararTransicion’ de la clase ‘Monitor’.

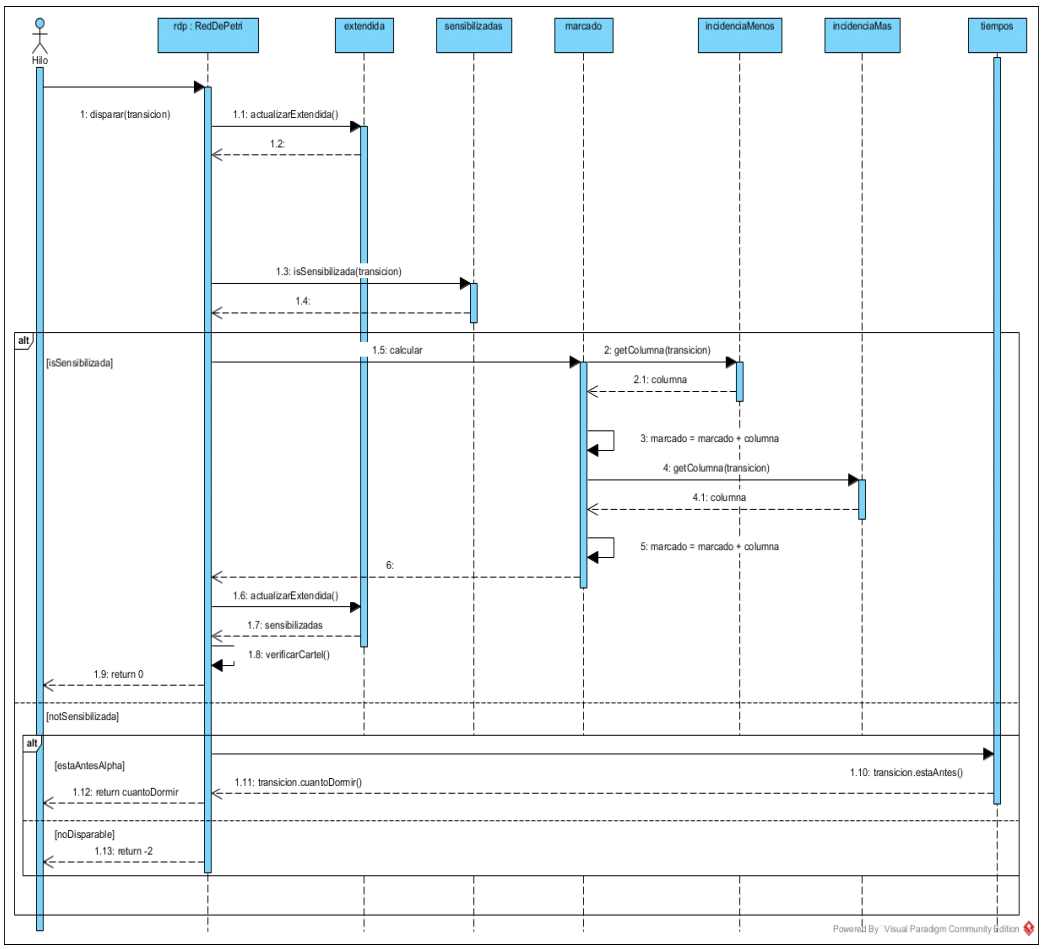


Figura 4: Diagrama del método ‘disparar’ de la clase ‘RedDePetri’.

Conclusiones y Resultados

A partir del modelo realizado en el software PIPE, fue posible realizar la simulación del funcionamiento de una playa de estacionamiento a través de un programa desarrollado en Java. La implementación de un monitor en el programa garantizó que los disparos de las transiciones se realizaran en forma secuencial. Esto permitió que tanto las invariantes de plaza como las de transiciones se cumplieran a lo largo de diferentes corridas del programa.

La lógica implementada dentro del diseño de la red a través de las matrices de transición, arcos lectores e inhibidores posibilitó definir las restricciones y el funcionamiento general de la red, en tanto que las políticas implementadas en el programa sirvieron para modificar el comportamiento de la simulación.

El uso de expresiones regulares permitió realizar el testing del funcionamiento del programa a partir de un registro (log) generado durante la ejecución del mismo.