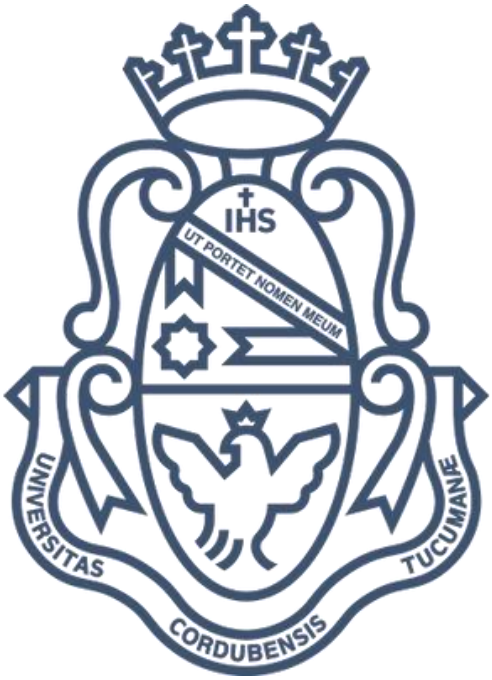
UNIVERSIDAD NACIONAL DE CÓRDOBA

FACULTAD DE CIENCIAS EXACTAS, FÍSICAS y NATURALES



PROGRAMACIÓN CONCURRENTE

Trabajo Práctico Final

"Control de Tráfico Vehicular en una Esquina con Redes de Petri"

- **Alumnos:**
  - Aguilar, Mauricio - 34.496.071
  - Tarazi, Pedro Esequiel - 35.035.736

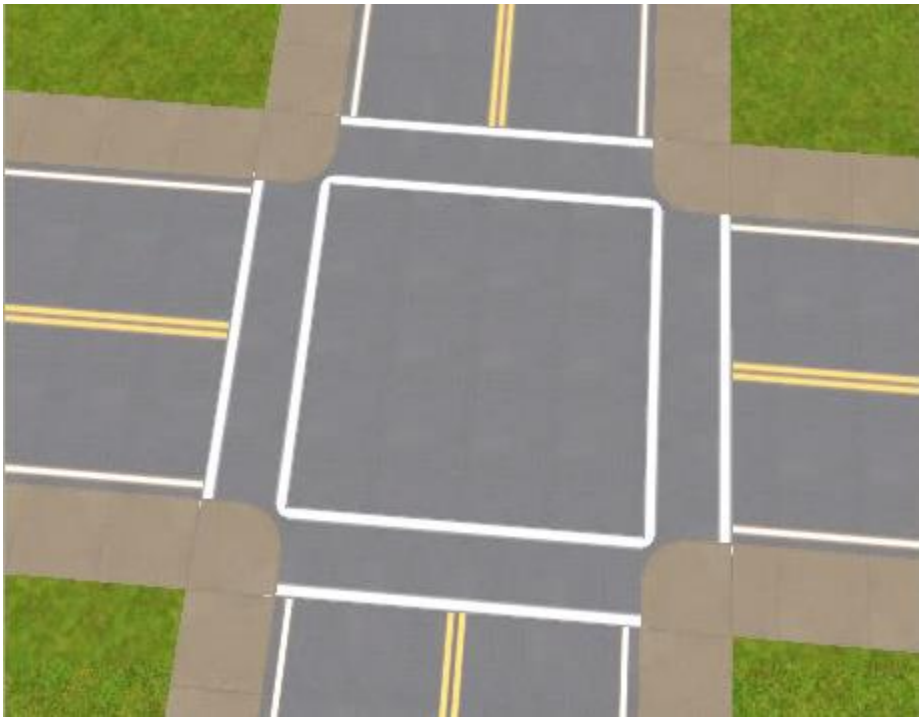
- **Carrera:** Ingeniería en Computación
- **Profesor:** Ing. Orlando Micolini
- **Año:** 2015

INDICE

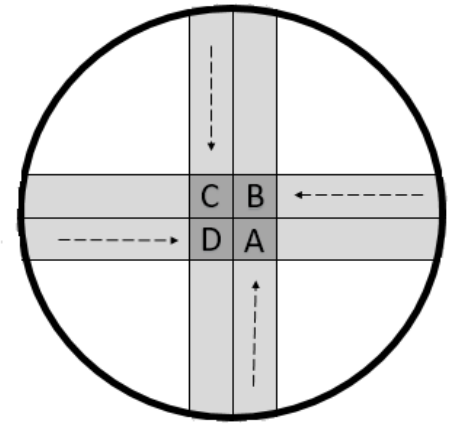
AGUILAR, MAURICIO
TARAZI, PEDRO ESEQUIEL

INTRODUCCIÓN

En el presente trabajo se diseña una RdP (Red de Petri) para poder simular el control de tráfico en una esquina cuando varios autos quieren acceder a la misma de manera concurrente y en distintas direcciones. Este análisis se hace en una esquina que tiene 4 carriles de ingreso y 4 carriles de salida, de modo que los autos pueden entrar y salir de la misma en todas las direcciones.



Se divide la esquina en cuatro porciones o espacios llamados A, B, C y D, recursos a administrar por la RdP. Los autos intentarán tomar los recursos A, B, C y D para poder hacer su camino, pero esto dependerá de si pueden o no hacerlo, por lo que se considera a la esquina como una Región Crítica.



Aplicamos entonces la RdP para administrar estos recursos y evitar el interbloqueo y la inanición, utilizando los conceptos de exclusión mutua y condiciones de sincronización.

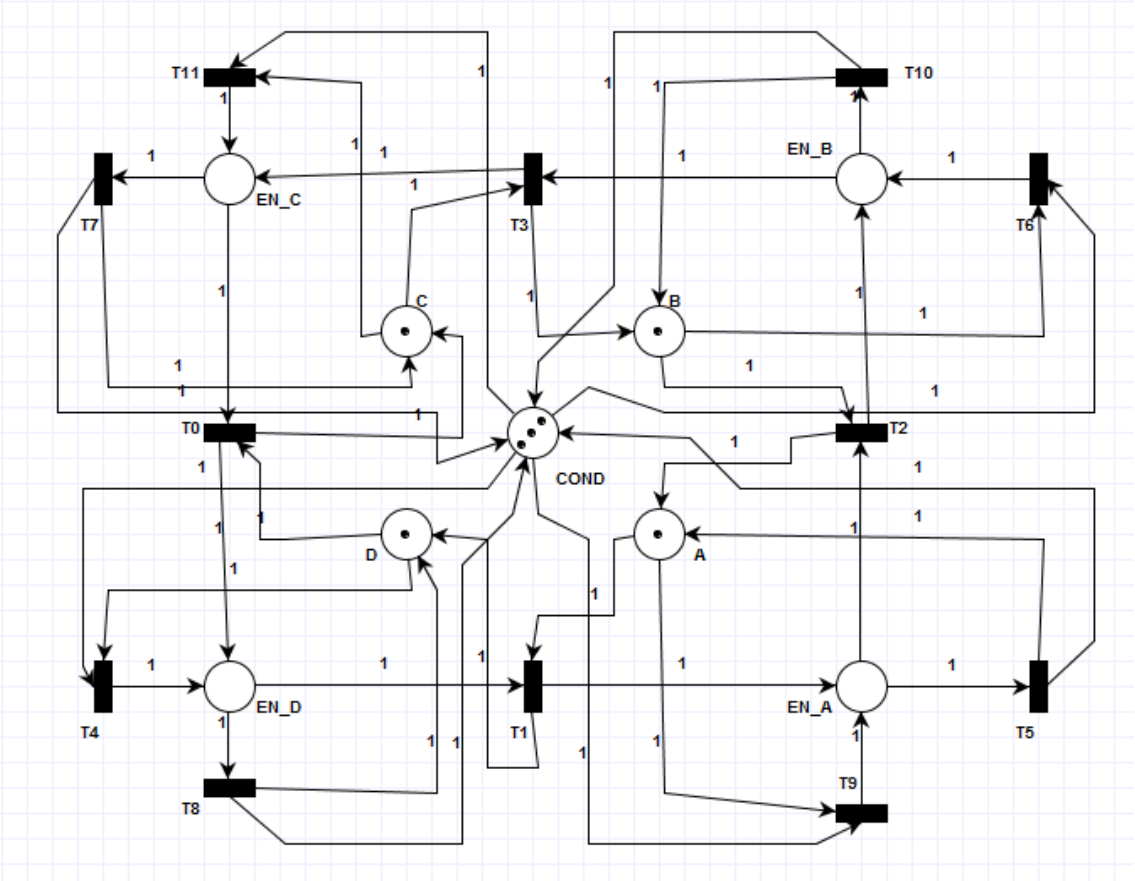
Respecto a los caminos, cada auto tiene tres caminos posibles diferentes para realizar: girar a la derecha, seguir derecho, o girar a la izquierda, mientras que el giro en U no está permitido.

Para realizar esto, se representa a los autos como procesos (hilos), y la esquina se representa con un Monitor. La lógica se implementa mediante una Red de Petri.

DESARROLLO

RED DE PETRI

Se muestra a continuación la RdP en cuestión, donde los recursos A, B, C y D se encuentran representados por las plazas del mismo nombre, y sus lugares correspondientes por las plazas EN_A, EN_B, EN_C, y EN_D. Estas plazas están conectadas a través de arcos (de peso unitario) con transiciones, que son disparadas por los autos.



Para evitar el interbloqueo, cada auto debe tomar de a un recurso por vez, y esperar que el siguiente recurso esté disponible la próxima vez que ingrese al Monitor.
Además, otra medida será que en la esquina se pueda tomar como máximo, tres de estos recursos a la vez. Estas condiciones se establecen con las plazas A, B, C, D, y COND respectivamente, y sus tokens iniciales.

La RdP es representada por las matrices positiva, negativa y de incidencia.

La matriz positiva I+ indica las relaciones entre transiciones que son salidas de plazas, con un número que se corresponde al peso del arco:

Forwards incidence matrix I^+

	T00	T01	T02	T03	T04	T05	T06	T07	T08	T09	T10	T11
A	0	0	1	0	0	1	0	0	0	0	0	0
B	0	0	0	1	0	0	0	0	0	0	1	0
C	1	0	0	0	0	0	0	1	0	0	0	0
D	0	1	0	0	0	0	0	0	1	0	0	0
EN_A	0	1	0	0	0	0	0	0	0	1	0	0
EN_B	0	0	1	0	0	0	1	0	0	0	0	0
EN_C	0	0	0	1	0	0	0	0	0	0	0	1
EN_D	1	0	0	0	1	0	0	0	0	0	0	0
COND	0	0	0	0	0	1	0	1	1	0	1	0

La matriz negativa I- indica las relaciones entre transiciones que son entradas de plazas, con un número que se corresponde al peso del arco:

Backwards incidence matrix I^-

	T00	T01	T02	T03	T04	T05	T06	T07	T08	T09	T10	T11
A	0	1	0	0	0	0	0	0	0	1	0	0
B	0	0	1	0	0	0	1	0	0	0	0	0
C	0	0	0	1	0	0	0	0	0	0	0	1
D	1	0	0	0	1	0	0	0	0	0	0	0
EN_A	0	0	1	0	0	1	0	0	0	0	0	0
EN_B	0	0	0	1	0	0	0	0	0	0	1	0
EN_C	1	0	0	0	0	0	0	1	0	0	0	0
EN_D	0	1	0	0	0	0	0	0	1	0	0	0
COND	0	0	0	0	1	0	1	0	0	1	0	1

La matriz de incidencia I es una combinación de las dos anteriores, e indica ambas cosas, las transiciones que son entradas a plazas con números negativos, y las transiciones que son salidas a plazas con números positivos:

Combined incidence matrix *I*

	T00	T01	T02	T03	T04	T05	T06	T07	T08	T09	T10	T11
A	0	-1	1	0	0	1	0	0	0	-1	0	0
B	0	0	-1	1	0	0	-1	0	0	0	1	0
C	1	0	0	-1	0	0	0	1	0	0	0	-1
D	-1	1	0	0	-1	0	0	0	1	0	0	0
EN_A	0	1	-1	0	0	-1	0	0	0	1	0	0
EN_B	0	0	1	-1	0	0	1	0	0	0	-1	0
EN_C	-1	0	0	1	0	0	0	-1	0	0	0	1
EN_D	1	-1	0	0	1	0	0	0	-1	0	0	0
COND	0	0	0	0	-1	1	-1	1	1	-1	1	-1

Esta matriz puede cambiarse antes de cada ejecución por medio de un archivo de texto llamado matrizI_default.txt

De acuerdo a lo antes expuesto en cuanto reglas para evitar el interbloqueo, el marcado inicial es el siguiente:

Marking

	A	B	C	D	EN_A	EN_B	EN_C	EN_D	COND
Initial	1	1	1	1	0	0	0	0	3

En consecuencia, las transiciones inicialmente sensibilizadas son:

Enabled transitions

T00	T01	T02	T03	T04	T05	T06	T07	T08	T09	T10	T11
no	no	no	no	yes	no	yes	no	no	yes	no	yes

POLÍTICA

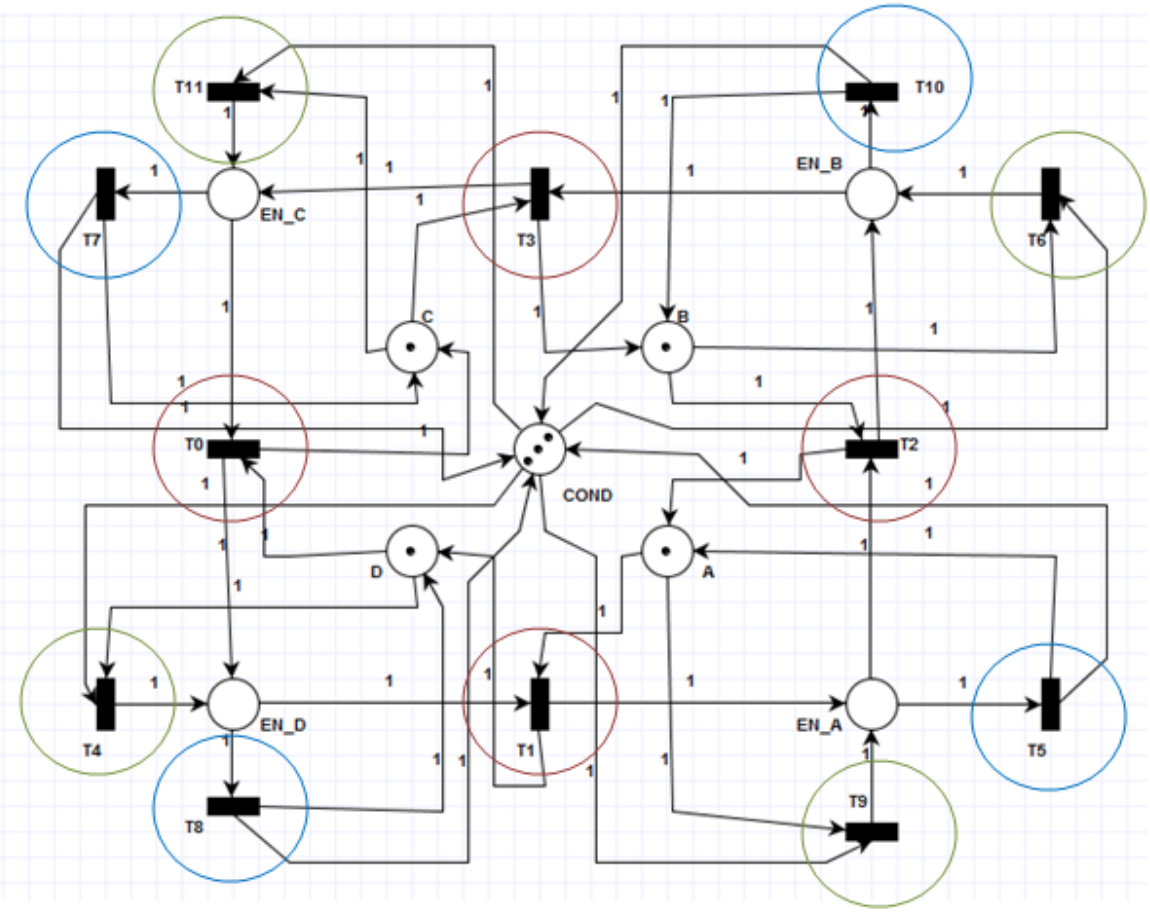
En cuanto a la inanición, se implementa una Política que determina qué auto tiene prioridad de dispararse. Esta política establece que las transiciones salientes de la esquina son las que tienen mayor prioridad de ser disparadas por un auto, a fin de que la esquina se libere los más rápido posible. En segundo lugar, se les da prioridad a las cuatro transiciones entrantes, para que la esquina esté llena la mayor parte del tiempo, utilizándose la capacidad máxima permitida de 3 autos. En tercer y último lugar tienen prioridad las transiciones interiores a la esquina, las que permiten que los autos se muevan en su interior. En adición y como punto importante para evitar la inanición, la política realiza entre cada disparo un cambio de prioridad para aquellas transiciones sensibilizadas y con autos en sus colas, que permanezcan un tiempo mayor a 50ms, dándole la máxima prioridad a esa transición para realizar un solo disparo. Esto evita que las transiciones que no consiguen obtener una oportunidad durante este período de tiempo, puedan realizar un disparo.

De acuerdo a esto, tenemos que:

Transiciones de mayor prioridad: T5, T10, T7 y T8

Transiciones de prioridad intermedia: T9, T6, T11 y T4

Transiciones de baja prioridad: T2, T3, T0 y T9



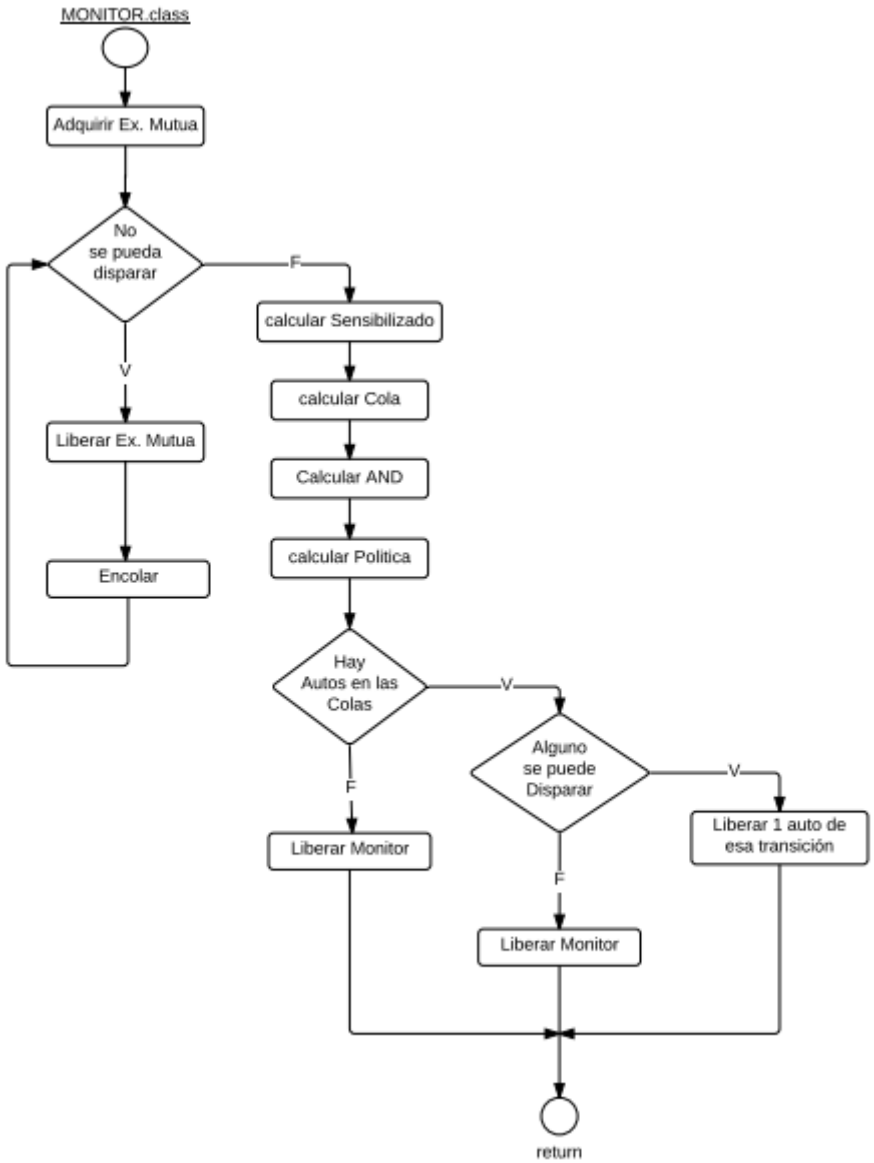
MONITOR

Debido a que los autos, que son hilos en ejecución, deben consultar la disponibilidad de los recursos de a uno por vez, es entonces que se hace uso de un monitor para cumplir esta tarea. La RdP creada representa la región crítica en un cruce, pero el monitor es quien controla que las consultas a esta RdP sea una a la vez, es decir, de a un auto y no todos juntos. Por eso, el monitor contiene un objeto RdP, el cual contiene un marcado inicial, además de la matriz de incidencia mostrada anteriormente, junto a otros atributos necesarios para la implementación del monitor.

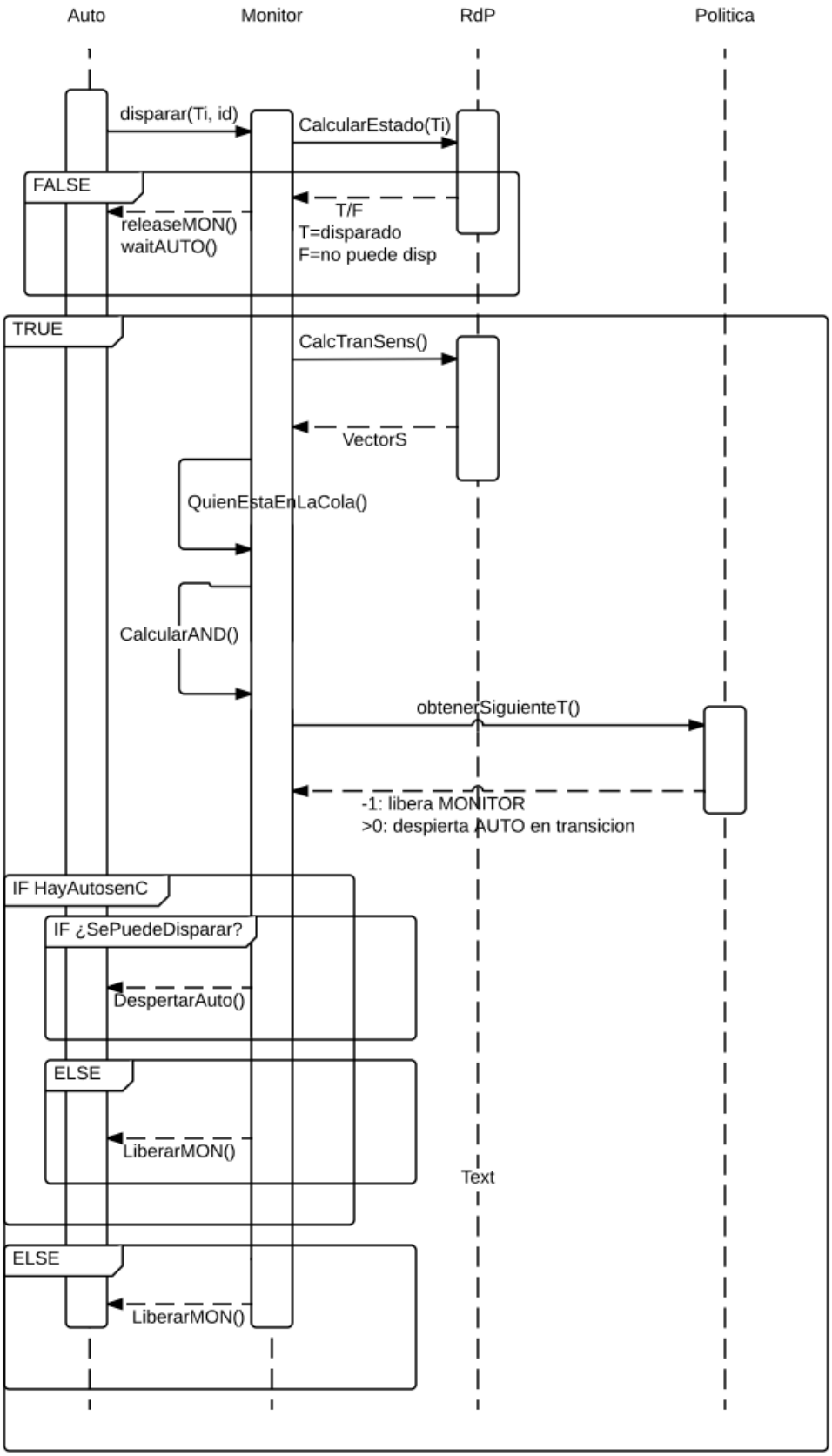
Dentro del monitor, cada transición de la RdP es representada mediante un Semáforo, contando este con una cola FIFO para los autos en espera de recursos. Mediante este semáforo, un auto intenta disparar una transición y en un bucle while se realiza una consulta sobre si el disparo fue realizado o no. Si el disparo no pudo ser realizado, el auto se “duerme” en la transición que intentó disparar a la espera del recurso.

El monitor contiene otros atributos además del objeto RdP, como, por ejemplo, un objeto Política que sirve para que un auto que logró disparar una transición, decida darle la exclusión mutua al mismo monitor, o a un auto que la Política decidió. Para saber cuál fue la decisión de la Política, se le pasa como argumento un vector que contiene el resultado de una operación AND entre dos vectores: el vector Sensibilizado, que contiene las transiciones que se pueden disparar, y el vector Colas, que contiene las transiciones que tienen al menos un auto en espera de recurso. Dependiendo del resultado de esta operación, el monitor decidirá qué auto se disparará después.

A continuación, se muestra un diagrama de flujo para entender el funcionamiento del Monitor:



Esto mismo se puede analizar también en el siguiente diagrama de secuencias, en el que se puede ver cómo van llamando y accediendo a los diferentes métodos de distintas clases:



Para más detalle, en el ANEXO se agrega el código del Monitor.

TESTING

Se realizaron varios test para probar el correcto funcionamiento del sistema. Se realizaron tanto UnitTest como SystemTest.

UNITTEST

Mediante los test unitarios se probó que los autos se agregan a la cola de espera del recurso cuando no pueden disparar una transición, y que la política decide correctamente qué transición disparar.

Para probar que un auto se agrega a la cola de una transición esperando que se libere un recurso, se crearon dos autos que disparan la misma transición. Luego de disparado el primero, el segundo auto debe quedar esperando que el auto que tomo el recurso, lo libere, y agregarse a la cola de la transición que quiere disparar.

Para probar la política, se simula que dos transiciones tienen autos esperando es sus colas, y que ambas están sensibilizadas. El método calcularPolitica () debe decidir qué transición disparar. El test fue exitoso, por lo que sabemos que el método funciona de manera correcta.

SYSTEMTEST

En los test de sistemas se probaron tres cosas: la Política, la Lógica, y los Tiempos. En la Política, se intenta probar que, el sistema responde correctamente a la decisión tomada por la Política. En la Lógica, se intenta probar que, dentro de cada espacio, no puede haber más de un auto, y que como máximo puede haber tres al mismo tiempo en la esquina. Y en los Tiempos, se intenta probar que el Tiempo de Respuesta del Monitor es bajo.

ID_Test	Test_01
Nombre del Test	POLITICA
Requerimiento	Se probará que la política predefinida responde adecuadamente a las exigencias del sistema.
Precondiciones	Las transiciones T4 y T0 deben tener un auto en sus respectivas colas, y dichas transiciones deben estar sensibilizadas. De esta manera, ambas pueden dispararse, pero la transición que se disparó antes, le dará la exclusión mutua a la transición que define la política. Ambas transiciones deben estar en conflicto debido a que necesitan el mismo recurso para poder dispararse.
Postcondición	La cola de T4 disminuye en un valor quedando sin autos en la misma, y dicha transición queda NO sensibilizada. La transición T0 queda en espera del recurso y por lo tanto queda NO sensibilizada.
Secuencia	- Crear un objeto Monitor - Creamos 4 objetos de tipo Auto, y le asignamos un camino a cada auto. Los caminos para cada auto son: A1: T11, T0, T8 A2: T11, T0, T8 A3: T4, T1, T5 A4: T4, T1, T5 (*) - Iniciamos los hilos.
Resultado Esperado	Se debe ejecutar la T4 primero.
Resultado Obtenido	Se disparó la transición T4.
Estado del TEST	EXITOSO.

(*) Se asignan estos 4 caminos para que se dé la situación en la que T0 y T4 tienen un auto esperando el recurso para dispararse. Entonces, cuando el recurso esté disponible, la política debe elegir disparar la transición que tiene más prioridad.

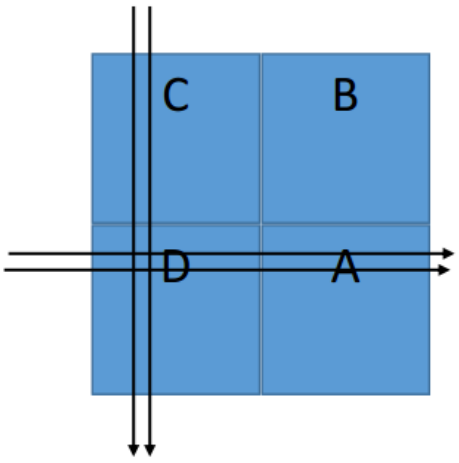


Figura que ilustra el recorrido de los autos inicializados, lo cual produce el conflicto por el recurso compartido

T Disp	A Disp	Autos en Colas											Sensibilizado											AND										
		0	1	2	3	4	5	6	7	8	9	10	11	0	1	2	3	4	5	6	7	8	9	10	11	0	1	2	3	4	5	6	7	8	9	10	11
T11	a1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T0	a1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T11	a2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T8	a1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
T4	a3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T1	a3	1	0	0	0	0	1	0	0	0	0	0	0	1	0	1	1	0	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
T4	a4	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T5	a3	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T1	a4	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T0	a2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T5	a4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T8	a2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ID_Test	Test_02
Nombre del Test	LOGICA
Requerimiento	Se probará que las plazas de la Red de Petri que corresponden a los espacios de las esquinas, tienen, como máximo, un auto en su interior. Además, se verifica que no pueden haber más de 3 autos en la esquina.
Precondiciones	Los espacios A, B, C y D, están vacíos, por lo tanto, los recursos están disponibles.
Postcondición	Los espacios A, B y C, tendrán un auto en su interior, pero las transiciones de ingreso a estas plazas estarán NO sensibilizadas y tendrán autos en sus colas de bloqueo.
Secuencia	- Crear un objeto Monitor - Crear 4 caminos. - Creamos 8 objetos de tipo Auto, asignando el mismo camino a cada par de autos. (**) - Iniciamos los hilos.
Resultado Esperado	Se realizan los disparos de los tres primeros autos, y el resto no puede disparar debido a que no puede haber más de un auto en cada plaza, y se encolan en sus transiciones de entrada. Además, el auto que quiere ingresar a D, no debe poder hacerlo porque ya hay 3 autos en la esquina.
Resultado Obtenido	Hay 1 auto en A, en B y en C, y en las transiciones de entrada a estas, hay autos en espera. Hay solo 3 autos en total en toda la esquina.
Estado del TEST	EXITOSO.

(**) Se crean 8 autos, donde 2 autos tienen el mismo camino asignado. De esta manera, se generarán esperas en las transiciones debido a que no puede haber más de 3 autos al mismo tiempo dentro de la esquina, y tampoco puede haber más de un auto dentro de un espacio o plaza.

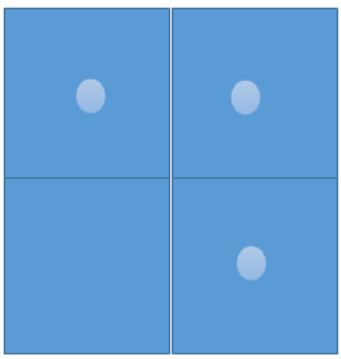
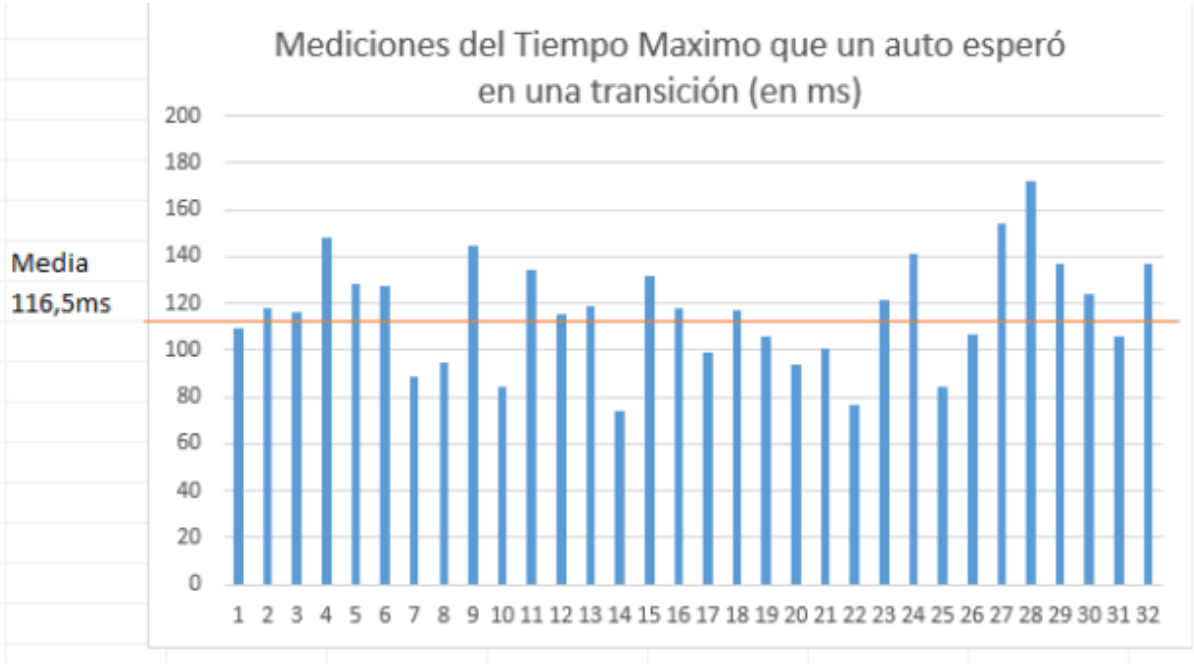


Figura que ilustra la esquina con la capacidad máxima utilizada:
máximo 3 autos por equina
máximo 1 auto por espacio

ID_Test	Test_03
Nombre del Test	TIEMPO_COLAS
Requerimiento	Se probará que el tiempo de respuesta de un monitor.
Precondiciones	La esquina estará inicialmente vacía y 20 autos intentan realizar su camino.
Postcondición	La esquina estará vacía y los 20 autos recorren su camino.
Secuencia	- Se crea un objeto Monitor - Se crean los 12 caminos posibles. (***) - Creamos 20 objetos de tipo Auto. (****) - Se asigna a cada auto un camino aleatoriamente. - Se inician los hilos.
Resultado Esperado	El tiempo de respuesta del Monitor será menor a 120 milisegundos.
Resultado Obtenido	Se realizaron 30 ejecuciones y la media fue de 115 milisegundos.
Estado del TEST	EXITOSO.

(***) Son los máximos posibles.
(****) Testeamos 20 autos. Se puede testear con más autos, pero el tiempo de respuesta será mayor, ya que cambiará debido al incremento en los autos creados.



EJECUCIÓN

A continuación, se muestra la ejecución del programa, donde se puede ver la interfaz gráfica. Además, se adjunta una captura de cómo se muestran los resultados de la ejecución. Vale aclarar que, en los resultados, se ven los datos disparo a disparo, por lo que su extensión aumenta considerablemente con el aumento en el número de autos. Se mostrará una ejecución de 10 autos.

Imagen tomada en un instante t cualquiera. Al finalizar la ejecución, la cantidad de autos ejecutados y la cantidad de autos que finalizaron es la misma. En rojo, se muestra el auto que ejecuto el disparo.

T Disp	A Disp	Autos en Colas											Sensibilizado											AND											PRIORIDAD											MARCADO ACTUAL										
		0	1	2	3	4	5	6	7	8	9	10	11	0	1	2	3	4	5	6	7	8	9	10	11	0	1	2	3	4	5	6	7	8	9	10	11	0	1	2	3	4	5	6	7	8	9	10	11	A	B	C	D	eA	eB	eC	eD	Cond
T11	A0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	1	0	1	0	1	0	2
T3	A1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	1	0	1	0	1	0	1
T6	A2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	1	0	1	0	1	0	1
T7	A0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	5	10	7	8	9	6	11	4	2	3	0	1	1	0	0	1	1	1	1	0	0	1
T4	A3	0	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	5	10	7	8	9	6	11	4	2	3	0	1	1	0	0	1	1	0	1	0	0
T5	A1	0	0	0	0	0	1	0	1	0	0	1	0	0	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	0	1	0	1	0	1	1
T3	A4	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	1	0	1	0	1	0
T10	A2	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	1	0	1	0	1	1
T6	A6	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	6	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	1	0	1	0	1	0
T5	A4	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	3	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	1	0	1	0	1	1
T3	A7	0	1	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	1	0	1	0	1	1
T10	A6	0	1	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	1	0	1	0	1	0
T11	A5	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	11	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	1	0	1	0	1	1
T2	A7	0	1	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	1	0	1	0	1	0	1
T1	A3	0	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	10	7	8	9	6	11	4	2	3	0	1	1	0	0	1	1	0	1	0	0	1
T7	A5	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	5	10	7	8	9	6	11	4	2	3	0	1	1	0	0	1	1	0	1	0	0	1
T4	A8	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	4	5	10	7	8	9	6	11	4	2	3	0	1	1	0	0	1	1	0	1	0	0
T3	A7	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	0	1	0	1	0	1	1
T5	A3	0	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	5	10	7	8	9	6	11	4	2	3	0	1	1	0	0	1	1	0	1	0	0
T9	A9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	3	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	1	0	1	0	1
T7	A7	0	1	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	10	7	8	9	6	11	4	2	3	0	1	1	0	1	0	1	0	1	0	1	1
T5	A9	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	10	7	8	9	6	11	4	2	3	0	1	1	0	0	1	1	0	1	0	0	1
T1	A8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	5	10	7	8	9	6	11	4	2	3	0	1	1	0	0	1	1	0	1	0	0
T2	A8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	5	10	7	8	9	6	11	4	2	3	0	1	1	0	0	1	1	0	1	0	0
T10	A8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	5	10	7	8	9	6	11	4	2	3	0	1	1	0	0	1	1	0	1	0	0

CONCLUSIÓN

Hemos comprendido cómo manejar e interactuar con una Red de Petri implementada en un Monitor. Como resultado del uso de esta herramienta se comprobó que es posible simular y administrar la concurrencia de autos de una esquina de manera sencilla e intuitiva.

Inicialmente una dificultad fue el dibujo del esquema de la red, por no saber identificar claramente los recursos y espacios, y hacerlos corresponder con plazas y transiciones. Luego pudimos observar que las Redes de Petri son fáciles de entender debido a la naturaleza gráfica y precisa del esquema de representación, y que se puede analizar el comportamiento del sistema con el gráfico y el marcado.

Al realizar este proyecto, pudimos darnos cuenta de la utilidad de los diagramas de clases, secuencia y flujo para la visualización y el entendimiento del problema. Esto simplifica el proceso de programación.

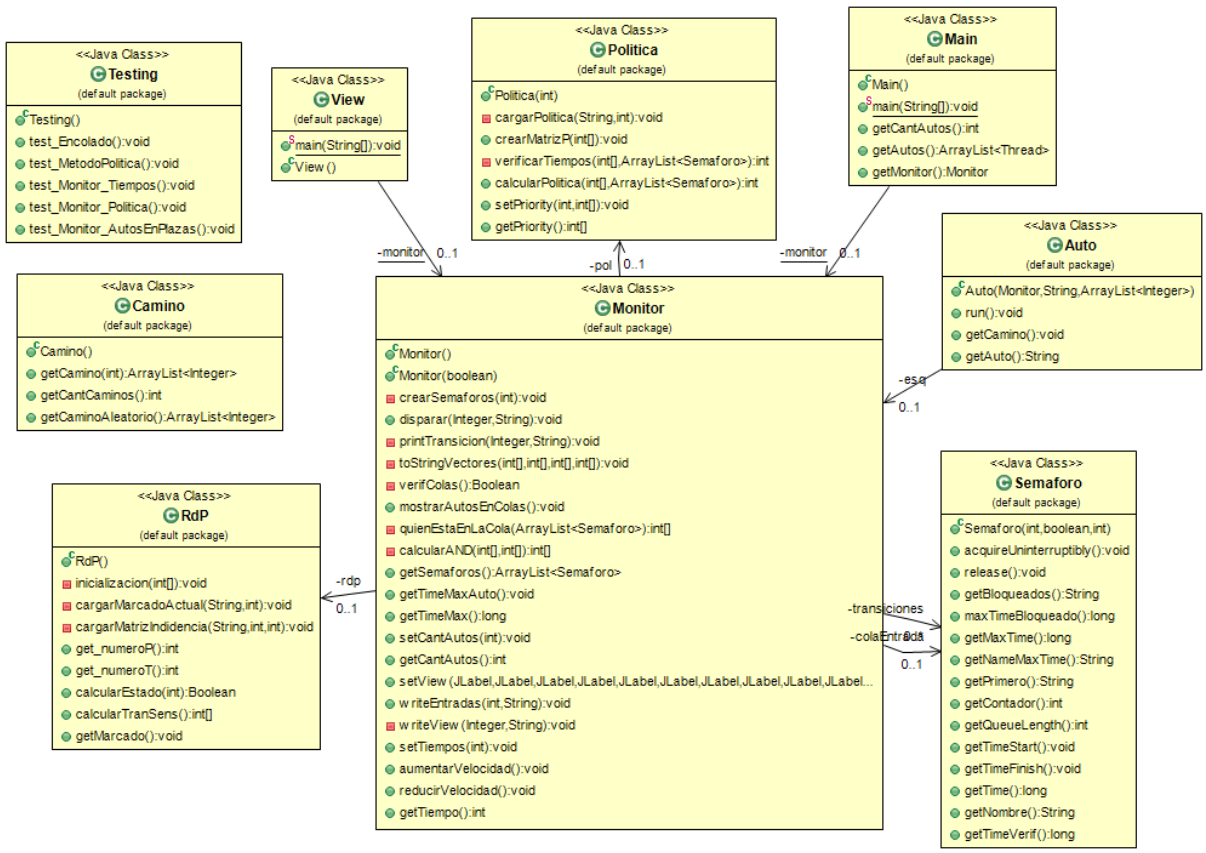
Destacamos también, que el uso de la prioridad que se le debe dar a cada una de las transiciones de una red, tiene mucha importancia en cuanto al comportamiento de la red durante su evolución.

Cuando las condiciones del sistema, la política, o incluso cuando la red en sí se modifica, es posible utilizar la misma herramienta simplemente cambiando los archivos correspondientes sin necesidad modificar ni una línea de código.

Es debido a esto que se puede concluir que las Redes de Petri son una gran herramienta para administrar procesos que necesitan compartir recursos, con una excelente mantenibilidad y una gran utilidad en infinidad de aplicaciones.

ANEXO A

DIAGRAMA DE CLASES



ANEXO B

CODIGO MONITOR

```
public class Monitor {

	private int outCars = 0; //Indicar de la cantidad de autos que terminaron su ejecucion
	private int cantP, cantT, cantA; //Cantidad de Plazas y Cantidad de Transiciones
	private Semaforo colaEntrada; //Cola de entrada al Monitor
	private ArrayList<Semaforo> transiciones = new ArrayList<Semaforo>(); //Array de Semaforos. 1 por c/T
	private RdP rdp;
	private Politica pol;
	private int[] vectorC, vectorS, vectorAND; //Vectores que guardan los vectores COLA, SENSIBILIZADO, y AND
	private int dispT; //Indicador de la proxima transicion a disparar segun la politica
	private JLabel pA, pB, pC, pD, pN, pS, pE, pO, psN, psS, psE, psO, fin;
	private boolean ejecucionDesdeView = false;
	private int tiempo=0;
	public Monitor(){
		rdp = new RdP(); //Creamos un objeto RdP (Red de Petri)
		cantP = rdp.get_numeroP();
		cantT = rdp.get_numeroT();
		pol = new Politica(cantT);
		crearSemaforos(cantT); //Creamos los semaforos segun la cantidad de transiciones
		vectorC = new int[]{0,0,0,0,0,0,0,0,0,0,0,0};
		vectorS = new int[]{0,0,0,0,0,0,0,0,0,0,0,0};
		vectorAND = new int[]{0,0,0,0,0,0,0,0,0,0,0,0};
	}

	public Monitor(boolean fuenteEjecucion){
		ejecucionDesdeView = true;
		rdp = new RdP(); //Creamos un objeto RdP (Red de Petri)
		cantP = rdp.get_numeroP();
		cantT = rdp.get_numeroT();
		pol = new Politica(cantT);
		crearSemaforos(cantT); //Creamos los semaforos segun la cantidad de transiciones
		vectorC = new int[]{0,0,0,0,0,0,0,0,0,0,0,0};
		vectorS = new int[]{0,0,0,0,0,0,0,0,0,0,0,0};
		vectorAND = new int[]{0,0,0,0,0,0,0,0,0,0,0,0};
	}
	/*
	 * Funcion que crea los semaforos segun la cantidad de transiciones que se les pase
	 */
	private void crearSemaforos(int t) throws NullPointerException{
		colaEntrada = new Semaforo(1,false,13);
		for(int j=0;j<t;j++)
			transiciones.add(new Semaforo(0,false,j)); // 0 semaforo en rojo. 1 semaforo en verde
	}

	/*
```

```
	* Funcion que llaman los Autos cuando quieren disparar una
transicion.
	* @transicion: transicion a disparar
	* @id: id del auto que disparar la transicion
	*/
      public void disparar(Integer transicion, String id) throws
InterruptedException {

		colaEntrada.acquireUninterruptibly(); //Monitor ocupado

		// Mientras no se pueda disparar
		while(!(rdp.calcularEstado(transicion))){
			colaEntrada.release(); //Liberamos cola de entrada del
monitor
			transiciones.get(transicion).acquireUninterruptibly();
//El auto que no pudo disparar

				//se duerme y se encola.
		}

		//Cuando se ejecutan las transiciones de salida, outCars suma
1.
		if(transicion==5 || transicion==10 || transicion==7 ||
transicion==8){
			outCars++;
		}
		if(ejecucionDesdeView){
			writeEntradas(transicion,id);
			writeView(transicion,id);
		}

		// Cuando se disparó...
		vectorS = rdp.calcularTranSens(); //Calcular vector
sensibilizado
		vectorC = quienEstaEnLaCola(transiciones); //Verifica que
transicion tiene autos en espera
		vectorAND = calcularAND(vectorS, vectorC); //Calcula la AND,
indicando transiciones que

//pueden dispararse y que tienen autos en espera
		printTransicion(transicion,id); //Se imprime mensaje
		//Cuando hay al menos un auto en espera en alguna transicion,
el boolean es TRUE
		Boolean hayAutosColas = verifColas();
		//Se calcula que transicion debe dispararse segun la politica
		dispT = pol.calcularPolitica(vectorAND,transiciones);

		//Se pasa la exlusion mutua a quien corresponda
		if(hayAutosColas){ //Si alguno de los autos encolados se puede
disparar, entonces lo despierto
			if(dispT>=0){
				transiciones.get(dispT).release();
			}
			else colaEntrada.release(); //Si no se puede disparar,
libero el monitor
		}
		else{ //Si no hay autos en las colas, se libera el monitor
			colaEntrada.release();
		}

	}
```

```
/*
	 * Funcion que devuelve el vector C, indicando que transiciones
tienen autos en espera
	 */
	private int[] quienEstaEnLaCola(ArrayList<Semaforo> colas){
		int[] vector = new int[]{0,0,0,0,0,0,0,0,0,0,0,0};
		Semaforo s;
		for(int i=0; i<cantT; i++){
			s = colas.get(i);
			if(s.getQueueLength()>0)
				vector[i] = 1;
		}
		return vector;
	}

	/*
	 * Funcion que retorna la AND entre S y C.
	 * S: transiciones SENSIBILIZADAS
	 * C: transiciones con autos en ESPERA
	 * AND: transiciones que tienen autos en ESPERA y SE PUEDEN DISPARAR
	 */
	private int[] calcularAND(int[] s, int[] c){
		int[] resultado = new int[]{0,0,0,0,0,0,0,0,0,0,0,0};
		for(int i=0; i<cantT; i++){
			resultado[i] = s[i] & c[i];
		}
		return resultado;
	}
}
```

CODIGO RDP

```
public class RdP {
	private int[][] incidencia;
	private int[] marcadoActual;
	private int A,B,C,D,EnA,EnB,EnC,EnD,Cond;
	public RdP(){
		cargarMatrizIndidencia("matrizI_default.txt",9,12);
		cargarMarcadoActual("marcado_default.txt",9);
	}

	private void inicializacion(int[] marcadoActual) {
		A=marcadoActual[0];
		B=marcadoActual[1];
		C=marcadoActual[2];
		D=marcadoActual[3];
		EnA=marcadoActual[4];
		EnB=marcadoActual[5];
		EnC=marcadoActual[6];
		EnD=marcadoActual[7];
		Cond=marcadoActual[8];
	}

	private void cargarMarcadoActual(String file_name, int numeroP) {
		int cant_plazas = numeroP;
		marcadoActual = new int[cant_plazas];
		try{
```

```
            FileInputStream fstream = new FileInputStream(file_name);    //
Abrimos el archivo
            DataInputStream entrada = new DataInputStream(fstream);
      // Creamos el objeto de entrada
            BufferedReader buffer = new BufferedReader(new
InputStreamReader(entrada));        // Creamos el Buffer de Lectura
            String strLinea;
            int j=0;
            // Leer el archivo linea por linea
            while ((strLinea = buffer.readLine()) != null)   {
                  String [] linea = strLinea.split(" ");  //Separamos la
linea por cada espacio y lo guardamos en un arreglo
                  for(j=0;j<cant_plazas;j++){
                        marcadoActual[j] = Integer.parseInt(linea[j]);
                  }
            }
            entrada.close();  // Cerramos el archivo
        }catch (Exception e){ //Catch de excepciones
            System.err.println("Ocurrio un error: " + e.getMessage());
        }
      }

      private void cargarMatrizIndidencia(String file_name, int numeroP,
int numeroT) {
            int cant_trans = numeroT;
            int cant_plazas = numeroP;
            incidencia = new int[cant_plazas][cant_trans];
        try{
            FileInputStream fstream = new FileInputStream(file_name);    //
Abrimos el archivo
            DataInputStream entrada = new DataInputStream(fstream);
      // Creamos el objeto de entrada
            BufferedReader buffer = new BufferedReader(new
InputStreamReader(entrada));        // Creamos el Buffer de Lectura
            String strLinea;
            int j=0, pos;
            // Leer el archivo linea por linea
            while ((strLinea = buffer.readLine()) != null)   {
                  String [] linea = strLinea.split(" ");
                  pos = 0;
                  for(int i=0;i<cant_trans;i++){
                        incidencia[j][pos] = Integer.parseInt(linea[pos]);
                        pos++;
                  }
                  j++;
            }
            entrada.close();  // Cerramos el archivo
        }catch (Exception e){ //Catch de excepciones
            System.err.println("Ocurrio un error: " + e.getMessage());
        }
    }

      public Boolean calcularEstado(int transicion){
            for(int i=0 ; i<get_numeroP() ; i++){
                  //Vemos que plazas van a una transicion en particular, y
si esas plazas tienen 0 token.
                  if (( incidencia[i][transicion] < 0 ) && ((
marcadoActual[i] - 1) < 0 )){
                        return false;
                  }
            }
```

```
		for(int i=0 ; i<get_numeroP() ; i++){
			marcadoActual[i] += incidencia[i][transicion];
		}
		return true;
	}

	public int[] calcularTranSens(){
		inicializacion(marcadoActual);
		int[] vectorS = new int[]{0,0,0,0,0,0,0,0,0,0,0,0};

		if(EnC==1 && D==1 && EnD==0)
			vectorS[0]=1; //T0 sensibilizada
		if(EnD==1 && EnA==0 && A==1)
			vectorS[1]=1; //T1 sensibilizada
		if(EnA==1 && B==1 && EnB==0)
			vectorS[2]=1; //T2 sensibilizada
		if(EnC==0 && EnB==1 && C==1)
			vectorS[3]=1; //T3 sensibilizada
		if(D==1 && Cond>0 && EnD==0)
			vectorS[4]=1; //T4 sensibilizada
		if(EnA==1)
			vectorS[5]=1; //T5 sensibilizada
		if(B==1 && Cond>0 && EnB==0)
			vectorS[6]=1; //T6 sensibilizada
		if(EnC==1)
			vectorS[7]=1; //T7 sensibilizada
		if(EnD==1)
			vectorS[8]=1; //T8 sensibilizada
		if(A==1 && Cond>0 && EnA==0)
			vectorS[9]=1; //T9 sensibilizada
		if(EnB==1)
			vectorS[10]=1; //T10 sensibilizada
		if(C==1 && Cond>0 && EnC==0)
			vectorS[11]=1; //T11 sensibilizadas
		return vectorS;
	}
```