

**PRESENTACIÓN TRABAJO FINAL**

PROGRAMACIÓN CONCURRENTE

Profesores: Dr. Ing. Orlando Micolini

    Ing. Luis Ventre

Integrantes:

Gon, Alexander Mateo

Lenta, Luis Alejandro

Orecchini Alem, Stefano Mauricio



**INDICE:**

**RESUMEN 3**

**INTRODUCCIÓN 3**

**ESQUEMA 4**

**IMPLEMENTACIÓN 4**

**DISEÑO 8**

**CONCLUSIÓN 16**

**RESUMEN:**

En este trabajo se buscó aplicar los conceptos aprendidos durante el cursado de la materia “Programación Concurrente”, y alguna que otra materia ya cursada, para poder aplicarlo a un problema de concurrencia real de Ingeniería, de manera que podamos resolver el mismo de la manera más satisfactoria posible.

Para ello, además, debimos investigar mucho sobre el trabajo de las redes de Petri temporales y cómo implementar las mismas en el lenguaje de programación orientado a objetos (JAVA).

**INTRODUCCIÓN:**

En este practico se debe resolver el problema planteado en (Naiqi and MengChu, 2010), “Sistema de manufacturación robotizado".

La red de Petri que modela a dicho sistema, debe modificarse con el fin de evitar interbloqueos. Luego simular la solución en un proyecto desarrollado con la herramienta adecuada (explique porque eligió la herramienta usada).

El sistema de manufacturación consiste en tres robots R1, R2 y R3 cuatro máquinas M1, M2, M3 y M4, tres tipos diferentes de piezas a procesar A, B y C, como se observa en la figura.

Las piezas provienen de tres contenedores de entrada distintos, I1, I2 e I3, de los cuales los robots las retiran, las colocan en las máquinas para su procesamiento y depositan en tres contenedores de salidas distintos, que son: O1, O2 y O3.

Los tiempos que emplean cada máquina para producir las piezas, son los siguientes:

M1=30, M2=5, M3=24, M4=10 (Para producir pieza A)

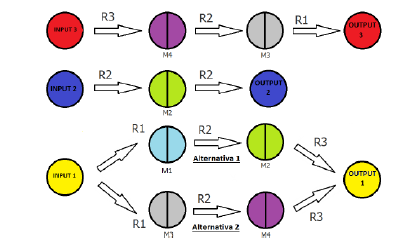
M2=15 (Para producir pieza B)

M3=18 (Para producir pieza C)

M4=5 (Para producir pieza D)

**ESQUEMA DEL PROBLEMA:**

En la siguiente figura, podemos ver de qué problema se trata, en una forma esquemática, fácil de entender.



**IMPLEMENTACIÓN:**

Antes de comenzar con la programación, agarramos la red de Petri, dada en el práctico, la cual corrimos con Pipe y nos dimos cuenta que se trababa en ciertas partes.

Los **BLOQUEOS PROPIOS** encontrados en la red fueron los siguientes:

Línea de Producción A: Esta línea no tenía interbloqueos.

Línea de Producción B: Esta línea se puede bloquear a sí misma. En caso que queramos volver a disparar la transición T21, luego de haber disparado las transiciones T21 y T22. Esto se debe a que la transición T23, posee el recurso M2 (Máquina 2), pero no posee el recurso R2 (robot 2), para que deposite la pieza en la máquina. En cambio T22, posee R2 (robot2), pero no posee M2(máquina 2), lo que quiere decir que el robot no puede dejar la pieza en la máquina, debido a que esta se encuentra trabajando. Para solucionar esto, se agregó otra plaza, la cual nos permite evitar que se dispare la transición T21, sin antes haberse disparado T23.

Línea de Producción C:

Esta línea no posee bloqueo propio.

Y los **INTERBLOQUEOS** de la red encontrados, fueron los siguientes:

Línea de Producción AB:

En caso que se ejecuten de manera intercalada la línea de producción A y B, puede darse un interbloqueo debido al acceso de A a la misma máquina y robot (M2, R2) que utiliza B.

En caso de la secuencia de ejecución T10, T11, T22 y T13, se produce un interbloqueo. Esto se debe a que la línea de producción B requiere utilizar el brazo del robot para terminar el procesamiento de la máquina 2, mientras que la línea A, posee el brazo del robot, pero necesita de la máquina 2 para poder depositar la pieza en la misma. La solución para esto, fue colocar una plaza que posee un solo recurso que controla el acceso a M2 y R2.

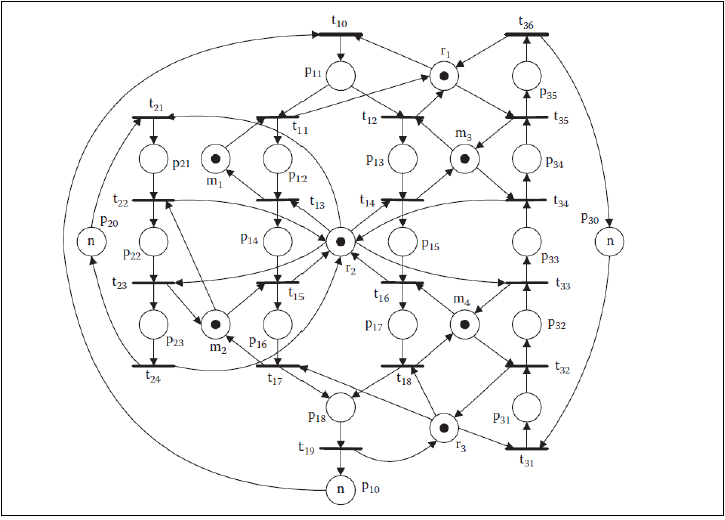
Línea de Producción AC:

En caso que se de la ejecución intercalada entre la línea A y la línea C, se produce un interbloqueo, debido a que las dos líneas utilizan los mismos recursos (Máquinas). La solución fue colocar una plaza que posee un recurso, que controla el acceso por completo a toda la línea de producción, es decir, si por algún motivo, A comienza primero, la línea C deberá esperar hasta que termine A y viceversa.

Luego de encontrar estos bloqueos mencionados anteriormente, procedimos a desarrollar la red de nuevo, aplicando conocimientos adquiridos en Modelos y Simulación, dándonos cuenta así, que las transiciones en la RdP, son lo mismo que las actividades vistas en Modelos.

De esta manera se nos facilitó el trabajo y pudimos desbloquear la red, de manera que al realizar nuevamente la simulación, corría sin problemas, es decir no había más loops en la misma.

En la siguiente figura podemos observar la red dada en el práctico:

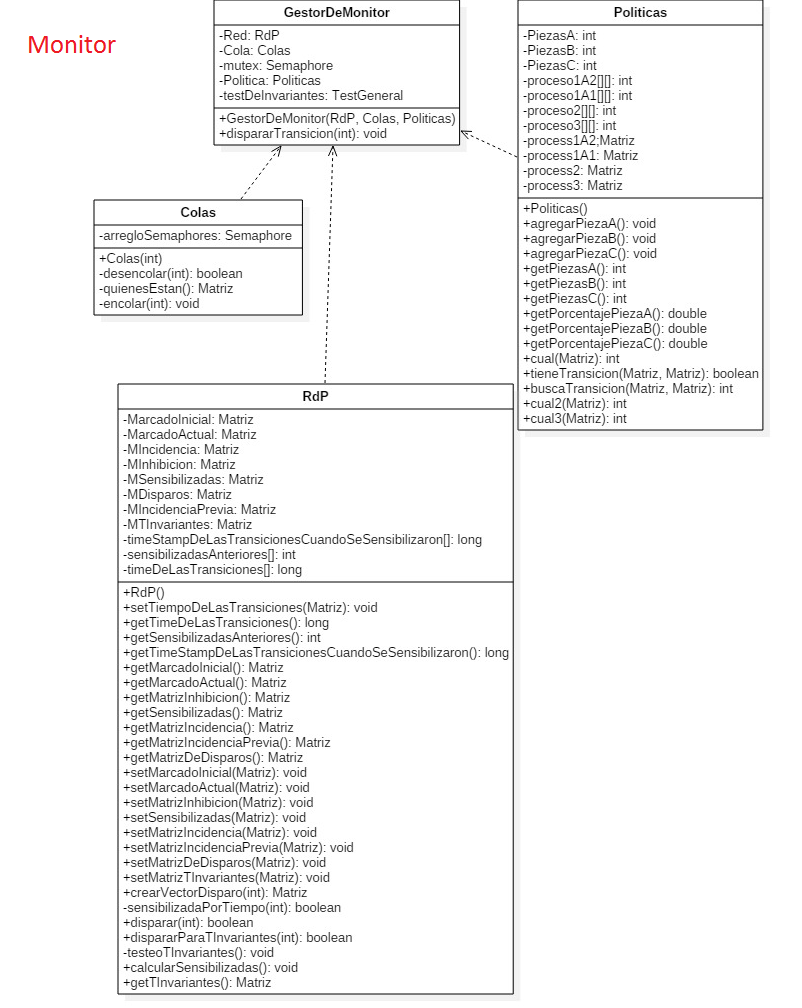


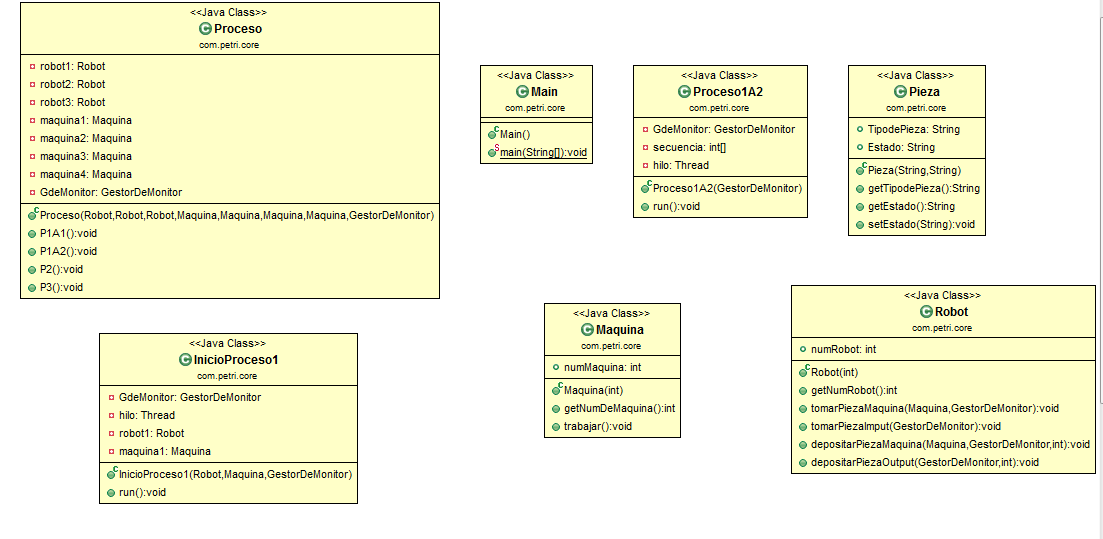
Y en la siguiente figura, se puede observar la red con los cambios pertinentes, para lograr la simulación sin interbloqueos.

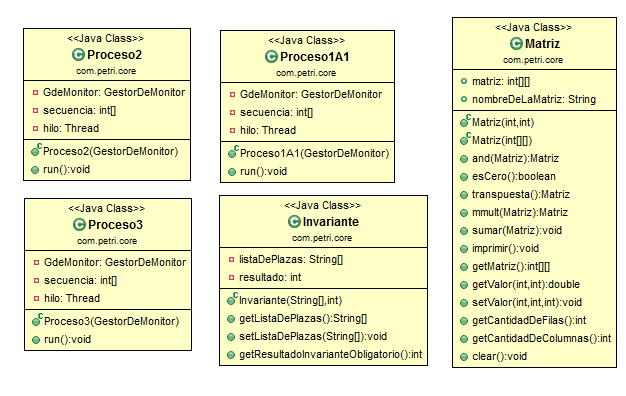
RED DE PETRI DESBLOQUEADA (AQUÍ)

**DISEÑO:**

**DIAGRAMA DE CLASES:**



Sistema Robot-Procesos-Máquinas:



En nuestro programa, podemos observar un GestorDeMonitor, el cual es el encargado de conseguir la exclusión mutua, mediante semáforos o coordinación de procesos.

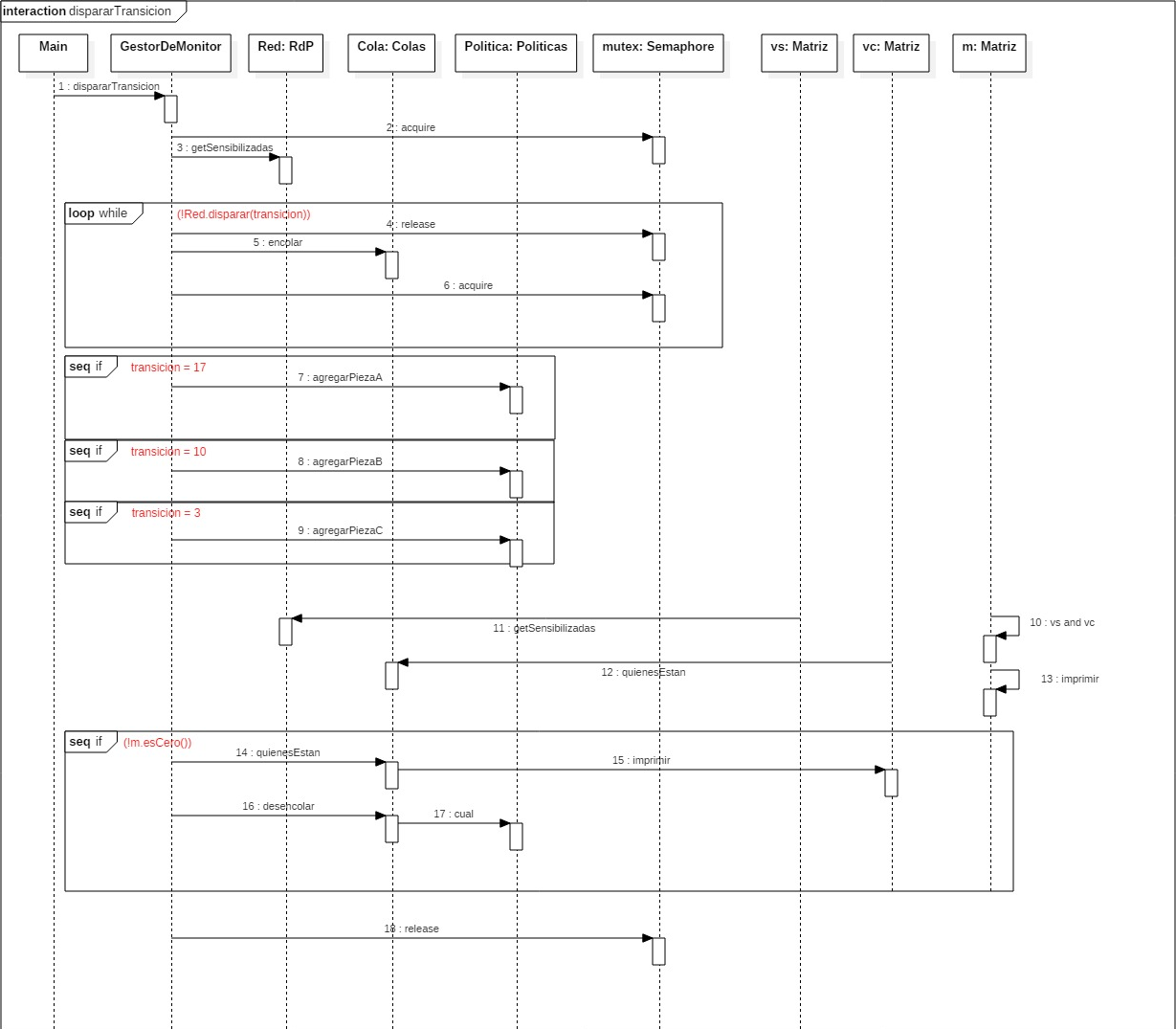
El monitor crea una RdP, la cual va cambiando de estados a medida que se van ejecutando las distintas transiciones. Cuando una transición quiere ejecutarse, debe preguntarle al monitor si puede ejecutarse o no, el cual lo determina, a través de la RdP. Dicho de otra manera, el monitor es el encargado de decidir que se ejecutará y que no.

Dentro del monitor, podemos encontrar políticas, que son las encargadas de regir el comportamiento del sistema y nos indican la prioridad que tiene la transición en caso que exista algún conflicto, es decir, en caso que dos o más transiciones tengan el permiso del monitor para ejecutarse y se encuentren listas para ello.

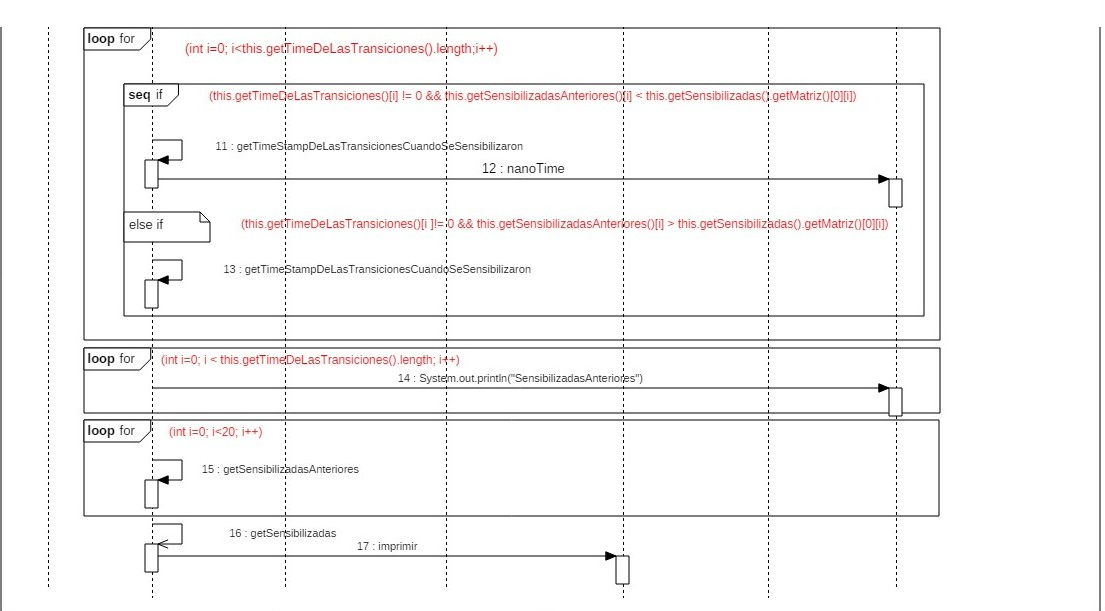
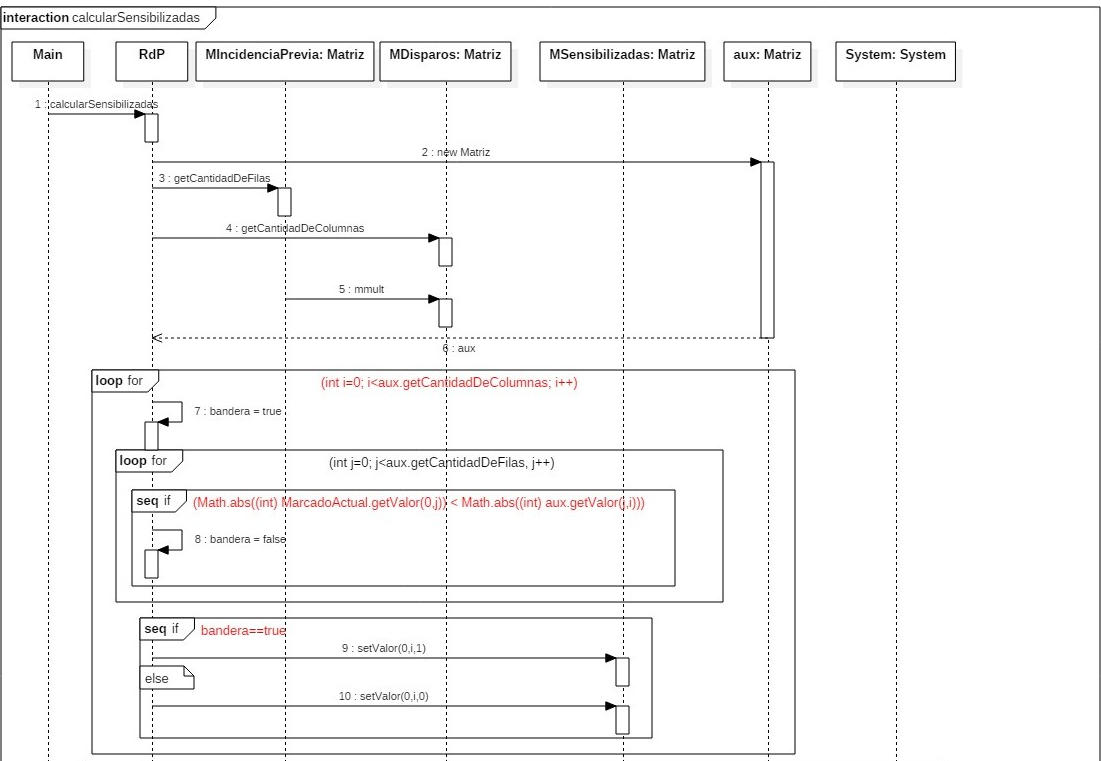
**DIAGRAMA DE SECUENCIAS:**

Con el objetivo de poder observar rápidamente como se interrelacionan los elementos más críticos del sistema, se realizaron una serie de diagramas de secuencia, los cuales se muestran a continuación:

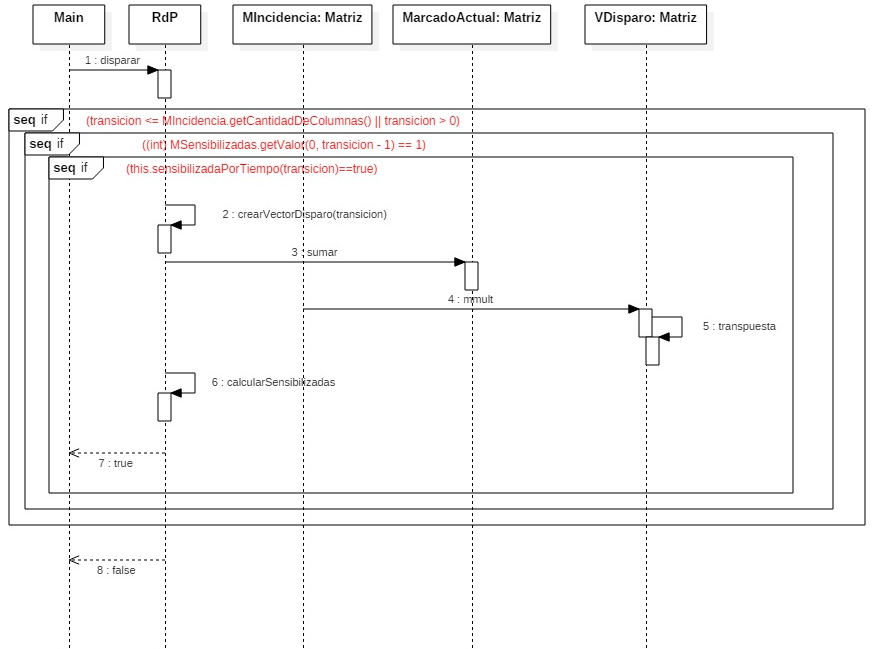
**GestorDeMonitor: dispararTransicion (Método que permite disparar una transición)**



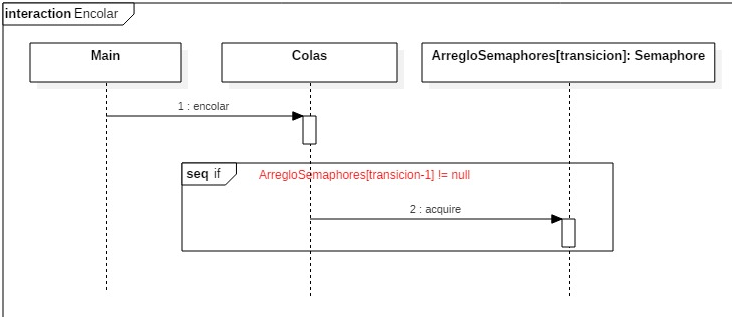
**RdP: CalcularSensibilizadas (Método que nos devuelve la matriz de que transiciones se encuentran sensibilizadas)**

****

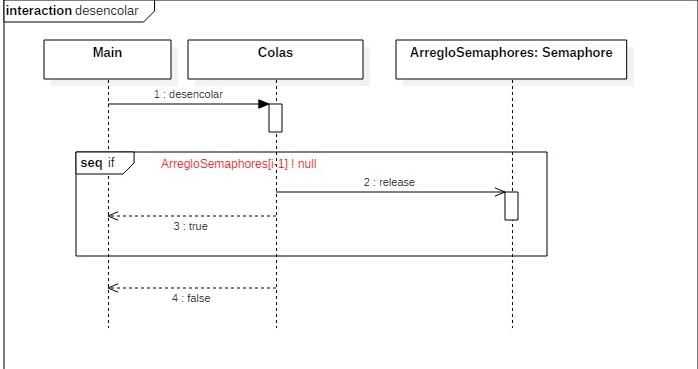
**RdP : disparar (Método que nos devuelve si se puede o no disparar una transición)**

****

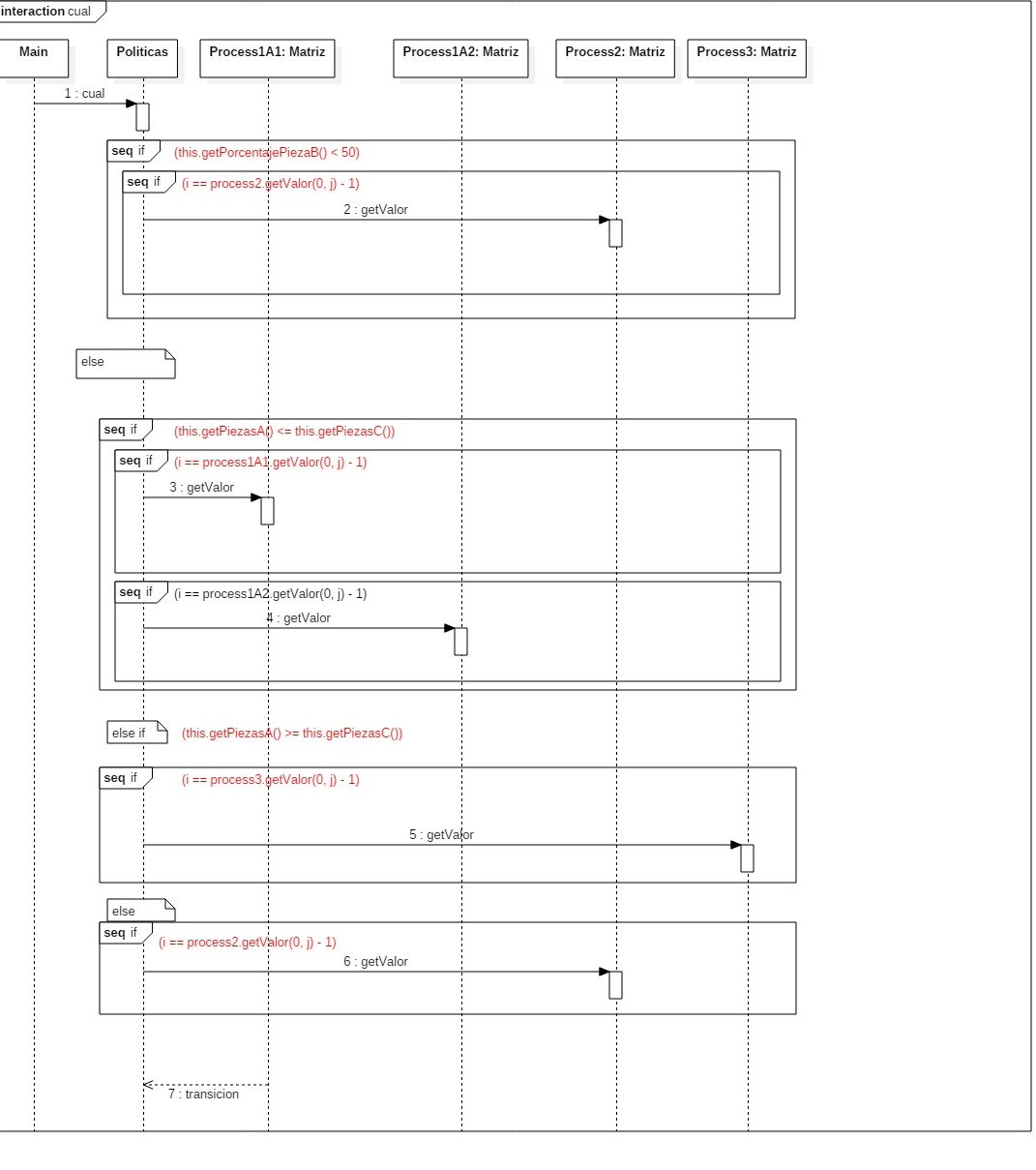
**Colas : Encolar (Método que encola los hilos que deben esperar su turno para dispararse)**

****

**Colas : Desencolar (Método que despierta los hilos para que se disparen)**

****

**Políticas : cual (Método que nos devuelve que transición se puede disparar, según los criterios de las políticas)**

****

**CONCLUSION:**

Como conclusión final de este trabajo, podemos decir que la materia Programación Concurrente nos permitió tener una noción de cómo encarar problemas reales de Ingeniería, donde se tienen gran cantidad de actividades concurrentes al mismo tiempo.

Además podemos decir que nos dimos cuenta que estos sistemas son altamente críticos a un fallo, por lo cual, debemos considerar minimizar al máximo, los mismos.

Por otro lado, aprendimos a utilizar herramientas de simulación, que no teníamos conocimientos algunos anteriormente, y que facilitan mucho la labor a la hora de hacer un sistema concurrente.

Por lo cual se puede decir que fue una gran experiencia el cursado de la misma y estamos seguros que usaremos todo lo aprendido, en un futuro no muy lejano.