Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Físicas y Naturales

Cátedra de Programación Concurrente

TRABAJO FINAL

"Sistema de Producción robotizado de piezas"

Profesores:

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

Alumnos:

Orecchini Alem, Stefano Mauricio.
Lenta, Luis Alejandro.
Gonn, Alexander.

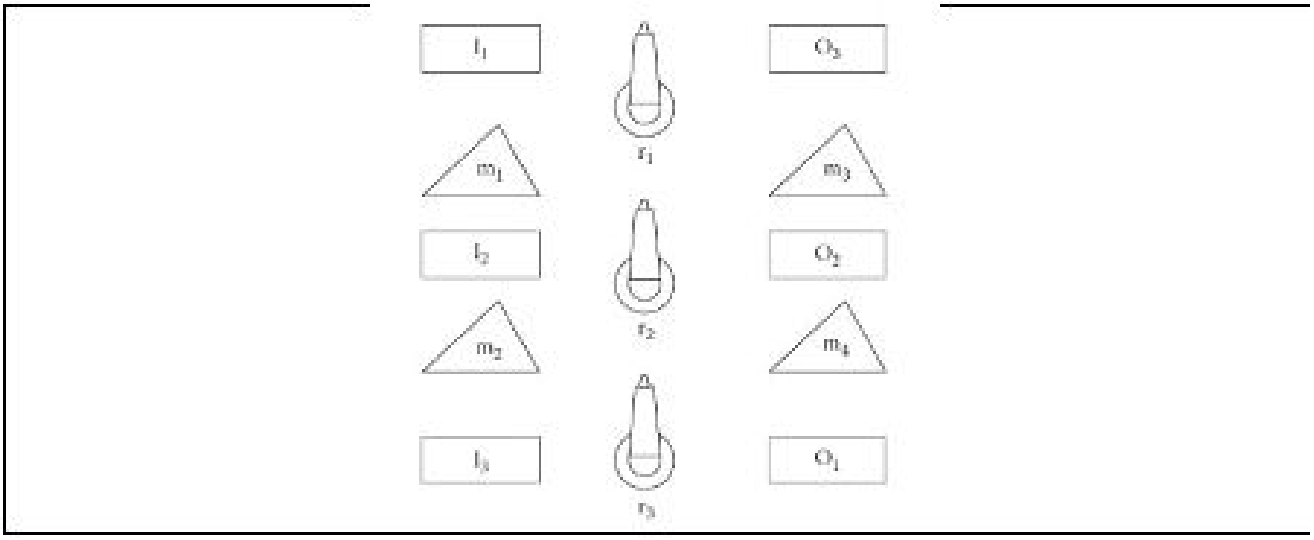
Indice:

Introducción:

A pedido de la cátedra se deberá de realizar la resolución del siguiente problema:
Una línea de producción robotizado en el cual existe concurrencia. La misma está limitada principalmente por los brazos robot que participan en el traslado de las piezas.
La solución que debemos de plantear consiste en evitar el interbloqueo de los brazos robots y no alterar el paralelismo de la línea de producción, la cual será modelada por medio de una Red de Petri.
Aplicaremos las propiedades matemáticas de la misma (la cual viene derivada de la teoría de grafos). Esto nos permite hacer un programa perfecto a nivel funcional y basado en un modelo matemático “Fuerte”
Por último deberemos implementar los tiempos de los eventos que se realizan dentro de la red o planta e implementar absolutamente todo en una lenguaje de programación orientado a objetos. en nuestro caso implementaremos JAVA.

Presentación del problema:

Este informe se centrará en dar solución a un “Sistema de manufacturación robotizado" planteado en Naiqi and MengChu, 2010, el cual 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.
Para llegar a su objetivo, cada tipo de pieza debe seguir una trayectoria de procesamiento distinta, definida de la siguiente forma:
Para producir la pieza A:

- I1 → M1 → M2 → O1
- I1 → M3 → M4 → O1

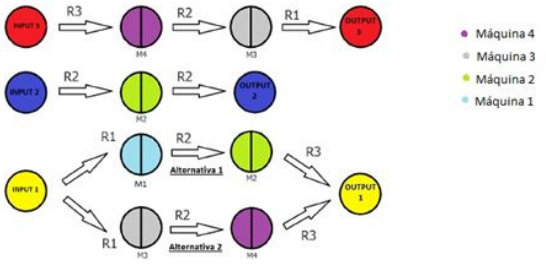
Para producir la pieza B:

- I2 → M2 → O2

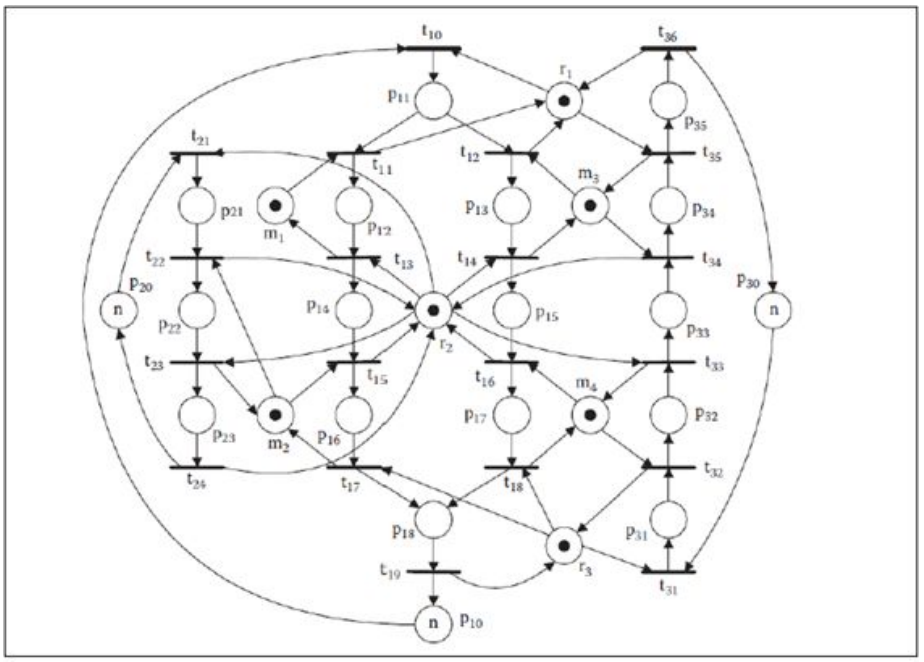
Para producir la pieza C:

- I3 → M4 → M3 → O3

Los robots tienen tareas definidas, cada operación de traslado de las piezas corresponde a un único robot. En la siguiente figura se pueden observar las distintas trayectorias para cada tipo de pieza:



Red de Petri modelada:



Se pide:

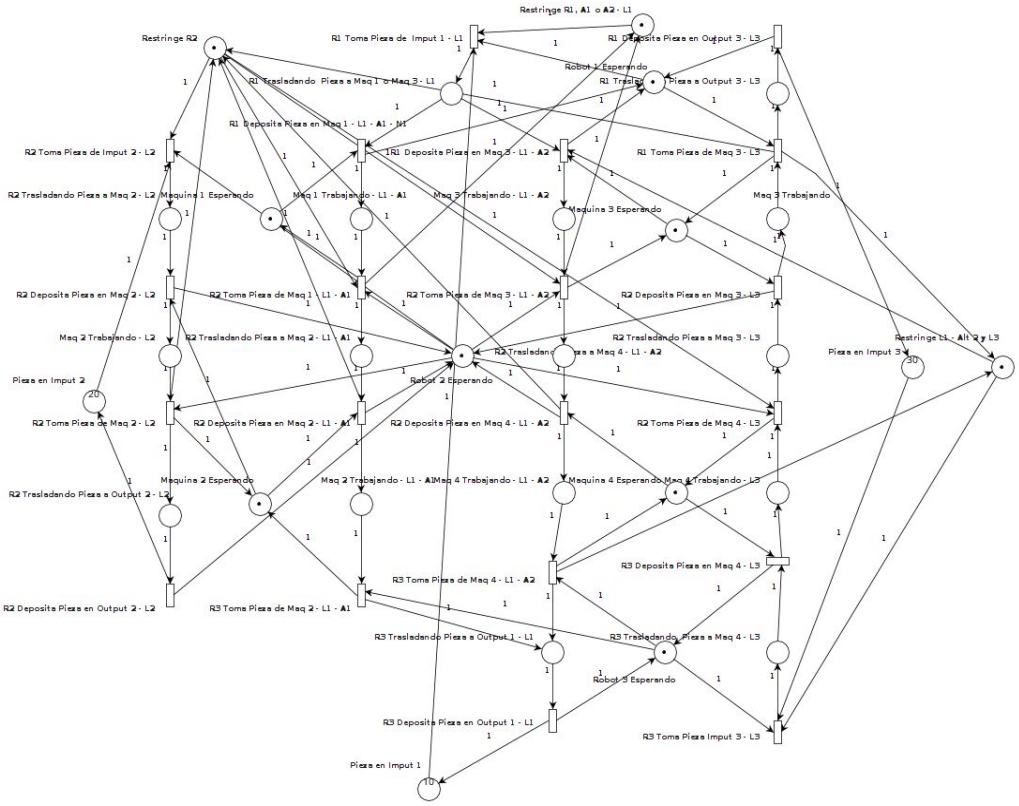
- Colocar las restricciones a la RdP para evitar el interbloqueo, mostrarlo con la herramienta.
- Colocar los tiempos de las máquinas en las transiciones correspondientes.
- Hacer la tabla de eventos y de estados o actividades.
- Determinar la cantidad de hilos necesarios (justificarlo).
- Implementar dos caso de Políticas para producir:
  - Por cada pieza A producir dos B y una C.
  - Por cada Pieza C producir tres A y dos B.
- Hacer el diagrama de clases y diagramas de secuencias.
- Hacer el código y testing.

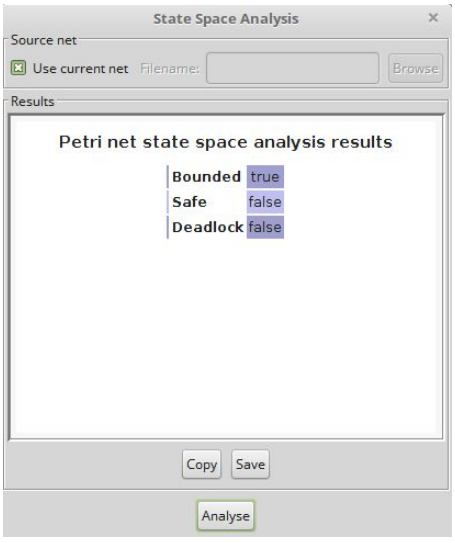
Resolución del problema:

Como bien sabemos, la red de Petri que modelamos anteriormente tiene lo que denominamos interbloqueo el cual se produce cuando una transición sufre de inanición, para ellos se procedió modelar la red desde cero basándonos en el modelo entregado por la cátedra anteriormente. El diseño del mismo fue efectuado en una primera instancia una herramienta llamada Pipe en su versión 4.3.0 ya que la misma nos permite verificar con exactitud si la red tiene interbloqueo o no.
Por último implementamos la red de Petri en TINA en su versión 3.4.4 ya que las transiciones temporales dentro del mismo programa se simulan mucho mejor que en Pipe

Implementación de la Red en PIPE:

Como mencionamos anteriormente en este software se puede analizar el interbloqueo. Lo que hicimos básicamente fue analizar la RDP y agregar unas plazas conectadas de una manera específica dentro de la red para evitar el interbloqueo.
A nivel conceptual enfocado a la consigna del problema, estas plazas, evitan que los brazos robot quieran depositar piezas dentro de máquinas que están ocupadas (que tienen una pieza dentro de ellas), esto genera que el brazo robot no tenga un lugar donde depositar la pieza generando así un interbloqueo.

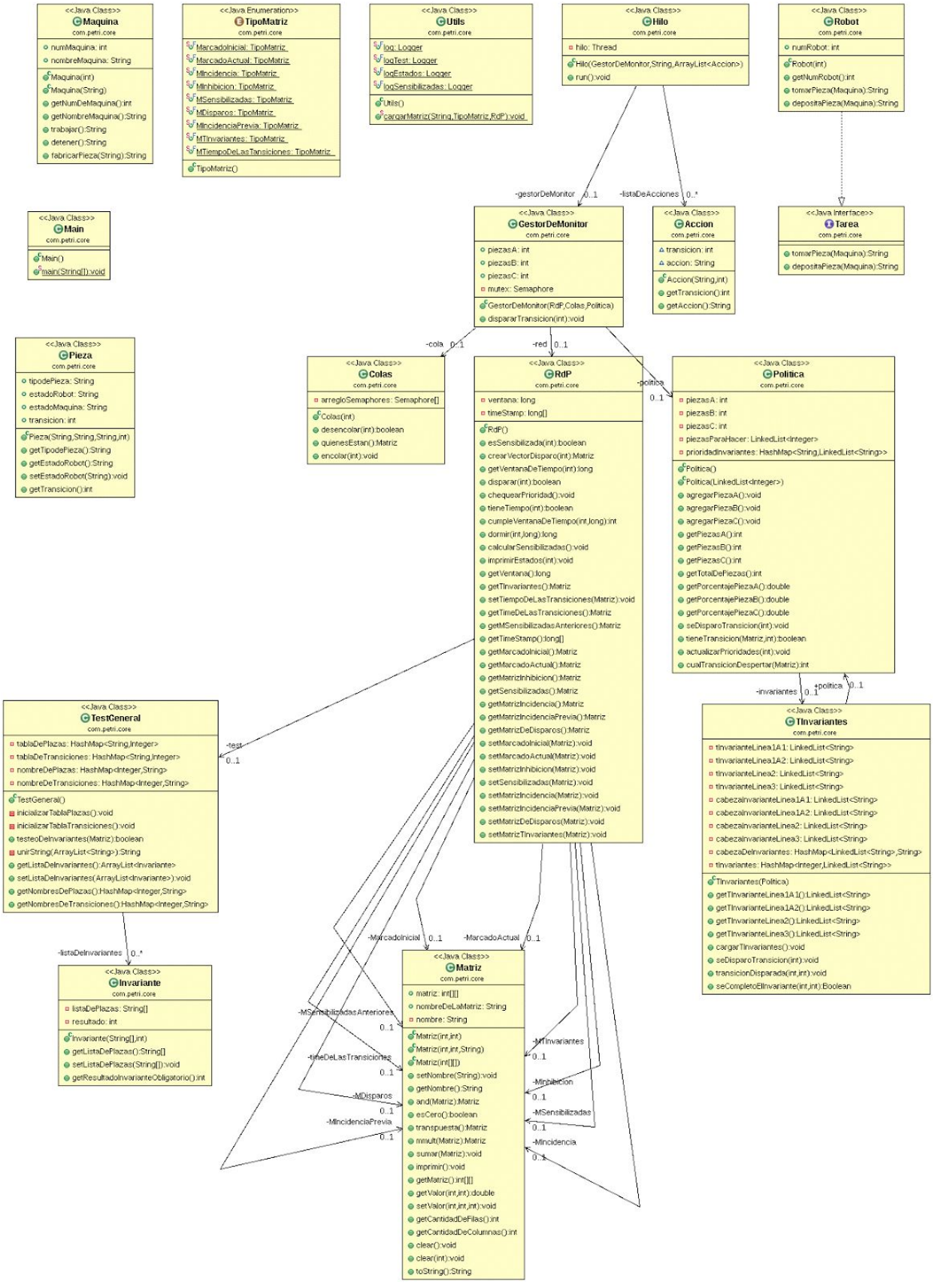




Bien, esta red es la que hemos analizado con PIPE y el resultado del análisis de invariantes es el siguiente

El dato que nos importa en este caso es el “Deadlock”, como da false entonces sabemos que no existe interbloqueo dentro de la RDP
Una vez sabiendo que la red no tenía interbloqueo procedimos a la estructuración del diagrama de clases para poder modularizar la mayor cantidad de código posible y poder efectuar una codificación adecuada.

Diagrama de clases:



T-Invariantes y análisis de los Threads(Hilos):

En esta parte, PIPE, nos arrojó los siguientes resultados

Petri net invariant analysis results

T-Invariants

R1 Deposita Pieza en Maq 1 - L1 - A1 - N1	R1 Deposita Pieza en Maq 3 - L1 - A2	R1 Deposita Pieza en Output 3 - L3	R1 Toma Pieza de Imput 1 - L1	R1 Toma Pieza de Maq 3 - L3	R2 Deposita Pieza en Maq 2 - L1 - A1	R2 Deposita Pieza en Maq 2 - L2	R2 Deposita Pieza en Maq 3 - L3	R2 Deposita Pieza en Maq 4 - L1 - A2	R2 Deposita Pieza en Output 2 - L2	R2 Toma Pieza de Imput 2 - L2	R2 Toma Pieza de Maq 1 - L1 - A1	R2 Toma Pieza de Maq 2 - L2	R2 Toma Pieza de Maq 3 - L1 - A2	R2 Toma Pieza de Maq 4 - L3	R3 Deposita Pieza en Maq 4 - L3	R3 Deposita Pieza en Output 1 - L1	R3 Toma Pieza de Maq 2 - L1 - A1	R3 Toma Pieza de Maq 4 - L1 - A2	R3 Toma Pieza Imput 3 - L3
0	0	0	0	0	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	1	1	0	0
0	1	0	1	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0
0	0	1	0	1	0	0	1	0	0	0	0	0	0	1	1	0	0	0	1

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

Tenemos 4 T-Invariantes.
Los T-Invariantes son aquellos que me dan una secuencia de transiciones, la propiedad de los T-Invariantes nos dice lo siguiente:
Si yo disparo una secuencia de transiciones debo de obtener el mismo marcado antes de disparar dicha secuencia y después de dispararla.
Para detectar lo hilos hay un patrón específico que, si bien no nos garantiza el máximo paralelismo, nos permite detectar un número de hilos principales.
La cantidad de hilos que yo puedo obtener es la cantidad de T-Invariantes y un hilo por cada transición que se repite en los T-Invariantes.
Siguiendo esta regla llegamos a la conclusión de que tenemos 5 Hilos
4 hilos son por los T-Invariantes
1 hilo es producto de que se repite la transición nº 3 (R1 Toma pieza de Input 1) en dos T-Invariantes

P-Invariantes:

Los P-Invariantes son otra propiedad de las Redes de Petri la cual nos asegura que no esta sucediendo ninguna anomalía dentro de la Red
Básicamente se maneja de la siguiente forma, existen “grupos de plazas” en los cuales uno encuentra una cierta cantidad de **tokens** dentro de la red, la sumatoria de ellos debe de ser constante **SIEMPRE** si no es así quiere decir que la red está inestable (a nivel software) por ende uno debería de inspeccionar el código a fondo pasa saber bien qué está sucediendo con el/los tokens faltantes y poder saber a donde “se está/n yendo”

NOTA: Recomiendo un log de cobertura completa para que uno pueda saber con una precisión y exactitud alta que está sucediendo.

Los P-Invariantes que nos arrojó el PIPE son los siguientes

P-Invariant equations

M(Maq 1 Trabajando - L1 - A1) + M(Maquina 1 Esperando) = 1

M(Maq 2 Trabajando - L1 - A1) + M(Maq 2 Trabajando - L2) + M(Maquina 2 Esperando) = 1

M(Maq 3 Trabajando) + M(Maq 3 Trabajando - L1 - A2) + M(Maquina 3 Esperando) = 1

M(Maq 4 Trabajando - L1 - A2) + M(Maq 4 Trabajando - L3) + M(Maquina 4 Esperando) = 1

M(Maq 1 Trabajando - L1 - A1) + M(Maq 2 Trabajando - L1 - A1) + M(Maq 3 Trabajando - L1 - A2) + M(Maq 4 Trabajando - L1 - A2) + M(Pieza en Imput 1) + M(R1 Trasladando Pieza a Maq 1 o Maq 3 - L1) + M(R2 Trasladando Pieza a Maq 2 - L1 - A1) + M(R2 Trasladando Pieza a Maq 4 - L1 - A2) + M(R3 Trasladando Pieza a Output 1 - L1) = 10

M(Maq 2 Trabajando - L2) + M(Pieza en Imput 2) + M(R2 Trasladando Pieza a Maq 2 - L2) + M(R2 Trasladando Pieza a Output 2 - L2) = 20

M(Maq 3 Trabajando) + M(Maq 4 Trabajando - L3) + M(Pieza en Imput 3) + M(R1 Trasladando Pieza a Output 3 - L3) + M(R2 Trasladando Pieza a Maq 3 - L3) + M(R3 Trasladando Pieza a Maq 4 - L3) = 30

M(Maq 3 Trabajando) + M(Maq 3 Trabajando - L1 - A2) + M(Maq 4 Trabajando - L1 - A2) + M(Maq 4 Trabajando - L3) + M(R2 Trasladando Pieza a Maq 3 - L3) + M(R2 Trasladando Pieza a Maq 4 - L1 - A2) + M(R3 Trasladando Pieza a Maq 4 - L3) + M(Restringe L1 - Alt 2 y L3) = 1

M(Maq 1 Trabajando - L1 - A1) + M(Maq 3 Trabajando - L1 - A2) + M(R1 Trasladando Pieza a Maq 1 o Maq 3 - L1) + M(Restringe R1, A1 o A2 - L1) = 1

M(Maq 2 Trabajando - L2) + M(Maq 3 Trabajando) + M(R2 Trasladando Pieza a Maq 2 - L1 - A1) + M(R2 Trasladando Pieza a Maq 2 - L2) + M(R2 Trasladando Pieza a Maq 3 - L3) + M(R2 Trasladando Pieza a Maq 4 - L1 - A2) + M(Restringe R2) = 1

M(R1 Trasladando Pieza a Maq 1 o Maq 3 - L1) + M(R1 Trasladando Pieza a Output 3 - L3) + M(Robot 1 Esperando) = 1

M(R2 Trasladando Pieza a Maq 2 - L1 - A1) + M(R2 Trasladando Pieza a Maq 2 - L2) + M(R2 Trasladando Pieza a Maq 3 - L3) + M(R2 Trasladando Pieza a Maq 4 - L1 - A2) + M(R2 Trasladando Pieza a Output 2 - L2) + M(Robot 2 Esperando) = 1

M(R3 Trasladando Pieza a Maq 4 - L3) + M(R3 Trasladando Pieza a Output 1 - L1) + M(Robot 3 Esperando) = 1

Son un total de 13 P-Invariantes y se deben de respetar **SI O SI** a lo largo de toda la ejecución del programa.

Implementación del código:

Una vez efectuado esta serie de pasos procederemos a implementarlo todo esto a nivel “Código”
Anteriormente habíamos mencionado varias características del programa llamado PIPE pero no mencionamos en ningún momento una de las características más importantes que nos va a servir para implementar nuestro software de manera efectiva y es la capacidad de dar todos los análisis que efectúa este mismo programa en un formato de archivo XML dándonos la capacidad de poder parsear el archivo mediante las etiquetas

o tags características de este documento. Esto nos va a poder permitir automatizar la carga de archivos.

Una vez cargados estos datos, emplearemos el “Gestor de Monitor", que es el encargado de administrar los recursos (máquinas y robots) de acuerdo a la lógica y la política planteados en dicho problema. El funcionamiento del mismo está basado en el uso de semáforos y colas. Cabe destacar que lo más importante es que administra los recursos críticos en función de las políticas cuando los demás hilos entran en conflicto. El gestor de monitor se encarga de “averiguar” mediante el uso de las ecuaciones de petri si se puede disparar una transición o no.

Las líneas de producción serán representadas a través de hilos, que ejecutarán indefinidamente una serie acciones para producir una pieza determinada, utilizando los recursos necesarios. De aquí, la necesidad de un administrador de recursos que maneje la concurrencia de los múltiples hilos.

La Red de Petri juega el papel fundamental de decidir qué transición se puede disparar y cuál no de acuerdo su estado, o lo que es lo mismo, decirnos qué acción se puede realizar y cuál no en determinado momento. Para esto se creó una clase RdP en donde se cargan las diferentes matrices obtenidas del software PIPE, como ser el Marcado Inicial, Incidencia Previa, Tiempos, etc.
A través de los métodos de esta clase podremos determinar si una transición cumple con el tiempo para poder dispararse o no y actualizar el estado de la Red(cambiando la matriz de marcado), actualizar el vector de sensibilizadas o actualizar los tiempos de las transiciones sensibilizadas, todo esto realizando los cálculos matemáticos correspondientes.
Otra ventaja que obtenemos de la Red de Petri es que cada vez que se produce una acción, o lo que es lo mismo se dispara una transición, registramos el estado en el que se encuentra la misma (a través de su marcado actual) y las acciones que se pueden realizar o no (a través de las sensibilizadas) ahora que la red ha cambiado. Todo esto queda guardado en dos Log, uno llamado Estados y otro Sensibilizadas.

Lo que nos permite realizar diferentes simulaciones y tener un control de todas las acciones que se realizaron y todos los estados por los cuales paso el sistema, permitiéndonos analizar, por ej, el funcionamiento de las líneas de producción luego de determinada cantidad de piezas hechas. Luego de varias simulaciones podemos ver como la línea de producción que hemos denominado Línea 1 Alternativa 2, la cual construye una pieza de tipo A, se ejecuta muy pocas veces en comparación con las demás líneas. Con un análisis poco profundo, una de las primeras soluciones que se presentan es la de quitar esta linea, ya que existe otra que se encarga de realizar la pieza A, pero tal vez esta solución no es la más conveniente para la empresa.

Gracias al software construido, se pueden generar nuevas simulaciones, quitando la línea, reemplazando máquinas, reemplazando robots, etc, y ver los resultados para determinar cual es la solucion que mas le conviene a la empresa.
Además, el Log de nombre Simulación, permite hacer un seguimiento “a nivel físico” del sistema, ya que en el mismo podemos ver cómo fue evolucionando cada una de las líneas de producción, que acción es la que se está llevando a cabo y en qué estado se encuentran cada uno de los recursos del sistema, es decir, que máquina está ocupada y cuál no, y lo mismo con los robots.

Durante el desarrollo del código se cometieron varios errores por ejemplo al comienzo, lograr implementar toda la clase RdP y sus métodos tuvo una cierta dificultad. Luego una vez implementada la clase de Gestor de Monitor, Colas, Robot y varias más, implementamos la clase Políticas, la cual nos dimos cuenta que había sido mal implementada a partir de las consultas con el Profesor. Lo que nos llevó a implementarla nuevamente con otro enfoque, utilizando muchas menos líneas de código y desacoplando el mismo para hacerlo más sencillo de leer en el caso de querer realizar cambios y más reutilizable. Lo cual hizo que crearamos una nueva clase llamada TInvariantes encargada de llevar un control de los T-Invariantes, además de cargarlos y testearlos al comenzar el programa.
Otra complicación la tuvimos al momento de implementar los tiempos en la clase RdP, lo que también nos demandó varias horas y varios cambios en el código.

Para comprender mejor toda la implementación, se realizaron los diagramas pertinentes que se encuentran al final de este documento.

Implementación de pruebas

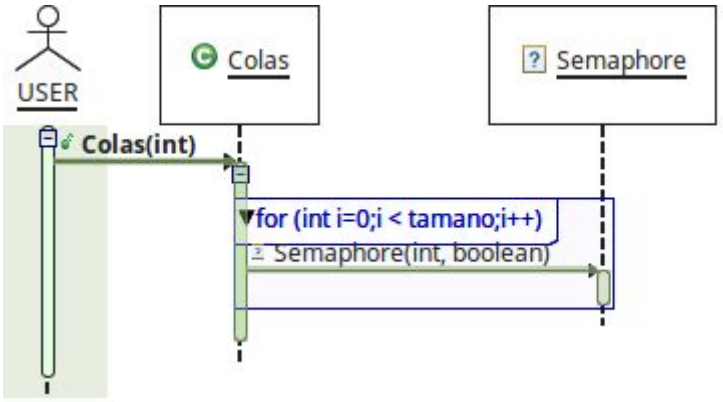
A partir de las explicaciones dadas anteriormente sobre T-Invariantes y P-Invariantes sabemos que debemos realizar dos test diferentes.
El primero para controlar los T-Invariantes, verifica que la secuencia de transiciones que componen al T-Invariante se disparen correctamente. Se implementó en la clase TInvariante, en el método cargarTInvariante, aprovechando este método para realizar dos pasos al mismo tiempo, testear los TInvariantes y cargarlos en diferentes listas dentro de la clase. El método se ejecuta en el constructor de la clase, por lo que cada vez que creamos una instancia de la misma nos aseguramos que los 4 TInvariantes fueron cargados exitosamente y pasaron el test.
Por otro lado, tenemos el test de los P-Invariantes. El mismo se realiza en tiempo de ejecución, ya que cada vez que se llama al metodo disparar de la clase RdP si la transición que es pasada por parámetro se puede disparar, el marcado de la red cambia. Como el control de los P-Invariantes implica que siempre debe haber la misma cantidad de token dentro de un grupo de plazas, cada vez que cambie el marcado de la Red, debemos realizar el test, por eso se decidió hacerlo de esa manera.

Los resultados de los test se verán reflejados en el Log llamado Test. Este Log contendrá toda la información en caso de que cualquier de los dos test nombrados anteriormente no sean satisfactorios. En caso de que ninguno falle, el log no contendrá nada, indicando que los test son exitosos.

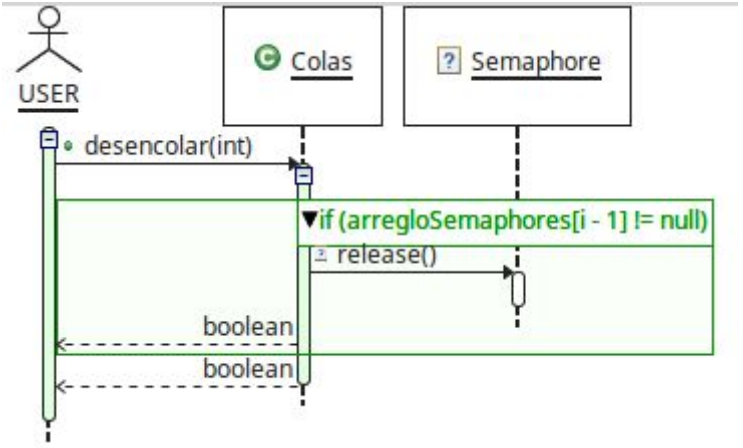
Diagramas de secuencia:

Clase Cola

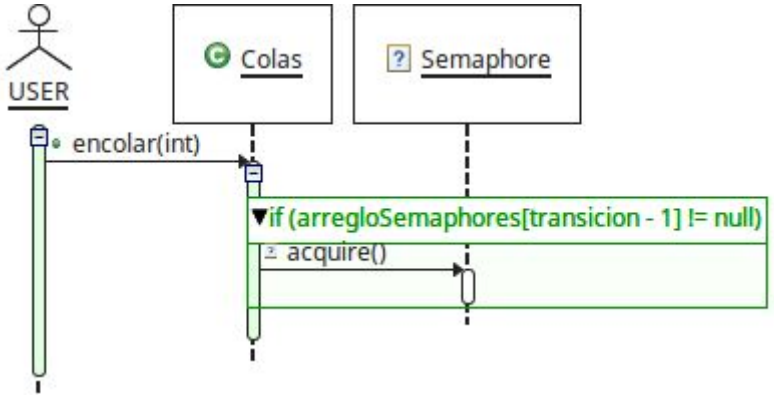
Constructor:



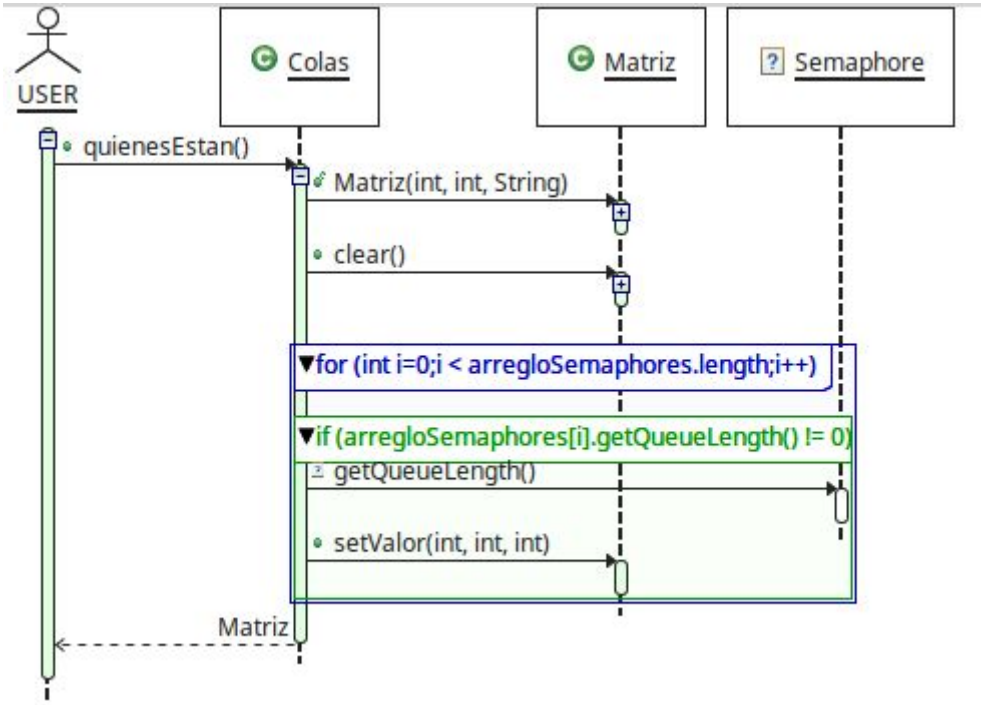
Método desencolar:



Método encolar:

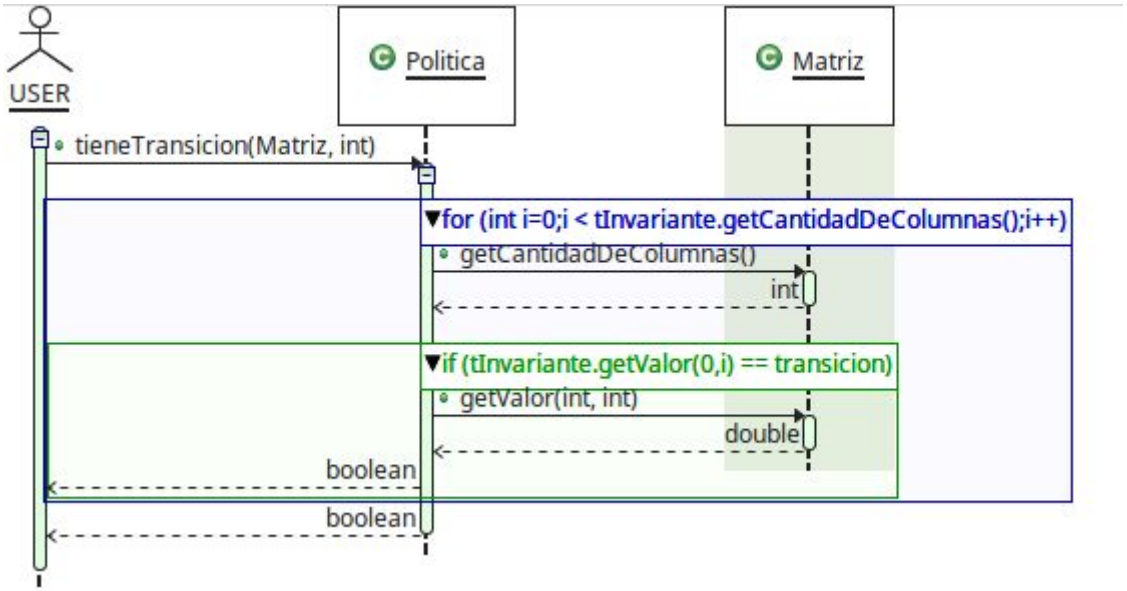


Método quienesEstán:

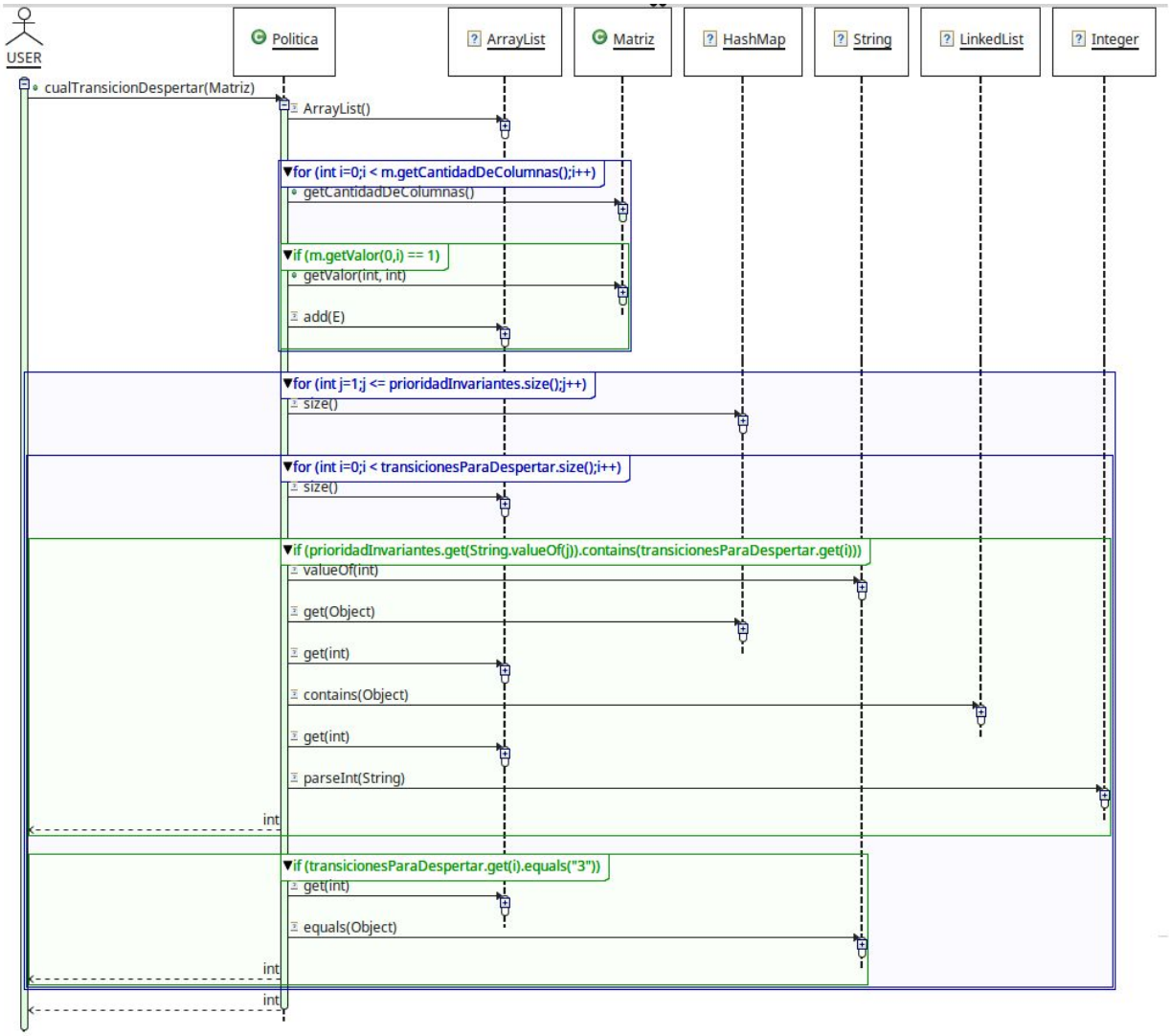


Clase Política:

Método tieneTransicion:

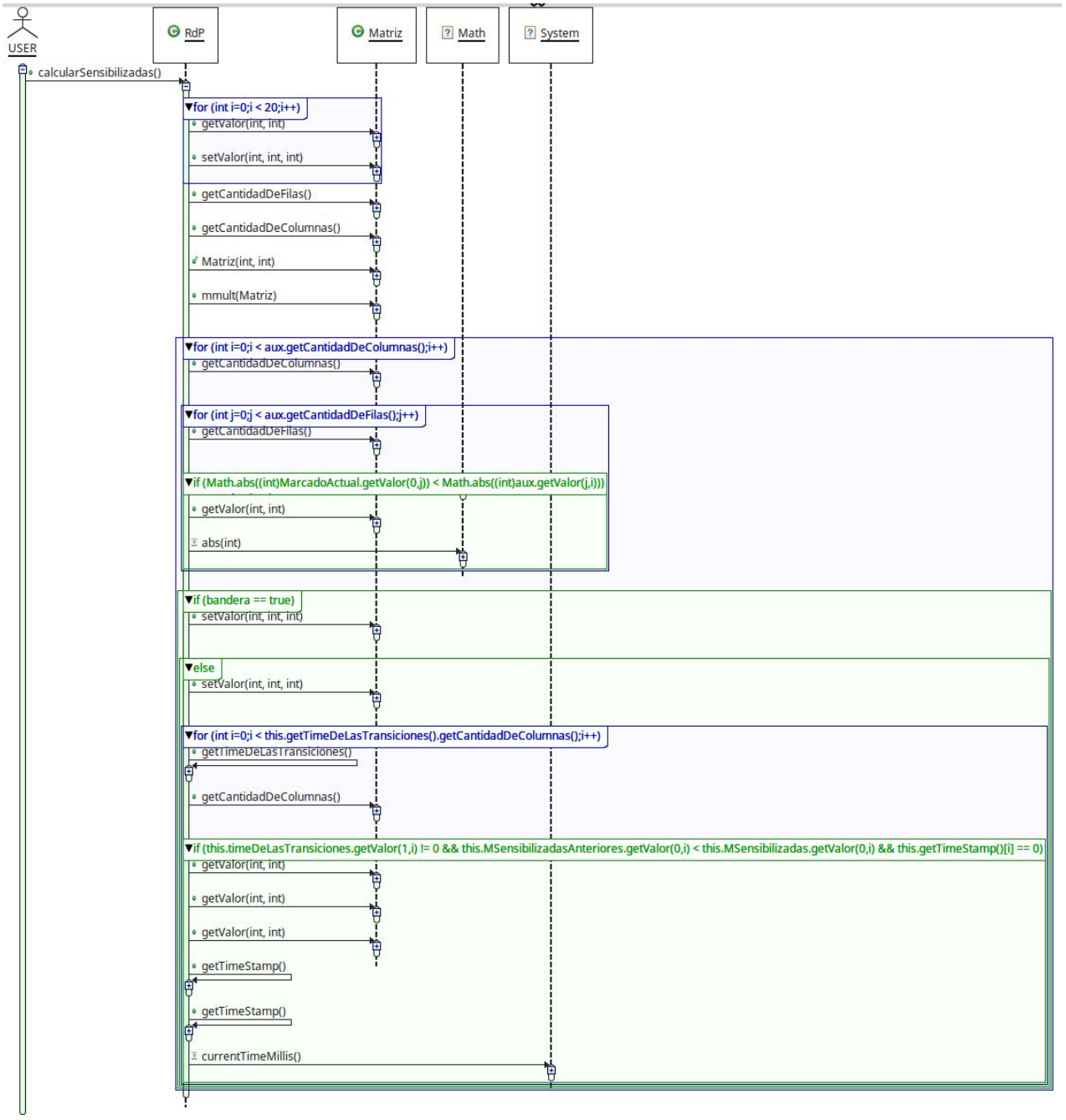


Método cualTransicionDespertar:

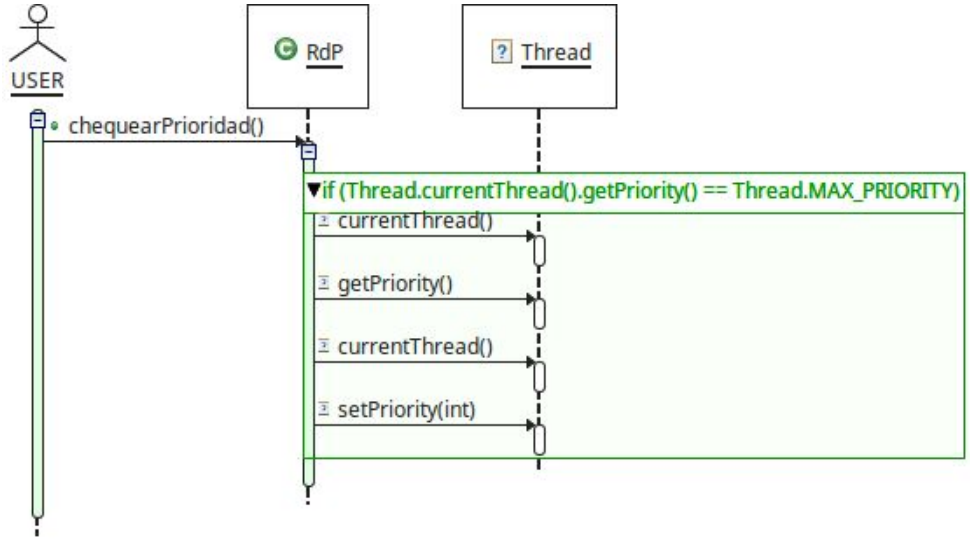


Clase Rdp:

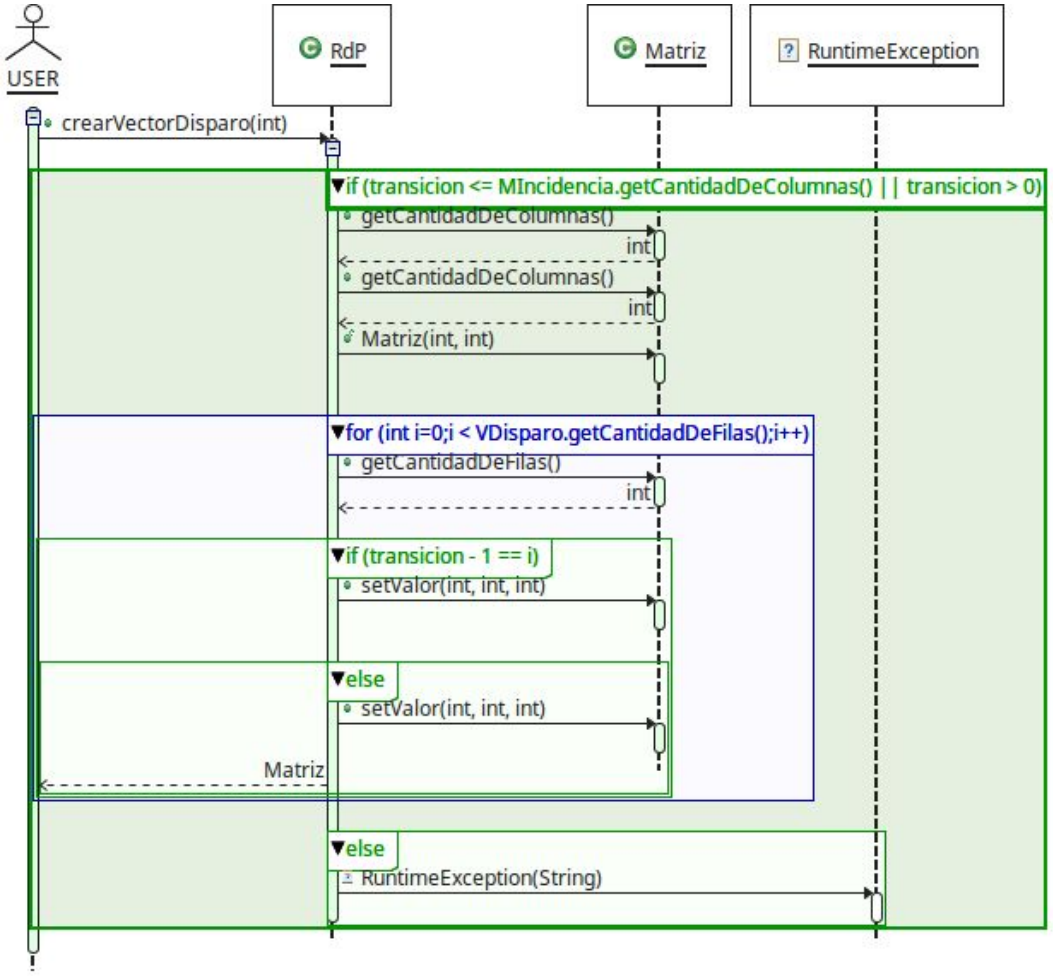
Método calcularSensibilizadas:



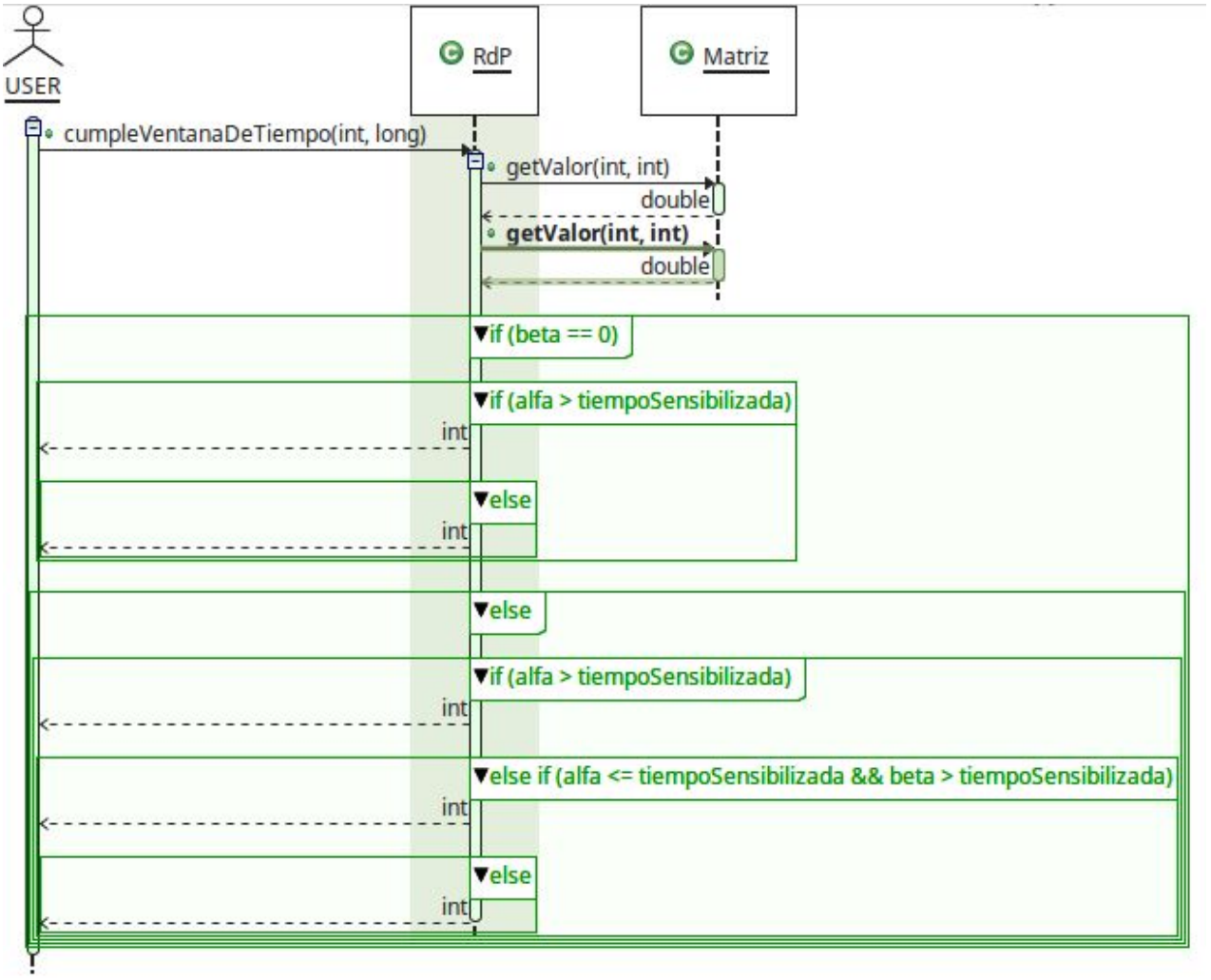
chequearPrioridad:



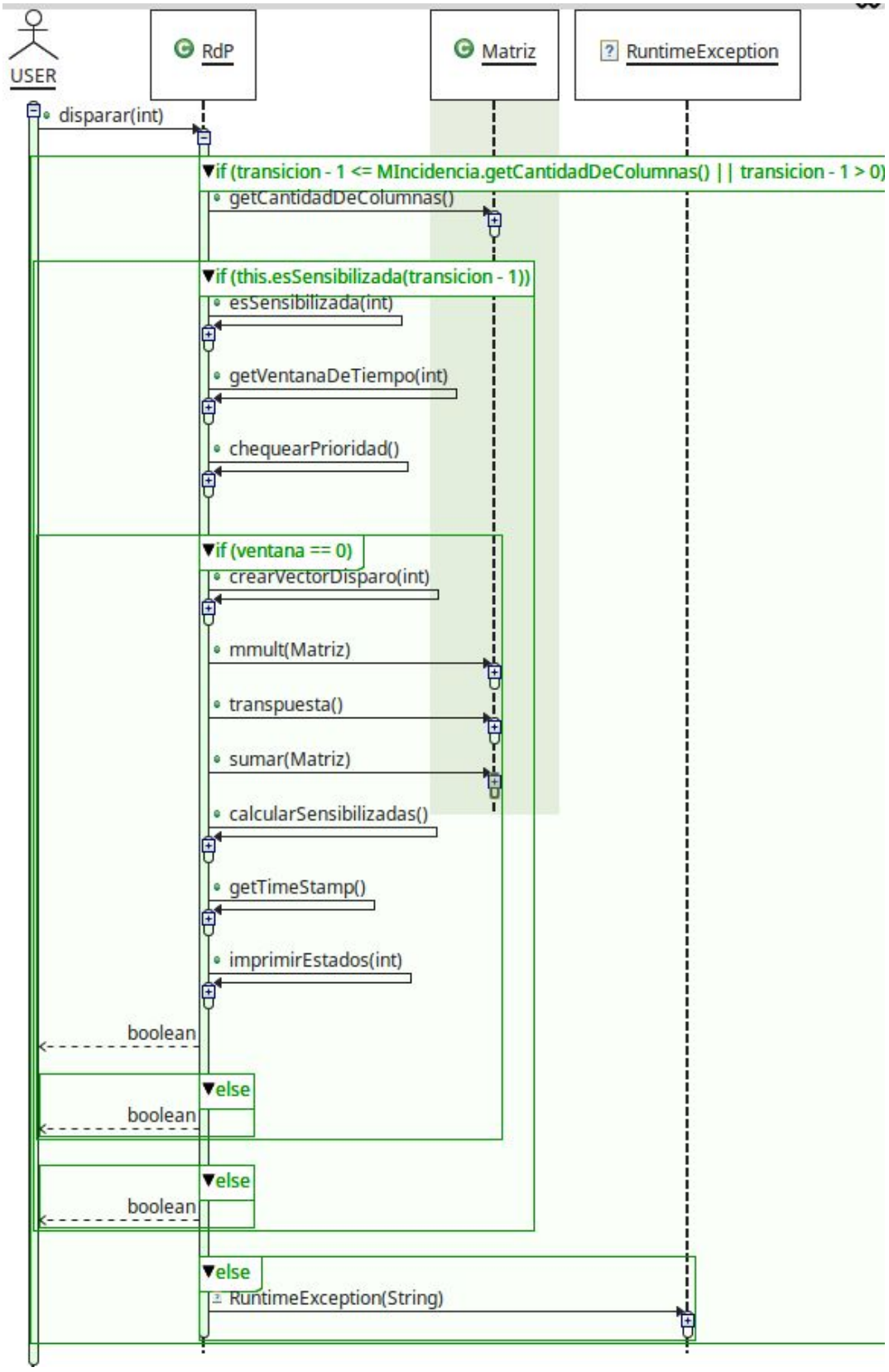
Método crearVectorDisparo:



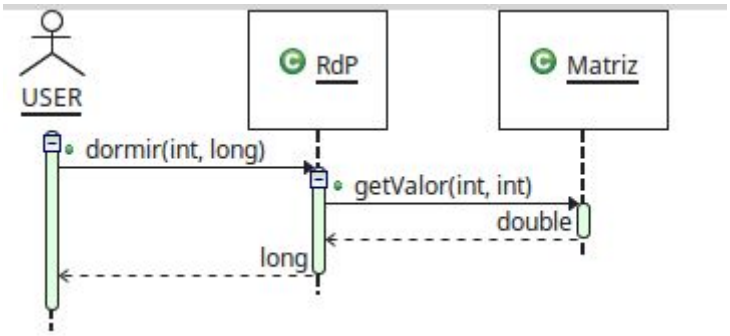
Método cumpleVentanaDeTiempo:



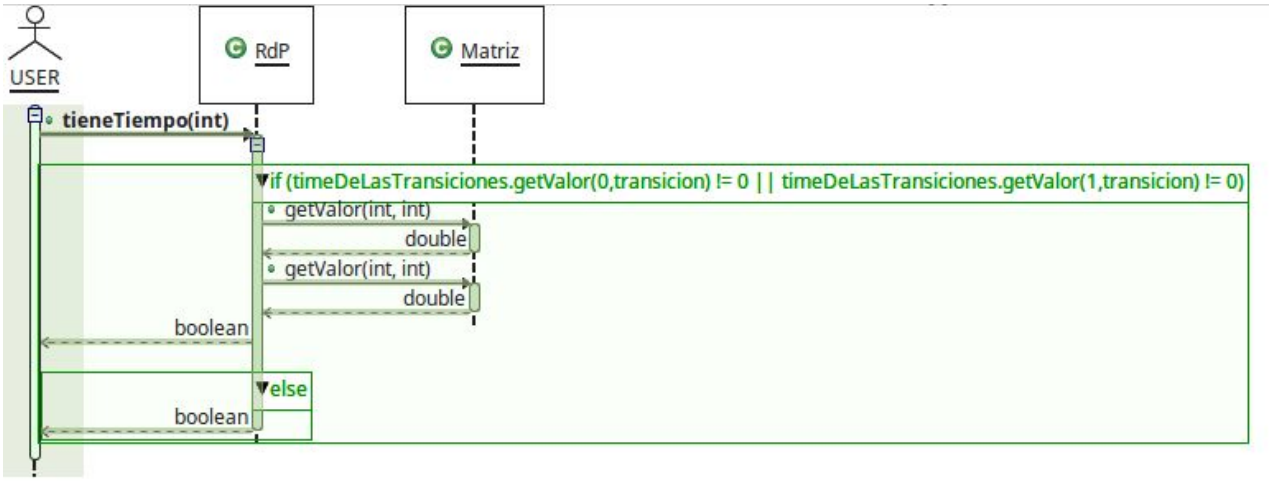
Método disparar:



Método dormir:

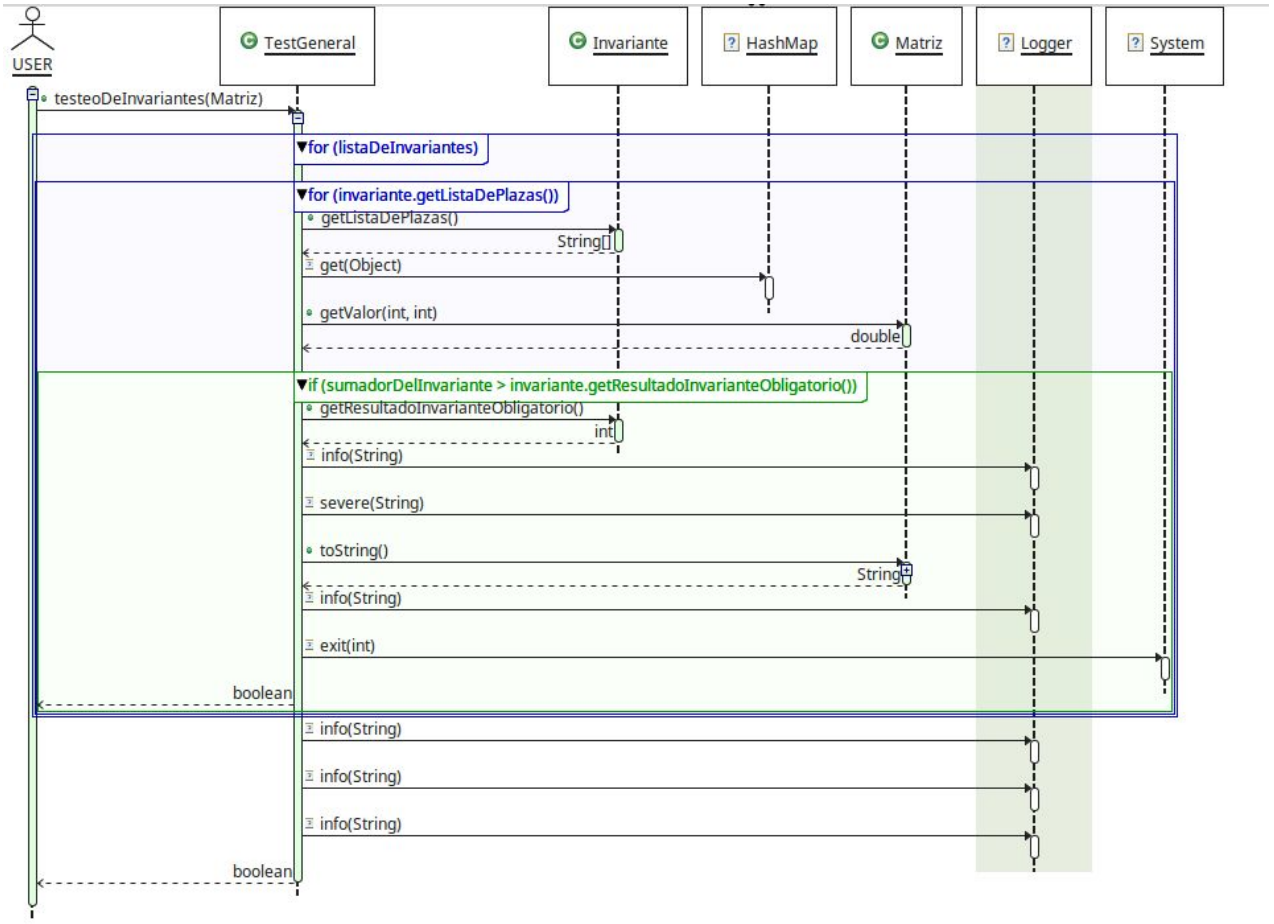


Método tieneTiempo:



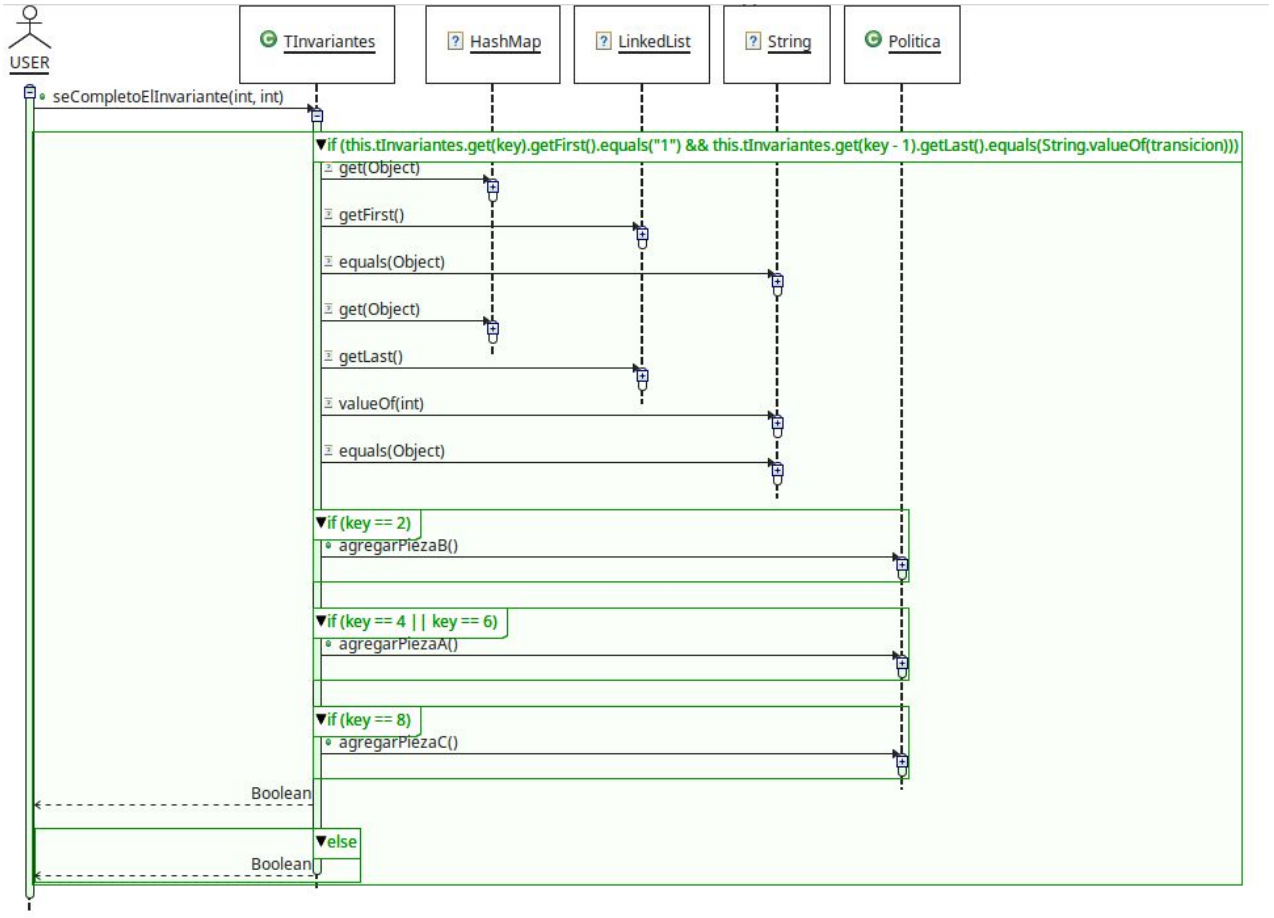
Clase TestGeneral:

Método testeoDeInvariantes:



Clase TInvariantes:

Método seCompletoElInvariante:



Conclusiones:

- Las Redes de Petri nos dan un sustento matemático para demostrar que no va a existir un interbloqueo en el software por error de modelado de un sistema y que siempre las líneas de producción van a funcionar sin interrumpirse y sin anular el paralelismo (a menos que sea necesario).
- Nos permiten paralelizar sin tener que hacer un exhaustivo análisis del problema incluyendo la dependencia de datos ya que uno puede modelar de manera gráfica un problema.
- Uno puede efectuar una simulación perfecta basándonos en un criterio/política particular.
- En esta red particularmente se puede observar que hay un circuito de producción que casi nunca se utiliza, se debería de implementar una mejora extra para que se utilice con mayor frecuencia por ejemplo: implementar otro brazo robot o mejorar los tiempos de mecanizado de las máquinas.
- Las redes de Petri es una herramienta de modelado muy potente pero uno debe de entender qué está haciendo ya que una plaza o transición puesta sin saber su significado implicaría desconocer algún estado, acción o evento de la RDP, el código podría seguir andando y no tener interbloqueo sin embargo estaría efectuando una acción desconocida.
- Si bien hicimos un programa para un problema específico, se puede ver claramente que si cargamos cualquier tipo de RDP nuestro software va a andar correctamente sin necesidad de reprogramar todo el código, sólo habría que modificar las políticas.
- Es mucho mejor siempre efectuar un diagrama de secuencia para entender lo que estamos haciendo y en especial para entender que ha programado un compañero de hecho es mucho más rápido leer el diagrama que ir debuggeando o inspeccionando el código ya que también fortalece la parte conceptual de lo que estamos haciendo.
- El desarrollo del programa junto con las consultas realizadas durante el proceso del mismo nos dio la posibilidad de afianzar conocimientos, como es la Programación Orientada a Objetos (POO), el diseño de diagramas UML, el uso de patrones de diseño como ser el patrón Strategy o el uso de la clase Collection de Java y otras más.