PROGRAMACIÓN CONCURRENTE 2015

TRABAJO FINAL

ACCIONAMIENTO DE UNA BOMBA DE AGUA EN UNA MINA

PROFESOR:

- Micolini, Orlando

INTEGRANTES:

- Prados, Santiago Agustín
- Perpetua, Patricio Rafael
- Viñas Viscardi, Dardo Ariel

INDICE

1 OBJETIVO

Nuestro objetivo es parametrizar un problema a través de las redes de Petri, para luego ser ejecutados en un procesador realizado en lenguaje Java.

El sistema a resolver, es el de una bomba de agua que extrae agua de un sumidero de una mina, que no puede ejecutar bajo la presencia de ciertos gases que pueden generar un siniestro.

2 INTRODUCCIÓN

Este trabajo, a diferencia de otros realizados anteriormente, nos presentó una enorme dificultad. Por momentos nos encontrábamos con la necesidad de hacer diagramas o tests que pensábamos de poca utilidad. Por otros, nos hizo darnos cuenta que programar de forma adecuada requiere de un procedimiento bien definido, que no era solo sentarse a codificar.

Podemos garantizar el correcto funcionamiento de una política de ejecución en redes de Petri. Además, hemos logrado desarrollar un procesador que ejecuta redes temporales, con transiciones automáticas, con arcos inhibidores, y que adquiere los datos para el funcionamiento de archivos exportados de un graficador visual de redes de petri.

Se extrega el código en Java del procesador terminado, junto a las redes que resuelven el problema planteado

3 DESARROLLO

Como se introdujo en un principio, lo primero en realizarse fue el programa en Java que fuera capaz de ejecutar una RdP, partiendo del archivo .html que devuelve el programa de código abierto "PIPE" (Platform Independent Petri net Editor) en su versión 4.3.0. El mismo, era capaz de ejecutar redes que cumplieran con la ecuación:

$$m_{i+1} = m_i + I \times \delta$$

Donde:

I : incidencia

m: marcado

δ: disparo

Por supuesto, esto no es suficiente para el problema, debido a la necesidad de utilizar arcos inhibidores, transiciones temporales y además, un sistema de políticas para establecer una prioridad de disparo entre transiciones.

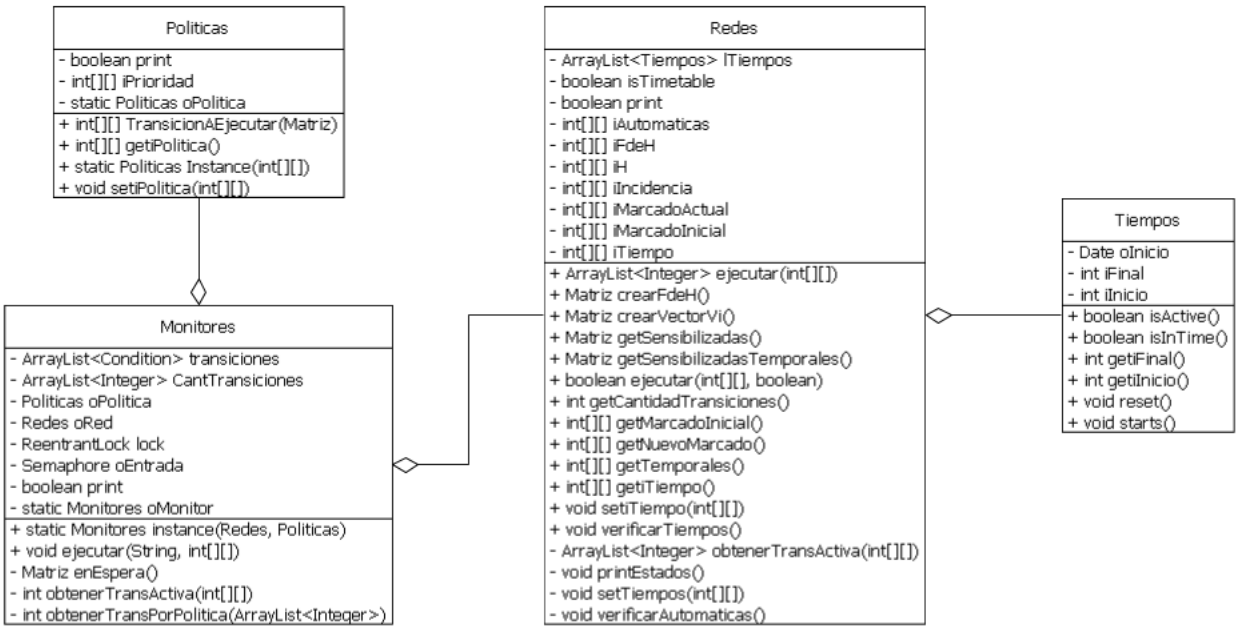
Se tomó, entonces, la decisión de tomar cada una de estas necesidades por separado, tomando cada una de ellas como un requisito del programa en Java, y realizando tests de cada una para seguir adelante con la elaboración de la RdP.

3.1 Programa en Java

3.1.1 Requerimientos

- RQ1: El programa no debe poder disparar una transición no sensibilizada.
- RQ2: El programa debe poder disparar las transiciones sensibilizadas.
- RQ3: Se necesita un método para conocer las transiciones sensibilizadas.
- RQ4: Se debe poder establecer prioridades de disparo entre transiciones.
- RQ5: Se debe poder disparar una transición sensibilizada por tiempo.
- RQ6: No se debe poder disparar una transición no sensibilizada por tiempo.
- RQ7: Se deben poder disparar transiciones de manera automática, en el instante que se sensibilizaron (transiciones automáticas que no requieren actor).

3.1.2 Diagrama de Clases



3.1.3 Procesador de Petri

El procesador desarrollado contiene dos constructores. Uno es para trabajar con redes temporales, y el otro con redes sin tiempo. En ambos casos se puede contar con transiciones automáticas. Los archivos con los que trabaja son los documentos extraidos del programa PIPE, que pueden ser exportados en formato .html. Las matrices de tiempo, automáticas y automáticas son cargadas desde archivos excel .xls

3.1.3.1 Redes con arcos inhibidores

Este tipo de redes, puede o no, contener transiciones automáticas, las cuales son solicitadas por el constructor al instanciar la red, pero en el caso que no las contenga, el programa simplemente las iniciará como *null* y procederá con la ejecución sin inconvenientes.

Para realizar un disparo, se tiene en cuenta la siguiente ecuación:

$$m_{i+1} = m_i + I \times [\delta \; and \; f_H(m_i)]$$

Donde:

- δ : disparo
- mi+1 : nuevo marcado
- I : incidencia
- $fH(mi)$: función de arcos inhibidores
- Vi : un vector que se forma en base al marcado, donde es 0 si en el lugar de la plaza hay tokens, o es 1 si la plaza está vacía

3.1.3.2 Redes con arcos inhibidores temporales

Al igual que las no temporales, este tipo de redes pueden contener transiciones automáticas, las cuales son solicitadas por el constructor al instanciar la red, pero en el caso que no las contenga, el programa simplemente las iniciará como *null* y procederá con la ejecución sin inconvenientes.

La diferencia está en el constructor, que solicita además un vector temporal, que tiene la misma dimensión que el vector de disparo.

Para realizar un disparo, se tiene en cuenta la siguiente ecuación:

$$m_{i+1} = m_i + I \times [\delta \; and \; f_H(m_i) \; and \; V_t \; and \; V_s]$$

Donde:

- δ : disparo
- mi+1 : nuevo marcado
- I : incidencia
- $fH(mi)$: función de arcos inhibidores
- Vi : un vector que se forma en base al marcado, donde es 0 si en el lugar de la plaza hay tokens, o es 1 si la plaza está vacía
- V_t : Vector temporal
- V_s : Vector de transiciones sensibilizadas

3.1.4 Monitor

El monitor tiene un semáforo de entrada y un conjunto de semáforos encerrados en un ArrayList (uno por cada transición de la RdP). Cuenta también con un método para saber si en las colas hay alguien en la cola.

En el constructor, recibe la RdP a ejecutar, y las políticas. A partir de la RdP, construye los semáforos de las colas, inicializadas en 0 (cero). El monitor utiliza el patrón singleton (solo se puede tener 1 monitor por programa), lo que nos facilita su llamada desde cualquier otra clase, ya que no se necesita un puntero hacia él.

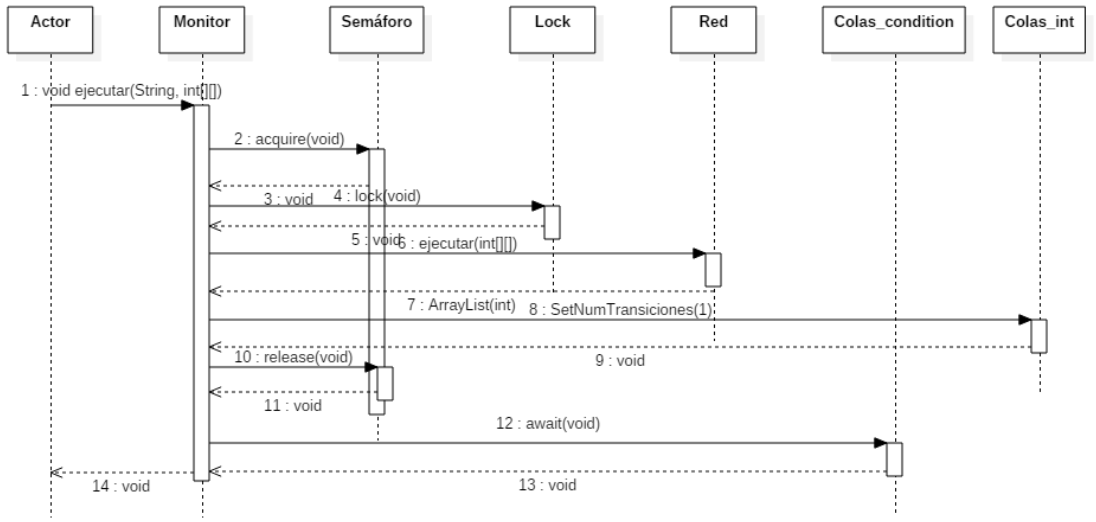
El único evento público es *ejecutar*, todo el resto son privados. Pueden suceder diferentes casos:

1. No se ejecutó. Se va a la cola
2. Se ejecutó. Verifica que no se haya liberado anteriormente alguien de su cola:
    a. No hay nadie en la cola de condición:
        i. Liberaron a alguien antes. Libero el lock, y me voy.
        ii. No liberaron a alguien antes, libero el lock y el semáforo.
    b. Hay alguien en la cola de condición:
        i. Si hay 1, libera el lock, pero no el semáforo.
        ii. Si hay más de 1, manda a todos los que esperan a *Políticas*, y la misma devuelve 1 a ejecutar, y liberal el lock.

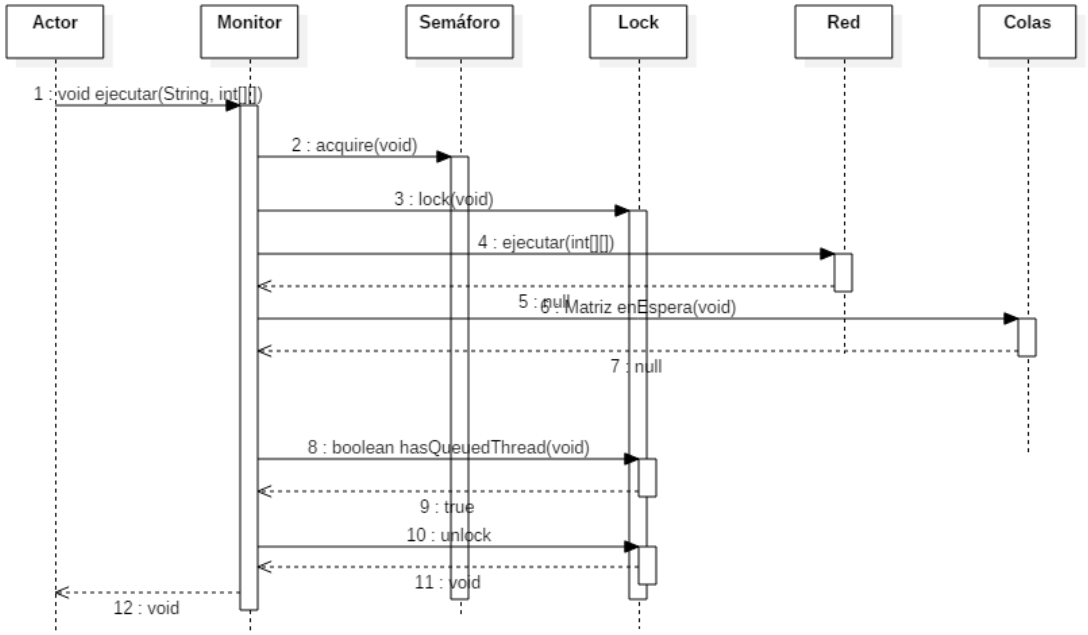
El semáforo de entrada solamente se libera cuando no hay nadie en el lock.

3.1.4.1 Diagramas de secuencia

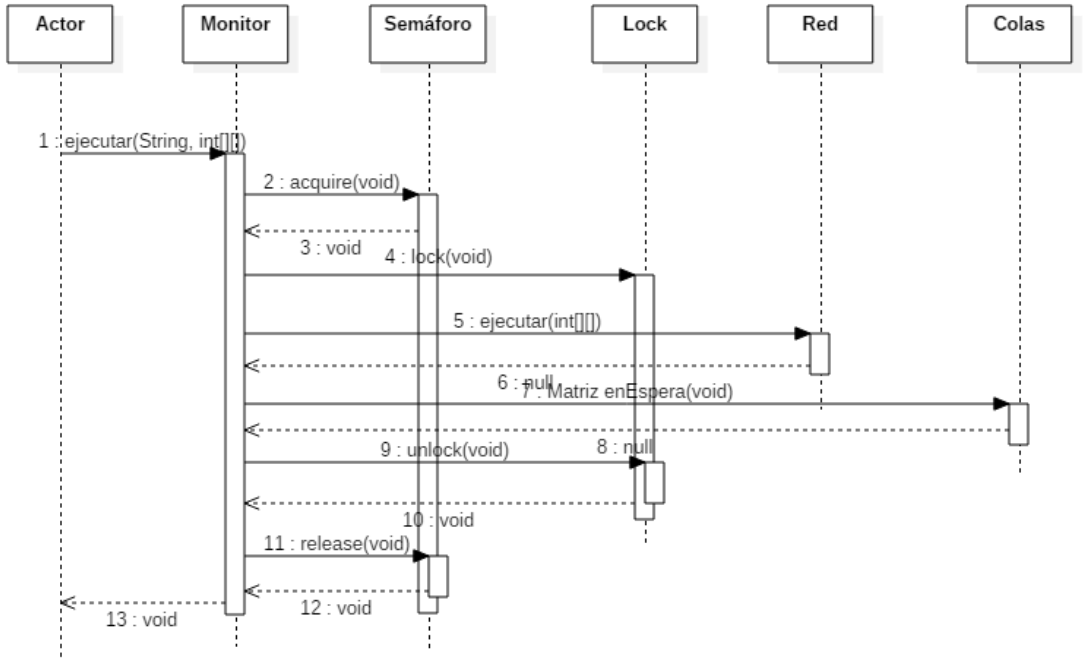
Caso 1:



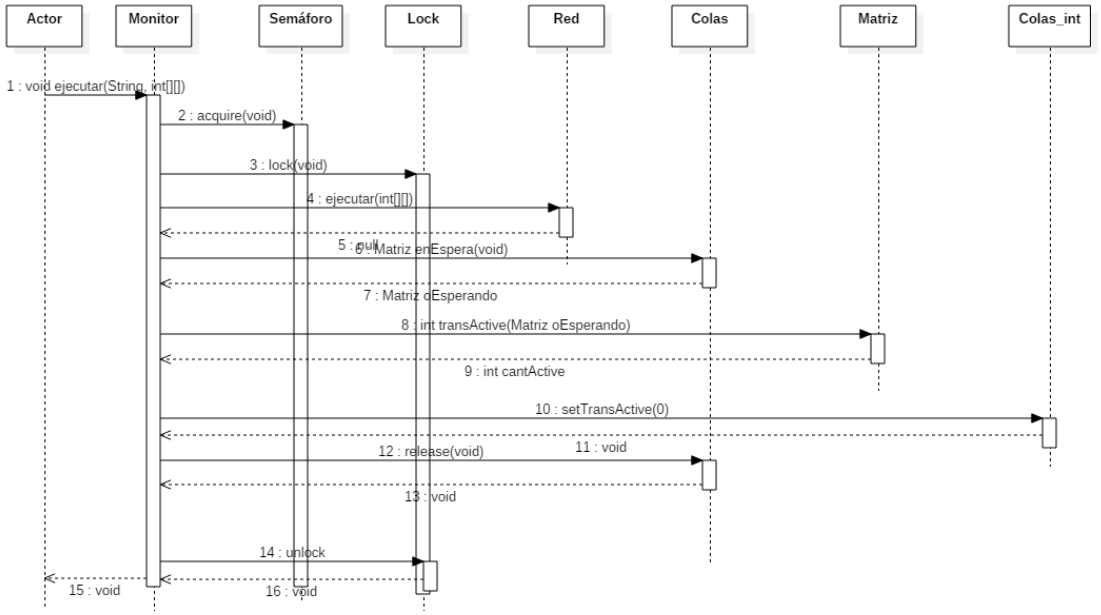
Caso 2 - a - i:



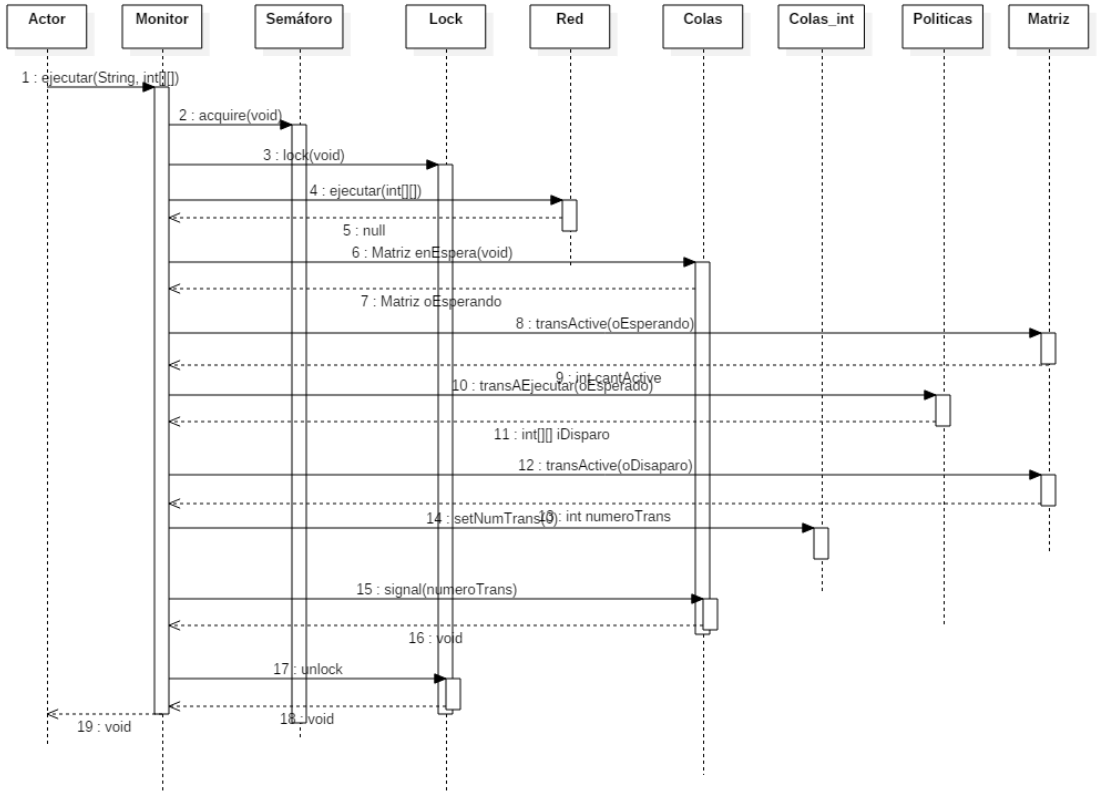
Caso 2 - a - ii:



Caso 2 - b - i:



Caso 2 - b - ii:



3.1.5 Políticas

Esta clase, al igual que el monitor, también es singleton. La misma cuenta con una matriz de prioridad, en la cual la fila representa la prioridad de ejecución (a menor fila, mayor prioridad), y las columnas representan el número de transición.

Para tomar la decisión, entonces multiplica a la matriz de prioridad con las transiciones sensibilizadas, con lo que devolverá solo 1 transición válida, que será la seleccionada para ejecutar.

3.2 Red de Petri

Para la elaboración de la RdP, en un comienzo se tomó al sistema como un todo, incluyendo a todos los sensores, junto al nivel de agua y accionamiento de la bomba. Por diferentes dificultades para llevarlo a cabo, incluyendo problemas con la simulación en PIPE de redes tan grandes, se consideró la opción de dividir al sistema en problemas más chicos. Al dividir el

problema, se llega a redes pequeñas que se conectan entre sí, por transiciones y plazas que comparten en común.

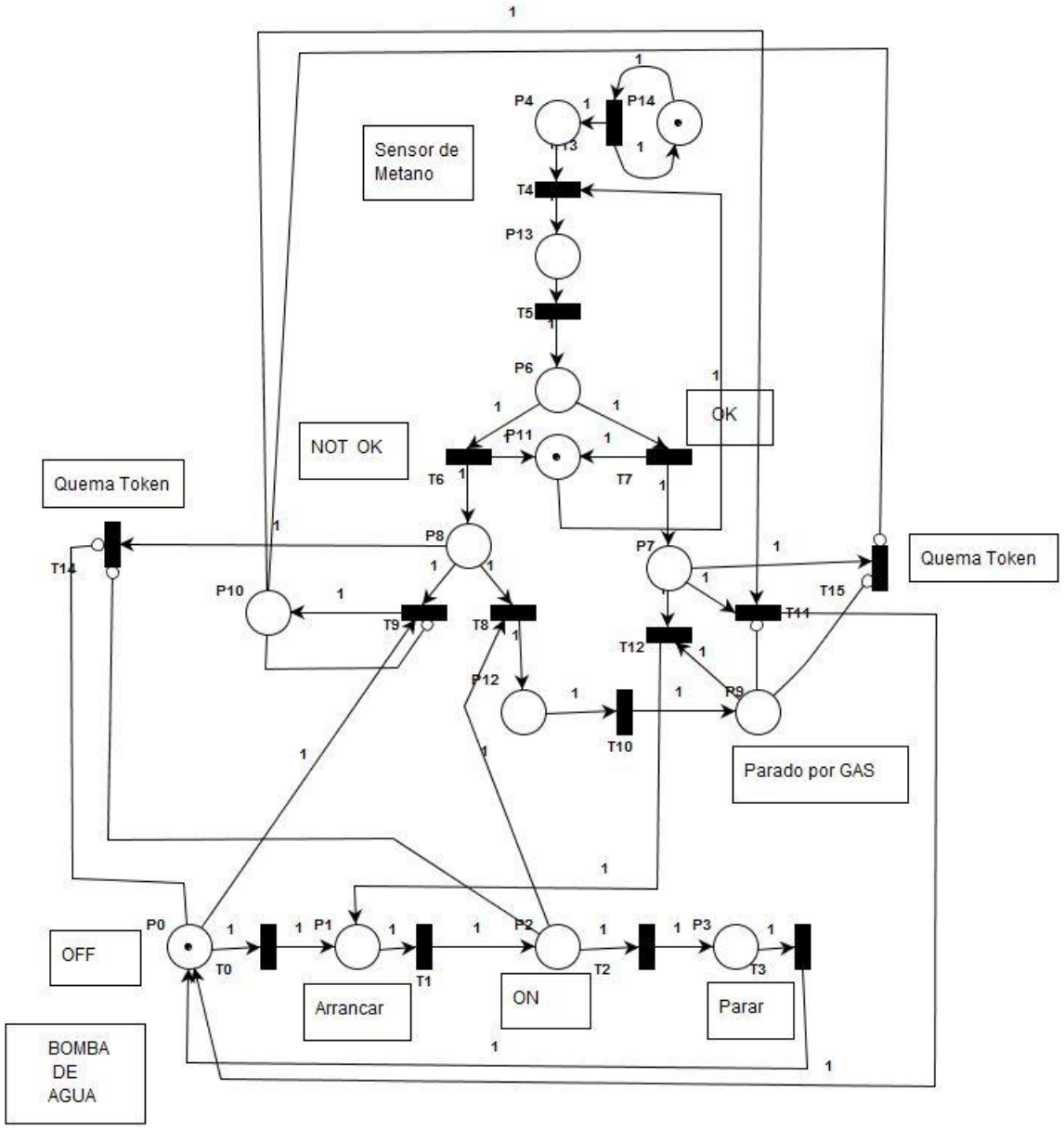
Para esto tenemos en cuenta el funcionamiento de los sensores de gases por un lado, y el carrito junto con el nivel de agua por otro. El sensor, es genéricos. Puede representar a cualquiera de los sensores presentados en el problema, lo que variaría entre ellos sería el tiempo de muestreo de cada uno.

Por otro lado, el carrito con el nivel de agua, que se encarga de encender o apagar la bomba si el nivel llega a niveles críticos, tanto altos, como bajos.

3.2.1 Requerimientos

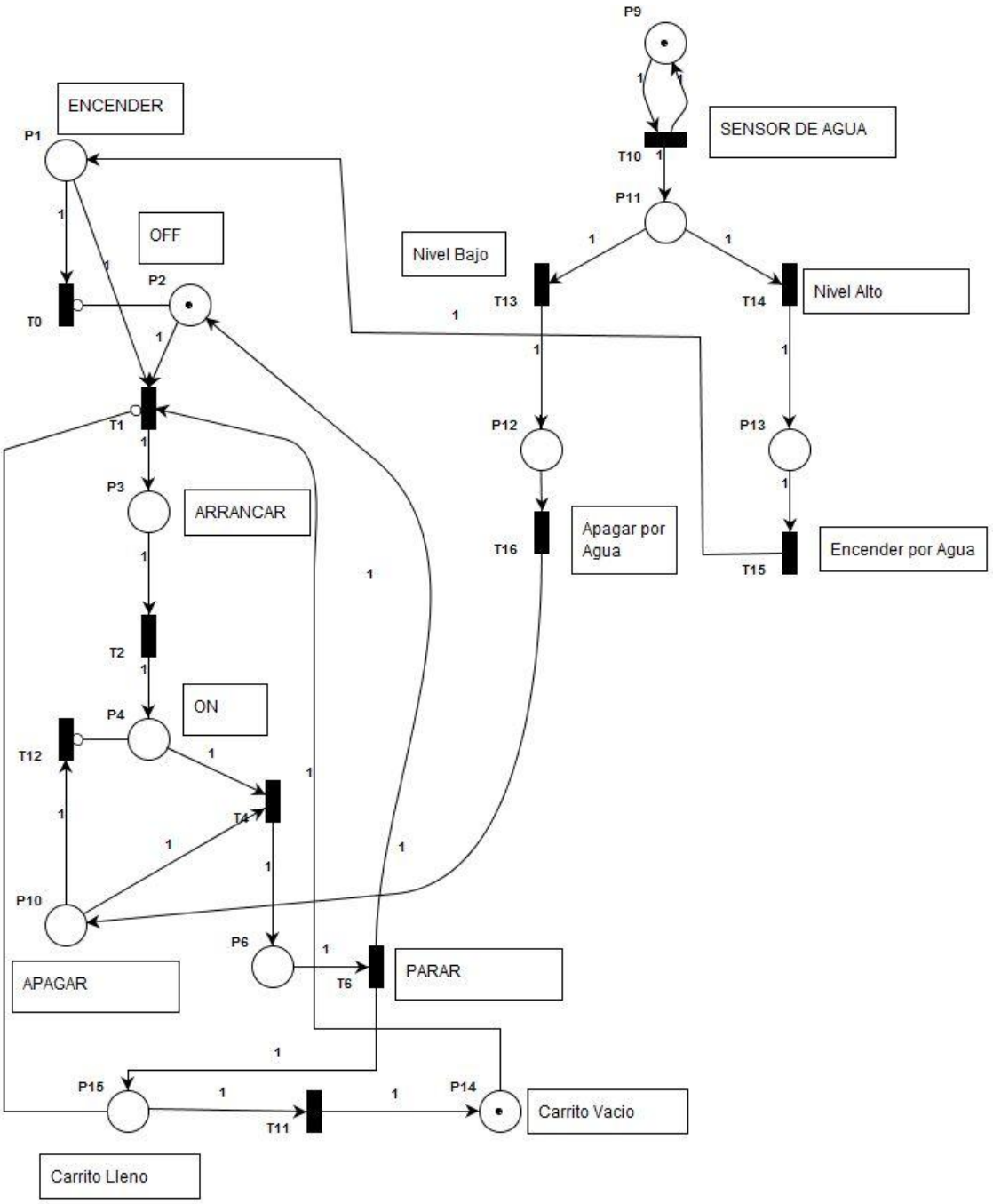
- La bomba se enciende por nivel superior o indicación de un operario.
- La bomba se apaga por nivel inferior o indicación de un operario.
- CO y Aire avisan por alarma
- Sistema se controla por una consola
- Todos los eventos del sistema son guardados para luego ser visualizados

3.2.2 El sensor de gas



La RdP que se muestra, representa a cualquiera de los sensores utilizados en el sistema. La unión de cada uno de ellos se realiza a través de las plazas {P0, P1, P2, P3}, que representan al encendido de la bomba, sea por un operario o por problema detectado por los sensores.

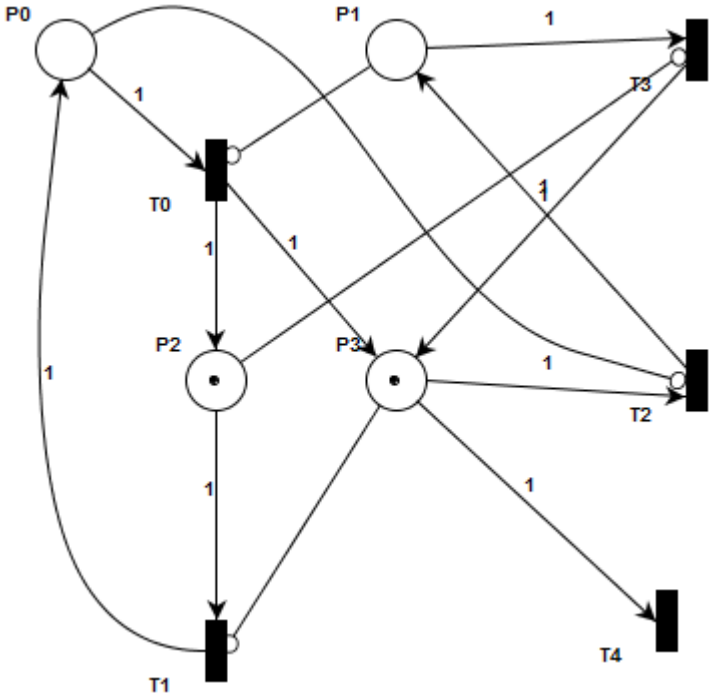
3.2.3 Nivel de agua y carrito



El accionamiento de la bomba se produce por indicación de un operario (T2) o por el sensor de nivel de agua. Este sistema se conecta con el de sensores de gases a través de las mismas plazas, en esta RdP representadas por {P2, P3, P4, P6}. El sistema no puede iniciar si el carrito que se encuentra en la bomba no está vacío.

4 TESTING

Red de prueba:



Marcado Inicial

P0	P1	P2	P3
0	0	1	1

Incidencia:

	T0	T1	T2	T3	T4
P0	-1	1	0	0	0
P1	0	0	1	-1	0
P2	1	-1	0	0	0
P3	1	0	-1	1	-1

Inhibición:

	T0	T1	T2	T3	T4
P0	0	0	1	0	0
P1	1	0	0	0	0
P2	0	0	0	1	0
P3	0	1	0	0	0

Requerimientos Funcionales:

Descripción del Test	RQ1: Disparar transición no sensibilizada
Ejecución	1. Se crea la red con la I,m0 y H de la RdP de prueba. 2. Se intenta disparar T3. 3. Se comprueba que el nuevo marcado sea igual al inicial
Resultado esperado	La transición no se puede disparar. Marcado no modificado.
Pass/Fail	PASS

Descripción del Test	RQ2: Disparar transición sensibilizada
Ejecución	1. Se crea la red con la I,m0 y H de la RdP de prueba. 2. Se intenta disparar T2. 3. Se comprueba que el nuevo marcado sea igual al inicial
Resultado esperado	La transición se puede disparar. Nuevo marcado: {0, 1, 1, 0}
Pass/Fail	**PASS**

Descripción del Test	RQ4: Comprobar Política
Ejecución	1. Se crea la red con la I,m0 y H de la RdP de prueba. 2. Se aplica la prioridad: T2 - T3 - T0 - T4 - T1 3. Se intenta disparar T2 y T4 4. Se comprueba que T2 sea la elegida para ejecutar
Resultado esperado	Se ejecutará T2
Pass/Fail	**PASS**

Descripción del Test	RQ3: Transiciones sensibilizadas
Ejecución	1. Se crea la red con la I,m0 y H de la RdP de prueba. 2. Se llama la función *getSensibilizadas().* 3. Se comprueba que las transiciones que devuelve sean T2 y T4
Resultado esperado	Las sensibilizadas son T2 y T4
Pass/Fail	**PASS**

Matriz Temporal

	α	β
T0	150	200
T1	0	0
T2	0	0
T3	0	0
T4	0	0

Descripción del Test	RQ5: Disparar Sensibilizada temporal
Ejecución	1. Se crea la red con la I,m0, H y el tiempo de la RdP de prueba. 2. En el momento en que la red es creada se inicializan los contadores de las transiciones sensibilizadas 3. Se espera el tiempo necesario para que T0 quede sensibilizada por tiempo 4. Se comprueba que T0 quedó sensibilizada temporalmente.
Resultado esperado	Se espera un 1 en el subíndice 0 del vector de ejecución temporal.
Pass/Fail	**PASS**

Descripción del Test	RQ6: Disparar No sensiblizada temporal
Ejecución	1. Se crea la red con la I,m0, H y el tiempo de la RdP de prueba. 2. En el momento en que la red es creada se inicializan los contadores de las transiciones sensibilizadas 3. Se comprueba que la transición T0 no esta sensibilizada, porque no pasó el tiempo necesario.
Resultado esperado	Se espera un 0 en el subíndice 0 del vector de ejecución temporal.
Pass/Fail	**PASS**

Descripción del Test	RQ7: Disparar Automáticas
Ejecución	1. Se crea la red con la I,m0 ,H y iAutomáticas de la RdP de prueba. Marcado inicial {0, 0, 1, 0, 1} 2. En el momento en que la red es creada se deberían disparar automáticamente los disparos previamente establecidos en el vector como automáticas 3. Sin generar ningún disparo, se comprueba que al crear la Red, el marcado haya sido modificado por las automáticas.
Resultado esperado	Se espera un marcado de {0, 1, 1, 0}
Pass/Fail	**PASS**

5 CONCLUSIONES

Al principio no le dimos el merecido respeto a los diagramas UML, por lo que nos focalizamos en resolver el problema a través de la programación. Al cabo de unos meses, nos dimos cuenta que sin contar con los diagramas se hace difícil hacer el seguimiento del programa. Decidimos entonces realizar los diagramas de clases y secuencia, lo cual fue de gran ayuda para el avance.

Muchas veces se rompían partes que se encontraban terminadas anteriormente, con lo que tuvimos la necesidad de utilizar test unitarios, lo cual nos ayudaba a corroborar que el avance que hiciéramos, sea por nuevas funcionalidades o corrección de errores, no estropeara el trabajo finalizado.

Se reforzaron además muchos conocimientos, se profundizo en el empleo de la herramienta de desarrollo NetBeans y PIPE para la realización de la Red. Lo más importante fue quizás la experiencia de hacer un programa tan extenso y complejo, a comparación a trabajos realizados anteriormente.