Programación concurrente 2018

Integrantes:

- Morales, Julian

TP Integrador

1. Problema

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. El piso superior posee una rampa de ascenso/descenso y solo puede haber un auto a la vez en la misma. 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 ingresan 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 ó 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.

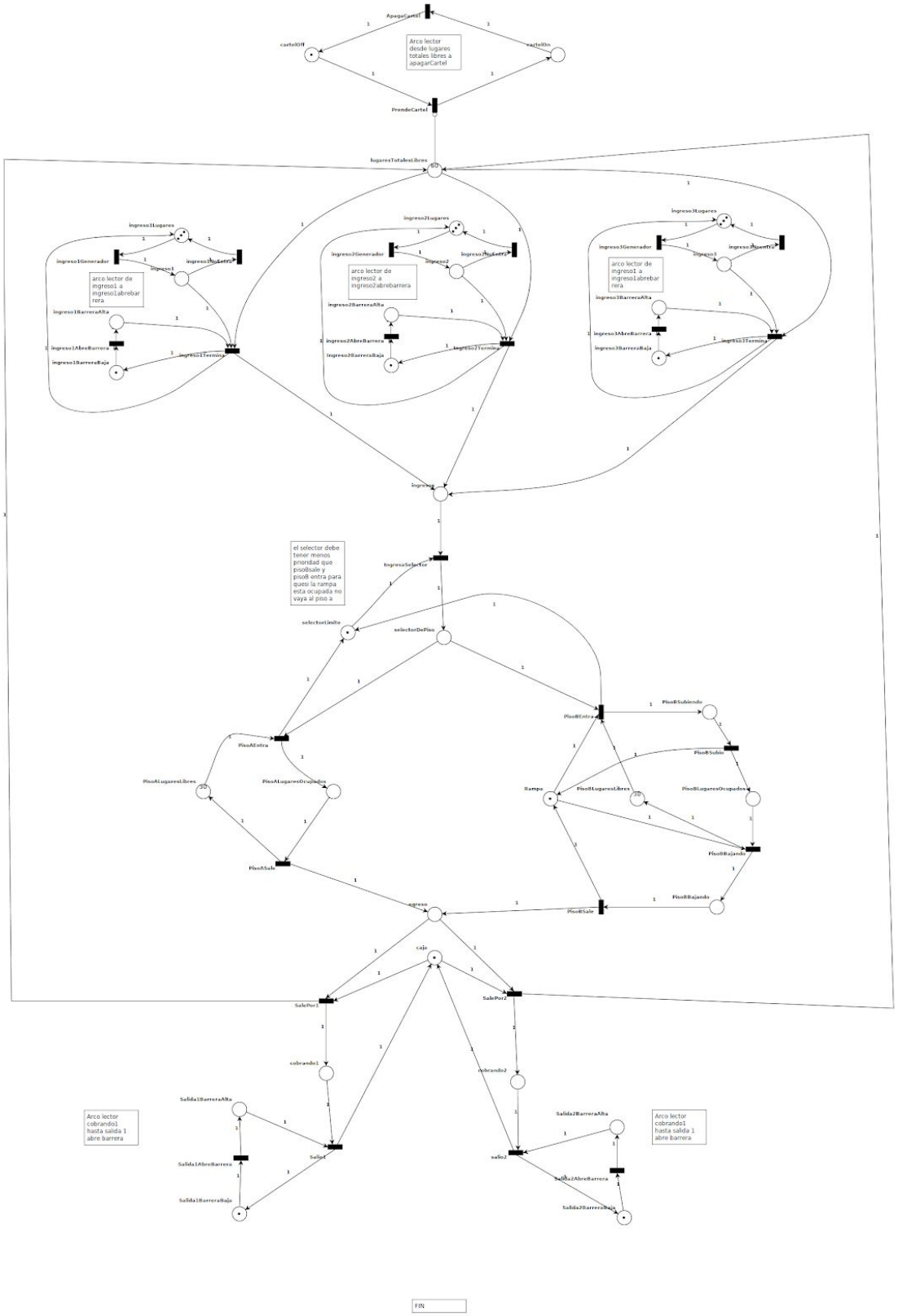
El sistema controlador debe estar conformado por distintos hilos, los cuales deben ser asignados a cada conjunto de responsabilidades afines en particular. Por ej. Ingreso de vehículos, manejo de barreras, etc.

1.1. Debe realizar

1) La red de Petri que modela el sistema.
2) Agregar las restricciones necesarias para evitar interbloqueos ni accesos cuando no hay lugar, mostrarlo con la herramienta elegida y justificarlo.
3) Simular la solución en un proyecto desarrollado con la herramienta adecuada (expliqueporque eligió la herramienta usada).
4) Colocar tiempo en las estación de pago caja (en la/s transición/es correspondiente/s).
5) Hacer la tabla de eventos.
6) Hacer la tabla de estados o actividades.
7) Determinar la cantidad de hilos necesarios (justificarlo)
8) Implementar dos casos de Políticas para:
    a) Prioridad llenar de vehículos planta baja (piso 1) y luego habilitar el piso superior. Prioridad salida indistinta (caja).

    b. Prioridad llenado indistinta. Prioridad salida a calle 2.
9) Hacer el diagrama de clases.
10) Hacer los diagramas de secuencias.

2. La red de Petri que modela el sistema



Adjunto se encuentra en la carpeta doc/PetriModel los modelos creados, siendo esta la tercera versión que cumple con los requerimientos pedidos. También se encuentran los fuente de las redes en formato XML para PIPEv4, el cual elegimos como herramienta ya que permite testear la veracidad de la red y obtener sus invariantes y matrices, que luego con un script transformamos de manera directa a un JSON que se carga en el programa con la configuración de la red y su estado inicial.

Un problema que posee la red, es que está modelando política con lógica, a través del selector límite, que limita la plaza del selector a 1. Esto se puede resolver bajando la prioridad del selector y subiendo la de los accesos a los pisos, para que nunca haya más de un 1 token. La razón del límite del selector es que luego de acceso a uno de los pisos, ambos deben estar desensibilizados y deben sensibilizarse a la misma vez, para que la política pueda tomar las decisión esperada.

3. Tablas de estados y eventos

Tabla de estados	
cartelOff	Cartel apagado
cartelOn	Cartel encendido
lugaresTotalesLibres	Lugares totales libres
ingreso1	Ingresa por calle 1
ingreso1Lugares	Cantidad de lugares en la entrada
ingreso1BarreraAlta	Abierta para que pase 1 auto
ingreso1BarreraBaja	Cerrada
ingreso2	Ingresa por calle 2
ingreso2Lugares	Cantidad de lugares en la entrada
ingreso2BarreraAlta	Abierta para que pase 1 auto
Ingreso2BarreraBaja	Cerrada
ingreso3	Ingresa por calle 3
ingreso3Lugares	Cantidad de lugares en la entrada
ingreso3BarreraAlta	Abierta para que pase 1 auto
ingreso3BarreraBaja	Cerrada

ingresos	Autos esperando para elegir piso
selectorDePiso	Auto en el selector de piso
selectorLimite	Limite de autos en el selector
PisoALugaresOcupados	Lugares ocupados piso a
PisoALugaresLibres	lugares libres piso a
Rampa	rampa de asc/desc piso b
PisoBSubiendo	Auto subiendo al piso b
PisoBLugaresLibres	Lugares lires segundo piso
PisoBLugaresOcupados	lugares ocupados piso b
PisoBBajando	Bajando del piso b
egreso	Saliendo del estacionamiento para ir a cobrar
caja	Cajero
cobrando1	Cobrando por calle 1
Salida1BarreraBaja	Barrera baja de la calle 1
Salida1BarreraAlta	Barrera alta de la calle 1
cobrando2	Cobrando por la calle 2
Salida2BarreraAlta	Barrera alta de la calle 2
Salida2BarreraBaja	**Barrera baja de la calle 2**

Tabla de eventos	
ApagaCartel	**Cuando hay lugares disponibles**
PrendeCartel	**Cuando no hay lugares libres disponibles**
ingreso1Termina	**Cuando el auto entra a la playa**
ingreso1AbreBarrera	**Cuando el auto quiere entrar a la**

	playa
ingreso1Generador	Autos que llegan
ingreso1NoEntra	Autos que llegan pero no entran
Ingreso2Termina	Cuando el auto entra a la playa
ingreso2AbreBarrera	Cuando el auto quiere entrar a la playa
ingreso2Generador	Autos que llegan
ingreso2NoEntra	Autos que llegan pero no entran
ingreso3Termina	Cuando el auto entra a la playa
ingreso3AbreBarrera	Cuando el auto quiere entrar a la playa
ingreso3Generador	Autos que llegan
ingreso3Noentra	Autos que llegan pero no entran
IngresaSelector	Cuando el auto debe seleccionar el piso
PisoAEntra	Entro al piso a
PisoASale	Salio del piso a
PisoBEntra	Entra al piso b si esta disponbile la rampa
PisoBSale	Sale del piso, debe esperar la rampa
PisoBSubio	Subiendo al piso por la rampa
PisoBBajando	Bajando al piso por la rampa
SalePor1	Sale por la calle 1
Salida1AbreBarrera	Se abre la barrera para quie salga
Salio1	Salio
SalePor2	Sale por calle 2
salio2	Salio
Salida2AbreBarrera	Abre la barrera para que salga

4. Cantidad de hilos

Utilizamos un hilo por transición menos aquella secuencia de transiciones que se disparan en un orden específico y necesario,como por ejemplo la salida por una de las calles, o la subida al piso de arriba. En la siguiente tabla se puede ver los hilos que usamos para la simulacion, donde esta:

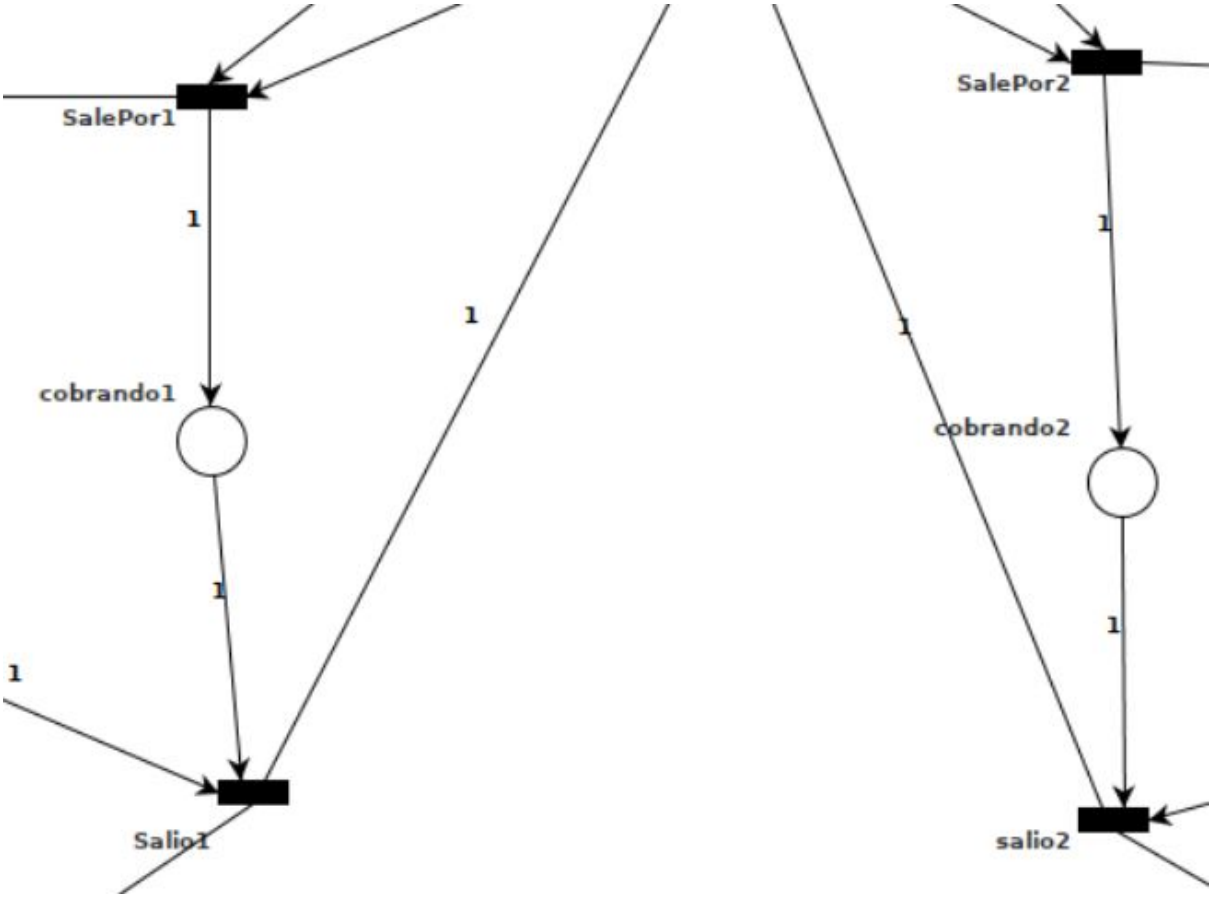
- El nombre del hilo que utilizamos,
- Secuencia de transiciones que debe disparar
- El tiempo que debe esperar antes de volver a disparar
- Cantidad de veces que debe disparar.

name	seq	sleepTime	nTimes
Cartel	[2,1]	0	0
Generador Ingreso 1	[5]	7	100
Ingreso1 no entra	[6]	100	0
Ingreso 1 abre barrera	[4]	0	0
ingreso 1 termina	[3]	0	0
Generador Ingreso 2	[9]	20	100
Ingreso 2 no entra	[10]	100	0
Ingreso 2 abre barrera	[8]	0	0
ingreso 2 termina	[7]	0	0
Generador Ingreso 3	[13]	12	100
Ingreso 3 no entra	[14]	100	0
Ingreso 3 abre barrera	[12]	0	0
ingreso 3 termina	[11]	0	0
ingreso al selector	[15]	0	0
Entra piso A	[16]	0	0
Sale Piso A	[17]	0	0
Piso b entra + sube	[18,20]	0	0
Piso b sale + baja	[21,19]	0	0
Sale por 1 + cobro y salio	[22,24]	0	0
Barrera salia 1	[23]	0	0
Sale por 2 cobro y salio	[25,26]	0	0
BArrera salida 2	[27]	0	0

4.1 El problema de creación de hilos/Hilos que no se encolan

El trabajo fue implementado con una clase de test de integración del monitor, que no contemplaba el caso de sincronización de hilos antes de arrancar. Esto produce que al trabajar con muchos hilos, se inicien en distinto orden en el que fueron invocados con start, y se disparen transiciones sin que todos los hilos estén esperando para disparar. Esto produce en los primeros milisegundos una evolución distinta a la esperada por las políticas.

Otro problema que se presentó es que en la utilización de muchos hilos con pocos cores, lleva a los threads a devolver el semáforo del monitor y cambiar de contextoantes que se encolen para pedir nuevamente disparar la transición. Esto lleva a que las políticas reciban datos de colas vacías cuando y devuelva un resultado no deseado por que el hilo no esta encolado. Por ejemplo en la salida de la playa, cuando se utiliza un hilo para disparar SalePor1-Salio1 y otro para SalePor2-salio2, (Donde salio1 y salio2 son temporales), el resultado de la ejecución se puede ver claramente un 50% de autos toma un camino, y el otro 50%, lo que pareciera ignorar las políticas, aunque en verdad es que los hilos disparan y no se alcanzan a encolar.



Cuando en vez de utilizar 2 hilos, utilizamos 4, los hilos siempre se encolan y el programa funciona como lo esperado. Ambas situaciones con los hilos se puede detectar

con el debugger, aunque la primera es más sencilla detectarla con un logueo de eventos de la clase logger.

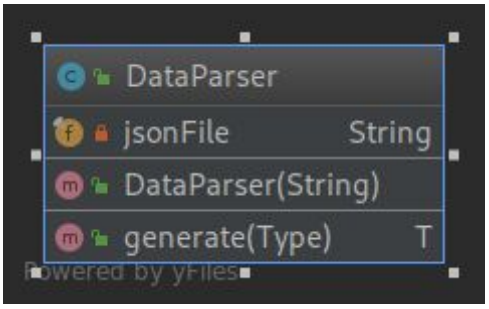
5. Diagrama de clases

El diagrama de clases es muy extenso, y al igual que los demás se encuentra en doc/Diagramas/Package Monitor. El monitor se encuentra compuesto por 5+1 módulos que pueden ser utilizados de manera independiente:

1. parser
2. logger
3. rdp
4. colas
5. policies
6. complex_test

5.1. Parser

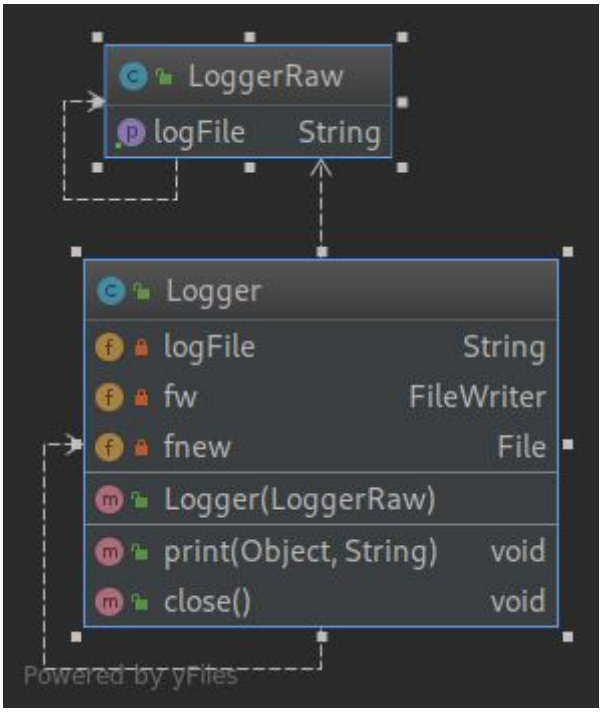
Contiene el DataParser, que es el encargado de generar los objetos necesarios para la configuración de los demás módulos a partir de un archivo de estructura tipo JSON.



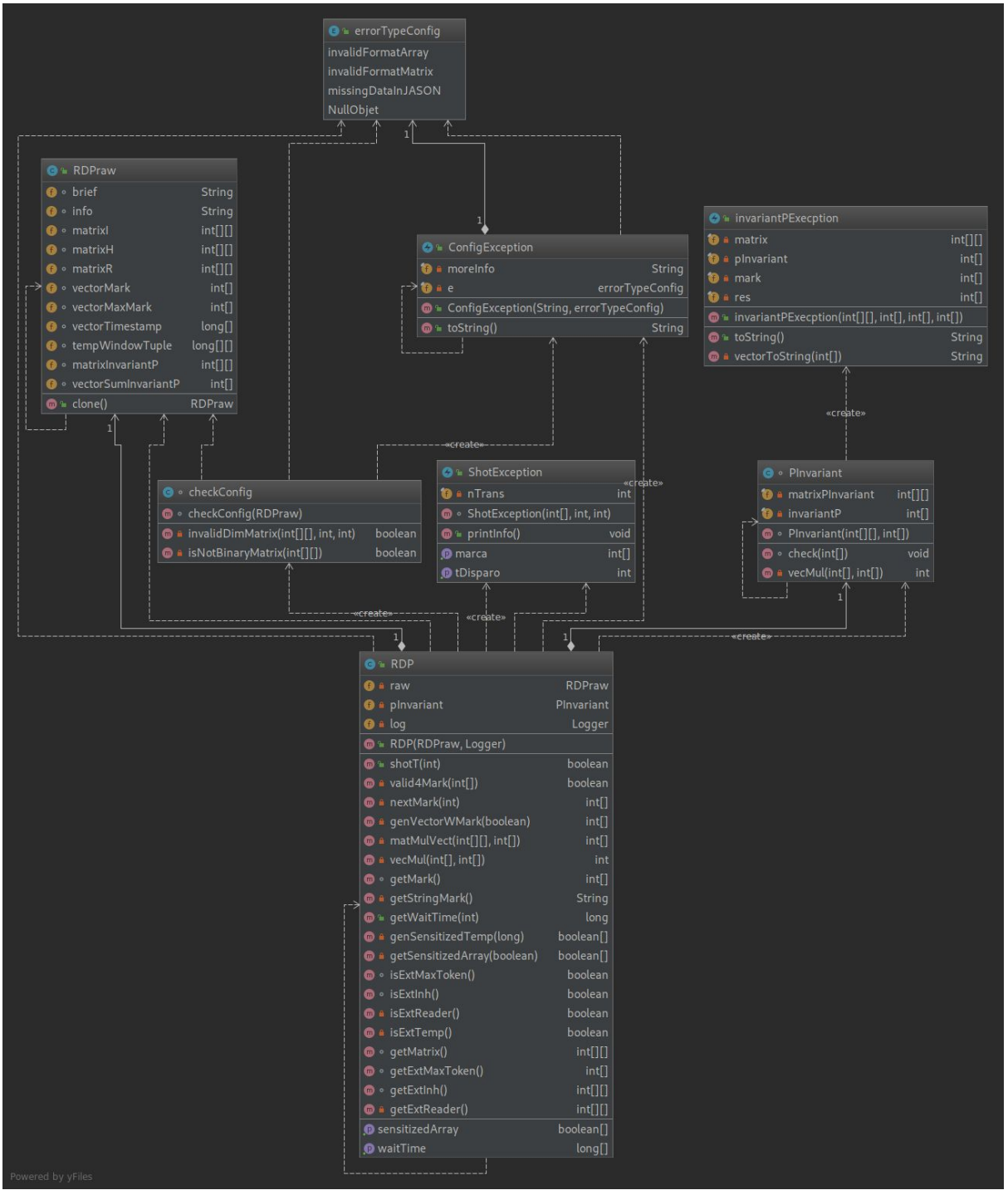
5.2. Logger

Utilizado para guardar todo tipo de eventos en todos los módulos, guarda el timestamp del evento, con la información de que hilo (id + nombre) genero el evento , desde que modulo genero el evento y el evento en sí mismo.

Si no se especifica en el Json la configuración, guardada en un archivo llamado log.txt en directorio de trabajo.



5.3. RDP

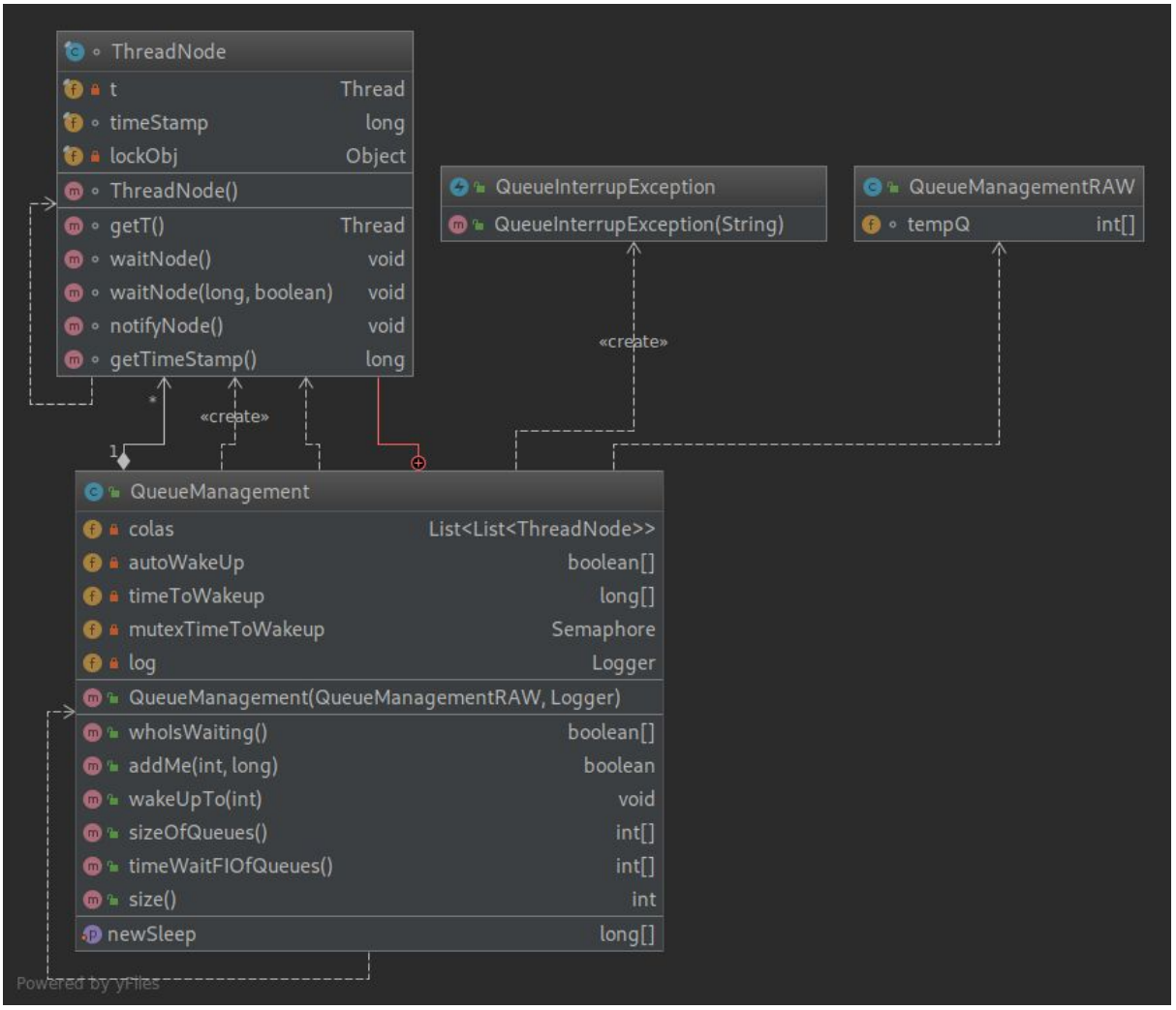


Modela la red de petri por completo, carga la configuración de la red desde RDPRaw, y Soporta características independientes (opcionales) de redes de petri extendidas como:

- Maxima cantidad de tokens por plaza
- Arcos inhibidores y lectores peso 1
- Transiciones sensibilizadas temporalmente

Tambien realiza un chequeo de invariantes de plaza luego de cada disparo y en la carga de la matriz. Se puede obtener el tiempo que falta para que se sensibilicen las transiciones temporales y también las sensibilizadas actualmente.

5.4. Colas

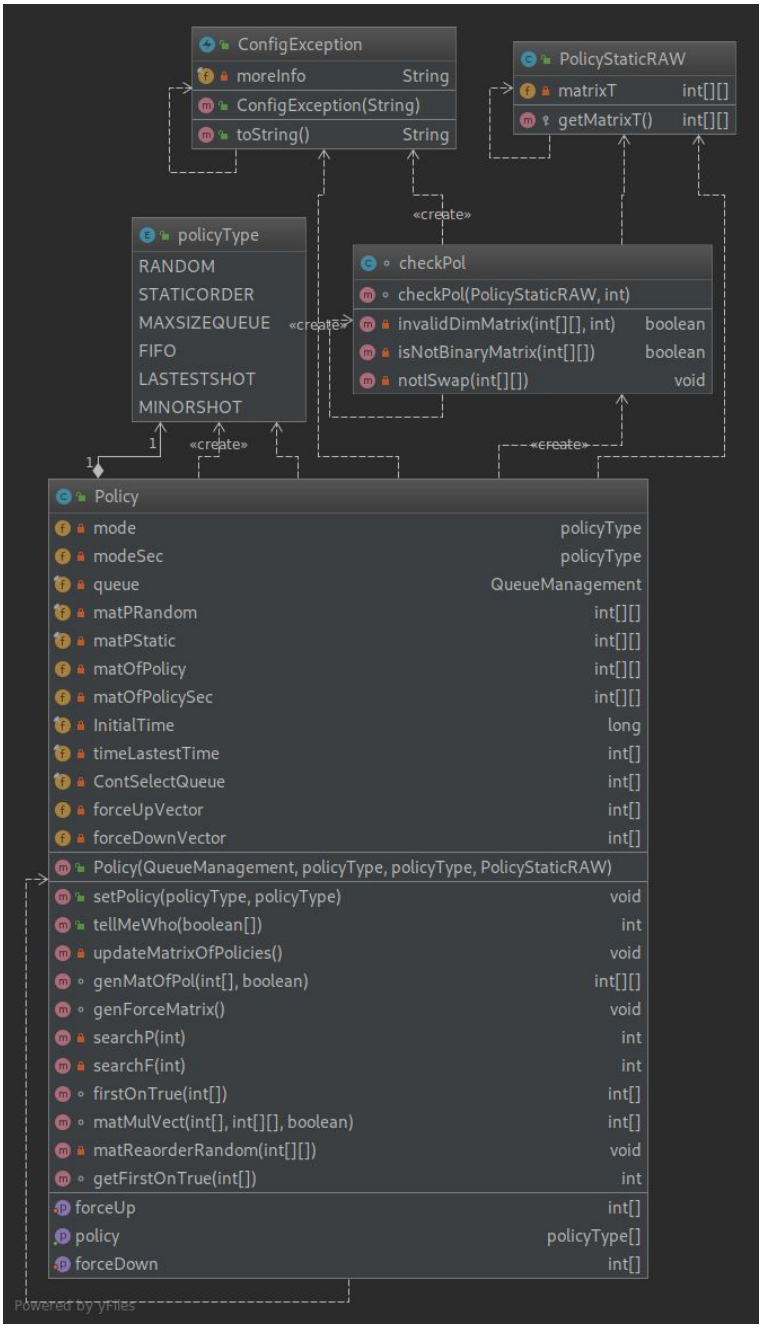


El Manejador de multiples colas (Listas FIFO) de threads del monitor, es creado con un numero de colas fijo y vacias. Cuando el thread vuelve a ready sale de la lista.
Posee mecanismos para:

- Agregar/Eliminar threads de una cola
- Consultar el tamaño de las colas
- Consultar Tiempo que lleva de espera del primer threads en cada cola (Utilizado para generar politicas dinamicas)
- Anadir threads por un determinado tiempo y que estos levanten solos.
- Anadir threads sin tiempo y luego asignarles tiempo para que levanten solos de manera dinamica.

- Soporta interrupciones externas al manejador o monitor que sacan el threads de la lista si se lo interrumpe cuando estaba esperando y larga una nueva excepción.

5.5. Policies



El manejador de políticas se puede crear opcionalmente a una cola asociada, donde la puede utilizar para obtener información y generar políticas de manera dinámica.

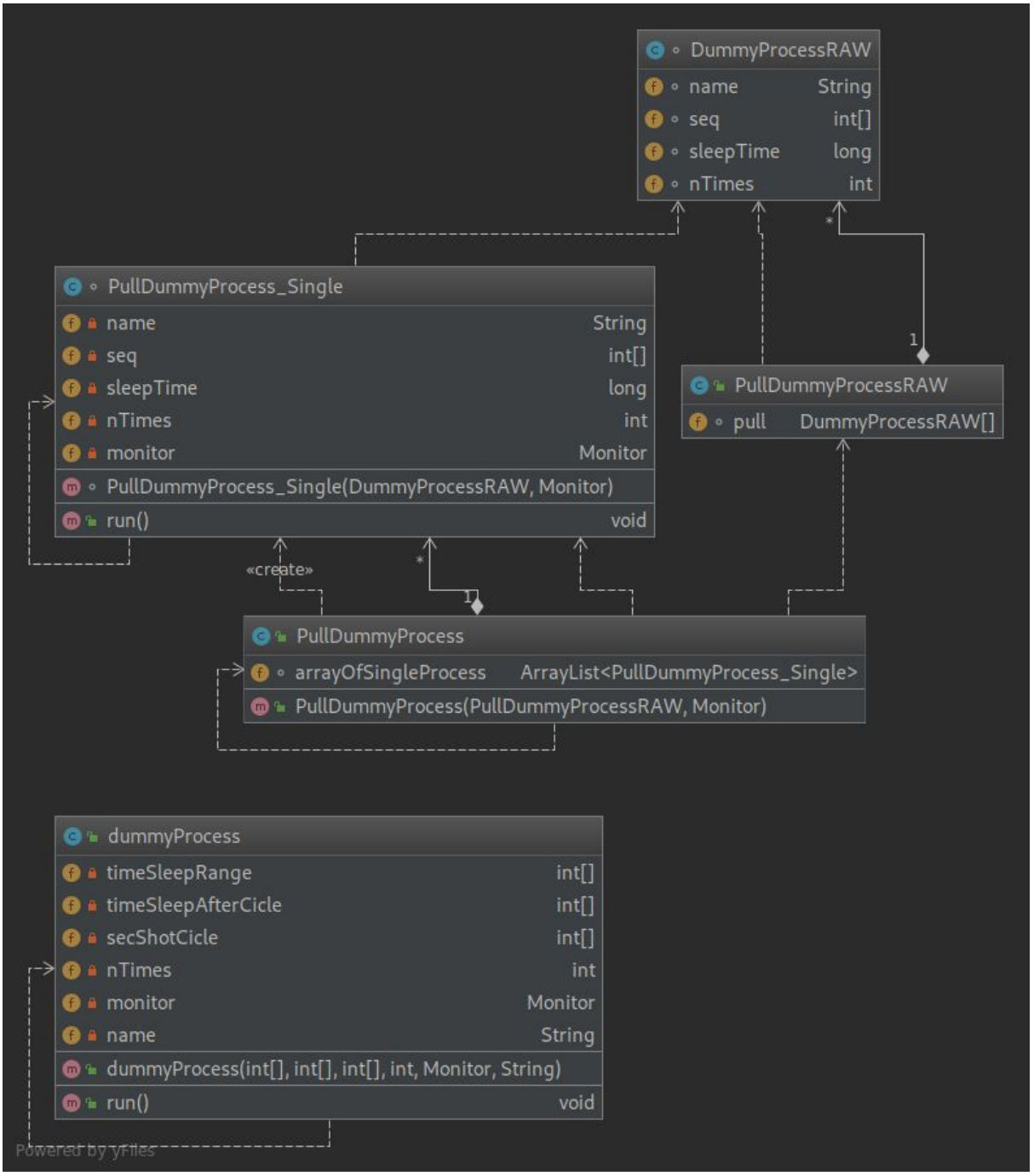
La política puede cambiar de manera dinámica y soporta 6 políticas primarias distintas:

1. **RANDOM**: Selecciona una cola al azar, todas tienen la mismas probabilidades de ser disparadas, convierte al sistema en no determinista. Cambia la prioridad en cada disparo.
2. **STATICORDER**: Selecciona la cola a disparar por un orden de prioridad fijo y estático. En caso de que no se especifique la matriz de prioridad, esta genera una matriz de priorirar de tipo identidad, donde la transición más baja tiene mas prioridad.
3. **MAXSIZEQUEUE**: Selecciona la cola más grande de las disponibles a disparar
4. **FIFO**: Selecciona la cola que hace mas tiempo esta esperando a ser disparada
5. **LASTESTSHOT**: Selecciona la cola que hace más tiempo que no fue selecciona
6. **MINORSHOT:** Selecciona la cola que entró en cola y fue disparada menos veces

En caso de empate en la política primaria (por ejemplo que 2 colas tengan el mismo largo), esta es desempatar con la política secundaria que solo puede ser **RANDOM** o **STATICORDER** ya que esta nunca puede entrar en conflicto.

Pero además de todo esto las políticas pueden ser combinadas y forzadas para que sean altas o bajas, es decir puede utilizarse las políticas primarias y secundarias combinadas sumadas a una política forzada, para que determinadas transiciones/colas tengan una política de prioridad fija, en un determinado orden (Alta o Baja) y el resto de las transiciones/colas se generen con las políticas primarias y secundarias

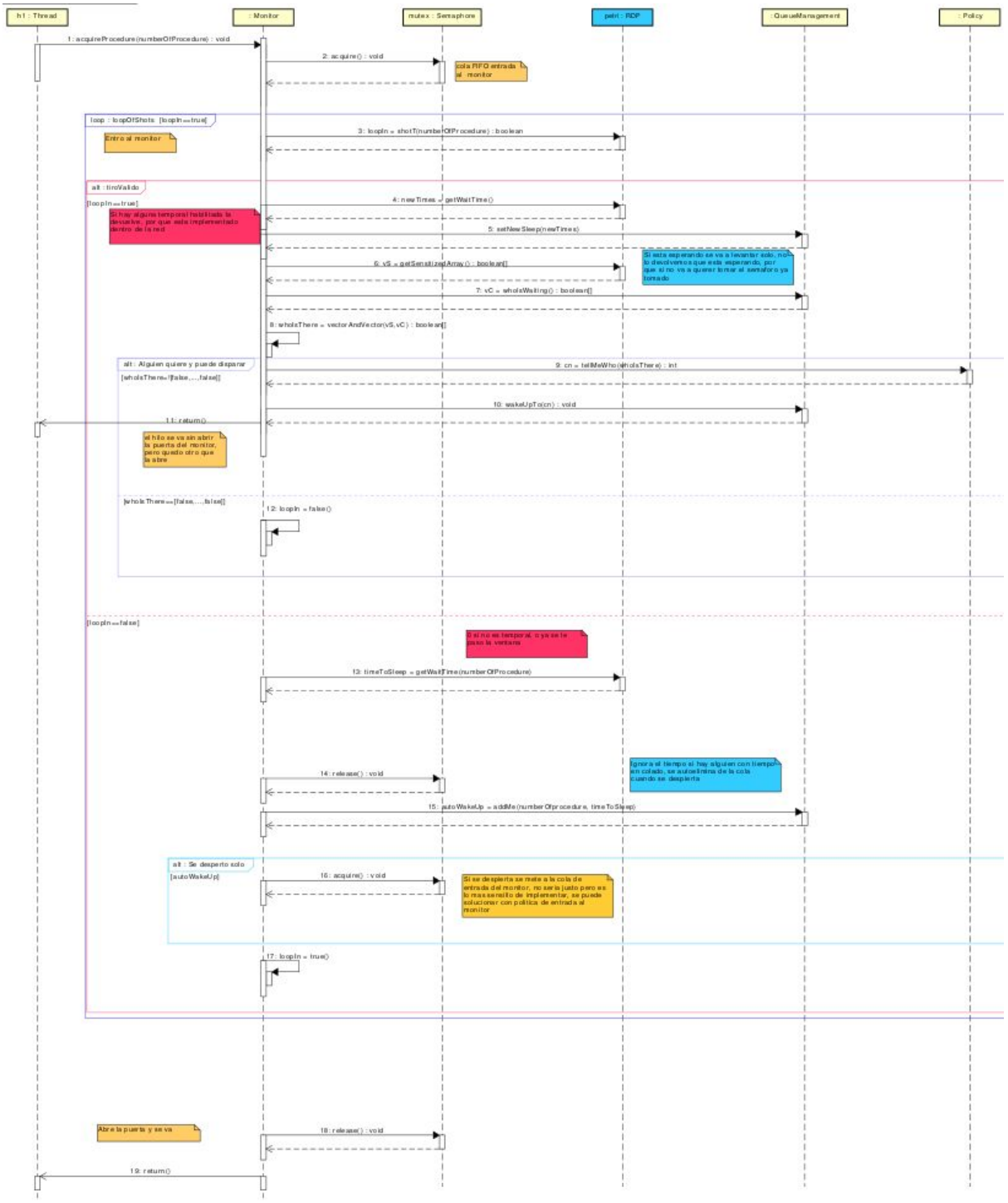
5.6. Complex_test



Utilizado para los test, y luego tambien sirvio para no crear nuevas clases a la hora de realizar el trabajo pedido, el objeto PullDummyProcessRAW es cargado de un json, que posee un array de DummyProcessRAW. Estos representan procesos dummy que poseen un nombre, una secuencia de disparo a intentar, un tiempo a esperar después de cada disparo antes de intentar volver a disparar y una cantidad de veces que pueden disparar.

6. Diagrama de secuencia del monitor

Los diagramas se encuentran con mejor calidad en la carpeta doc/Diagramas, y son 3. El primero de ellos, es el monitor sin tiempo, el segundo es con tiempo que es el que fue implementado y el tercero es una idea de como resolver las políticas en las temporales en sistemas más complejos, pero no fue implementado no solo por la complejidad, si no por que la red actual no usaría las funciones que resuelve este último diagrama.



7. Estructura de configuración del JSON

El diagrama de clases es muy extenso, y al igual que los demás se encuentra en doc/Diagramas/Package Monitor

Cargada desde un archivo json. Donde el primer lugar representa la transicion o la plaza uno dependiente el caso. Las matrices de incidencia estan ordenadas por filas que representan las plazas y columnas que representan las transiciones. Ningun valor puede ser salteado.

N: Cantidad de transiciones
M: cantidad de plazas

Campos obligatorios del JSON:

- Matriz de incidencia I [dim: MxN]
- Vector de marcado [dim: M]

Campos opcionales del JSON:

- Matriz de Incidencia H [dim: NxM]
    - Los valores deben ser binarios
    - Filas representan transiciones
    - Columnas representan plazas
    - 1 Si hay un arco inhibidor entre entre la plaza y la transicion
    - 0 Si no hay relacion
- Matriz de Incidencia R [dim: NxM]
    - Los valores deben ser binarios
    - Filas representan transiciones
    - Columnas representan plazas
    - 1 Si hay un arco lector entre la plaza y la transicion
    - 0 Si no hay relacion
- Vector de tokens maximos por plaza [dim: M]
    - Cada valor representa una plaza

    - Lo valores deben ser 0 para sin restriccion o un numero mayor a 0 para setear un maximo de tokens en esa plaza
- Matriz de de politicas T [dim: NxN]
    - Politica estatica, plana. Utilizada para crear las Policies.
    - Los valores deben ser binarios
    - La matriz es una matriz identidad de NxN con las filas cambiadas de orden, donde el orden reprensta la prioridad,
      esto es:
        - La matriz es cuadrada
        - Cada colunas representa una transicion
        - En cada fila/columnas hay y solo hay un 1 (No pueden ser todos ceros y no pueden tener mas de uno)
        - La posicion del 1 representa el nivel de prioridad
        - Las tranciciones no pueden tener igual prioridad (No pueden haber 2 filas/columnas iguales)
- Vectores de ventana temporal [dim: 2xN]
    - Cada columna es asignada a una transicion
    - La primera fila es el inicio de la ventana temporal.
    - la otra fila representa el fin de la ventana temporal.
- Matriz y vector de invariantes de plaza
    - Deben ser ambos utilizados o ninguno, funcionan en conjunto
    - Matriz de invariantes de plaza. [dim: IPxM]
        - Cada fila representa un conjunto de de plazas que formar el invariante (IP Invariantes)
        - Cada columna representa con un 1 si forma parte de algun invariante o no (0).
        - El la dimencion de la fila es la cantidad de plazas M
    - Vector de invariantes de plaza. [dim: IP]

        - el valor 's' del vector representa la suma invariante de plas plazas en la matriz de invariantes

    - El invariante 's' se chequea haciendo el productor interno entre la fila 's' de la matriz y el vector de marcado, donde el resultado del (escalar), es el valor 's' del vector de invariantes.

- matrixInvariantT, matriz de invariantes de plaza, de tamano ITxN
    - Cada fila representa un conjunto de de transiciones que forman el invariante (IT Invariantes)
    - Cada columna representa con un 1 si forma parte de algun invariante o no (0).
    - El la dimencion de la fila es la cantidad de transiciones N

- Pull de procesos, opcional para pruebas, un vector que tienen los siguientes datos cada entrada:
    - "name": "T4-T6, leen",
    - "seq" : [4,6],
    - "sleepTime" : 0,
    - "nTimes" : 0

7.1 Ejemplo de configuración

```
{
  "brief" : "Ejemplo 6: Un escritor Multiples lectores simultaneos",
  "info" : "Conjunto de procesos (P0), que pueden leer (P4) simultaneamente o escribir (P3)",
  "logFile": "log_ej6_conTiempo.txt",
  "tempQ": [0,0,0,0,1,1],
  "matrixI" :  [
    [-1, -1, 0, 0, 1, 1], //P1  procesos Idle
    [ 1,  0,-1, 0, 0, 0], //P2 Quieren escribir
    [ 0,  0,-1, 0, 1, 0], //P3 recurso
    [ 0,  0, 1, 0,-1, 0], //P4 escribiendo
    [ 0,  0, 0, 1, 0,-1], //P5 leyendo
    [ 0,  1, 0,-1, 0, 0]  //P6 quire leer
  ],
  "vectorMark"     : [7, 0, 1, 0, 0, 0], // 7 procesos
  "vectorMark"     : [0, 0, 0, 0, 0, 0], // Sin max por plaza
  "matrixH" :  [ // Al revez que I, transiciones por plaza
    [ 0,  0, 0, 0, 0, 0], //t1 Agregarse a la cola de escritura
    [ 0,  0, 0, 0, 0, 0], //t2 Agregarse a la cola de lectura
    [ 0,  0, 0, 0, 1, 0], //t3 Entrar a escribir (La escritura queda inhibida si hay alguien leyendo
P5)
    [ 0,  0, 0, 0, 0, 0], //t4 Entrar a leer
    [ 0,  0, 0, 0, 0, 0], //t5 Termina de escribir
```

```
    [ 0,  0, 0, 0, 0, 0]  //t6 Termina de leer
  ],
  "matrixR" :[ // Al revez que I, transiciones por plaza
    [ 0,  0, 0, 0, 0, 0], //t1 Agregarse a la cola de escritura
    [ 0,  0, 0, 0, 0, 0], //t2 Agregarse a la cola de lectura
    [ 0,  0, 0, 0, 0, 0], //t3 Entrar a escribir
    [ 0,  0, 1, 0, 0, 0], //t4 Entrar a leer Si el recurso (p3) no fue tomado para escritura
    [ 0,  0, 0, 0, 0, 0], //t5 Termina de escribir
    [ 0,  0, 0, 0, 0, 0]  //t6 Termina de leer
  ],
  "matrixT" :[
  //t1-t2-t3-t4-t5-t6 ** Escritor
    [0, 0, 0, 0, 0, 1],         // Un escritor con multiples lectores
    [1, 0, 0, 0, 0, 0],         // Prioridad escritor cuando:
    [0, 0, 1, 0, 0, 0],         // t6>t1>t3>t5>t2>t4
    [0, 0, 0, 0, 1, 0],         // Prioridad lector cuando:
    [0, 1, 0, 0, 0, 0],         // t5>t4>t2>t6>t1>t3
    [0, 0, 0, 1, 0, 0]
  ],
  "matrixInvariantP" :[
    [ 1,  1, 0, 1, 1, 1],
    [ 0,  0, 1, 1, 0, 0]
  ],
  "vectorSumInvariantP" : [7, 1],
  "tempWindowTuple" : [
    [    0,    0,    0,    0, 10,  5], // Tiempo de lectura 500ms, tiempo escritura 1000
    [    0,    0,    0,    0,    0,    0]
  ],
  // pull Threads
  "pull" : [
    // T1 - Quien escribir
    {
      "name": "T1, Quieren escribir",
      "seq" : [1],
      "sleepTime" : 50,
      "nTimes" : 10
    },
    // T3-T5 - Escriben
    {
      "name": "T3-T5, Escriben",
      "seq" : [3,5],
      "sleepTime" : 0,
      "nTimes" : 0
    },
    // T2 - Quien leer
    {
      "name": "T2, Quieren Leer",
      "seq" : [2],
      "sleepTime" : 20,
      "nTimes" : 20
    },
    // T4-T6 - leen
    {
      "name": "T4-T6, leen",
      "seq" : [4,6],
      "sleepTime" : 0,
      "nTimes" : 0
    }
  ]
}
```

8. Otros script / tInvariant

Con bash podemos filtrar el log y ver los disparos que fueron fueron realizados con exito, ordenarlos por tiempo, filtrar solo los numeros de transiciones, ordenarlos por transicion y contar cuantas veces se disparo cada transicion de la siguietne forma:

```
log=$(grep -E "ShotEvent .* true" ../log_tp2018.txt | sort -k1 | awk -F  "|"  '{print $6}' | awk
'{print $2}' |  sort -k2 | uniq -c);
echo "$log"
```

Si eso lo guardamos en un archivo, (vdis.txt) luego con el script de python checkInv.py, se encarga de leer la configuración de la red, quitar los invariantes (todo desde el archivo de configuracion json), y el resto de transiciones multiplicarlas por la matriz de incidencia, a la que luego sumamos el marcado y corroboramos que el mercado siga siendo el mismo que nos dio el ultimo disparo exitoso en el log.

9. Conclusiones

El trabajo nos sirvió para aprender a separar el modelo de la planta de las políticas, es decir la lógica (Lo que puedo hacer) y las políticas (lo que quiero hacer dentro de lo que puedo). En nuestro caso para modelar la lógica usamos las redes de petri pero podemos cambiar el modulo y utilizar otro mecanismo para modelar la misma. Este módulo contiene el estado del sistema y con él todas las sentencias condicionales que forman el modelo de la planta.

La ventaja de la utilización de las redes de petri es su comprobación matemática, poder asegurar parte de los requerimientos a través de funciones y demostraciones, lo que garantiza su funcionamiento. La otra parte de los requerimientos estará dado en las políticas.