Tarea nº3 Lenguajes de Programación

Víctor Daniel Ríos Tapia - 18.568.652-3

Damian Andrés Reyes Cordero - 18.950.453-5

24 Octubre 2017

1. Resumen

En el presente informe se explicará el concepto de 'red de Petri', además de su función, y sus usos; posteriormente se procederá a modelar, y programar, un sistema mediante el uso de dicha red.

2. Objetivo

Explicar y entender el concepto de 'red de Petri', tanto a nivel de funcionamiento como de usos. Modelar un equipo de desarrollo de software, que trabaja bajo ciertas condiciones, mediante el uso de una red de Petri. Programar dicho modelamiento en cualquier lenguaje de programación.

3. Introducción

Una red de Petri es una representación gráfica (o matemática) que sirve para modelar sistemas que, generalmente, poseen propiedades concurrentes, no deterministas, comunicacionales, o de sincronización. Dicha representación gráfica se hace utilizando 4 símbolos:

- Plaza (círculo): Lugar / espacio / estado del sistema.
- Transición (rectángulos vertical): Evento que permite relacionar plazas, causando cambios en el sistema.
- Arco (Flecha): Une plazas y transiciones para poder armar el modelo de red, y poder expresar el sentido del flujo del sistema. Máximo puede haber un arco uniendo una transición con una plaza.
- Token (Punto): Marca que se ubica dentro de las plazas, y que pueden representar distintas cosas según el sistema que se esté modelando.

Como se mencionó anteriormente, una red de Petri se utiliza para modelar un sistema; y dicha red debe crearse en base al problema a resolver.

Una red de petri debe estar diseñada para tener diversos estados (plazas) dependiendo de las distintas acciones que requiera el sistema. Para pasar de una plaza a otra siempre se hará a través de una transición (condición / evento); por lo tanto, no pueden existir dos plazas unidas directamente.

La simulación de los cambios de estados se realiza mediante el movimiento del token. El token puede pasar de una plaza A a una plaza B, sí y solo sí, se cumple el evento que indica la transición que une dichas plazas.

Teniendo todo esto en cuenta, a continuación se procederá a diseñar una red de petri para un sistema que simula un equipo de desarrollo de software cuya forma de trabajar varía dependiendo de ciertas condiciones / restricciones. Dicho sistema será programado utilizando el lenguaje de programación Java.

4. Desarrollo

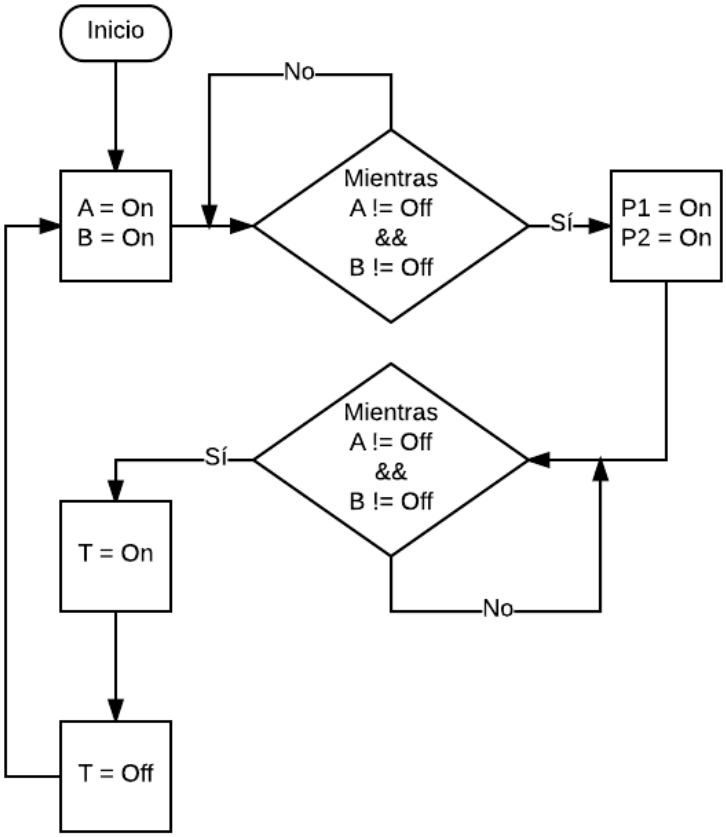
El problema planteado nos pide desarrollar una red de Petri que represente el funcionamiento de un equipo de desarrollo de software, el cual está constituido por un analista (A), un diseñador (D), dos programadores (P1 y P2), y un tester (T). Dicho equipo de desarrollo necesita que se cumplan las siguientes restricciones para poder avanzar con sus proyectos:

- A y D pueden trabajar de forma independiente.
- P1 y P2 solo pueden trabajar una vez terminen A y D.
- T solo puede trabajar una vez que terminen P1 y P2.
- T siempre encuentra nuevos problemas, e itera nuevamente hacia A y D.

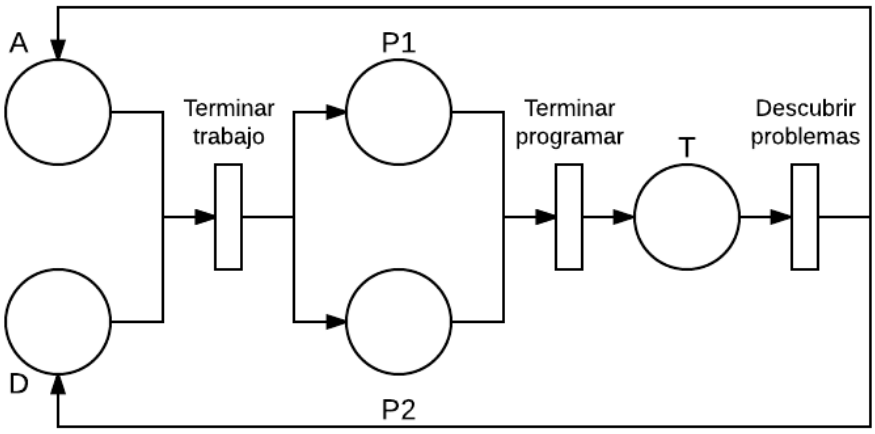
Si analizamos el problema, y describimos como una cadena de acciones, podemos inferir lo siguiente:

- Tanto A como D pueden pueden empezar a trabajar en cualquier momento, ya que son independientes.
- P1 y P2 solo pueden trabajar una vez terminen A y D, por lo tanto, si A terminase primero que D, hay que esperar a que D termine y posteriormente llamar a trabajar a P1 Y P2. En el caso de que D termine primero que A, se hace lo mismo pero al revés.
- Como no se menciona nada acerca de las condiciones de trabajo de P1 y P2, y dado que ambos se desempeñan en lo mismo, diremos que estos comienzan a trabajar, y que terminan de hacerlo, al mismo tiempo.
- T puede empezar a trabajar una vez que P1 y P2 terminen.
- Cuando T termine su trabajo, derivará el proyecto nuevamente a A y D; volviéndose a repetir todo el proceso.

Realizando un pequeño diagrama de flujo, para poder ver así el funcionamiento del proceso de manera más simplificada y posteriormente pasar dicho diagrama a una red de Petri, obtendremos algo como lo que se muestra en las siguientes dos imágenes:

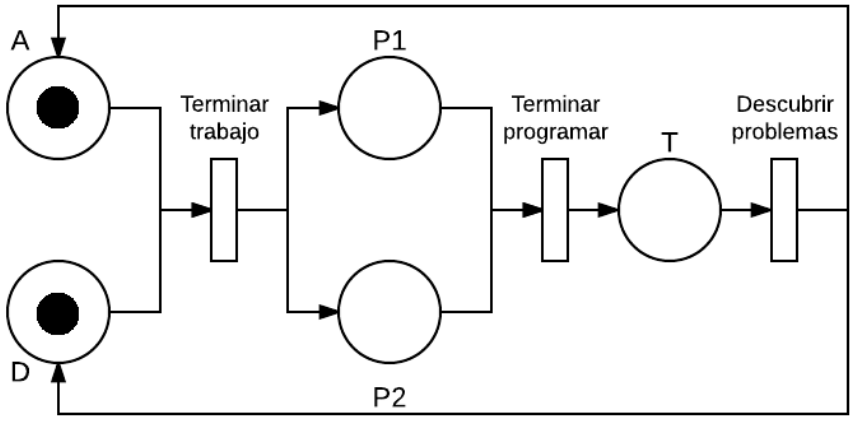


de flujo.png

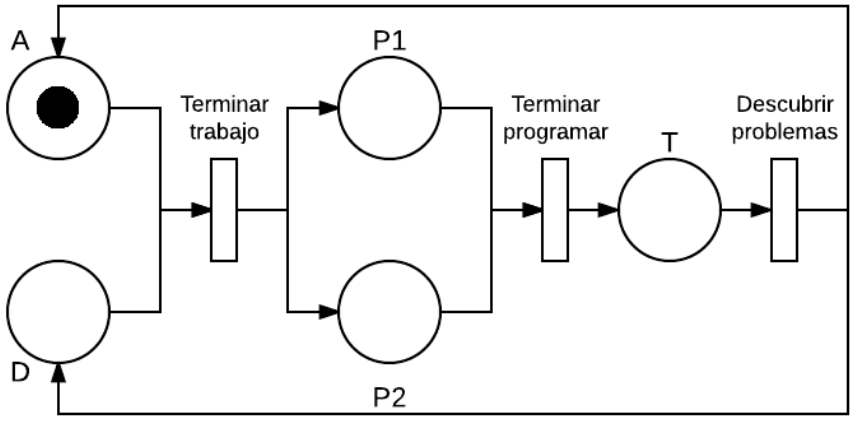


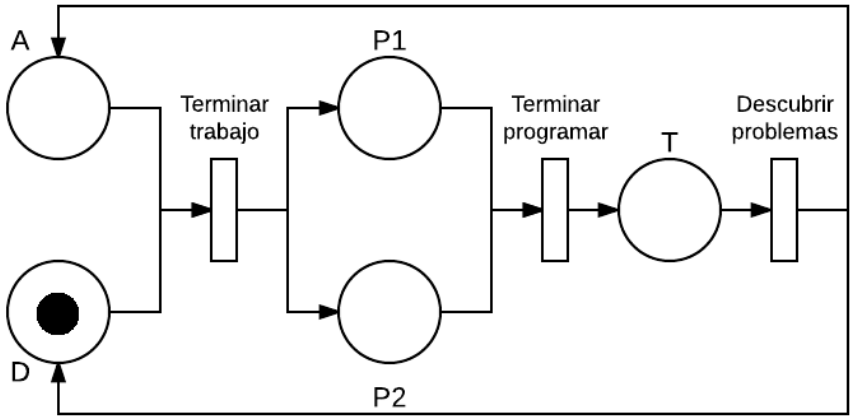
Petri.png

Revisando el proceso, el primer paso es poner a trabajar a A y D.

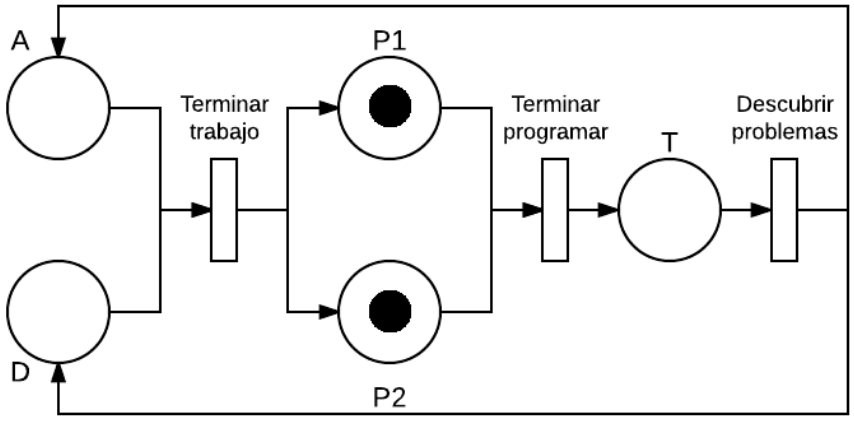


En esta sección pueden darse 2 casos. El primer caso es que A termine antes que D, lo que significa que, dadas las condiciones del problema, para que P1 y P2 comiencen a trabajar, estos deberán esperar, junto a A, a que D termine con su trabajo. El segundo caso es lo contrario, si D termina antes que A, para que P1 y P2 comiencen a trabajar, deberán esperar junto a D a que A termine de trabajar. Ambos casos se ven ilustrados en las dos imágenes siguientes:





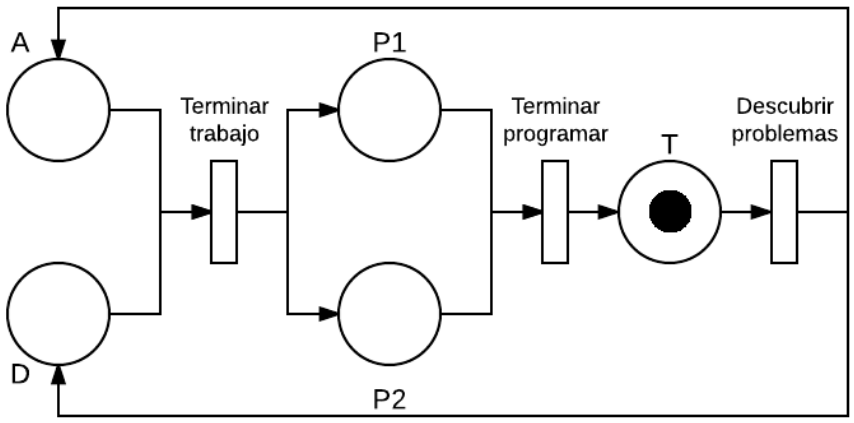
Una vez que A y B terminan con su trabajo, es el momento de que P1 y P2 comiencen con el suyo.



Como se dijo en un comienzo, supondremos que P1 y P2 trabajan al mismo tiempo y que, por ende, también terminarán de trabajar al mismo tiempo; lo que implica que T tendrá que esperar dicho momento para comenzar a trabajar.

Por otro lado, si nos ponemos en el caso de que nuestro supuesto es errado, y de que P1 y P2 no necesariamente trabajan al mismo tiempo, lo único que pasaría sería que existen 2 probabilidades, la primera es que P1 termine antes que P2, y la segunda es que P2 termina antes que P1; en cualquier caso, lo único que hay que hacer es esperar, tal como se mencionó en el caso de A y D.

Finalmente, diremos que T comienza a trabajar cuando P1 y P2 terminan con su trabajo, tal como se muestra en la siguiente imagen:



Como T siempre descubre problemas, desde T se vuelve a iterar, llevando el proyecto nuevamente a manos de A y D, y repitiendo todos los pasos anteriormente nombrados.

A continuación se mostrará el código realizado para el modelo propuesto; dicho código está programado en Python, y será presentado en 2 partes, para una mayor claridad visual.

Sección uno:

```
import threading
import time

globvar = 0

def analista():
    print "iniciando analista"
    time.sleep(2)
    print "terminando analista"
    return

def disenador():
    print "iniciando diseno"
    time.sleep(1)
    print "terminando diseno"
    return

def programador(tiempo):
    print "inicio
{}".format(threading.current_thread().getName())
    time.sleep(tiempo)
    print "termino
{}".format(threading.current_thread().getName())
    return
```

```
def tester(tiempo):
    global globvar
    print "inicio
{}".format(threading.current_thread().getName())
    datoinput = raw_input('Escriba "y" para aprobar en caso
contrario otra letra: ')
    if datoinput == 'y':
        globvar = 1
        print "proyecto completo!!"
    else:
        time.sleep(tiempo)
        print "termino
{}".format(threading.current_thread().getName())
```

De forma simple y abreviada, diremos que en esta sección se presentan los distintos hilos a utilizar: Analista, Diseñador, Programador, y Tester. Cada uno de dichos hilos posee una 'frase' que nos indica que ha comenzado a ejecutarse, un valor que representa la cantidad de tiempo que durará dicha ejecución, y otra frase que representa cuando el hilo se ha terminado de ejecutar.

Sección dos:

```
   while globvar == 0:

        a = threading.Thread(name="analista", target=analista,
args=())
        d = threading.Thread(name="diseno", target=disenador,
args=())

        p1 = threading.Thread(name="programador1",
target=programador, args=(3, ))
        p2 = threading.Thread(name="programador2",
target=programador, args=(2, ))

        t = threading.Thread(name="tester", target=tester,
args=(3, ))

        a.start()
        d.start()

        a.join()
        d.join()

        p1.start()
        p2.start()
        p1.join()
        p2.join()

        t.start()
        t.join()
```

En esta sección se comienza por crear los distintos hilos. A cada hilo se le otorga una letra como identificador (a, d, p1, p2, t), un nombre para el hilo (analista, diseno, programador1, programador2, tester), y el 'target' que indica al hilo cual 'función base', de la sección uno, debe utilizar para ejecutarse.

Cuando se llama a los distintos identificadores (letras) y se les aplica la función 'start', se procede a ejecutar el hilo correspondiente. Cuando se aplica la función 'join', de forma muy simplificada, podemos decir que se indica al sistema debe 'esperar' a que los hilos continuos terminen antes de seguir con el resto de la ejecución del programa.

Todo esto se ejecuta dentro de un while debido a las indefinidas iteraciones que debe realizar el equipo de desarrollo.

5. Conclusión

Como se mencionó anteriormente, una red de Petri es ideal para poder modelar sistemas, especialmente para aquellos que poseen propiedades recursivas y / o concurrente; dada la facilidad que éste modelo nos brinda para el entendimiento del sistema.

Por otro lado, diseñar una red de Petri no es tan sencillo como suena. Es necesario analizar bien el sistema que se ha de modelar, y adaptarse correctamente al uso de los 4 símbolos que nos permite utilizar. Desde un punto de vista global es necesario entender correctamente el funcionamiento del sistema y los distintos estados que éste pueda tomar, para poder acomodar de forma correcta las plazas, transiciones, y arcos que sean necesarios, y de esta manera poder definir bien los posibles caminos que podrá recorrer el token al momento de la 'simulación'.

Si todo lo anteriormente nombrado se realiza de forma detallada, definiendo correctamente las plazas, transiciones, arcos, y tokens; es posible usar las redes de Petri para modelar, de forma visualmente sencilla y legible, diversos tipos de sistemas, inclusive aquellos complejos que utilizan concurrencia y recursividad.

6. Bibliografía

- Documentación sobre Python: https: //docs.python.org/2/library/threading.html.
- Uso de las redes de Petri: https://www.uaeh.edu.mx/scige/boletin/icbi/n1/e4.html.