

## **Proyecto: Redes de Petri**

### **Resumen**

El siguiente informe describe brevemente las decisiones de diseño e implementación del proyecto 1 en la asignatura Redes y Sistemas Distribuidos usando el lenguaje de programación **go**, el mismo consiste en la implementación de una librería para la simulación de sistemas distribuidos usando redes de Petri.

### **Objetivos generales:**

1. Diseñar e implementar un simulador distribuido de Redes de Petri.
2. Implementar sincronización conservativa mediante el mecanismo de LEFs para redes de petri distribuidas.

### **Implementación**

Para la implementación de este proyecto se realizaron modificaciones al código aportado como base para el mismo. El proyecto está conformado por dos requerimientos principales:

1. El primer requerimiento consistió comprobar el funcionamiento de la librería para simulación centralizada de redes de petri, para lograrlo se estudió el código y las distintas funciones que conforman la librería para conocer su funcionamiento y utilidad. Por último se procedió a corregir los errores en el test y main para la ejecución exitosa de la librería.
2. Para la segunda parte se diseñó e implementó una simulación distribuida de redes de petri basadas en simulación conservativa(pesimista) la cual evita restricción de causalidad local, y avanza en tiempo solo cuando está seguro que no van a llegar eventos con tiempo menor, ya que los eventos son procesados en orden. Esto es posible dado que se trabaja con una red estática (no hay creación dinámica de procesos lógicos) y se usa TCP para garantizar fiabilidad y demás ventajas que ofrece el protocolo en el envío y recepción de mensajes entre las distintas subred. Cada proceso es responsable del envío de un mensaje que contienen una estampilla de tiempo con el tiempo mínimo aceptado en mensajes posteriores.

**Shamuel Manuel Manrique Aquino**  
**NIP: 802400**

Para la implementación se dividió la problemática en trabajos más simples, primero nos enfocamos en subdividir la red de petri y distribuirla en varias máquinas y segundo manejar la simulación de forma conservativa mediante el mecanismo de LEFs. Para llegar a las siguientes soluciones planteadas. A Continuación se muestran imágenes del pseudocódigo que representa la estructura principales de las funciones entre ellas algunas fueron modificadas y otras creadas desde cero en el proyecto.

La primera de ellas es la función simular, encargada de llevar el recuento de la simulación con un tiempo inicial 0 y tiempo final n (número de veces a simular) introducido por el usuario. Esta función orquesta la simulación llamando a las funciones requeridas para llevarla a cabo la misma de forma distribuida.

```
BEGIN SIMULATE (#Ciclos a Simular)
  Init Time
  Update Sensitive
  FOR (init, endTime):
    PRINT PetryNet
    FIRST TIME sleep

    THERE EVENTS
      CHECK EVENT:
        REMOTE(negative):
          Negative(IDGlobal)
          Pre:
            SEND(msm, msm.To)
        LOCAL:
          LOCAL MANAGEMENT
    END THERE EVENTS

    UPDATE SENSITIVE

    NOT THERE Sensitive
    INIT WAIT AGENTS
    SEND MSM other subNet
    END WAIT AGENTS

    CHECK HAS ALL LOOKOUT(Sleep)

    NOT THERE EVENTS
    ADVANCE TIME
      1: CHECK EVENTS
      2: GET MIN_TIME_LOOKOUT
      COMPARE MIN AND NO NEGATIVE TIME
      RETURN
    END ADVANCE TIME
  END THERE
END FOR
CLOSE RECEIVE
END SIMULATE
```

## MESSAGE ORCHESTRATION

### USING FUNCTIONS SEND AND RECEIVE OF PREVIEW PROJECT

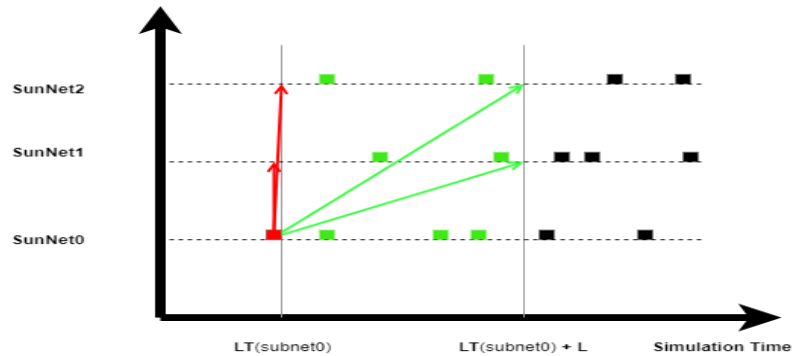
```
BEGIN RECEIVE(*SimulationEngineDist, connection)
  INIT CONNECTION(CONNECT.port)
  CHECK RECEIVE:
    MESSAGE:
      CALL SimulationEngineDist.TreatMessage(msm)
    DEFAULT:
      Error
  END RECEIVE

BEGIN TREATMESSAGE(msm)
  GET MSM.pack
  SWITCH (pack):
    EVENT:
      UPDATE IDGlobal
      ADD Event
    TIME:
      UPDATE LOOKOUT
    ID(NULL):
      TIME ESTIMATE TODO SOMETHING
      SEND(msm, msm.GetFrom)
  END SWITCH
END TREATMESSAGE
```

Las funciones mostradas son una versión mejorada de las funciones implementadas para la práctica 1 de la asignatura. El objetivo de esta función es realizar una serie de acciones dado un determinado tipo de paquete. Entre las acciones a ejecutar se encuentran:

1. Si es un evento lo añade a la lista de eventos y trata el mismo.
2. Si es una estampilla de tiempo actualiza el diccionario de lookahead.
3. Si es un mensaje null, calcula el lookahead y lo envía.

El código también cuenta con otras funciones de peso que fueron modificadas para su funcionamiento en sistemas distribuidos. Como por ejemplo: Esperar agente encargado de enviar msm null solicitando lookahead esto ocurre en el caso inicial en el que la transición está activada o no tiene acciones. Por ende para garantizar que se mantiene el principio conservativo y que no llegara una estampilla de tiempo menor, luego de haber procesado un evento. Al recibir un mensaje null la red procede a calcular su lookahead y lo envía como respuesta.



Los mensajes que contienen como paquete lookahead son una especie de garantía que permite a cada una de las subredes ejecutar avances significativos en sus eventos internos de forma segura. Una vez recibido los lookahead y conociendo el estado de las demás subredes, el que está activo procede a disparar los eventos, si el evento afecta una transición que no pertenece a su red entonces busca en su post el ip y envía el evento.

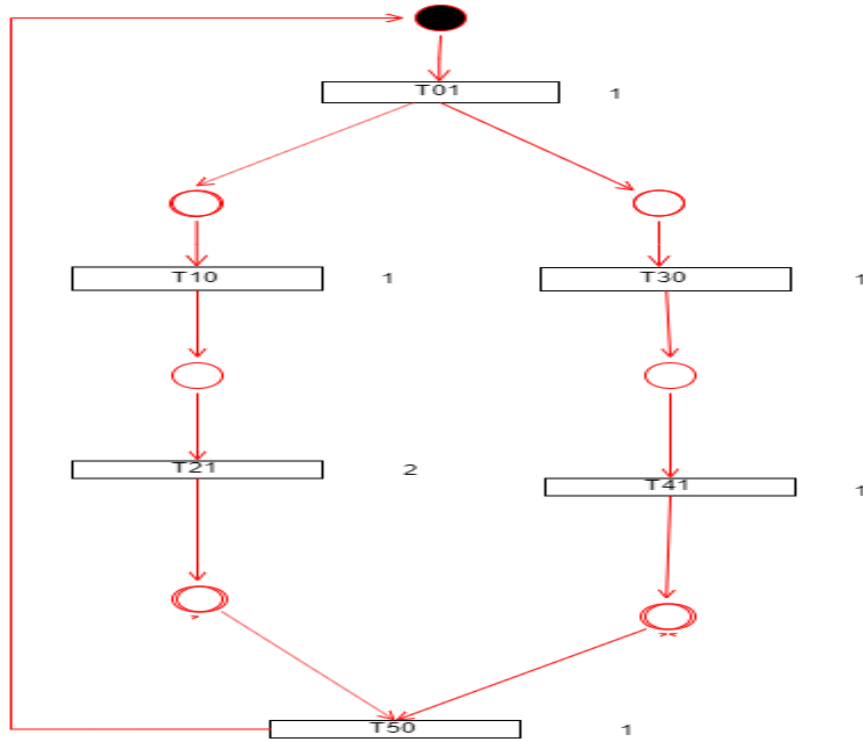
Otro punto importante es la recepción de todos los lookahead para garantizar una secuencia ordenada con respecto al tiempo lógico y avanzar en el tiempo. El cálculo del tiempo local se lleva a cabo sumando al tiempo actual con el tiempo de disparo, mientras que para calcular el tiempo global se procede a tomar la estampilla de menor tiempo de los lookahead en caso que no tengas eventos por procesar. El programa termina una vez se haya alcanzado el número de simulaciones, imprimiendo el número de transiciones disparadas y su respectivos tiempos.

A Continuación una representación de la inicialización de una subred.

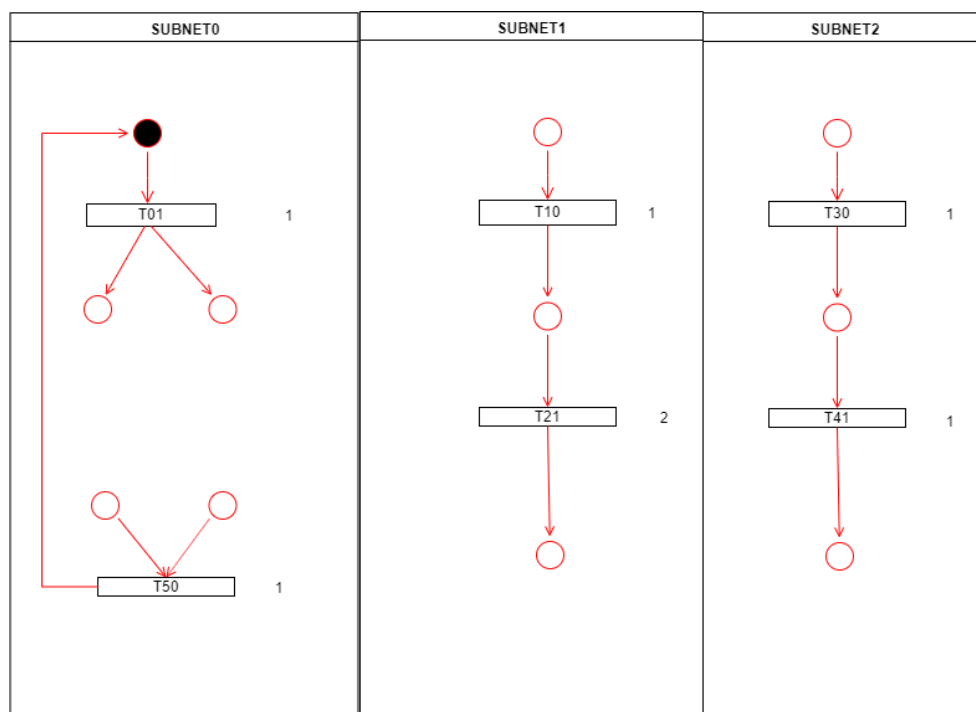
```
NET netNumber
LEFSDIST:
  SUBNET connection(IP, IDSUBNET)
  t1: TransitionDist
    IDGlobal: #IDGlobal
    IDGlobal: #IDLocal
    IiValorLef: Value
    IiShotDuration: T
    IiListactes:
      TransitionConstant #Const,
      TransitionConstant #Const,
  t2: {...}
  .
  .
  tn: {...}
  PRE:
    #IDGlobal: connection(IP, IDSUBNET)
    #IDGlobal2: connection(IP, IDSUBNET)
  POST:
    #IDGlobal: connection(IP, IDSUBNET)
  END SUBNET
END LEFSDIST
END NET
```

**Shamuel Manuel Manrique Aquino**  
**NIP: 802400**

A continuación se desarrollará un pequeño ejemplo que muestra la evolución del contenido de la estructura de datos durante el proceso de simulación. Tomando como caso base el ejemplo planteado en el enunciado del proyecto. Supongamos que un simulador distribuido conservativo procede de la siguiente Red de Petri.



A partir de esta Red de Petri se define 3 subredes quedando de la siguiente manera la distribución.



Cada una de las subredes se simulará en tres máquinas distintas con un mecanismo de comunicación usando paxos.

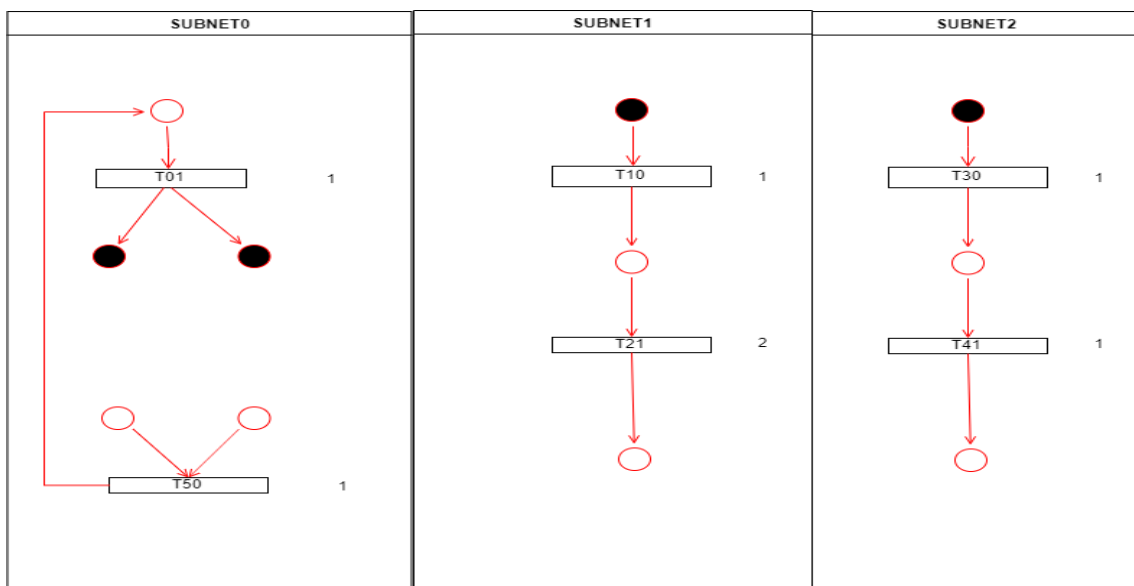
A Continuación la tabla de marcas dividida por subred:

SubNet		SubNet		SubNet	
Transición	Marcas Requeridas	Transición	Marcas Requeridas	Transición	Marcas Requeridas
T01	0	T10	1	T30	1
T50	2	T21	1	T41	1

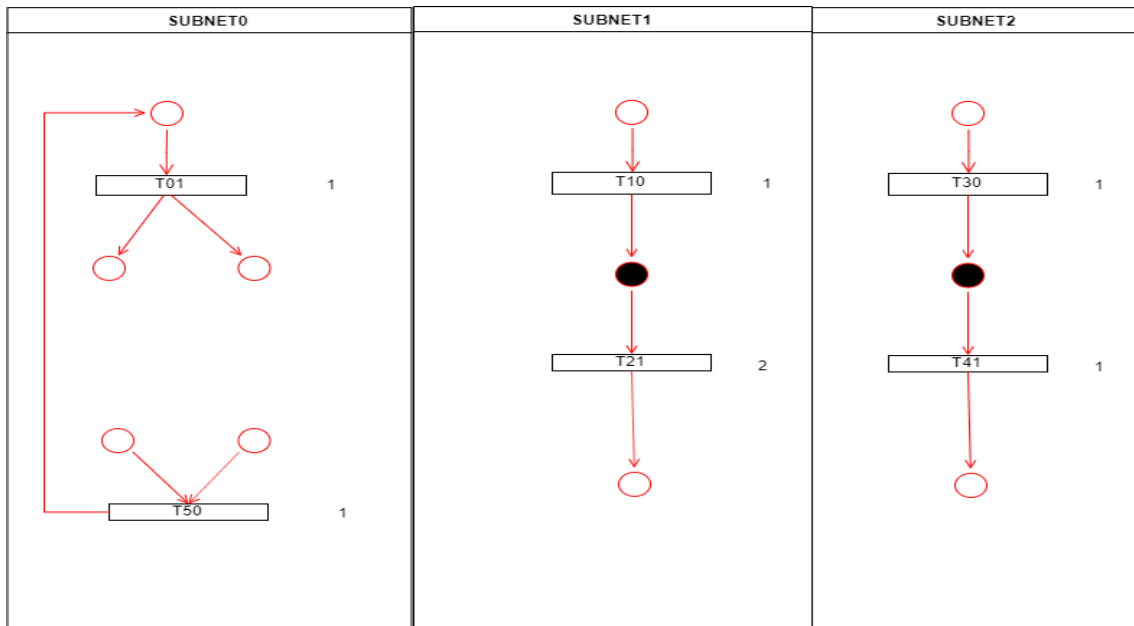
Las tabla de constantes de cada una de las transiciones clasificadas por red se describen en la tabla a continuación.

Red	Transiciones	Marcas Requeridas	Constantes a Enviar (T0, T1, T2, T3, T4, T5)						Pre	Post
SubNet 0	T0	$1 - m(p_0)$	$1(T)$	$-1(T+1)$		$-1(T+1)$			Subnet 1	Subnet 1
	T5	$2 - m(p_3) - m(p_6)$	$-1(T+1)$					$2(T)$	Subnet 2	Subnet 2
SubNet 1	T1	$1 - m(p_1)$		$1(T)$	$-1(T+1)$				Subnet 0	Subnet 0
	T2	$1 - m(p_2)$			$1(T)$			$-1(T+1)$		
SubNet 2	T3	$1 - m(p_4)$				$1(T)$	$-1(T+1)$		Subnet 0	Subnet 0
	T4	$1 - m(p_5)$					$1(T)$	$-1(T+1)$		

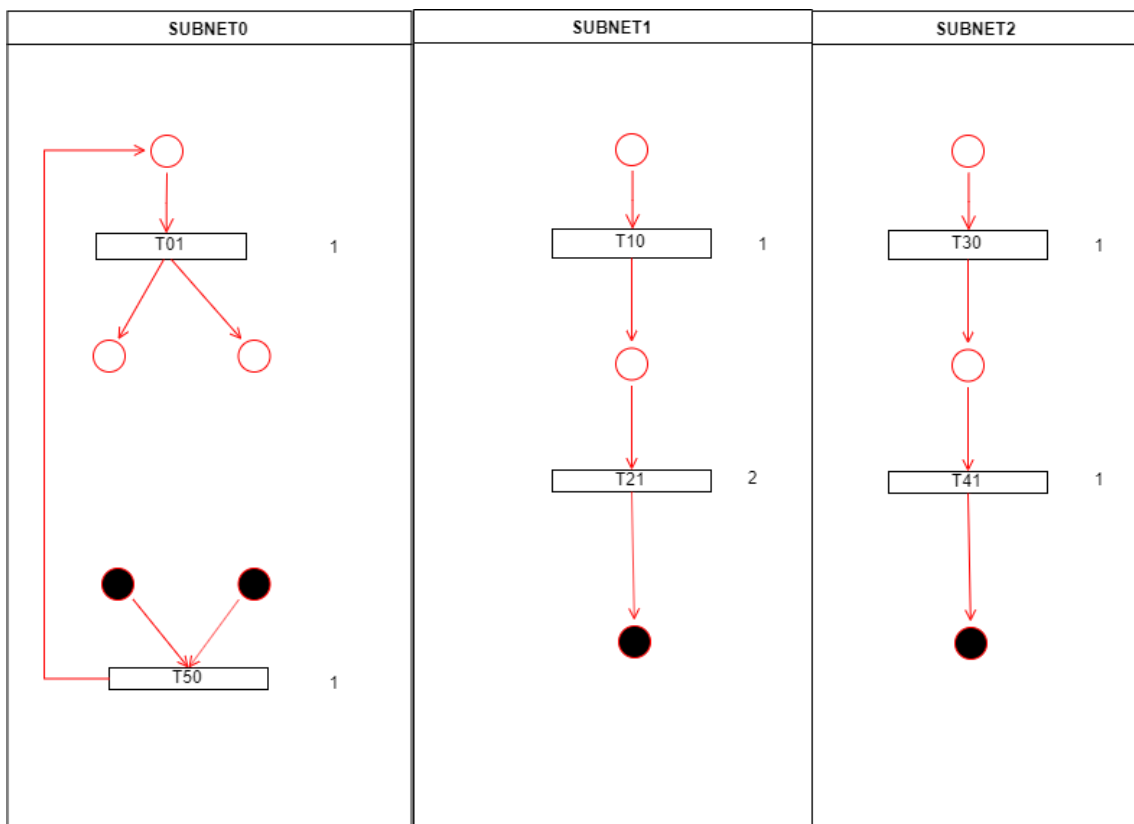
La simulación inicia en tiempo cero y procede a ejecutar todos los pasos mencionados anteriormente como se muestra en la primera imagen de las transiciones. Una vez disparada la primera transición T01 se genera una serie de eventos que son enviados a sus respectivas subredes y el tiempo es actualizado sumándole el tiempo de disparo en este caso 1. En paralelo la subred Subnet 1 y Subnet 2 ejecutan sus eventos.



Para el tiempo  $T=2$  ambas transiciones T21 Y T41 se han disparado pero ocurre un evento interesante que la transición T50 quedará bloqueada hasta que llegue el token de la transición T21 en el tiempo  $T=4$



La transición T50 estará lista para ser disparada en tiempo  $T=4$  y habilitará nuevamente a T01 en el tiempo  $T=5$ .





Para este proyecto se realizó una serie de pruebas incrementando el número de transiciones por subred y el número de subredes. Se tuvo que adaptar los tiempos de espera en la recepción de los lookahead dado que cuando se ejecutaban en las máquinas del laboratorio no se obtuvieron las salidas correspondientes. Teniendo en cuenta que se repitió este mismo ejemplo y uno nuevo agregando dos subredes para hacer un total de 5.