Shamuel Manuel Manrrique Aquino
NIP: 802400

Proyecto:

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 redes de petri en sistemas distribuidas.

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 mediante la comunicación de envío y recepción de mensajes entre las distintas subred. Cada proceso es responsable del envío de un mensaje que contienen una estampilla de tiempo que el mínimo tiempo aceptado en mensajes posteriores.

Shamuel Manuel Manrrique Aquino
NIP: 802400

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

La primera de ellas es la función simular, se 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 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
```

Shamuel Manuel Manrrique Aquino
NIP: 802400

MESSAGE ORCHESTRATION

USING FUNCTIONS SEND AND RECEIVE OF PREVIEW PROJECT

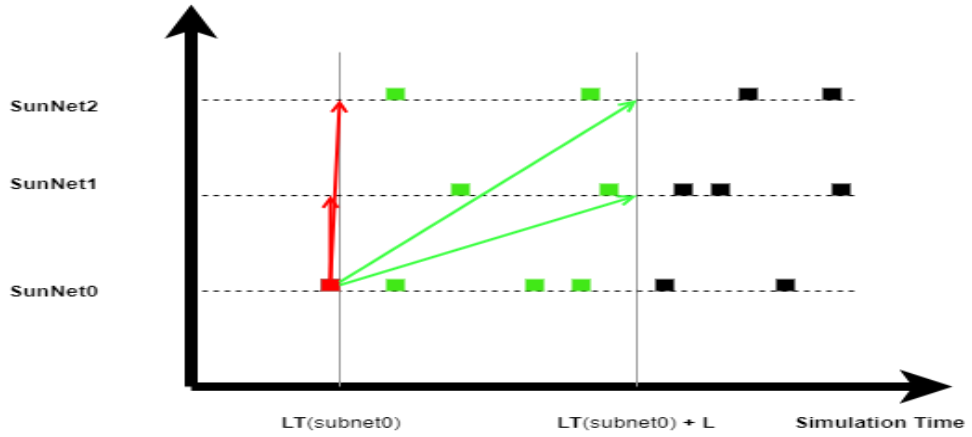
```
BEGING RECEIVE(*SimulationEngineDist, connection)
    INIT CONNECTION(CONNECT.port)
    CHECK RECEIVE:
        MESSAGE:
            CALL SimulationEngineDist.TreatMenssage(msm)
        DEFAULT:
            Error
END RECEIVE

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

Las siguientes funciones es llamada del received implementado para la práctica número uno 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 evento y trata el mismo.
2. Si es un estampilla de tiempo actualiza el diccionario de lookahead.
3. Si es un mensaje null, calcula el lookahead y lo envía.

El codigo tambien cuenta con otras funciones de peso que fueron modificadas para su funcionamiento en sistemas distribuidos. Como por ejemplo: Esperar agente de encarga 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 conservativos y garantizar que no me llegara una estampilla de tiempo menor, luego de haber procesado un evento. Al recibir un mensaje null la red procede a calcular su look ahead y lo envía como respuesta.

Shamuel Manuel Manrrique Aquino
NIP: 802400



Los mensaje que contienen como paquete look ahead son una especie de garantía que permite a cada una de las subred 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.

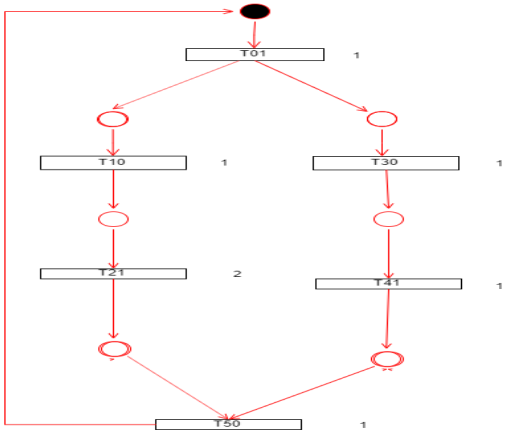
Otro punto importante es la recepción de todos los lookahead para garantizar una secuencia ordenada con respecto al tiempo lógico y avanzar en el tiempo. El cálculo del tiempo local se lleva a cabo sumando al tiempo actual el tiempo de disparo, por otro lado para calcular el tiempo global se procede 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 solicitado imprimiendo el número de transiciones disparada y su respectivo tiempo.

A Continuación una representación de la inicialización de una subred.

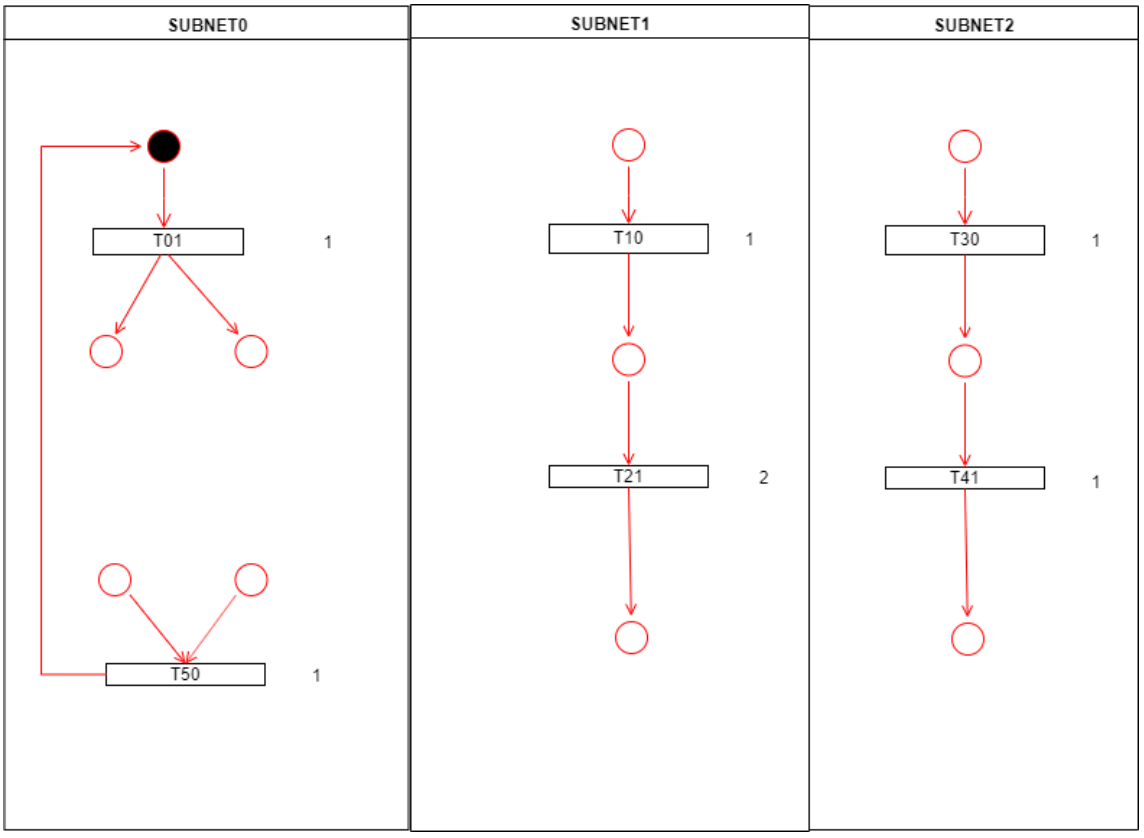
```
NET netNumber
    LEFSDIST:
        SUBNET connection(IP, IDSUBNET)
            t1: TransitionDist
                IDGlobal:       #IDGlobal
                IDGlobal:       #IDLocal
                IiValorLef:     Value
                IiShotDuration: T
                IiListactes:
                    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 Manrrique Aquino
NIP: 802400

A continuación se desarrollara un pequeño ejemplo que muestra la evolución del contenido de la estructura de datos durante el proceso de simulación. Tomando como caso base el planteado en el enunciado del proyecto y ay con el caso base del proyecto, Supongamos que un simulador distribuido conservativo que 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.



Shamuel Manuel Manrrique Aquino
NIP: 802400

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

A Continuación la tabla de marcas dividida por subred:

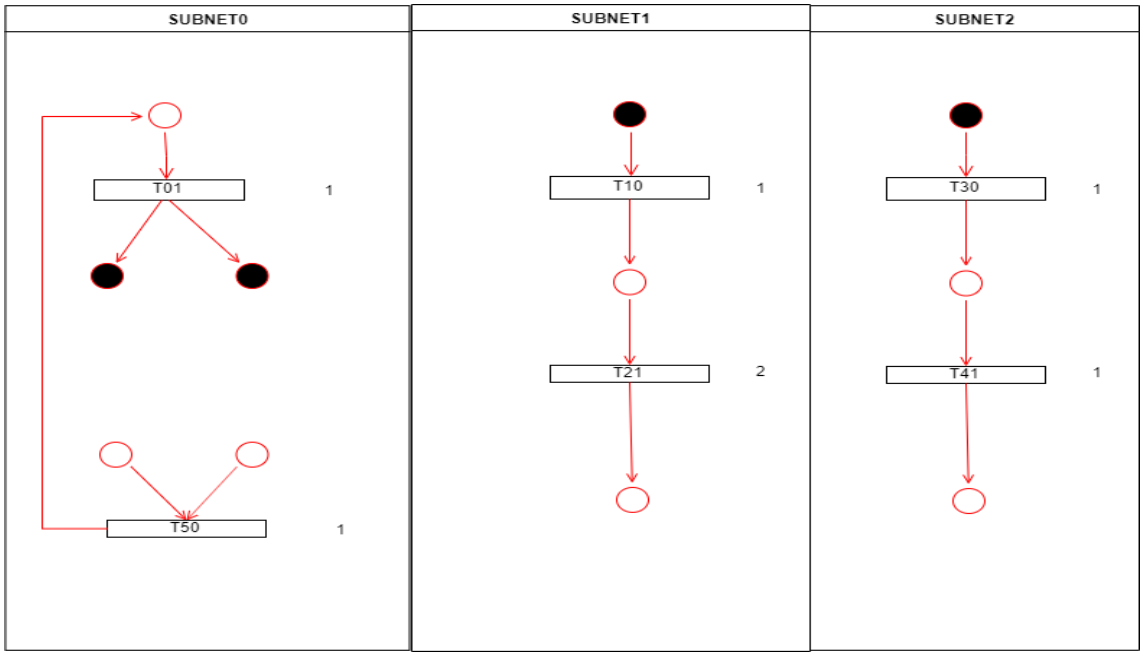
SubNet		SubNet		SubNet	
Transición	Marcas Requeridas	Tiempo	Marcas Requeridas	Tiempo	Marcas Requeridas
T00	0	T10	1	T30	1
T91	2	T21	1	T41	1

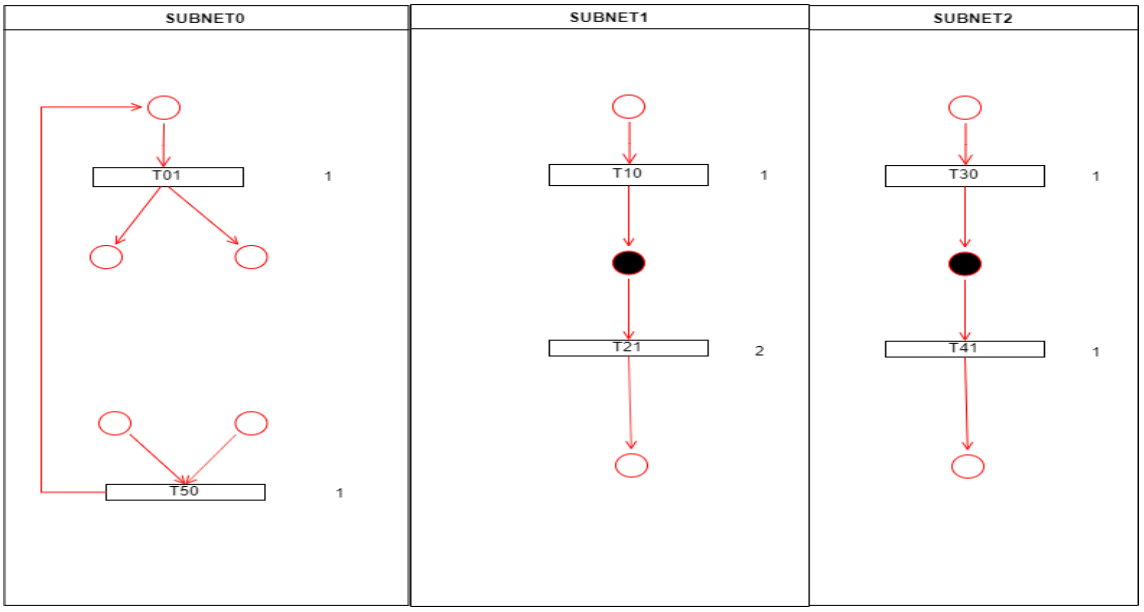
Las tabla de constantes de cada una de las transiciones clasificadas por red se describen en la tabla a continuación.

Red	Transiciones	Marcas Requeridas	Constantes a Enviar (T0, T1, T2, T3, T4, T5)						Pre	Post
SubNet 0	T0	1 – m(p0)	1(T)	-1(T+1)		-1(T+1)			Subnet 1	Subnet 1
	T5	2 – m(p3) – m(p6)	-1(T+1)					2(T)	Subnet 2	Subnet 2
SubNet 1	T1	1 – m(p1)		1(T)	-1(T+1)				Subnet 0	Subnet 0
	T2	1 – m(p2)			1(T)			-1(T+1)		
SubNet 2	T3	1 – m(p4)				1(T)	-1(T+1)		Subnet 0	Subnet 0
	T4	1 – m(p5)					1(T)	-1(T+1)		

Shamuel Manuel Manrrique Aquino
NIP: 802400

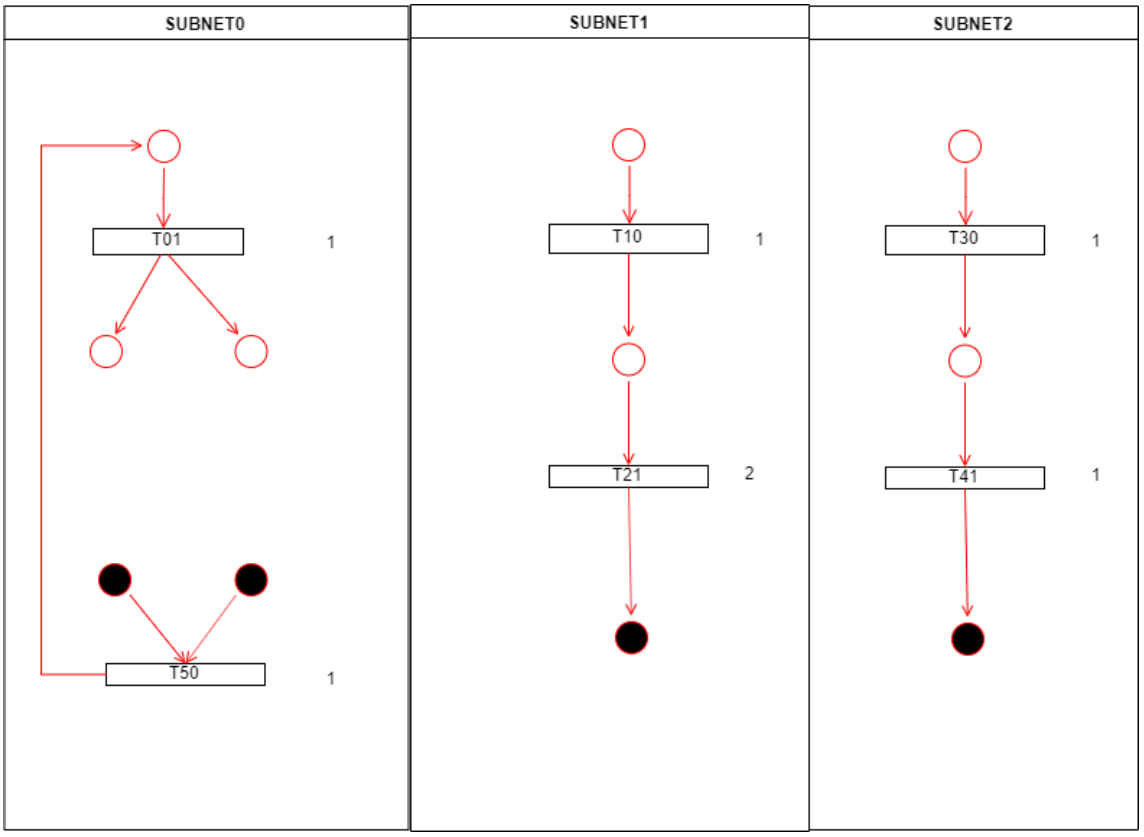
La simulación inicia en tiempo cero y procede a ejecutar todos los pasos mencionados anteriormente como se muestra en la primera imagen de las transiciones. Una vez disparada la primera transición T01 y se generó una serie de eventos que son enviados a sus respectivas subredes y el tiempo es actualizado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





Shamuel Manuel Manrrique Aquino
NIP: 802400

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 dispara en tiempo T=4 y habilitará nuevamente a T01 en el tiempo T=5.

Para este proyecto se realizó una serie de pruebas que 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 ejecutaban las salidas correspondientes.