UNIVERSIDAD DE ZARAGOZA
ESCUELA DE INGENIERÍA Y ARQUITECTURA

Práctica 2 - Simulador Distribuido de Redes de Petri

Nombre estudiante: *Alberto Mur López*

Curso: *Redes y sistemas distribuidos*
Docente: *Unai Arronategui*
Fecha de entrega: *January 9, 2024*

INTRODUCCIÓN

El objetivo de esta práctica es desarrollar un simulador de redes de Petri distribuido. Para ello deberemos hacer uso de los algoritmos planteados por Chandy, Misra y Bryant que definieron el problema de la sincronización y una solución. La misión de la simulación paralela de eventos discretos (PDES) es acelerar la ejecución de un modelo aprovechando características de concurrencia. Pero estoy viene acompañados de una seria de retos y problemas que hay abordar.

Partiendo de un simulador centralizado donde se ejecuta una red de Petri utilizando **Linear Enabling Function** (LEF). Aquí no vamos a entrar en detalle de cómo funciona la ejecución de este modelo de ejecución de redes de Petri. Pero si nos quedaremos con que es un modelo de ejecución temporal que tiene las siguientes características:

- Tiene un reloj que avanza
- Genera eventos que pueden sensibilizar transiciones
- Dispara transiciones sensibilizadas que pueden generar eventos

En el momento que introducimos el concepto de reloj o tiempo en un sistema distribuido, es muy seguro que nos encontremos ante un problema de sincronización tarde o temprano. La distribución de una red de Petri según el modelo de LEF haciendo que haya lugares compartidos entre los distintos procesos lógicos. Un ejemplo lo tenemos en la la Figura 1. En dicha figura hay dos subredes que comparten lugares. Cuando la transición p0.1 de la subred0 genere un evento, este debe ser enviado a la subred1.

Existen diversas técnicas de simulación distribuida de eventos discretos. En esta práctica implementaremos un simulador con sincronización conservativa el cuál esta publicado en este repositorio de GitHub[1].

[1] `https://github.com/mursisoy/distributed-petri-net-simulator`

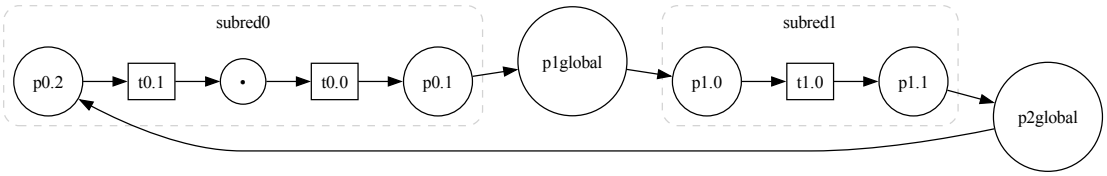

Figure 1: Red de petri distribuida en 2 subredes

SIMULADOR CON SINCRONIZACIÓN CONSERVATIVA

La sincronización con simulación conservativa, se basa principalmente en que no se traten eventos fuera de orden. Esto es, el reloj de la simulación no avanza hasta que no se pueda garantizar que no se van a recibir eventos anteriores o iguales al reloj actual por parte de otros procesos lógicos.

Para garantizar la ausencia de *deadlocks*, se utiliza el envió de mensajes *Null* y el *lookahead*. Estos mensajes se envían para indicar el límite inferior de tiempo de la simulación de cada proceso lógico y es la garantía de que dicho proceso lógico no va enviar eventos con tiempo a anterior a dicha marca.

Hemos una implementación en Golang. Donde se han creado dos ejecutables, `dsim-launcher` y`dsim-node`. El ejecutable `dsim-launcher` se encarga de leer los archivos con la definición de la red de Petri y un listado de nodos. Y a continuación mediante `ssh` lanza tantos procesos `dsim-node` como subredes haya, siempre y cuando el número de nodos sea suficiente.

Para que el algoritmo pueda funcionar correctamente, para cada proceso lógico es necesario saber:

- De que otros procesos lógicos va a recibir eventos
- Los procesos lógicos a los que debe enviar eventos

La simulación se realiza hasta un tiempo de ciclo final. Para cada paso de la simulación

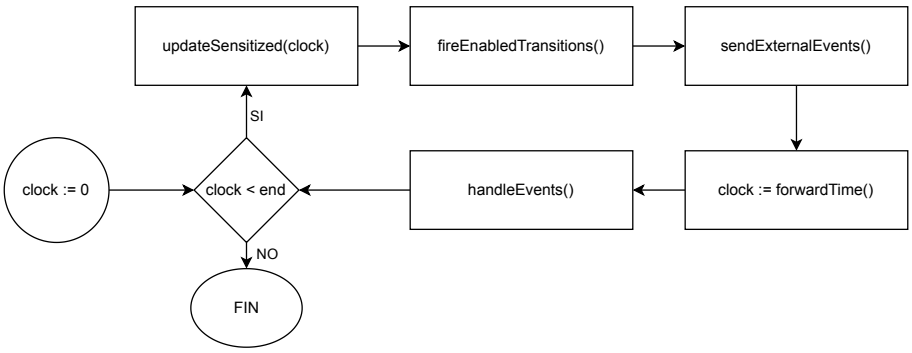

Figure 2: Flujo de procesamiento de eventos en la simulación para cada proceso lógico

En el momento de enviar eventos externos, si no se ha generado algún evento para alguno de los procesos lógicos a los que se les envía eventos, se les envía un mensaje

Null con el reloj local más el *lookahead*.

En el punto de avanzar el reloj de la simulación. Es donde tiene lugar la ejecución del algoritmo. El motor mantiene una estructura de datos para cada proceso lógico del que tiene que recibir datos donde guarda el último valor de *lookahead* conocido y una cola de mensajes con eventos entrantes. Se coge el menor de todos los valores entre el reloj actual, eventos locales y valores de *lookahead* recibidos. Para cada proceso lógico cuyo valor coincida con ese mínimo, si la lista de eventos esta vacía, la ejecución se bloquea hasta recibir un evento de ese proceso o actualizar su valor de *lookahead*.

Para visualizar una ejecución de la simulación en *ShiViz*, seguir las instrucciones del anexo A.

1 Conclusión

En esta práctica hemos profundizado en el campo de la simulación distribuida de eventos discretos. Se ha podido comprobar de primera mano la complejidad de estos sistemas y hemos implementado una solución a algunos de los problemas que aparecen cuando se trabajan con sistemas distribuidos. No obstante, además de esta técnica de simulación con sincronización conservativa existen otras técnicas, con mayor complejidad, como la simulación con sincronización optimista donde se pueden procesar mensajes fuera de tiempo.

A Visulización de ShiViz

Shiviz es una herramienta muy útil para la depuración, visualización y comprensión de sistemas distribuidos. Utilizando relojes lógicos vectoriales, ofrece una visualización de los eventos que ocurren entre los diferentes procesos lógicos.

Para visualizar el progreso de esta simulación ir a ShiViz[2] y cargar el archivo disponible en este repositorio de GitHub[3]

[2] `https://bestchai.bitbucket.io/shiviz/`

[3] `https://github.com/mursisoy/distributed-petri-net-simulator/blob/main/report/shiviz.log`