Universidad Nacional de Córdoba

Informe Trabajo Final

Ingeniería de Software

Aplicación de las técnicas de Ingeniería de Software sobre un proyecto inconcluso.

Autores:
Crístian Gutierrez (35.104.714)
Esteban Morales (35.104.714)
Fabiola Campos (35.258.183)
Gianfranco Barbiani (33.414.224)

Supervisor:
Ing. Martín Miceli

19 de junio de 2014

Índice

1. Introducción

En el presente informe se expone el desarrollo del trabajo final para la cátedra Programación Concurrente que se cursa en el primer cuatrimestre del cuarto año de la carrera Ingeniería en Computación en la Facultad de Ciencias Exáctas Físicas y Naturales de la Universidad Nacional de Córdoba, Argentina. Este trabajo tiene por consigna la simulación de una celda flexible de producción mediante el modelado por redes de petri y la implementación concurrente programada en Java. Tal implementación incluye la programación de los procesos productivos que ejecutan sus tareas de forma paralela y el diseño de un monitor para garantizar la sincronización de la adquisición y devolución de los recursos compartidos.

2. Consigna

Sea una planta industrial donde se intenta producir tres tipos de piezas distintas mediante el uso de una celda de producción flexible que incluye tres robots, cuatro máquinas, tres almacenes de materiales y tres almacenes de piezas producidas. Debe tener en cuenta las siguientes reglas de producción para cada pieza:

- Pieza 1: Para producir una unidad de pieza (1), una unidad de material del almacen I_1 debe someterse a los porcesos de la máquina M_1 y la máquina M_2. Finalmente se colocan las piezas producidas en el almacen = O_1.

$$I_1 \rightarrow M_1 \rightarrow M_2 \rightarrow O_1$$

- Pieza 2: Existen dos reglas para producir una unidad de pieza (2), una unidad de material del almacen I_2 debe someterse a los porcesos de la máquina M_4 ,la máquina M_3 y la máquina M_1 ó bien pasar por las máquinas M_2, M_3 y M_1. Finalmente se colocan las piezas producidas en el almacen = O_2.

$$I_2 \rightarrow M_4 \rightarrow M_3 \rightarrow M_1 \rightarrow O_2$$

$$I_2 \rightarrow M_2 \rightarrow M_3 \rightarrow M_1 \rightarrow O_2$$

- Pieza 3:Para producir una unidad de pieza (1), una unidad de material del almacen I_3 debe someterse a los porcesos de la máquina M_3 y la máquina M_4. Finalmente se colocan las piezas producidas en el almacen = O_3.

$$I_3 \rightarrow M_3 \rightarrow M_4 \rightarrow O_3$$

Los encargados de transportar los materiales, productos intermedios y finales son los robots, en este caso se cuenta con tres y cuyas responsabilidades se detallan a continuación:

- Robot 1 (R1):$[I_1, M_1, M_3, O_2]$
  - Carga la máquina M_1.
  - Libera la máquina M_3.
  - Quita materiales del almacen I_1.
  - Deposita las piezas (2) en el almacen O_2.
- Robot 2 (R2):$[I_3, M_1, M_2, M_3, M_4]$
  - Libera la máquina M_1.
  - Libera la máquina M_2.
  - Libera la máquina M_4.
  - Carga la máquina M_3.
  - Carga la máquina M_4.
  - Quita materiales del almacen I_3.
- Robot 3 (R3):$[M_2, M_4, 0_1, 0_3]$
  - Carga la máquina M_2.
  - Carga la máquina M_4.
  - Libera la máquina M_2.
  - Libera la máquina M_4.
  - Deposita las piezas (1) en el almacen O_1.
  - Deposita las piezas (3) en el almacen O_3.

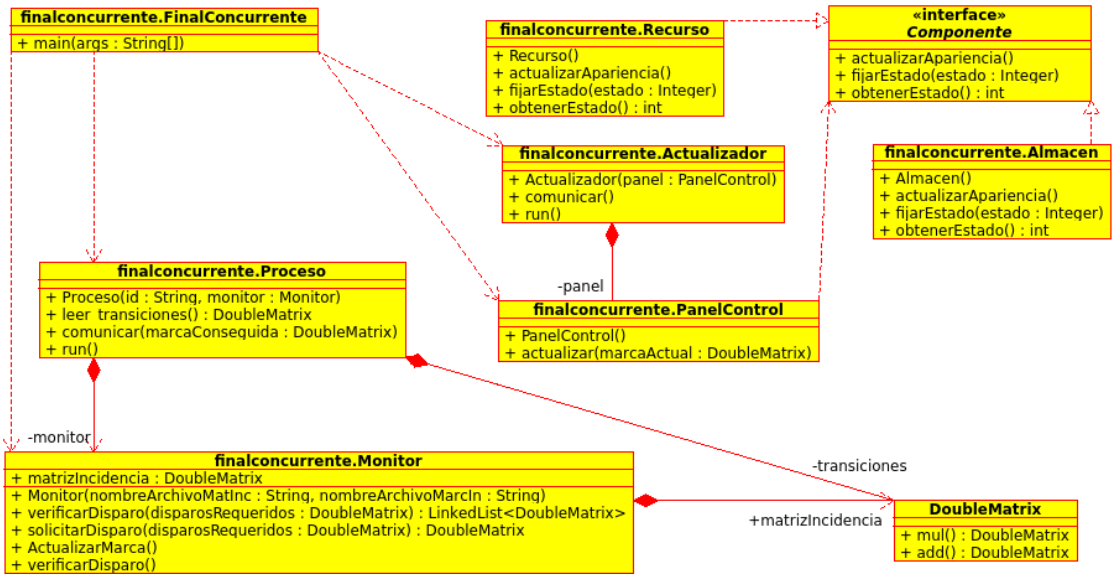
Se pide como consigna modelar el sistema con una red de petri, luego debe programarse la solución y emplearse un monitor para la sincronización de los procesos concurrentes.

3. Clases

Con el objetivo de modularizar el código y procurar el encapsulamiento de cada parte, se decidió la implementación de las siguientes clases:

- Actualizador.java
- Componente.java
- FinalConcurrente.java
- Monitor.java
- PanelControl.java
- Proceso.java
- VentanaPanel.java

Se muestra a continuación el diagrama de clases y luego se desarrollará una breve descripción de las clases más importantes y se detallan los métodos más relevantes.



3.1. Actualizador

Es el hilo encargado de actualizar los valores de la pantalla, cada vez que se ejecuta un disparo. Recibe la nueva marca del sistema enviada por el proceso que realizo el disparo, a través de la implementación de Sockets. El actualizador ejecuta el método actualizar de la clase PanelControl con esta nueva marca.

3.2. Monitor

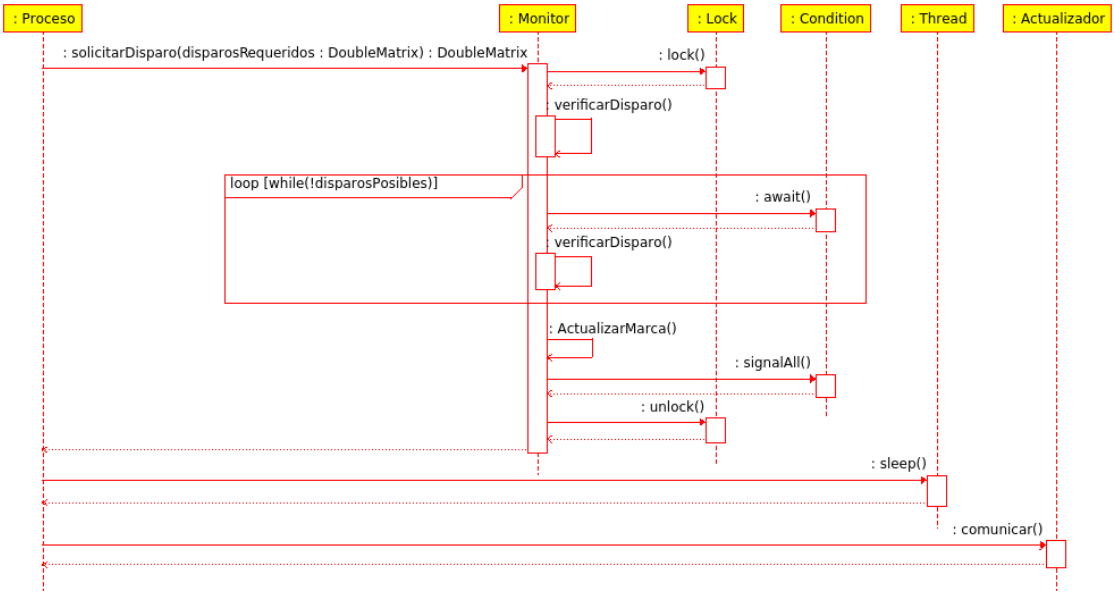
Este objeto es el que se encarga de administrar los recursos del sistema, y determinando los disparos que sean posibles realizar para cada proceso. Al inicializarse cargara la configuración del sistema desde un archivo, que contiene la matriz de incidencia. Y desde otro archivo leerá la marca inicial del sistema. Los métodos principales del monitor son verificarDisparo y solicitarDisparo:

- SolicitarDisparo():
  Este método es llamado por los procesos, se hace uso de los métodos lock() y unlock() para garantizar la exclusión mutua. Dentro de este método se llama a verificarDisparo(), que devolverá una lista con todos los disparos que son posibles de efectuarse en ese instante, de entre todos los permitidos para el proceso. En caso de no haber disparos posibles, el proceso se duerme en la cola de condición. Si hay disparos posibles, se determina uno al azar, y se dispara actualizando la marca. Luego se despierta a los procesos que estén dormidos en la cola de condición, y por ultimo se libera la exclusión mutua con un unlock(). Se retorna al proceso con una lista de dos elementos, el disparo ejecutado y la nueva marca.

- VerificarDisparo():
  Este método es llamado desde solicitarDisparo(), recibe un vector que indica las transiciones permitidas para el proceso. Determina cuales son factibles de realizarse, sin que se produzca algún problema de concurrencia (Interbloqueo, inanición). Y retorna una lista con todos los disparos que son posibles de realizarse.

3.3. Proceso

Se crearan tres hilos que corresponderán a cada linea de producción. Al inicializarce leerán las transiciones que pueden ejecutar desde un archivo. Durante su ejecución solicitaran disparos al monitor, y una vez que el monitor les da los recursos ejecutan el método comunicar(), para que se actualice la interfaz gráfica. A continuación se muestra el diagrama de sequencia de ejecución del proceso.



4. Validación del Sistema y Pruebas

Para validar el sistema se usa el algoritmo iterativo de análisi de Sifones. Para ello se calculan en el simulador todos los sifones posibles. De todo el conjunto deben discriminarse otros subconjuntos:

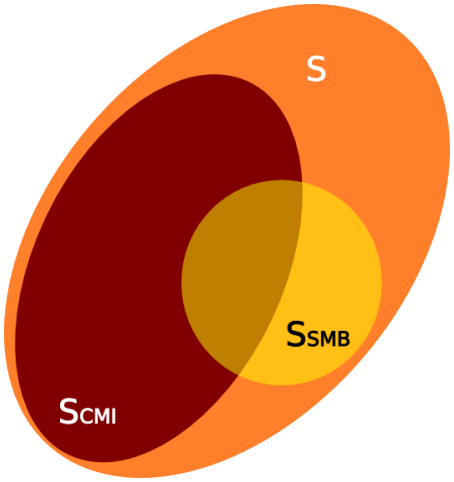
- Conjunto de Sifones que tienen marca inicial distinta de cero.

$$S_{CMI} = \{\forall s \in S/t \equiv 0 \wedge s \neq 0\}$$

- Conjunto de Sifones que tienen marca igual a cero al momento del bloqueo.

$$S_{SMB} = \{\forall s \in S/t \equiv BLOQ \wedge s \equiv 0\}$$

La intersección de ambos conjuntos (Sifones que tenían marca al incio de la simulación y están vacíos al momento del bloqueo) son los sifones que deben controlarse o limitarse para poder asegurar la usencia de bloqueos.
En el caso de nuestra implementación puede observarse una plaza adicional con marca inicial 3 (tres) que agregamos con este propósito. Así como comentamos en la sección de creación de la red de petri, puede interpretarse como un control necesario para el funcionamiento sin bloqueos.



5. Conclusión

El desarrollo del presente trabajo práctico muestra una forma de implementación de una celda de producción flexible a partir del modelo en redes de petri y sincronización de los recursos mediante el uso de un monitor. Implementar la consigna nos permitió estudiar una alternativa de rápido desarrollo para la construcción de simulaciones. El uso de un vector de marcas sustituye y encapsula la gestión de los semáforos o condiciones de una forma limpia y sencilla. El protocolo de disparo de la red se resume al cálculo de marcas posibles por lo que resultó de codificación trivial y segura.
Otra opción válida para evaluar la factibilidad del disparo y el cálculo de la nueva marca es reemplazar la multiplicación de la matriz de incidencia con el vector de disparo tan solo por el acceso a la colúmna de la matriz respecto del ínidice correspondiente a la transición de disparo.
Sin embargo, al implementar la ecuación de forma estrictamente matricial nos

permite, en una mejora futura, incluir el procesamiento de prioridades y tiempos específicos en los estados de la simulación.
Los resultados de las simulaciones son consistentes y coherentes. En una somera comparación la simulación de producción de 100 piezas de cada tipo lleva 1802 pasos de simulación mientras que en el simulador que se usó en clases necesita 1800, esto muestra que, en promedio, nuestra implementación alcanza los mismos resultados. Desde un enfoque industrial este mismo software podría utilizarse como controlador de los tiempos de los procesos y como monitor de permisos para regular el comportamiento de la celda.

Referencias

[1] Kristensen, Lars M. and Petrucci, Laure *Applications and Theory of Petri Nets* 32nd International Conference PETRI NETS 2011, Newcastle, UK, June 2011

[1] Naiqi Wu and MengChu Zhou *System Modeling and Control with Resource-Oriented Petri Nets* 6000 Broken Sound Parkway NW, Suite 300, 2010 *Process- vs. Resource- Oriented Petri Nets*, 6:71–81.