Generación de Código de Sistemas Concurrentes a partir de Redes de Petri Orientadas a Procesos

Orlando Micolini[#1], Member, IEEE, María F. Caro[#2], Ignacio Furey[#3], y Marcelo Cebollada[#4]

[#]Laboratorio de Arquitectura de Computadoras, FCEFyN-UNC, Córdoba, Argentina

[1]omicolini@compuar.com
[2]mflorenciacaro88@gmail.com
[3]fureynac@gmail.com
[4]mcebollada@gmail.com

Abstract. —Actualmente, se utilizan sistemas concurrentes que cuentan con múltiples procesadores y múltiples hilos y/o procesos ejecutándose simultáneamente. Las redes de Petri surgen como una manera gráfica, sencilla y con una sólida base formal matemática, para el modelado de dichos sistemas concurrentes.
El objetivo de este trabajo es generar el código de los procesos secuenciales de un sistema concurrente y paralelo, modelado por una red de Petri, la cual guía la ejecución secuencial de dichos procesos, y puede ser conducida por los procesadores de Petri. Esto permite resolver la concurrencia y el paralelismo del sistema con la red de Petri, y simplificar el diseño de los procesos secuenciales, desacoplando la parte secuencial de la parte paralela. Para esto, se desarrolló un framework con el cual se puede diseñar y generar el código de un sistema real concurrente y paralelo. El framework desarrollado se ha probado con distintos problemas modelados con redes de Petri orientadas a procesos (POPN), que son lo suficientemente generales y con características diferentes, que representan una amplia diversidad de escenarios; en este trabajo se presenta un caso para el cual también se obtuvieron programas con mejores desempeño que con semáforos, como se muestra a continuación.

Keywords: —Framework, Multicore, Procesador de Petri Redes de Petri, Sistemas reactivos.

1 INTRODUCCIÓN

Actualmente, la ejecución paralela es necesaria en una gran cantidad de sistemas, ya que en general éstos están formados por múltiples procesadores [1, 2]. Esto es lo que actualmente nos motiva al estudio, investigación y desarrollo de soluciones de sistemas concurrentes y paralelos. Una red de Petri es una herramienta gráfica con una base matemática y formal, la cual permite modelar los estados locales y globales de sistemas reactivos y las posibles transiciones entre ellos [3]. Dicho modelo gráfico

se corresponde con una ecuación de estado, por medio de la cual se define el siguiente estado del sistema; dependiendo del estado actual y de las transiciones que se disparen, esto se realiza con un cálculo algebraico. Tradicionalmente los sistemas concurrentes son resueltos con el uso de semáforos y/o monitores [4]. Nuestra propuesta es ejecutar la red de Petri, haciendo uso del cálculo algebraico, y hacer concordar el modelo y la implementación. Con este cambio en el enfoque, la solución obtenida con el modelo de redes de Petri es la misma que la implementada con el framework. Este último genera el código de la programación paralela, la comunicación con el procesador de Petri y el esqueleto de los códigos secuenciales, disminuyendo la codificación, sus pruebas y verificación.

En este trabajo se desarrolla un framework que genera el código fuente de procesos que interactúan con la red de Petri, que es ejecutada en cualquiera de los procesadores y simulador de procesador de Petri desarrollados en el Laboratorio de Arquitectura de computadoras (FCEFyN, UNC) [5-7]; de este modo, se facilita al usuario el trabajo de desarrollo de sistemas concurrentes y reactivos mediante el uso de redes de Petri.

2 Conocimiento Previo y Antecedentes

Un procesador de Petri es un componente encargado específicamente de la ejecución de redes de Petri. Puede estar implementado en hardware o en software.

Como antecedentes a este trabajo, existen implementaciones en hardware usando la matriz de adyacencia como [8, 9]. Estas soluciones no soportan: programación en tiempo real, programación de prioridades, brazos con peso, brazos inhibidores, plazas acotadas, brazos de reset, detección de interbloqueo, disparos automáticos, etc. También se han implementado con lenguajes de alto nivel como en [10, 11], con las mismas restricciones que las anteriores. Aybar y Altuğ en [12] usa red de Petri temporal, ejecutadas con un lenguaje de alto nivel, también con las mismas restricciones.

En el Laboratorio de Arquitecturas de Computadoras se desarrollaron múltiples procesadores de redes de Petri implementados en IP-Core, para la ejecución de redes de Petri jerárquicas [13] mediante hardware, y también se desarrollado un software de simulación del procesador de red de Petri jerárquicas [14]. Este procesador se utilizó en este trabajo como componente para la ejecución de la red de Petri.

En los trabajos de referencia [15-17] se ha desarrollado una herramienta para la generación de código automático para sistemas concurrentes, pero la propuesta no contempla ejecución paralela. Esencialmente, se diferencia de nuestro desarrollo, en la programación del procesador de Petri o del simulador, puesto que nuestro desarrollo realiza la programación en forma directa (con las matrices). Esto permite, incorporar distintos tipos de brazos como transiciones temporales; la decisión del disparo se realiza en dos ciclos de reloj y en paralelo, la programación de la prioridad es directa, y ambas puede realizarse en tiempo de ejecución, y la ejecución es paralela. Con respecto a la comunicación con los procesos nuestro sistema incluye el disparo automático en la entrada y él no informe en la salida y su programación no requiere código. Y no menos importante es que no se ha cambiado la semántica de la red de Petri que ejecuta el sistema.

2.1 Etiquetas de Transición

Los procesadores de Petri requieren de algún mecanismo para especificar el comportamiento que presentará cada transición en el sistema. Por tal motivo, en el proyecto [14] se creó una nomenclatura de doble etiqueta. La Figura 1 ilustra las convenciones utilizadas para describir las transiciones del sistema.

Para poder solicitar la ejecución explícita de transiciones al procesador de Petri, y consultar si éstas fueron ejecutadas o no, se trabaja con el concepto de colas de entrada y colas de salida. El procesador de Petri tiene una cola de entrada y una de salida. La cola de entrada tiene la función de almacenar las transiciones cuyos disparos se solicitaron explícitamente para ser ejecutados, y proveer a la red la información que necesita para determinar que transiciones se encuentran pendientes en la cola. La cola de salida tiene la función de almacenar las transiciones ejecutadas por la red para que los procesos puedan en cualquier momento consultar su estado.

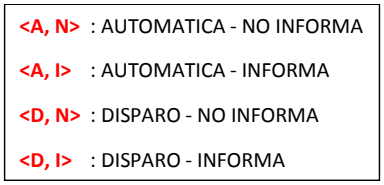

Figura 1. Nomenclatura de etiquetas de transiciones.

Es importante notar que las etiquetas de transición son binarias. Se requieren dos bits para generar los cuatro estados posibles de cada etiqueta, descriptos en la Figura 1, de esta forma se define el comportamiento de cada transición con la menor complejidad posible.

2.2 Procesos

Considerando que el trabajo está enfocado en el modelado de sistemas concurrentes con POPN [18]. Para el desarrollo del componente encargado de la generación de código surge la necesidad de definir los procesos que forman parte del sistema.

Por lo tanto, se deben especificar la cantidad de procesos del sistema, sus responsabilidades y tipo. Se han diferenciado dos tipos de procesos: “procesos runnable” (adoptando el “runnable” según la definición del lenguaje Java para hilos), que definen la dinámica y orden de ejecución del sistema, y los “procesos no runnable” (objetos) que representan a los recursos del sistema. En la generación de código fuente del sistema, los procesos “runnable” se convierten en objetos hilos ejecutables y los de tipo “no runnable” en objetos.

3 Framework “CodGenTPN”

“CodGenTPN” es el nombre del framework desarrollado. Este framework usa las redes de Petri y redes de Petri con Tiempo, para el modelado de sistemas concurrentes y paralelos e integra las principales funcionalidades para el diseño y desarrollo de

sistemas. Le provee al usuario una interfaz para el diseño, simulación y análisis de dichas redes, sumando a esto la generación de los códigos de ejecución concurrente y secuencial del sistema modelado.

Las funcionalidades de *"CodGenTPN"* son:

- Crear, editar, simular y validar redes de Petri y redes de Petri con Tiempo.
- Definir procesos del sistema.
- Generar código del sistema.

Nos centraremos en los dos últimos ítems, puesto que el primer ítem se obtuvo integrando TINA [19] a este proyecto.

3.1 Definir Procesos

La definición de los procesos del sistema es necesaria para la generación de código. Para lo cual, se desarrolló una interfaz gráfica de usuario (IGU) de definición de procesos, donde el usuario define la semántica de los procesos.

La definición de un proceso implica crearlo, asignarle un nombre, determinar el tipo de proceso, "runnable" o "no runnable", y determinar las transiciones que lo componen.

Cada transición, como se dijo anteriormente, tiene asignada una etiqueta. Las transiciones que son "disparo" y/o "informa", son las que deben ser relacionadas con los procesos. Tanto el "disparo", el "informe", el orden secuencial, como el nombre del método deben ser configurados, según corresponda el caso.

1. "Disparo": se deben marcar las transiciones a las cuales el proceso debe pedir explícitamente el disparo.
2. "Informe": se deben marcar las transiciones por las cuales el proceso debe esperar un informe de disparo efectuado antes de continuar con la ejecución.
3. "Secuencia de disparo e informe": secuencia en el que el proceso pide los disparos y espera los informes. Existen dos tipos de secuencia. Secuencia de las precondiciones, se realizan por única vez al inicio del proceso, y secuencia cíclica, que se repiten en el mismo orden durante toda la vida del proceso. La numeración es única para cada proceso, es decir que se pueden intercalar los pedidos de disparo con las esperas de los informes .
4. Métodos "Disparo/Informe": se define el nombre del método que se llamará cuando se pide un disparo o se recibe un informe. Para todos los disparos o informes de un proceso se deberá definir un método. Dichos métodos son los que el usuario deberá completar una vez generado el código, con las acciones específicas que correspondan al método.

Para simplificar el diseño del sistema, se introdujo como restricción del framework, que el pedido del disparo de una transición pertenezca sólo a un proceso, y que la comunicación de la resolución de una transición (un informe) sea comunicada a un sólo proceso, que no necesariamente sean los mismos. De esta forma, por medio de las etiquetas de transición, se vinculan los procesos del sistema con las transiciones de la red de Petri que lo modela.

3.2 Generar Código

Puesto que el tipo de problemas que se abordan con el framework son POPN, donde existen componentes reales como robots, máquinas, entradas, salidas, piezas, etc.; y la programación resultante es programación orientada a objetos (OOP), se deben construir clases que representen a estos objetos, las cuales deben ser runnables o no según su comportamiento. Además existe una clase principal (Main) que incluye el método main() para inicializar el sistema, y el objeto que nos vincula con el procesador de Petri. En los objetos del sistema, el usuario define los métodos que son las tareas propias del sistema.

Para la generación de código se optó por Java, debido a su facilidad para el manejo de hilos y concurrencia, y su portabilidad. También C++ es adecuado.

El código generado se comunica con un procesador de Petri para sincronizarse según la estructura de la red de Petri, y con los componentes reales del sistema para controlarlos; a esta última comunicación la define el usuario.

La clase Main se encarga de inicializar los objetos del programa, que son: una instancia del componente procesador de Petri y una instancia por cada proceso definido.

El código de los objetos posee métodos con dos objetivos diferentes: los que se comunican con el procesador de Petri, y los que define y completa el usuario que cumplen tareas específicas del sistema. Además, los objetos "runnable" implementan el método abstracto run(), cuya responsabilidad es realizar secuencias de ejecución relacionadas (controladas por el procesador de Petri), que son independientes de otras secuencias de ejecución.

Y por último, el objeto que vincula al sistema con la red de Petri, programa al procesador con los archivos de descripción de la red de Petri, que son archivos de texto plano con matrices y vectores que definen la estructura de la red. Y es responsable de la comunicación entre los objetos del sistema y el procesador de Petri.

3.3 Flujo de Trabajo del Framework

La Figura 2 expone el flujo de trabajo del proceso de diseño a modo de ejemplo, puesto que es preciso realizar varias iteraciones entre las distintas etapas, este flujo es:

- Definición de los requerimientos del sistema a desarrollar.
- Creación o edición de una red de Petri que modela al sistema.
- Definición de etiquetas, que vinculan las transiciones con el flujo de ejecución.
- Validación de la red de Petri, realizando análisis de alcanzabilidad y análisis estructural.
- Definición de los procesos del sistema según su responsabilidad (nombre, tipo, transiciones que lo componen).
- Generación del código de control de los procesos del sistema.
- Completar el código: se deben completar las tareas específicas del problema en particular para cada método.
- Verificar el correcto funcionamiento del software final.

Es importante para obtener un buen resultado, validar la red de Petri antes de generar el código. Ya que si la red de Petri es errónea (presenta interbloqueo, inanición, etc.), el código generado también lo será.

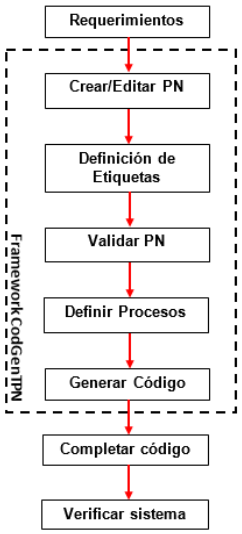

Figura 2. Flujo de trabajo del Framework CodGenTPN

4 CASO DE APLICACIÓN: "SISTEMA DE MANUFACTORIZACIÓN ROBOTIZADO"

La Figura 3, representa un caso de estudio típico de manufacturación robotizado flexible.

4.1 Introducción

En este caso se estudia el problema y solución planteada en [18, 20], "Sistema de manufacturación robotizado". La red de Petri, que se muestra en la Figura 3, modela a dicho sistema; la que ha sido modificada de la original con el fin de evitar que ocurra interbloqueo, para implementar la solución con CodGenTPN.

El sistema de manufacturación consiste en tres robots, cuatro máquinas y tres tipos diferentes de piezas a procesar, como se observa en la Figura 3.

Los robots retiran las piezas de tres inventarios I1, I2 e I3, y las colocan en las máquinas para su procesamiento. Cuando finaliza el proceso, las retiran y las colocan en las salidas O1, O2 y O3. Los robots también trasladan las piezas de máquina en máquina en las distintas etapas del proceso. Por lo que, para la manufactura, cada tipo de pieza debe seguir una trayectoria de procesamiento distinta, como se observa en la Figura 3.

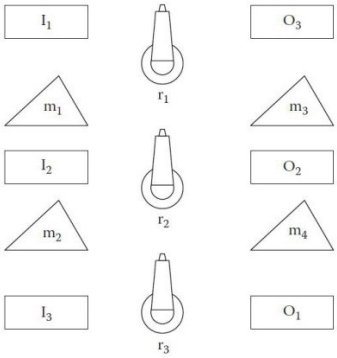

Figura 3 Diagrama de la celda de fabricación.

Las trayectorias de las piezas son:

- I1 (alternativa 1) → M1 → M2 → O1
- I1 (alternativa 2) → M3 → M4 → O1
- I2 → M2 → O2
- I3 → M4 → M3 → O3

4.2 Desarrollo con el framework CodGenTPN

La red de Petri en la Figura 4 modela el sistema de manufacturación robotizado. Las plazas de p21 a p23 representan los estados discretos de las etapas de la línea de producción dos, las plazas de p11 a p18 representan los estados discretos de las etapas de la línea de producción uno y las plazas de p31 a p35 representan los estados discretos de las etapas de la línea de producción tres. Las plazas p10, p20 y p30 representan a los inventarios de cada línea de producción, y las marcas de cada una son la cantidad máxima de productos que pueden estar simultáneamente en cada línea de producción.

El disparo de las transiciones (t20_R2, t21_M2, t22_R2, t23) causa movimientos de tokens entre las plazas de p21 a p23, que se corresponde con la circulación del producto entre las distintas etapas de la línea dos, el disparo de las transiciones (t101_R1, t11_M1, t112_R2, t113_M 2, t114_R3, t121_M3, t122_R2, t123_M4, t124_R3, t102) causa movimientos de tokens entre las plazas de p11 a p18, que se corresponde con la circulación del producto entre las distintas etapas de la línea uno, y el disparo de las transiciones (t30_R3, t31_M4, t32_R2, t33_M3, t34_R1, t35) causa movimientos de tokens entre las plazas de p31 a p35, que se corresponde con la circulación del producto entre las distintas etapas de la línea tres. Las plazas de M1 a M4 y de R1 a R3 denotan la disponibilidad de los recursos del sistema. Las plazas (c1, c2, c3, c4) son el supervisor de prevención de interbloqueo, que previenen el vaciado de los sifones marcados [21].

Definición de etiquetas: La mayoría de las etiquetas de transiciones de la red de Petri de la Figura 4 son del tipo disparo, informe < D, I >, con diferentes nombres que

indican tareas o eventos asociados para agregar semántica y facilitar la lectura, pero en general, todos los pedidos de disp1aros de las transiciones se efectúan cuando un robot o una maquina termina la tarea en curso, y todos los informes de disparo de las transiciones indican cuando se debe comenzar la próxima etapa de producción.

Las etiquetas de las transiciones que comienzan un proceso de producción son de disparo automático, es decir que se van disparando a medida que están sensibilizadas. Esto se definió para obtener la mayor performance de producción del sistema y automatizar la inicialización. Por último, las etiquetas de las transiciones que finalizan cada línea de producción, las cuales no informan el disparo debido a que esto no es necesario ya que el proceso termino.

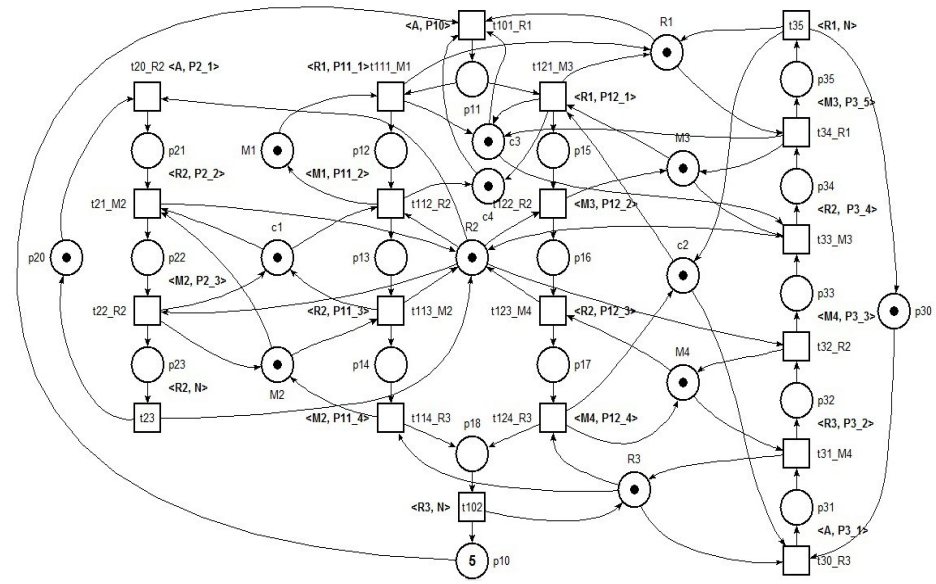

Figura 4 Red de Petri: sistema de manufactorización automatizada

Definición de procesos: Considerando que el diseño de la red de Petri es POPN [18], la solución consiste en definir un proceso del tipo “no runnable” por cada recurso del sistema y un proceso “runnable” por cada etapa independiente de cada línea de producción. Siendo una etapa independiente de un proceso de producción aquella que se puede ejecutar en paralelo con el resto de las etapas del mismo proceso.

Por lo tanto, se definieron siete procesos “no runnable” correspondientes a las máquinas M1, M2, M3 y M4 y a los robots R1, R2 y R3. Por otro lado, se definieron ocho procesos “runnable” correspondientes a: Producto 1: un proceso para iniciar la producción, cuatro procesos para la alternativa 1 con cuatro etapas independientes y un proceso para la alternativa 2 con cuatro etapas no independientes. Producto 2: un proceso para las tres etapas no independientes. Producto 3: un proceso para las cinco etapas no independientes.

5 Resultados Finales

Las clases que el framework CodGenTPN generó para este proyecto son:

- Main: encargada de inicializar el sistema.
- Maquina1 a Maquina4: cada una encargada de controlar las tareas a realizar por las máquinas de la celda de fabricación según corresponda para cada producto. Además, se comunica con la clase que se comunica con el procesador de Petri.
- Robot1 a Robot3: cada una encargada de controlar las tareas a realizar por los robots de la celda de fabricación según corresponda para cada producto. Además, se comunica con la clase que se comunica con el procesador de Petri.
- Producto10: encargada de controlar las acciones a realizar en la primera etapa de producción para el producto 1 (esta etapa de producción comienza una vez disparada la transición t101_R1, previa a la "bifurcación" de las alternativas). Esta clase se comunica con la clase que se comunica con el procesador de Petri para esperar el informe de disparo de la transición t101_R1 y luego llamar al robot1 que realice la tarea de la primera etapa de producción.

Las clases Producto11e1 a Producto11e4, Producto12 Producto2 y Producto3 son similares a las especificada en el ítem anterior.

Una vez obtenido el código, las únicas modificaciones realizadas fueron completar con código secuencial los métodos generados en cada clase, de manera que realicen lo requerido en cada caso.

Se obtuvo un sistema paralelo que controla la producción de tres productos diferentes en una celda de fabricación, controlando la exclusión mutua de los recursos, la sincronización entre las etapas de producción y evitando el interbloqueo.

Tabla 1. Resultados globales de la simulación

Tiempo total de simulación (ms)	191280
Total de productos realizados	255

Para la simulación, a cada máquina se le asignó una demora de 1000ms para la ejecución de una etapa de cualquier proceso de producción, y a cada robot una demora de 100ms para trasladar un objeto de un punto al otro. De esta forma, el tiempo de producción mínimo de los productos 1 y 3 es de 2300ms y el tiempo mínimo de producción del producto2 es de 1200ms. Cabe destacar que los tiempos mínimos se determinaron considerando el caso que el producto en particular no sufra demoras por falta de recursos. Las Tabla 1 y Tabla 2, muestran los resultados obtenidos para una simulación de 3 minutos 11 segundos de duración. De la Tabla 2 se observa que los totales obtenidos para los productos 1−2 y 3 son considerablemente más bajos que los totales obtenidos para los productos 1−1 y 2. Lo cual es esperable ya que entre los dos primeros existe exclusión mutua.

El incremento de los tiempos promedio con respecto a los tiempos mínimos es muy diferente de un producto a otro. Esto debido a los diferentes recursos que utilizan cada uno y la disponibilidad de ellos. En el caso de los productos 3 y 1−2 se nota poco incremento debido a que entre ellos tiene exclusión mutua y utilizan dos máquinas que no comparten con otros productos. Es decir, una vez que comienza, dispone de las

máquinas M3 y M4 sin demoras. Además de compartir los robots R1 y R3 únicamente con el producto 1−1. Por otro lado, el producto 2 tuvo el mayor incremento, debido a que utiliza dos veces al robot R2, el cual es muy requerido. Observando los throughput obtenidos en la Tabla 2, se nota que el producto 1−1 obtuvo un throughput cercano al doble que el resto, consecuencia del paralelismo entre sus etapas de producción. En cambio, en los productos 1−2, 2 y 3 no existe paralelismo entre sus etapas, resultando en un throughput mucho menor.

TABLA 2. RESULTADOS INDIVIDUALES DE LA SIMULACIÓN.

Producto	Total de productos	t mínimo [ms]	t promedio [ms]	Diferencia t mínimo y medio	Throughput [productos/tmin]
Producto 1-1	85	2300	3187	38,6%	1,02
Producto 1-2	46	2300	2312	0,5%	0,55
Producto 2	86	1200	2112	76,0%	0,54
Producto 3	38	2300	2366	2,9%	0,46

Vale aclarar que, para la presentación de los resultados se diferenció entre producto 1−1 y producto 1−2 a modo de diferenciar los resultados de las dos alternativas de producción. Sin embargo, por definición del problema, el producto final obtenido de ambas alternativas es el mismo.

En conclusión, se logró desarrollar y simular un sistema de control de una celda de manufactura robotizada, netamente concurrente, programando únicamente de manera secuencial y con una red de Petri. El desarrollo convencional de este sistema requiere de 23 semáforos y la carga computacional que demanda es de un 24% a un 40% superior por semáforo con respecto a una solicitud de disparo o consulta en el procesador de Petri, como ha sido medido en [6].

6 Conclusiones

Este desarrollo se obtuvo en base a diferentes casos de estudio, de los cuales sólo uno se expone en este escrito.

Se logró mostrar que es posible obtener el código fuente de los procesos de un sistema real, concurrente y paralelo, modelado con una red de Petri que se ejecuta en el procesador de Petri.

Como resultado de este trabajo, en base al procesador de Petri, se desarrolló un framework que engloba desde la creación de la POPN, que modela el sistema hasta la obtención del código fuente de las clases de dicho sistema.

Además con este enfoque, de manera sencilla, con las etiquetas de transición se vincularon los procesos y recursos reales del sistema con los hilos y objetos del programa. Concluyendo entonces, que existe una vinculación simple, directa y eficiente para solucionar problemas reales y complejos de programación paralela con concurrencia.

De acuerdo con los casos resueltos, el framework desarrollado simplifica la programación concurrente para el diseño de sistemas concurrentes y paralelos orientados a procesos. Se obtuvieron resultados exitosos con el código secuencial generado, al cual sólo se le completaron las tareas que ejecutan acciones específicas de los problemas.

Hacemos notar que el uso del framework, para el diseño y desarrollo de sistemas concurrentes y paralelos, simplifica la programación concurrente, la codificación, el testing y la claridad del programa. En consecuencia los tiempos de desarrollo se reducen, ya se facilitan los cambios de requerimientos en la POPN que modela el sistema.

El framework valida e integra a la red de Petri en forma directa, y permite la generación de objetos e hilos requeridos por el programa. Lo cual asegura que el sistema generado es estructuralmente correcto y de calidad.

En todos los casos en que no haya que modificar ninguna de las transiciones cuyas etiquetas de informe o disparo formen parte de algún proceso del sistema o agregar una nueva, sólo se debe cambiar los archivos de descripción de la red de Petri (matrices y vectores), con los que se define la nueva red. Sin necesidad de cambiar el código generado del sistema, aún si este ya fue completado y/o modificado por el usuario. Estas matrices, también se pueden cambiar en tiempo de ejecución.

Por último, durante el desarrollo de los casos de estudio, concluimos que el framework no sólo funciona para el caso de sistemas concurrentes y paralelos, sino que también es posible diseñar y obtener el código de control de sistemas secuenciales.

Como trabajos futuros, se está rediseñando el sistema, basado en el uso de patrones de diseño y esencialmente se incluirá un patrón observer para desacoplar las transiciones de los eventos y poder reprogramar las relaciones con las transiciones y los eventos en tiempo de ejecución. En este momento el sistema está en producción con el fin de determinar la usabilidad y se ha determinado que se requiere perfeccionar la interface de usuario e incorporar nuevas propiedades de usabilidad.

Referencias

1. R. A. B. David R. Martinez, M. Michael Vai, *High Performance Embedded Computing Handbook A Systems Perspective*. Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, Massachusetts, U.S.A.: CRC Press, 2008.
2. M. Domeika, *Software Development for Embedded Multi-core Systems*. 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA Linacre House, Jordan Hill, Oxford OX2 8DP, UK, 2008.
3. M. Diaz, *Petri Nets Fundamental Models, Verification and Applications*. NJ USA: John Wiley & Sons, Inc, 2009.
4. J. F. González, *Java 7 Concurrency Cookbook*: BIRMINGHAM - MUMBAI, 2012.
5. P. M. Micolini Orlando, Gallia Nicolás A., Alasia Melisa A., "Procesador de Petri para la Sincronización de Sistemas Multi-Core Homogéneos," *CASECongreso Argentino de Sistemas Embebidos,* pp. 3-8, 2012.
6. N. G. M. Pereyra, M. Alasia and O. Micolini, "Heterogeneous Multi-Core System, synchronized by a Petri Processor on FPGA," *IEEE LATIN AMERICA TRANSACTIONS,* vol. 11, pp. 218-223, 2013.
7. J. N. y. C. R. P. Orlando Micolini, "IP Core Para Redes de Petri con Tiempo," *CASIC 2013,* pp. 1097-110, 2013.

8. H. S. Murakoshi, M. ; Ding, G. ; Oumi, T. ; Sekiguchi, T. ; Dohi, Y., "A high speed programmable controller based on Petri net," *IEEE, Industrial Electronics, Control and Instrumentation,* vol. 3, pp. 1966 - 1971, 1991.
9. M. S. Hideki, K. N. Tatsumasa, D. O. Yasunori, F. ANZAI, N. KAWAHARA, T. TAKEI, and T. WATAN ABE, "Hardware Architecture for Hierarchical Control of Large Petri Net," *IEEE,* pp. 115-126, 1993.
10. R. Piedrafita Moreno and J. L. Villarroel Salcedo, "Adaptive Petri Nets Implementation. The Execution Time Controller," in *Workshop on Discrete Event Systems Göteborg*, Sweden, 2008, pp. 300-307.
11. F. Xianwen , X. Zhicai, and Y. Zhixiang, "A Study about the Mapping of Process-Processor based on Petri Nets," *Anhui University of Science and Technology,* 2006.
12. A. Aydın and I. Altu˘g, "Deadlock Avoidance Controller Design for Timed Petri Nets Using Stretching," *IEEE SYSTEMS JOURNAL,* vol. VOL. 2, pp. 178-189, JUNE 2008.
13. K. Jensen and L. M. Kristensen, "Coloured Petri Nets Modelling and Validation of Concurrent Systems," *Springer* 2009.
14. E. A. Orlando Micolini, Sergio H. Birocco Baudino, y Marcelo Cebollada, "Reducción de recursos para implementar procesadores de redes de Petri," *en Jaiio 2014,* 2014.
15. R. Pais, Barros, J.P. ; Gomes, L., "From Petri net models to C implementation of digital controllers " *Emerging Technologies and Factory Automation. ETFA 2005. 10th IEEE Conference on,* 2010.
16. R. Pais, Barros, J.P. ; Gomes, L., "A Tool for Tailored Code Generation from Petri Net Models," *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on,* vol. 1, pp. 857-864, 2005.
17. W. L. Weizhi Liao "Automatic concurrent Program Generation from Petri nets," *International Symposium on Distributed Computing and Applications to Business, Engineering & Science,* pp. 34 - 39, 2013.
18. M. Z. Naiqi Wu, *System Modeling and Control with Resource-Oriented Petri Nets*. Boca Raton, FL, 2010.
19. LAAS/CNRS. (2013). *TINA webpage*.
20. N. W. a. M. Zhou, "PROCESS VS RESOURCE-ORIENTED PETRI NET MODELING OF AUTOMATED MANUFACTURING SYSTEMS," *Asian Journal of Control,,* vol. 12, p. 267 280, 2010.
21. I. P. Nedjeljko Peri, " An Algorithm for Deadlock Prevention Based on Iterative Siphon Control of Petri Net," *ATKAAF,* 2006.