Algoritmos para determinar cantidad y responsabilidad de hilos en sistemas embebidos modelados con Redes de Petri S 3 PR

Conference Paper · October 2021

CITATIONS
5

READS
794

2 authors:

Luis Orlando Ventre
National University of Córdoba
41 PUBLICATIONS **49** CITATIONS
SEE PROFILE

Orlando Micolini
National University of Córdoba
65 PUBLICATIONS **93** CITATIONS
SEE PROFILE

Algoritmos para determinar cantidad y responsabilidad de hilos en sistemas embebidos modelados con Redes de Petri S^3PR

Ing. Luis Orlando Ventre[1], Dr. Ing. Orlando Micolini[1]

[1]Laboratorio de Arquitectura de Computadoras, FCEFyN-Universidad Nacional de Córdoba
Av. Velez Sarfield 1601,CP-5000, Córdoba, Argentina
{luis.ventre, orlando.micolini}@unc.edu.ar

Abstract. La evolución de la tecnología, el uso del IoT, y los requerimientos reglamentarios de la industria impactan en el diseño de sistemas embebidos convirtiéndolo en complejo y desafiante e imponiendo métodos formales para su desarrollo. Más aun considerando el reducido time-to-market, es necesario minimizar los tiempos de desarrollo. En este escenario los sistemas deben ser concurrentes y seguros para aprovechar el rendimiento de las modernas arquitecturas multicore. Las Redes de Petri son un reconocido y adecuado lenguaje de modelado, análisis y ejecución de sistemas reactivos, paralelos y concurrentes. Para potenciar los esfuerzos del modelado, se utiliza el modelo para obtener automáticamente parte de la implementación del sistema. En este trabajo se presenta un conjunto de algoritmos, a partir de un sistema modelado con Redes de Petri, para determinar automáticamente los hilos y responsabilidades de ejecución; esto tiene por objetivo mitigar los tiempos de desarrollo y reducir los errores de programación.

Keywords: Determinación automática de hilos, Redes de Petri, Generación de código, Sistemas embebidos, IoT.

1 Introducción

Actualmente los estándares de la Industria 4.0 [1] demandan la utilización de técnicas formales en el diseño de sistemas embebidos críticos, reactivos (RS) y dirigidos por eventos (EDA) [2]. Este escenario impone que el diseño cumpla con complejos requerimientos no funcionales ya que son sistemas multi-hilos, concurrentes e interactúan con variables y eventos del propio sistema y del mundo exterior, donde los datos y eventos son heterogéneos y no determinísticos [3].

Determinar la secuencia de ejecución de estados consecutivos en los sistemas multi-hilos considerando sus combinaciones de ejecución con otros hilos, aun si estos son de estado finito, presentan un problema importante e intrínsecamente difícil de resolver. Así como la complejidad de determinar el número de hilos, las variables locales asociadas a los mismos, y las variables compartidas por estos, son también problemas poco explorados y determinantes de resolver en el diseño de RS y EDA.

Las fases del diseño de un sistema embebido incluyen el desarrollo del modelo basado en un conjunto de requerimientos [4]. Este modelo es el subyacente para otras etapas, incluida la etapa de codificación y testing de la aplicación [5]. Las redes de Petri (RdP) extendidas y no autónomas son un lenguaje de modelado de propósito general que admite el modelado de sistemas reactivos, concurrentes y paralelos independientemente de la plataforma.

En el diseño de RS y EDA la transformación del modelo en software implica un trabajo de traducción, gestión del uso de recursos y determinación de la cantidad y responsabilidad de los hilos. Esto se logra a través de sucesivas iteraciones con el fin de solucionar errores potenciales de interpretación e implementación.

En este trabajo y con la finalidad de cerrar esta brecha se propone una metodología para determinar automáticamente el número de subprocesos activos máximos simultáneos, así como determinar el número de subprocesos máximos requeridos y la responsabilidad de los mismos. Esto se obtiene a partir de un sistema modelado con RdP y sus invariantes de transición (ITs). Los algoritmos propuestos como metodología tienen la finalidad de reducir los tiempos de desarrollo, mitigar errores de codificación y contribuir en la generación automática de código; como así también, evaluar y gestionar la asignación de recursos en el sistema. Esta metodología es aplicable a sistemas modelados con una clase de RdP denominada Sistema de Procesos Secuenciales Simples con Recursos (S^3PR) la cual es adecuada para sistemas embebidos que comparten recursos y sincronización.

De acuerdo con nuestra investigación de documentos, no se han encontrado algoritmos con las funcionalidades para la determinación automática de la cantidad y responsabilidad de los hilos a partir del modelo del sistema realizado con una RdP. Como antecedente a este trabajo, y como aporte de este grupo de investigación, se puede encontrar el desarrollo de un Procesador de Petri (PP) modular que ejecuta RdP ordinarias en [6], una metodología para el desarrollo de sistemas embebidos basada en RdP [7] y el desarrollo de un PP extendido modular con un algoritmo para determinar la cantidad de hilos en ese procesador [8]. A diferencia de este último, en el actual trabajo se presentan tres algoritmos los cuales son de aplicación general para ser ejecutados sin la necesidad de un PP. Asimismo, en [9] se realizó un estudio sobre más de 70 referencias que hacen uso de la RdP para la solución de RS y EDA y no se encontraron metodologías similares a las aquí presentadas.

La siguiente sección presenta los objetivos de este trabajo; mientras que en la sección 3 se exponen la metodología y herramientas. Luego, en la sección 4, los algoritmos propuestos, mientras que en el apartado 5 se expone un caso de aplicación y los resultados, y finalmente en el apartado 6 las conclusiones y trabajos futuros.

2 Objetivos

En el modelo del sistema, realizado con una RdP, se encuentra explícita la lógica del sistema. La etapa de codificación transforma este modelo en software, lo que implica esfuerzos iterativos para interpretar, transcribir y refinar el modelo. Esto conlleva una carga de esfuerzo y tiempo en las etapas de desarrollo, que se pretenden mitigar.

El objetivo principal e innovador de esta propuesta es contribuir a la generación automática de código con la determinación de la cantidad y responsabilidad de los hilos en la ejecución, en concordancia con el paralelismo intrínseco del modelo; el cual tiene la capacidad de expresión de una máquina de Turing. Como así también gestionar el uso y la asignación de recursos en el sistema. Esta propuesta, mantiene todas las propiedades verificadas en el modelo, ya que se ejecuta la ecuación de estado extendida [10] de éste. Para esto se utiliza un monitor como mecanismo de control de concurrencia. La importancia del uso de este mecanismo radica en desacoplar la lógica de la política y las acciones, lo que resulta en un sistema modular, simple, mantenible y verificable. Estas metodologías tienen como finalidad garantizar un diseño seguro, correcto, eficiente y robusto de los sistemas embebidos y sus aplicaciones.

3 Metodología y herramientas.

Existen varias herramientas de modelado, entre las que se encuentran: diagramas UML [11] y RdP [12, 13]. Los diagramas UML brindan las características necesarias, en parte, pero esencialmente carecen de mecanismos de verificación formal que garanticen estrictamente el cumplimiento de requerimientos críticos, los cuales son aspectos fundamentales a ser implementados en los RS y EDA.

Dado que las RdP extendidas y no autónomas [14] permiten modelar la concurrencia, sincronización, el estado local y global y el paralelismo, es posible verificarlas formalmente, son ejecutables [9] y escalables cuando se expresan con la ecuación de estado extendida [10]; se ha considerado el formalismo más conveniente y se ha seleccionado como herramienta de modelado y lenguaje de ejecución.

3.1 Redes de Petri

Una RdP, denotada como *PN*, es una quíntupla [5] definida por:

$$PN = (P, T, I^+, I^-, M_0) \tag{1}$$

Dónde:

- $P = \{p_1, p_2, \dots, p_n\}$ es un conjunto finito, no vacío, de plazas.
- $T = \{t_1, t_2, \dots, t_m\}$ es un conjunto finito, no vacío, de transiciones.
- I^+, I^- son las relaciones de incidencia de salida y entrada de las plazas, la Matriz de Incidencia es:

$$I = I^+ - I^- \tag{2}$$

- $M_0 = [m_0(p_1), m_0(p_2) \dots, m_0(p_n)]$ es el marcado inicial de la red.

RdP sincronizadas o no autónomas. Las RdP Sincronizadas introducen eventos, y son una extensión de las RdP [14, 15].

Ecuación de estado. La ecuación de estado de una RdP, con n plazas y m transiciones con brazos con peso mayor o igual a uno y marca inicial M_0 es:

$$M_{j+1} = M_j + I * \sigma \quad (3)$$

Siendo: σ el vector disparo, con dimensión $m \times 1$. Cuando se dispara una transición sensibilizada, se puede calcular el siguiente estado usando la ecuación (3).

Conflicto entre transiciones. Los conflictos entre transiciones de una RdP sincronizada [14] ocurren cuando dos o más transiciones se encuentran sensibilizadas, sus eventos asociados suceden simultáneamente y el disparo de una de ellas desensibiliza la otra transición. Estos conflictos son resueltos con una política de prioridades.

S^3PR. Para definir una RdP S^3PR es necesario primero definir los conceptos de RdP S^2P y S^2PR. Se define una RdP S^2P como una red que modela procesos secuenciales simples; las RdP S^2PR, modelan procesos secuenciales simples con recursos; y finalmente las RdP S^3PR son la composición neta de RdP S^2PR a través de un conjunto de plazas comunes (recursos). Una explicación específica de las distintas subclases de estas RdP se encuentra en [16].

3.2 Monitor de Concurrencia

En los sistemas RS/EDA se necesitan mecanismos para organizar el acceso exclusivo a los recursos y para sincronizar y comunicar las tareas. Uno de los mecanismos más naturales, elegantes y eficientes para la sincronización y la comunicación, especialmente para los sistemas multicore, es el monitor [17].

En este desarrollo, el monitor, gestiona los eventos en exclusión mutua con el fin de determinar cuál acción ejecutar y cuando; para lo cual se basa en dos componentes que son: la lógica y la política. Es importante destacar que no es su responsabilidad la ejecución de las acciones, dado que estas son ejecutadas por los hilos.

3.3 Arquitectura del sistema de la solución

Los componentes de la arquitectura del sistema (RS/EDA) que da soporte a los objetivos de este trabajo se observan en la Fig. 1 y son: eventos de software, eventos físicos, monitor de concurrencia, manejador de eventos, RdP (lógica), política, hilos y sus acciones asociadas. Esta arquitectura es modular, sus componentes están desacoplados y sus interfaces claramente definidas. De esta manera se simplifica su diseño, gestión y control, lo que la hace mantenible, refactorizable y asegura que el formalismo de la RdP se mantiene.

En la Fig. 1 se muestran las interacciones entre los componentes de la arquitectura y a continuación se describirán las responsabilidades de los principales componentes:

Manejador de eventos. Es el módulo encargado de recibir los eventos/estímulos del sistema y del exterior. Dado que los eventos son responsables de desencadenar una acción que depende del estado del sistema, es necesario que este módulo direccione el evento recibido al hilo correspondiente.

Monitor. Gestiona el acceso de los hilos en exclusión mutua con el fin de determinar cuál acción ejecutar y cuando; para lo cual utiliza la lógica (RdP) y la política. Es importante destacar que no es su responsabilidad la ejecución de las acciones.

Red de Petri. Modela y representa la lógica del sistema. Es el mecanismo que utiliza el monitor para determinar a partir de los eventos y del estado del sistema, las acciones posibles a ejecutar por los hilos.

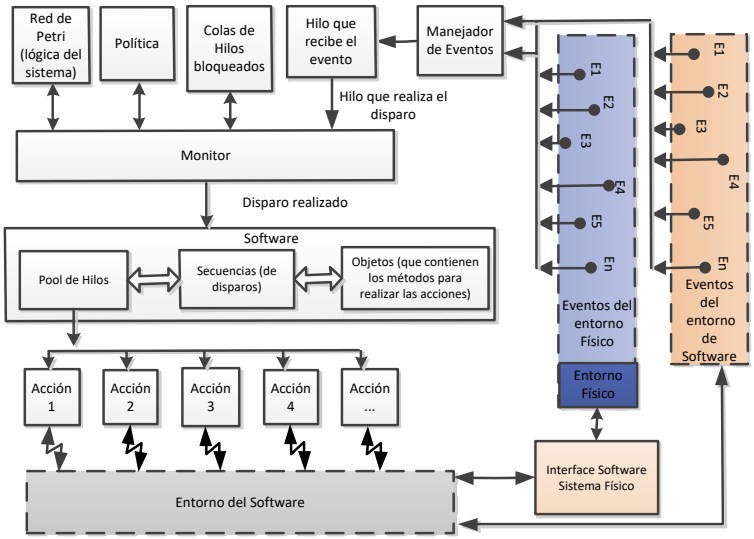

Fig. 1 Componentes de la arquitectura del sistema de la solución

Política. Es el mecanismo que utiliza el monitor para solucionar los conflictos del sistema con un propósito específico. Decide entre las posibles acciones ejecutables, cual es la próxima a ejecutar.

Colas de hilos bloqueados. Su responsabilidad es bloquear los hilos, con las solicitudes de disparo de una transición requerida. Es necesario que a cada transición se le asigne una cola, puesto que cada transición opera como una variable de condición [17]

Acciones. Son las tareas a realizar por cada hilo de acuerdo al estado en que se encuentra el sistema.

Secuencia de disparos. Representa a las transiciones que se corresponden con los segmentos de ejecución determinados por los algoritmos propuestos en éste trabajo.

4 Algoritmos de la solución.

La hipótesis de este trabajo se basa en el hecho de que un modelo elaborado con una RdP no autónoma es un conjunto de instrucciones, ecuaciones y restricciones o reglas para generar el comportamiento de E/S de un sistema. Es decir, el modelo se describe

como estado, transiciones y mecanismos para aceptar trayectorias de entrada y generar trayectorias de salida en función de su estado.

La definición, en términos de especificaciones del sistema, tiene la ventaja de una base matemática sólida y una semántica inequívocamente definida. Para especificar un comportamiento, el modelo necesita agentes. Se trata básicamente de un sistema informático capaz de ejecutar el modelo. El mismo modelo, expresado en formalismo, puede ser ejecutado por diferentes agentes, permitiendo así la portabilidad e interoperabilidad a un alto nivel de abstracción.

En este proyecto, los hilos son los agentes encargados de ejecutar el modelo haciendo uso de la ecuación de estado [10], para generar el comportamiento deseado.

En el modelo descripto con una RdP S^3PR los ITs se corresponden con los procesos para llevar a cabo un ciclo en el sistema, por lo cual los algoritmos propuestos utilizan están propiedades estructurales para la determinación de la cantidad y responsabilidad de los hilos de ejecución. Estos hilos ejecutan las transiciones consecutivas de cada IT y adquieren los estados entre estas transiciones. Estos estados están asociados a las acciones que el sistema debe ejecutar.

4.1 Algoritmo para la determinación de hilos máximos activos simultáneos.

1) Obtener los IT de la RdP y para cada IT realizar:
2) Obtener el conjunto de plazas asociadas al IT en análisis.

$$PI_i = \cup_{\forall t \in Inv} \bullet t \cup \cup_{\forall t \in Inv} t \bullet \quad (4)$$

Dónde: PI_i representa el conjunto de plazas asociadas al iésimo IT.

3) Determinar las plazas relacionadas a acciones de cada IT. Para esto es necesario eliminar del conjunto PI_i, las plazas que son restricciones, recursos e idle:

$$PA_i = PI_i - \left\{ PI_{\text{restricciones}_i} \cup PI_{\text{recursos}_i} \cup PI_{idle_i} \right\} \quad (5)$$

Dónde: PA_i representa el conjunto de plazas de acciones asociadas al iésimo IT.

4) Del árbol de alcanzabilidad de la RdP, se debe obtener MA, el cual es el conjunto de todos los marcados posibles de todos los conjuntos de plazas PA_i.
5) De cada marcado posible (estado) del conjunto MA, se debe realizar la suma de las marcas. De todas estas sumas, se debe buscar la de mayor valor (marcado máximo). Esta será la cantidad máxima de hilos activos simultáneos en el sistema.

4.2 Algoritmo para determinar la responsabilidad de los hilos.

El algoritmo propuesto comienza con el análisis de la estructura de los IT de la red. La asignación de responsabilidad de ejecución de los ITs varía de acuerdo a si éstos son estrictamente lineales (secuenciales) o presentan conflictos (forks) y/o uniones (joins). En los casos donde el IT presente fork/join la responsabilidad de ejecución se fracciona en diferentes segmentos. Al igual que en el algoritmo anterior, los segmentos están compuestos por subconjuntos de plazas de acción, las cuales no incluyen recursos, restricciones ni plazas idle.

A continuación se analizarán cada uno de estos casos:

1) En el caso que la estructura de la red presente un IT lineal, es decir no comparte transiciones con ningún otro IT, la responsabilidad de ejecución de este IT es asignada a un único segmento de ejecución.
2) En el caso de que dos (o más) ITs compartan transiciones en conflicto estructural (fork). La responsabilidad de ejecución de los ITs es segmentada. Esto tiene como ventaja que se elimina la lógica interna del hilo para decidir frente a un conflicto, por lo que la decisión del conflicto es tomada por solo por el componente responsable, el cual es la política. La responsabilidad de ejecución de este IT es asignada a distintos segmentos de ejecución; un segmento antes del conflicto (fork) y dos (o más) segmentos posteriores.
3) En el caso de que dos (o más) ITs tengan en sus estructuras una plaza de unión (join). La responsabilidad de ejecución de los ITs es segmentada. Hasta el punto de unión (join) en dos o más segmentos, uno por IT, y después de la unión (join) solo un segmento de ejecución extra. Esto tiene como ventaja que mejora el paralelismo de la ejecución, dado que permite la ejecución de los segmentos posteriores y anteriores simultáneamente.

4.3 Algoritmo para la determinación de hilos máximos por segmento.

Para el cálculo de la cantidad de hilos máximos necesarios por segmento:

1) Obtener los segmentos de los IT de la RdP con el algoritmo de la sección 4.2.
2) Determinar el conjunto de plazas de cada segmento denominado PS_i; con el árbol de alcanzabilidad de la RdP, se debe obtener MS_i, el cual es el conjunto de todos los marcados posibles del iésimo segmento.
3) Del conjunto de marcados MS_i, se debe seleccionar el marcado máximo. Esta será la cantidad máxima de hilos necesarios de ese segmento.
4) Para obtener el número máximo de hilos necesarios el sistema, es necesario sumar los hilos máximos necesarios de todos los segmentos de ejecución.

5 Caso de aplicación y resultados

A continuación se utilizará como ejemplo la red S^3PR del autor Huang [18], para detallar los pasos de aplicación de cada algoritmo. Esta red ha sido modificada ya que en su versión original posee deadlock, es decir existe un estado o marcado a partir del cual la red no puede continuar evolucionando. Para evitar esto, se agregó la plaza de restricción P14 y los arcos correspondientes que se observan en la Fig. 2.
Se comienza con el algoritmo de la sección 4.1:
Las plazas idle de esta red S^3PR son: idle={P1,P8} y los recursos={P6,P7;P12,P13} y restricciones={P14}.

1) Los IT de esta red son:
   IT_1={T1,T2,T4,T6} IT_2={T1,T3,T5,T6} IT_3={T7,T8,T9,T10}.
2) Obtenemos el conjunto de plazas PI de cada IT:
   PI_1={P1,P2,P3,P5,P6,P13,P14} PI_2={P1,P2,P4,P5,P7,P13,P14}

PI_3={P8,P9,P10,P11,P12,P13,P14}

3) Obtenemos el conjunto de plazas de acción PA de cada IT:

PA_1={P2,P3,P5} PA_2={P2,P4,P5} PA_3={P9,P10,P11}

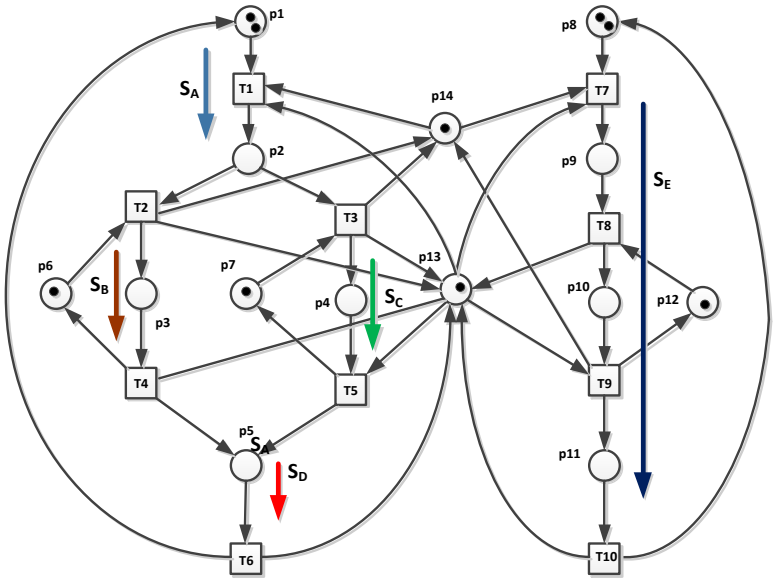

Fig. 2 RdP de ejemplo para aplicación de algoritmos (autor: Huang)

4) Obtenemos el conjunto de estados MA del conjunto de plazas PA donde PA = {P2,P3,P4,P5,P9,P10,P11}, y MA se observa en la Tabla 1.

Tabla 1. La tabla ***MA*** debe enumerar todos los marcados posibles, debido a la extensión de la misma a modo de ejemplo solamente se muestran los 4 primeros estados con su suma.

P2	P3	P4	P5	P9	P10	P11	SUMA
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
0	0	0	0	1	0	0	1
0	1	0	0	0	0	0	1
…	…	…	…	…	…	…	…

De todos los marcados posibles, se busca el marcado máximo, este valor determina la máxima cantidad de hilos activos simultáneos, para el caso de la Fig. 2 son 3 hilos.

A continuación se aplica el algoritmo descripto en la sección 4.2 para determinar los segmentos de ejecución y responsabilidades de cada hilo; éstos se encuentran etiquetados en la Fig. 2 como S_A, S_B, S_C, S_D y S_E.

1) Se observa en la RdP de la Fig. 2 que el IT_3 del sistema cumple la condición caso 1 del algoritmo, por lo cual se define el segmento de ejecución S_E y el subconjunto de plazas de acción asociadas al segmento $PS_E = \{P9, P10, P11\}$.
2) Se observan en la RdP que los IT_1 e IT_2 del sistema cumplen la condición caso 2 del algoritmo, por lo cual se determinan los siguientes segmentos de ejecución y plazas de acción asociadas:
   -Segmento previo al conflicto (fork) como segmento S_A y $PS_A = \{P2\}$.
   -Segmento Izquierdo como segmento S_B y $PS_B = \{P3\}$.
   -Segmento Derecho como segmento S_C y $PS_C = \{P4\}$.
3) Se observan en la RdP que los dos IT mencionados en el punto 2, cumplen además el caso 3 del algoritmo por lo cual se define un último segmento de ejecución luego de la unión (join), como segmento S_D y $PS_D = \{P5\}$.

Una vez determinados los segmentos de ejecución, se determinan los hilos máximos necesarios por segmento de ejecución. Para ello se aplica el algoritmo de la sección 4.3, el cual implica determinar los marcados máximos de los conjuntos de marcados MS_i para las plazas de acción de cada segmento de ejecución PS_i.

Para el caso de la RdP planteado en la Fig. 2 es: $Max(MS_A)$=1, $Max(MS_B)$=1, $Max(MS_c)$=1, $Max(MS_D)$ =1 y $Max(MS_E)$ =1.

Una vez determinados los hilos máximos por segmento, la suma de todos estos determina la máxima cantidad de hilos necesarios del sistema. En el caso de la red de la Fig. 2 es igual a cinco hilos.

Estos resultados han sido validados aplicando ésta metodología en las redes presentadas en [16, 19, 20] entre otras.

6 Conclusiones y trabajos futuros

En este trabajo se diseñaron un conjunto de algoritmos los cuales a partir del modelo de un sistema realizado con una RdP S^3PR no autónoma, determinan la cantidad de hilos necesarios en el software, la responsabilidad de los mismos y la máxima cantidad de hilos activos simultáneos. El diseño de la arquitectura propuesta, conserva las propiedades formales del modelo del sistema y mantiene la capacidad de expresión de una máquina de Turing. Asimismo los hilos y responsabilidades determinadas por este conjunto de algoritmos preservan estas propiedades.

En este trabajo se han validado los algoritmos propuestos con 16 redes distintas del tipo S^3PR, arrojando resultados similares a los obtenidos en el caso presentado. De esta validación es posible aseverar que estos algoritmos son eficientes, debido a que la complejidad computacional es equivalente a la resolución de un sistema de ecuaciones lineales de dimensión igual a la matriz de incidencia, esta es necesaria para determinar los IT de la RdP mientras que la complejidad del árbol de alcanzabilidad de una RdP S^3PR corresponde a una combinación de procesos finitos secuenciales simples. Además son precisos ya que definen sin ambigüedad un proceso para determinar el objetivo de cada uno de ellos; son determinísticos puesto que responden del mismo modo frente a las mismas condiciones y son finitos debido a que se garantiza su finalización. Estos algoritmos son importantes en las primeras fases del diseño de un sistema, dado que el número de hilos y sus responsabilidades

permiten establecer a priori parámetros del hardware, el grado de paralelismo y tiempos de respuesta para la implementación en sistemas embebidos. Esta metodología propuesta automatiza parte del proceso de diseño y generación del código mitigando así los tiempos de desarrollo y los errores de codificación.

Como trabajo futuro se mencionan las principales líneas de investigación para extender la presente metodología: incluir en los algoritmos las métricas de paralelismo y consumo de memoria entre diferentes modelos de un mismo sistema. Asimismo se está trabajando en un conjunto de algoritmos para determinar en forma automática el control sobre las RdP que presentan un estado de deadlock y automatizar la determinación de políticas para el manejo de los conflictos en las RdP.

Referencias

1. Schwab, K., The fourth industrial revolution. 2017: Currency.
2. Halbwachs, N., Synchronous programming of reactive systems. 2013: Springer Science.
3. Munir, A., A. Gordon-Ross, and S. Ranka, Modeling and optimization of parallel and distributed embedded systems2015: John Wiley & Sons.
4. Zeigler, B.P., A. Muzy, and E. Kofman, Theory of modeling and simulation: discrete event & iterative system computational foundations2018: Academic press.
5. Diaz, M., Petri nets:fundamental models, verification and applications2013:John Wiley & Sons.
6. Phd. Micolini O., L.O. Eng. Ventre, and E.N. Eng. Daniele. Modular Petri Net Processor for Embedded Systems. (CACIC 2017) Revised Selected Papers. 2018. Springer CCIS.
7. Phd. Micolini, O., L.O. Eng. Ventre, and M. Eng. Ludemann. Methodology for design and development of Embedded and Reactive Systems Based on Petri Nets. in 2018 IEEE Biennial Congress of Argentina (ARGENCON). 2018. IEEE.
8. Eng. Ventre, L.O. and O. Phd. Micolini, Extended Petri Net Processor and Threads Quantity Determination Algorithm for Embedded System, in Communications in Computer and Information Science CCIS, E.J. Pesado P., Editor 2021, Springer, Cham: CACIC 2020.
9. Micolini, O., ARQUITECTURA ASIMÉTRICA MULTI CORE CON PROCESADOR DE PETRI, in Informatica2015, UNLP: UNLP La Plata, Argentina.
10. PhD. Micolini, O., et al. Ecuación de estado generalizada para redes de Petri no autónomas y con distintos tipos de arcos. in XXII (CACIC 2016).
11. Selic, B. and S. Gérard, Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems2013: Elsevier.
12. Zhou, M. and N. Wu, System modeling and control with resource-oriented Petri nets. Vol. 35. 2018: Crc Press.
13. Siewert, S., Real time embedded components and systems2016: Cengage Learning.
14. David, R. and H. Alla, Discrete, continuous, and hybrid Petri nets 2010, Springer Science.
15. Micolini, O., Phd Thesis:Arquitectura asimétrica multicore con procesador de Petri, 2015: La Plata.
16. Liu, G. and K. Barkaoui, A survey of siphons in Petri nets. Information Sciences, 2016.
17. Peter A. Buhr, M.F., Monitor Classification. ACM Computing Surveys 1995. 27(1):
18. Huang, Y.S., Y. Pan, and P. Su, Transition-based deadlock detection and recovery policy for FMSs using graph technique. ACM Transactions on Embedded Computing Systems, 2013.
19. Timotei, A. and J.M. Colom. A New Approach to Prevent Deadlock in S3PR Nets with Unreplicable Resources. in ICORES. 2013.
20. Zhong, C.F. and Z.W. Li, Design of liveness-enforcing supervisors via transforming plant petri net models of FMS. Asian Journal of Control, 2010. 12(3): p. 240-252.