## Synthesis of Concurrent Programs Based on Supervisory Control

* Marian V. Iordache
  + School of Engineering and Eng. Tech.
  + LeTourneau University
* Panos J. Antsaklis
  + Department of Electrical Engineering
  + University of Notre Dame

Technical Report

September 2009

## Resumen

* Este documento describe una aplicación de los métodos de control de supervisión (SC) a la síntesis de programas concurrentes y presenta el trabajo actual sobre este tema de investigación.
* En En particular, se presta especial atención al desarrollo de software que aplica el SC al programa síntesis.
  + Este trabajo está motivado por las dificultades encontradas para escribir programas correctos en el contexto de la concurrencia. Escribir programas correctos es esencial para el desarrollo de software así como para todas las demás aplicaciones de ingeniería en las que se utilizan lenguajes formales para el diseño del sistema.
  + En el contexto de la concurrencia, el SC puede ayudar abordando cuestiones como la exclusión mutua, la vida y la justicia.
* En el enfoque propuesto aquí, SC se aplica a Petri netos (PN) de procesos concurrentes. Entonces, la lógica de control resultante se convierte en código.
  + Los PN son modelos formales desarrollados en la informática para el modelado de sistemas concurrentes.
  + En los sistemas de control, los PNs han sido utilizados en el contexto del SC de sistemas de eventos discretos y se han desarrollado poderosos resultados teóricos.
* Sin embargo, estos resultados aún no han sido aplicado a los problemas de la informática para los que se crearon los PN.
* El principal objetivo de este El trabajo de investigación es aplicar las herramientas de SC a la síntesis automática de código de programación basado en un especificación del programa de alto nivel.
* El objetivo es reducir el esfuerzo de programación teniendo más de los requisitos de nivel superior implementados automáticamente.
* Por un lado, el sistema automático el código generado es correcto por construcción y por otro lado, el programador sólo tiene que manejar especificaciones de alto nivel más simples.

## Visión general

La programación es esencial en el diseño del sistema de ingeniería contemporáneo. Sin embargo, se sabe que el desarrollo de programas correctos es difícil y costoso, especialmente en el contexto de la concurrencia. Por lo tanto, las herramientas que pueden automatizar el proceso de programación en mayor grado son de interés, para reducir el esfuerzo de programación y aumentar el número de características del producto que son correctas por construcción. Este documento presenta un proyecto que trata de la aplicación de los métodos de control de supervisión a la síntesis de programas que son correctos por construcción.

La teoría del control supervisor (SC) ha sido desarrollada en el contexto de los sistemas de control, para los sistemas de eventos discretos (DES). Dado un modelo de DES, llamado planta, y una especificación, los métodos SC se aplican para diseñar un supervisor que restrinja el funcionamiento de la planta para que se cumpla la especificación. El supervisor se asegura de que el funcionamiento de la planta satisfaga en todo momento la especificación, con sujeción a las restricciones impuestas por la planta. Normalmente, estas restricciones se refieren a los acontecimientos que pueden ser controlados u observados por el supervisor.

Las redes de Petri (PN) representan una clase de modelos de DES, que junto con los autómatas se han utilizado en el contexto del SC. Las PNs se utilizan aquí como los modelos DES de las especificaciones y de la planta.

Las PNs son una elección natural para el modelado de la concurrencia y hay numerosos resultados en el SC de las PNs. Por otra parte, también es posible beneficiarse de los resultados del SC obtenidos para los autómatas, ya que los autómatas son un caso especial de PNs .

El enfoque de este proyecto es aplicar el SC para generar automáticamente segmentos de código de programación. El enfoque se ilustra en la figura 1. En primer lugar, la planta y la especificación del SC se extraen de una especificación de nivel superior, escrita en un lenguaje de especificación de alto nivel (HLL). A continuación, el SC se aplica para generar el supervisor. Finalmente, la planta y el supervisor se traducen a código de programación. Todos estos pasos se llevan a cabo de forma transparente y automática, basados en la especificación de alto nivel. El beneficio de este enfoque es que el programador se centraría en descripciones concisas de alto nivel, en lugar de los detalles de la implementación de nivel inferior, que es más compleja. Por supuesto, no todos los tipos de especificaciones pueden ser manejados por un enfoque SC.

Sin embargo, el problema de la automatización de la síntesis de programas de alto nivel plantea cuestiones intrínsecamente relacionadas con la SC, ya que varios requisitos de alto nivel, como la equidad, la ausencia de bloqueos y la exclusión mutua, pueden considerarse especificaciones de la SC.

|  |
| --- |
|  |
| Figura 1: Una planta DES y una especificación SC se extraen de la especificación del usuario. La especificación SC describe el objetivo de la supervisión. Después de aplicar las herramientas SC, el DES resultante de bucle cerrado puede ser transformado en código de programación. Mientras que aquí nos centramos en la síntesis de programas, el mismo enfoque podría ser utilizado en otros contextos, como la generación de código para la implementación de hardware o la lógica de escalera para la fabricación de aplicaciones. (supervisory control (SC)) |

En lo que respecta a la aplicación de los métodos de SC, observamos que muchos de los problemas que aborda SC son de una dificultad considerable. Además, a veces no hay algoritmos conocidos que sean satisfactorios en todos los aspectos. Por ejemplo, al buscar todos los estados alcanzados por una PN es posible diseñar supervisores que sean óptimos con respecto a varios criterios. Sin embargo, la optimización se realiza a expensas de una gran complejidad de cálculo, debido a que el número de estados alcanzables podría ser extremadamente grande, si es finito. Por lo tanto, puede ser interesante utilizar en su lugar métodos que no busquen los estados alcanzables sino que analicen la estructura de los PN. Sin embargo, tales métodos pueden ser sub-óptimos. Además, pueden ser aplicables sólo a ciertas clases especiales de PNs. Por lo tanto, uno de los objetivos a largo plazo de este proyecto es aplicar diversos métodos de SC y determinar los criterios para seleccionar entre los métodos en función del contexto. A pesar de esas dificultades, habida cuenta de la relación de los problemas de tipo SC con la síntesis de programas, esta labor parece necesaria para lograr un alto grado de automatización en el desarrollo de programas informáticos.

El documento está organizado de la siguiente manera. Los resultados de la literatura conexa se examinan en la sección 2. La representación PN de los programas se describe en la sección 3. Las cuestiones relacionadas con el diseño del lenguaje de especificaciones de alto nivel se abordan en la sección 4. En la sección 5 se da un ejemplo. Los métodos de control de supervisión se tratan en la sección 6. Por último, en la sección 7 se describe un enfoque para la generación de código. Se puede encontrar información adicional sobre este proyecto en [1]. Se remite al lector al apéndice para una introducción a los PN .

## Revisión de la literatura

Debido a la popularidad de los microprocesadores multinúcleo, la programación concurrente es un tema de creciente importancia. Hay herramientas que pueden utilizarse para acelerar la ejecución de código en los procesadores multinúcleo, como [44]. Sin embargo, el desarrollo de programas concurrentes todavía se considera difícil.

La aplicación del SC, tal como se propone aquí, podría ayudar automatizando ciertos aspectos del desarrollo de programas concurrentes, especialmente los aspectos relacionados con la coordinación de tareas concurrentes. Nótese que el desarrollo de especificaciones concurrentes sigue siendo necesario incluso cuando existen buenas herramientas para la ejecución eficiente de código secuencial en sistemas multiprocesadores. Para ilustrar esta necesidad, consideremos el ejemplo de los filósofos del comedor[[1]](#footnote-1). Una posible solución secuencial sería implementar un protocolo de anillo de fichas en el que el filósofo que tiene la ficha pueda comer.

Una herramienta que convierta el código secuencial en código paralelo podría hacer implementaciones muy eficientes de los procesos asociados con un filósofo que toma los tenedores o come o piensa. Sin embargo, no cambiaría el protocolo: el programa generado seguiría permitiendo que sólo un filósofo comiera a la vez. Por el contrario, si se desarrolla una especificación paralela, se podría permitir que dos filósofos comieran al mismo tiempo. Además, las herramientas que convierten el código secuencial en código paralelo seguirían siendo muy útiles, ya que podrían mejorar el tiempo de ejecución de los segmentos secuenciales de código (como los que se ocupan de recoger los tenedores, comer y pensar). Nótese que nuestro proyecto se ocupa del desarrollo de especificaciones concurrentes y no de la conversión de código secuencial a código paralelo.

### Síntesis de Programas Concurrentes Basada en SC

En la literatura, estrechamente relacionado con la teoría SC está el enfoque para la síntesis de programas para sistemas reactivos, como en [69, 90] y las referencias en él. El problema es sintetizar un programa basado en una especificación descrita en la lógica temporal. La especificación debe ser satisfecha para todas las posibles entradas del entorno. En términos de la terminología de SC, un programa correspondería a un supervisor.

En su mayor parte, este trabajo de síntesis de programas aún no se ha reducido a la práctica [68]. En este proyecto usamos PNs en lugar de la lógica temporal para las especificaciones de SC. Sin embargo, a más largo plazo, podrían incorporarse también otros tipos de métodos, según proceda, para aumentar el ámbito de aplicación de este proyecto.

Se ha realizado una cantidad significativa de trabajo en el modelado y análisis de programas concurrentes utilizando PNs, como en [14] y las referencias en él. Se ha creado una herramienta de software PEP para el desarrollo, verificación y simulación de programas paralelos [2, 45, 99]. La estructura de la herramienta PEP ha sido descrita en tres niveles: el nivel de desarrollo, el nivel neto y el nivel de análisis. En el nivel de desarrollo la entrada del usuario se acepta en forma de programas escritos en uno de los lenguajes B(PN)2 [16] y SDL [39] o como autómatas finitos paralelos [46]. En el nivel de red, las especificaciones se representan como redes de alto nivel, expresiones de cálculo de caja de Petri [14], y PN de bajo nivel. Las redes M [15, 17] representan las redes de alto nivel y los PN seguros2 los PN de bajo nivel. En el nivel de análisis, la verificación del modelo y otros métodos de verificación pueden aplicarse a los PN de bajo nivel.

Comparando nuestro enfoque con el enfoque de la herramienta PEP, observe que la entrada está descrita por un lenguaje de especificación en nuestro trabajo y por un lenguaje de bajo nivel en PEP. Nuestro enfoque podría utilizarse para ayudar al programador a escribir una especificación de bajo nivel, mientras que la herramienta PEP puede utilizarse para verificar una especificación de bajo nivel.

El problema de programación, que trata del orden de ejecución de las tareas basado en una especificación concurrente, ha sido abordado en base a los modelos PN en [33, 32, 55, 73, 77, 78, 95, 96, 103, 114]. Típicamente los resultados están en la ejecución secuencial de programas concurrentes, como es el caso en las plataformas con un solo recurso de ejecución. Sin embargo, también se ha considerado la ejecución paralela [33, 73].

También hay resultados para las especificaciones en tiempo real, basados en un enfoque para la síntesis de controladores híbridos [7] y la heurística [55]. Típicamente se utiliza el análisis de alcanzabilidad para la síntesis, aunque también hay resultados estructurales, como en [78].

Nótese que un schedule puede ser visto como la política de supervisión aplicada por un supervisor del DES, como en [100]. Por lo tanto, estos resultados podrían aplicarse en el marco del SC de nuestro proyecto. Sin embargo, tenga en cuenta que en nuestro proyecto el SC debe ser usado para ayudar al programador a escribir programas concurrentes, no sólo para resolver el problema de programación basado en un programa concurrente dado. Además, la intención es centrarse en los métodos estructurales de SC, en un intento de evitar el problema de la explosión de estado de los métodos basados en la accesibilidad.

También se está trabajando en el análisis de tareas de programas escritos en el lenguaje Ada. La verificación basada en el modelado PN aparece en [13, 20, 82]. Hay tanto métodos estructurales [13] como enfoques basados en la accesibilidad [82, 83, 108]. Los métodos que podrían ser usados para modelar programas concurrentes con PNs aparecen en [97, 109]. Nótese que nuestro proyecto está en la síntesis correcta por construcción, no en la verificación.

Relacionado está también el trabajo sobre el diseño de código de hardware/software de [9, 10]. Allí, la especificación está escrita en un lenguaje como Esterel [49], del cual se extrae una red de máquinas de estados finitos de diseño en código (CFSM) [29]. Es interesante notar que las redes de CFSM podrían ser modeladas por PNs seguras. En comparación con nuestro proyecto, mientras que los procesos individuales serán modelados por máquinas de estado, la composición paralela de los modelos de proceso será una PN no necesariamente segura. Además, pretendemos que la PN segura 2A sea una PN en la que todas las marcas sean menores o iguales a una.

Síntesis de los programas concurrentes basados en el SC 4 para utilizar las especificaciones a un nivel más alto. En nuestro proyecto, el lenguaje de especificaciones es para permitir algunas características del lenguaje declarativo, ya que parte del código de nivel inferior debe ser generado automáticamente.

Recientemente, se han realizado trabajos de investigación para detectar y corregir situaciones de bloqueo en el software basado en modelos PN y métodos para la aplicación de la vida en los PNs [110]. El enfoque puede describirse de la siguiente manera. Dado un programa, se extrae un modelo de PN. Luego, se genera un supervisor de aplicación de la vida. Finalmente, el supervisor de aplicación de la vida se implementa mediante líneas de código adicionales en el programa original. Este enfoque ha sido implementado en la herramienta de software GADARA. Al igual que en nuestro enfoque, se aplican los métodos de SC para la aplicación de la vida. Sin embargo, en nuestro proyecto nos ocupamos de la síntesis del programa en lugar de programas ya escritos. Por lo tanto, en nuestro proyecto SC se aplica no sólo para la aplicación de la vida, sino también para automatizar la generación de código para otros requisitos que pueden ser expresados en términos de especificaciones SC.

Mientras que la síntesis de programas es en general un tema difícil, las restricciones de tiempo real presentan un desafío adicional para las aplicaciones integradas. La cuestión del tiempo se percibe como un posible obstáculo en el desarrollo de la próxima generación de sistemas incorporados en red [72]. El marco de trabajo de SC parece ser adecuado para la programación con restricciones en tiempo real. Sin embargo, en la actualidad nos centramos en las especificaciones no temporales y aplazamos las especificaciones en tiempo real para trabajos futuros. En la literatura, se han utilizado PNs cronometrados para la verificación [7, 21, 41] y también para SC [27, 42, 94, 93].

Se han propuesto numerosos métodos de SC para la supervisión de los PN. Se han publicado documentos de encuesta [52, 61]. Los métodos de SC difieren en las suposiciones que hacen, el tipo de especificaciones que consideran y la complejidad de los cálculos. Así pues, el contexto determina cuáles son los métodos más apropiados. Por lo tanto, con el fin de explotar mejor el poder de los resultados del SC, el paso del SC en nuestro enfoque (Figura 1) no se limitará a un solo enfoque del SC. Aunque nuestro enfoque actual se centra en los enfoques de SC de [62], la incorporación de otros métodos de SC es de interés para el futuro.

Hasta cierto punto, la aplicación de los métodos de SC a la ingeniería de software ha sido considerada durante algún tiempo [74, 75]. Algunos de los métodos propuestos en el CS, como el control de predicados para los cálculos distribuidos [106] y los enfoques de programación antes mencionados, pueden considerarse métodos de SC [64].

Además, algunos de los enfoques utilizados para generar programas informáticos de control están relacionados con el SC. Así, un enfoque para la generación de software de control basado en modelos de sistemas de condiciones aparece en [8, 51, 98]. Dado un modelo de sistema de condiciones y un lenguaje de especificaciones que describe una secuencia de estados que el sistema debe seguir, el software de control se genera automáticamente [4]. Los problemas específicos del SC en este contexto se abordan en [48]. El software de control también puede generarse utilizando la herramienta Supremica [65, 5] basada en especificaciones y métodos de autómatas finitos. Nótese que en nuestro proyecto, usando modelos PN, es posible tomar ventaja de ambos métodos PN y métodos de autómatas, ya que los autómatas representan el espacio de alcanzabilidad de los PN. Además, en comparación con [8, 51, 98], tenemos la intención de utilizar especificaciones más generales.

Entre las herramientas de software desarrolladas para SC, se pueden mencionar las siguientes. De especial interés para este proyecto es la caja de herramientas de SC [57] desarrollada en Matlab, que contiene métodos de SC para PNs. Esta caja de herramientas ha sido convertida a C y se ha incluido en el software desarrollado para este proyecto.

También hay otros trabajos sobre herramientas de software para el SC de PNs [3, 40]. Además, la herramienta del sistema de condiciones de [51, 4] también puede ser vista como una herramienta de software para PN, debido a la relación de los sistemas de condiciones con las redes P/T. Algunas de las herramientas de SC que dependen de los modelos de autómatas son TCT [111], UMDES [70], y Supremica [5]. La síntesis de controladores basada en especificaciones lógicas temporales también puede ser realizada [47].

## Petri Net Representación de los programas

Dado que este proyecto implica métodos SC para PNs, de especial interés aquí es la forma precisa en que los programas son representados usando PNs. La subsección 3.1 describe varias formas en las que los PNs podrían representar programas. En la subsección 3.2 se incluye una comparación de PNs y autómatas. Luego la forma específica en que los programas se representan en nuestro proyecto se describe en la sección 3.3 .

### Posibles enfoques para la modelización de PN

Hay varias maneras de obtener modelos de PN.

* Extraer un modelo PN de un programa de bajo nivel.
  1. Extraer un modelo PN completo de un programa.
  2. b) Extraer sólo la estructura del programa como estructura PN .
* Comenzar con un modelo PN.
  1. Empiece con una estructura PN en la que los segmentos de programa estén asociados a cada lugar.
  2. Desarrollar un lenguaje de especificaciones compatible con una estructura PN.

El enfoque 2(b) es el que pretendemos seguir. El enfoque 2(a) también es de interés, como se verá más adelante. A continuación, examinaremos con más detalle estos cuatro enfoques y su relación con este proyecto.

Con respecto al enfoque 1(a), nótese que los PN finitos no pueden modelar completamente programas arbitrarios, ya que no tienen la potencia de una máquina de Turing. Los programas con un número limitado de estados, así como algunos programas con un número ilimitado de estados, pueden ser modelados por PNs finitos .

Para ciertos lenguajes de programación, un enfoque que permite convertir programas en PNs seguros está disponible [14] e implementado en la herramienta PEP [2]. El PN es finito si las variables están definidas en un dominio finito [16]. El enfoque 1(a) no debe ser confundido con el enfoque 2(b), que trata de un lenguaje de especificación de alto nivel (HLL), no con un lenguaje de programación de bajo nivel.

Recordando la distinción entre ambos, un HLL describe objetivos (el qué), mientras que un lenguaje de bajo nivel describe implementaciones objetivas (el cómo).

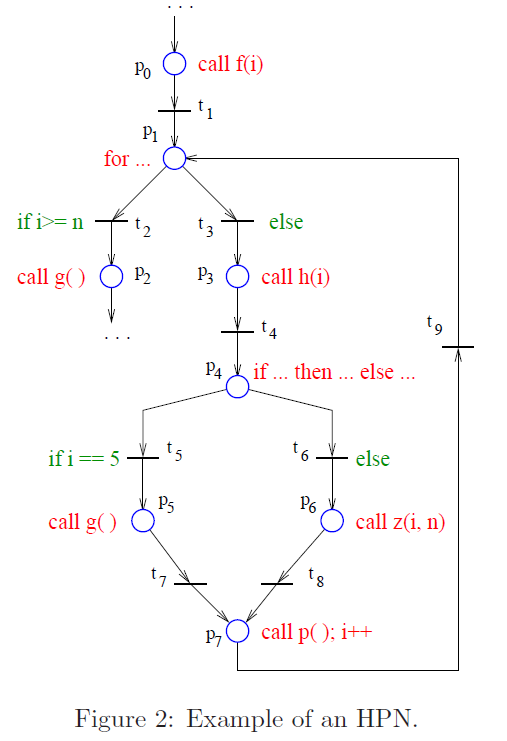
Se obtienen modelos de PN menos complejos si sólo se extrae la estructura del programa. Este es el enfoque 1(b). En este enfoque, tratamos con un PN de alto nivel (HPN) en el que los lugares y/o transiciones están etiquetados por las operaciones que representan. Se podrían representar tanto la ejecución secuencial como la concurrente de programas, donde esta última se puede obtener del gráfico de precedencia.

(Los gráficos de precedencia se discuten, por ejemplo, en [89].) Este enfoque de modelización se ha utilizado para representar programas C concurrentes en [77] y el comportamiento de las tareas de los programas Ada en [13, 82, 97] .

En la Figura 2 se muestra un ejemplo de un HPN.

Otra posibilidad es escribir desde el principio el programa en un formato HPN en el que los lugares se asocian con segmentos de programa. Este es el enfoque 2(a). Para las máquinas de estado, esta idea se ha implementado hasta cierto punto en herramientas como Stateflow de Matlab y en lenguajes de especificación como SDL y UML. Esta idea es atractiva porque algunos problemas se resuelven fácilmente definiendo primero los estados y las transiciones entre los estados. Por ejemplo, esos problemas pueden surgir en el contexto de los protocolos de comunicación. Nótese que no sólo las máquinas de estados sino también los PN son de interés en los problemas de los protocolos de comunicación [34, 105, 87, 18] .

En este proyecto se utiliza el enfoque 2 b). Basándose en una especificación escrita en un HLL, se extraen un modelo de PN y una especificación de SC (Figura 1). Como se mencionó en la sección 4, el HLL puede contener segmentos de código de nivel inferior. Tales segmentos de código se asociarán con lugares de la PN extraída. En este sentido, el modelo PN puede considerarse como uno de los HPN del enfoque 2(a). En la sección 3.3 se ofrece una descripción detallada de la representación del PN utilizada en este proyecto.



## Otros modelos DES

Este proyecto implica el uso de modelos PN para el SC. La mayoría de la literatura sobre el SC utiliza autómatas para los modelos DES. La elección de los PN para este proyecto tiene varias ventajas. En primer lugar, los PN son modelos naturales de procesos concurrentes. Además, al usar PNs, es posible aprovechar no sólo los métodos eficientes de los autómatas sino también los métodos de PN. Un modelo PN puede ser convertido en un autómata equivalente por medio del gráfico de accesibilidad. A la inversa, los autómatas pueden considerarse como un tipo particular de PN seguro. Sin embargo, los PN suelen ser representaciones considerablemente más compactas de los sistemas que los autómatas. De hecho, para algunos PN los autómatas equivalentes no son ni siquiera finitos. Por lo tanto, en principio, puede ser posible tener un método de PN de complejidad exponencial que sea más rápido que un método de complejidad polinómica en el autómata equivalente. Cabe mencionar también que, por lo general, los modelos de PN son más fáciles de obtener que los modelos de autómatas, ya que la composición de los componentes de los autómatas podría considerarse como la generación del gráfico de accesibilidad de un PN.

Los PN seguros son PN cuyo vector de marcado es un vector binario (tiene sólo 0 y 1 elementos).

Cualquier autómata o colección de autómatas puede considerarse como un PN seguro. En lugar de utilizar PNs seguros, en este proyecto se utilizan las redes de P/T más generales. De esta manera, el marcado del PN podría utilizarse para modelar variables enteras no negativas, en las que no es necesario delimitar las variables. En las redes de PN seguras la modelización de variables delimitadas es más compleja y no es posible la modelización de variables no delimitadas. Por ejemplo, una variable que iguala la diferencia entre el número de piezas que entran en un sistema de fabricación y el número de piezas procesadas con éxito es ilimitada.

Una variable de este tipo no puede ser modelada en un PN seguro o en un autómata finito, aunque puede ser fácilmente representada por la marca de un lugar de un PN.

### Representación interna de los programas

Esta subsección describe la forma precisa en que los PNs se usan para representar programas en este proyecto. En este proyecto, un programa consiste en un número de procesos que se ejecutan simultáneamente. La estructura de cada proceso está representada por un PN. Los lugares de la PN corresponden a las operaciones realizadas por el proceso. Las transiciones pueden ser etiquetadas por condiciones, indicando qué transición debe tomarse cuando hay una elección. Cada ficha de PN corresponde a un proceso e indica el estado actual del proceso. A medida que un testigo se mueve de un lugar a otro, la ejecución del proceso progresa de un conjunto de operaciones a otro. Por lo tanto, los distintos lugares del PN corresponden a diferentes etapas de la ejecución del proceso. A lo largo de este documento, un HPN (PN de alto nivel) denotará un PN en el que los lugares están etiquetados con instrucciones y las transiciones con condiciones. En la figura 2 se muestra una ilustración de un HPN.

En general, las transiciones PN pueden tener múltiples lugares de entrada y múltiples lugares de salida. El efecto de disparar tales transiciones se precisa describiendo la estructura del PN por medio de tuplas de la forma (p1, t, p2), (p, t), y (t, p), donde p1, p2, y p significan lugares y t una transición.

* Una tupla (p1, t, p2) indica que el PN tiene un arco de p1 a t y de un arco de t a p2. Además, cuando se dispara la transición t, un proceso en la etapa p1 continúa con la etapa p2.
* Un par (p, t) indica que el PN tiene un arco de p a t. Además, cuando se dispara la transición t, termina un proceso en la etapa p.
* Un par (t, p) indica que el PN tiene un arco de t a p. Además, cuando se dispara la transición t, se crea un nuevo proceso y el proceso comienza en la etapa p.

La descripción anterior puede aplicarse a los PN con pesos arbitrarios, ya que se podrían utilizar arcos repetidos para indicar pesos superiores a uno. Por ejemplo, si una transición t implica las tuplas (p1, t), (p1, t, p2), (p1, t, p2), entonces el arco (p1, t) tiene el peso 3 y el arco (t, p2) tiene el peso 2.

Obsérvese que cuando un lugar p tiene múltiples transiciones de salida, si las transiciones están etiquetadas con condiciones, un proceso en la etapa p seleccionará la siguiente transición a disparar basándose en las condiciones que etiquetan las transiciones. Por el contrario, si las transiciones no tienen condiciones y no hay un código asociado a p para seleccionar la siguiente transición, se dice que el lugar p no es determinístico. Para un lugar no determinístico la elección de la siguiente transición la hace el supervisor.

Las transiciones con un solo lugar de entrada se disparan inmediatamente, a menos que sean controladas por un proceso de supervisión.

Sin embargo, las transiciones controladas por un supervisor o que impliquen más de un lugar de entrada no pueden ser despedidas inmediatamente. Más bien, un proceso envía una solicitud para disparar dicha transición y luego espera el permiso. Una vez que se concede el permiso, el proceso continúa con la siguiente etapa.

Tengan en cuenta que una PN puede tener más de un símbolo. Cada ficha del PN corresponde a una instancia diferente del programa asociado al PN. Tenga en cuenta también que se permiten varios símbolos en el mismo lugar. Esta situación corresponde a múltiples procesos en la misma etapa de ejecución.

|  |
| --- |
|  |
| Figura 3: Los PN que representan programas son una composición de componentes de máquinas de estado. Obsérvese que las transiciones con la misma etiqueta están compuestas. |

Examinando la forma en que los PN se utilizan para representar programas, queda claro que las posibles etapas y transiciones de cualquier proceso forman una máquina de estados. Es decir, para cualquier posición inicial dada de un testigo en el PN, una máquina de estados describirá las posibles etapas y transiciones del proceso asociado al testigo. Además, el PN puede verse como una composición paralela de máquinas de estado. Para una definición formal de la composición paralela se remite al lector al apéndice A, en la página 29. En la figura 3 se muestra un ejemplo de composición de los componentes de la máquina de estado en una PN.

Sin perder la generalidad, se supondrá que las PN que describen los procesos tienen una estructura de máquina de estados. A diferencia de la definición típica de las máquinas de estado, nótese que aquí se permiten marcas y pesos de arco arbitrarios. Además, nótese que la composición paralela de las máquinas de estado puede dar lugar a un PN arbitrario, que no es necesariamente una máquina de estado.

## El lenguaje de las especificaciones

Como se ha mencionado anteriormente, en nuestro enfoque (Figura 1) la especificación se da en un lenguaje de especificación de alto nivel (HLL). Esta sección describe los objetivos, las limitaciones y el trabajo relacionado con el diseño de un HLL.

En primer lugar, observe que el resultado de la síntesis del software consiste en una serie de procesos de aplicación y un proceso de coordinación (Figura 4). En la figura 4, el proceso coordinador representa al supervisor generado por medio de SC. Además, los procesos M corresponden a las definiciones de los procesos que figuran en la especificación. El supervisor (coordinador) intercambia mensajes con los demás procesos para asegurarse de que su funcionamiento respeta las limitaciones indicadas en el archivo de especificación. El número de procesos es variable. Algunos procesos pueden terminar y se pueden crear nuevos procesos, como se describe en el pliego de condiciones. La aplicación se inicia con el inicio del supervisor. A continuación, el supervisor inicia otros procesos hijos, según lo determinado en la especificación. Si bien la figura 4 muestra un único proceso coordinador, es posible un enfoque descentralizado o distribuido utilizando los métodos SC correspondientes.

|  |
| --- |
|  |
| Figure 4: Implementation of the specification. |

Nótese que aquí se hace una distinción entre los procesos y los tipos de procesos. Varios procesos pueden tener el mismo tipo de proceso, es decir, el mismo código ejecutable. El código del supervisor y el código de los tipos de proceso se genera como se muestra en la figura 5. Si bien el enfoque que se propone aquí no se limita a un sistema operativo particular o a un lenguaje de bajo nivel, los programas se desarrollan actualmente para Unix utilizando C para el lenguaje de bajo nivel. Nótese que los archivos se generan usando la arquitectura mostrada en la Figura 6.

Como se muestra en la Figura 6, en base a la especificación, se extraen una serie de PNs de alto nivel (HPNs) y una especificación de control de supervisión (SC). Observe que el uso de HPNs en lugar de colocar redes de transición (redes P/T) es necesario debido a que estas últimas no tienen la potencia de las máquinas de Turing.

Así, los procesos están representados por PNs en los que los lugares se asocian con código de bajo nivel y las transiciones con condiciones. Sin embargo, esto significa que las herramientas de síntesis sólo abordan la parte de la especificación expresada por las PN. Por lo tanto, las herramientas SC sólo garantizarían la corrección del subproblema asociado a la estructura PN extraída del programa HLL. Esto se debe a que las herramientas SC no tienen en cuenta las secciones de código de bajo nivel. Las secciones de código de bajo nivel incrustadas en la especificación son simplemente copiadas, según corresponda, a los archivos de salida. En este sentido, nuestro enfoque se asemeja al de otras herramientas de síntesis de programas, como los generadores de analizadores léxicos y los generadores de analizadores [6, 76].

Mientras que el HLL permitirá secciones de código de bajo nivel, el HLL tiene que proporcionar otras formas de especificar las partes del software que son difíciles de escribir manualmente, para que se generen automáticamente. Así pues, en el contexto de la programación concurrente, la HLL tiene que abordar las diversas limitaciones de sincronización que puedan ser necesarias.

El papel del HLL es permitir programas que sean a la vez compactos y muy legibles. La HLL debería permitir a los usuarios que no están familiarizados con los PNs generar fácilmente el código correcto. Además, se espera que una especificación escrita en la HLL sea considerablemente más compacta que la representación PN de la especificación y mucho más compacta que el resultado de los pasos de generación de código y SC.

De hecho, la especificación de alto nivel no detallaría la forma de aplicar requisitos como la exclusión mutua o la vivacidad. Esos detalles serían manejados por las herramientas de SC. De este modo, el usuario se centraría más en lo que debe hacerse y menos en cómo debe hacerse. Además, como la especificación de alto nivel es más compacta, el programador tendría menos código para comprobar si hay errores.

Para describir explícitamente la representación PN de un programa, se definió un lenguaje de especificación de bajo nivel (LLS). El LLS describe cómo se representan internamente los programas de especificación en nuestro software.

Además, el LLS es importante para propósitos de depuración. También proporciona una forma de incluir descripciones de bajo nivel en las especificaciones de HLL. En comparación con la HLL, la LLS proporciona una descripción explícita de los HPN que representan la especificación y una descripción explícita de las restricciones.

Por lo tanto, una especificación HLL debería ser considerablemente más compacta que una especificación LLS. Es probable que las especificaciones de HLL se traduzcan a especificaciones de LLS. La relación entre el HLL y la LLS es similar a la relación entre un lenguaje de alto nivel y un lenguaje ensamblador.

En la figura 7 se muestra una descripción simplificada de la LLS. El usuario define los tipos de proceso, donde un tipo de proceso puede ser externo o interno. Nótese que el código se genera sólo para los procesos internos.

Los procesos externos describen las restricciones en el funcionamiento de la aplicación, tales como las restricciones impuestas por el hardware. Es importante definir todas las restricciones. Por un lado, esto puede simplificar las operaciones realizadas por los métodos SC. Por otro lado, permite a SC proporcionar soluciones que evitan todas las posibilidades de bloqueo, ya que se sabe que las restricciones pueden crear estados de bloqueo.

|  |
| --- |
|  |
| Figure 5: How the software synthesis tools are applied. |

Cada tipo de proceso está descrito por una estructura de red de Petri de alto nivel (HPN). En esta estructura, los lugares se asocian con las operaciones (como las llamadas a funciones). Además, cuando un lugar tiene múltiples transiciones de salida, las transiciones se asocian con condiciones (como las condiciones en una declaración de "if-then-else").

La LLS permite declarar procesos o grupos de procesos. En un grupo de procesos, cada proceso comparte el mismo HPN. Tenga en cuenta que cada proceso de un grupo de procesos es un testigo en la HPN. Por lo tanto, una declaración de proceso implica una HPN con una sola ficha y una declaración de grupo de procesos implica una HPN con varias fichas. Las marcas iniciales de los HPN se definen explícitamente. Aunque en este documento no se utilizan hilos, es posible implementar un grupo de procesos como un único proceso que consiste en varios hilos.

Como se ha mencionado anteriormente, la estructura del HPN de un tipo de proceso se describe mediante una enumeración de tuplas de la forma (p1, t, p2), (p, t) y (t, p). En el Apéndice B, en la página 31, figura un ejemplo de una especificación descrita en el lenguaje de bajo nivel. Obsérvese que al describir las estructuras de las PN como una enumeración de tuplas (p1, t, p2), (p, t), y (t, p), la PN se describe como una composición de componentes de máquinas de estado (por ejemplo, la figura 3).

|  |
| --- |
|  |
| Figure 6: Internal structure. HPN stands for high level PN. |

### Ejemplo

Supongamos que el HLL describe el software de control de un proceso de montaje en una aplicación de fabricación.

En este proceso dos componentes A y B se ensamblan en un componente C, de la siguiente manera.

Un robot toma una parte A y la coloca en un transportador, si el transportador está parado y ninguna otra parte A está en el transportador. Otro robot toma una parte B y la coloca en el transportador en el mismo lugar si el transportador está detenido y no hay otra parte B. Entonces, las dos partes A y B se ensamblan. Luego, después de que se enciende el transportador y se retira el producto ensamblado, puede comenzar un nuevo ciclo. El transportador no debe moverse desde el momento en que se coloca una parte A o B hasta el momento en que se ensamblan las piezas.

Refiriéndose a la figura 1, la especificación del SC corresponde a los requisitos de que sólo se coloque una parte A (B) en el transportador, que las piezas se coloquen cuando el transportador esté detenido y que el transportador no se mueva desde el momento en que se coloca una parte A o B hasta el momento en que se ensamblan las piezas. Por lo demás, la especificación describe la secuencia de procesamiento y corresponde a la descripción de la planta.

La función de la herramienta de análisis es extraer un modelo de PN de la planta y la especificación de SC basado en una descripción formal de la especificación anterior. Una posible solución es el modelo PN de la planta que se muestra en la figura 8 y la especificación SC que se da en las desigualdades (1)-(4). En el modelo de planta de la Figura 8, la secuencia de procesamiento se muestra a la izquierda y los estados del transportador a la derecha. Para incorporar el efecto de los retrasos en el procesamiento, un paso de procesamiento se modela por una transición controlable, una transición incontrolable y un lugar, como se muestra en la Figura 9. La transición controlable se dispara cuando se emite el comando y la transición incontrolable se dispara después de que se ha ejecutado el comando. Por ejemplo, en la figura 8, t1 se dispara cuando se emite el comando de encender el transportador y t2 se dispara cuando el transportador está encendido.

Las desigualdades de la figura 9, en el marcado del PN, expresan los requisitos restantes de la especificación.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Figure 8: Plant model. | Figure 9: Desigualdades | |

La desigualdad (1) expresa el requisito de que sólo se coloque una parte A en el transportador. Además, (2) describe el requisito de que el transportador no se mueva desde el momento en que se coloca una parte A hasta el momento en que se ensamblan las partes A y B. Las desigualdades (3) y (4) expresan los requisitos similares para las partes B.

|  |  |
| --- | --- |
| Las desigualdades (1)-(4) corresponden a la especificación PN que se muestra en la figura 10. El PN que representa la especificación tiene la propiedad de que la composición paralela de la planta con la especificación satisface los requisitos deseados. Obsérvese que las transiciones de la especificación y la planta que están marcadas con el mismo símbolo corresponden al mismo evento. |  |
| Figure 10: PN representing the SC specification |

|  |  |  |
| --- | --- | --- |
| Obsérvese que las desigualdades (1)-(4) no son la única forma de expresar la especificación SC. El siguiente sistema de desigualdades es equivalente. Sin embargo, es más complejo y da lugar a una representación de PN considerablemente más compleja, debido a las disyunciones [63]. |  |  |
| Figure 11: inequalities equivalent | |

Una dirección de investigación a largo plazo en el desarrollo de la herramienta de análisis es sobre cómo obtener la representación más eficiente de las especificaciones de SC.

En el apéndice B, en la página 31, figura una descripción de la versión de este ejemplo. Allí se describe la planta como la composición paralela de cinco componentes de la máquina de estado, cada uno de los cuales corresponde a un proceso.

## Control de supervisión

En el control de supervisión (SC), un supervisor se diseña en base a un modelo de planta y una especificación.

El supervisor se asegura de que el funcionamiento de la planta satisfaga en todo momento la especificación, con sujeción a las limitaciones impuestas por la planta. En su forma más simple, la supervisión de los PN implica un PN de la planta, como el PN de la figura 8 (11), y una especificación dada en términos de marcado de desigualdades, como las desigualdades (9)-(12). La especificación está escrita de forma compacta en la forma Lμ ≤ d, donde L es una matriz, μ es el vector de marcado, y d es un vector de columna. Entonces, si D es la matriz de incidencia de la PN, el supervisor es una PN de matriz de incidencia Ds = -LD. Por ejemplo, suponiendo la planta de la figura 8 y la especificación dada por las desigualdades (9)-(12), el supervisor que implementa la especificación se muestra en la figura 11, donde el supervisor consiste en los lugares C1, C2, C3 y C4 .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
|  | | |
| Figura 8 | | Desigualdad |  | |

En general, el problema del SC es considerablemente más difícil debido a las limitaciones impuestas por la planta y a las complejas especificaciones. Típicamente, las restricciones impuestas por la planta se refieren a los eventos que pueden ser controlados u observados por el supervisor.

El problema de los filósofos de comedor podría utilizarse como ejemplo. Desde el punto de vista de un agente que controla el acceso a las bifurcaciones, una transición entre los estados "piensa" y "hambriento" es incontrolable e inobservable. Además, una transición entre "comer" y "pensar" es incontrolable pero observable. Además, una transición que modela el acceso a un "tenedor" puede considerarse controlable y observable. La especificación del problema de los filósofos de la comida podría ser que la "inanición" es imposible, es decir, que cualquier filósofo hambriento eventualmente accede a los tenedores.

Las transiciones incontrolables y/o inobservables pueden ser necesarias en cualquiera de los siguientes contextos: - Un entorno descentralizado, en el que las transiciones de una entidad son inobservables e incontrolables para las otras entidades.

- Un entorno de sistema incorporado en el que las transiciones son controlables cuando pueden ser controladas por los actuadores y observables cuando pueden ser detectadas sobre la base de la información de los sensores .

- Una transición asociada a una interrupción puede considerarse incontrolable (como en [31]) .

- Para ciertos problemas de SC (como la aplicación de la vida útil), las transiciones etiquetadas por condiciones deben considerarse incontrolables .

Pueden ser necesarios varios tipos de especificaciones, como la imposición de la habitabilidad o la reversibilidad, la garantía de la exclusión mutua, las limitaciones formales del lenguaje y otras. El problema de la SC suele ser fácil de resolver cuando todas las transiciones son controlables y observables y no se dan requisitos de habitabilidad o reversibilidad. Por consiguiente, la mayor parte de los esfuerzos de investigación se han dirigido a los problemas de SC que implican especificaciones de habitabilidad y/o controlabilidad y observabilidad parciales.

### Métodos de control de supervisión

Se ha dedicado un esfuerzo significativo al desarrollo de métodos para el SC de PNs y autómatas. Nótese que los autómatas finitos son un caso especial de PNs y que el gráfico de alcanzabilidad de un PN es un autómata equivalente (no necesariamente finito). Por lo tanto, los métodos desarrollados para los autómatas son importantes para el análisis de la alcanzabilidad de los PN. El gráfico de alcanzabilidad de un PN puede no ser finito y, cuando es finito, puede tener un tamaño que se relaciona exponencialmente con el tamaño del PN. Por esta razón, se ha trabajado mucho en los métodos de SC que evitan el análisis directo de la alcanzabilidad de los PN. Se dice que tales métodos SC son estructurales, ya que se basan en la estructura del PN más que en el gráfico de alcanzabilidad.

Una descripción de los métodos estructurales disponibles se puede encontrar en los documentos del estudio [52, 61]. Para esta clase de métodos, hay resultados sobre los siguientes tipos de especificaciones:

* + Vivacidad (liveness) (en cualquier estado alcanzable, cualquier transición debe ser eventualmente disparable)
  + limitaciones de seguridad (safety constraints):
    - exclusión mutua (generalizada),
    - restricciones lingüísticas (que requieren que las palabras generadas por la planta pertenezcan al lenguaje especificado),
    - restricciones basadas en el estado (debe evitarse un conjunto de estados prohibidos)
  + ciertos tipos de restricciones de seguridad (certain types of safety constraints) para la supervisión descentralizada/distribuida.

En lo que respecta a la vivacidad, se está trabajando mucho en los métodos para evitar el estancamiento, la prevención y la recuperación. Varios de los principales enfoques aparecen en [113]. En el contexto de las PN, la mayoría de los métodos para hacer cumplir la vivacidad se propusieron para clases especiales de PN, modelando sistemas de asignación de recursos, como en [71, 104, 11, 112, 37, 85, 86, 107, 92]. Entre ellos, [86] también pueden utilizarse para la controlabilidad parcial. Para los modelos generales de PN y/o la controlabilidad y observabilidad parcial el método de [62, 58] es de interés. Este enfoque consiste en un procedimiento que identifica y elimina de forma iterativa las situaciones de bloqueo. Sin embargo, no se garantiza su terminación.

La investigación sobre las limitaciones de seguridad ha dado lugar a métodos para la representación de las especificaciones por los PN y a métodos de aplicación de las limitaciones de seguridad en el marco de la controlabilidad y la observabilidad parciales. Muchos de los métodos de aplicación de las limitaciones de seguridad se refieren a especificaciones de exclusión mutua generalizada. En esas especificaciones se exige que el marcado μ de una PN satisfaga una serie de desigualdades lμ ≤ b, donde b es un número entero y l es un vector entero. Algunos métodos abordan el problema general pero no son óptimos [79, 80, 26]. Los métodos subóptimos pueden generar supervisores no menos restrictivos. Además, es posible que los métodos subóptimos no sean capaces de encontrar soluciones para determinados problemas, incluso cuando existen soluciones. Además, hay otros métodos destinados a clases especiales de PN y especificaciones, que son óptimos, como en [28, 25, 35, 43, 54, 67, 101, 102]. Entre los métodos diseñados para diferentes tipos de conjuntos prohibidos mencionamos [19, 53]. Un enfoque subóptimo para las restricciones del lenguaje PN aparece en [61, 62].

Para la supervisión descentralizada/distribuida, se pueden utilizar los métodos de supervisión de [62, 60]. En base a la especificación dada (centralizada), se obtienen supervisores descentralizados que operan de manera autónoma.

Este enfoque puede utilizarse para los entornos descentralizados que excluyen la comunicación entre supervisores y para los entornos distribuidos en los que se permite la comunicación. Cuando los supervisores dependen de la comunicación para operar correctamente, es necesaria la sincronización distribuida.

No obstante, una solución distribuida puede implicar menos comunicación que una solución centralizada.

### Aplicación de los métodos SC

Refiriéndonos al paso SC de nuestro enfoque (ver Figura 1), el trabajo de implementación ha comenzado con la familia de métodos estructurales de [62]. Esta familia de métodos es general tanto en términos de descripción de la planta como de clase de especificaciones. Además, la síntesis puede llevarse a cabo en entornos generales de incontrolabilidad e inobservancia y muchos de los resultados pueden adaptarse a los entornos descentralizados/distribuidos. No obstante, cabe señalar que un objetivo a largo plazo de este proyecto es aprovechar todos los métodos disponibles, incluidos los basados en la accesibilidad. Por lo tanto, de especial interés para la labor futura es el desarrollo de algoritmos que identifiquen casos especiales para los que se disponga de un enfoque óptimo. En la labor futura se deberían abordar también estrategias para seleccionar entre diversos enfoques sobre la base de su tiempo de terminación estimado.

Entre las especificaciones consideradas en el SC de las PN, las especificaciones de tipo de lenguaje son especialmente importantes, debido a su generalidad. Una especificación de lenguaje está representada por un PN que genera el lenguaje especificado. La especificación PN introduce restricciones en la secuencia en que pueden ocurrir los eventos de la planta.

Estas limitaciones pueden describirse algebraicamente. Se ha demostrado [59] que cada lugar de un PN describe una restricción de la forma hq +cv ≤ b, donde b es un entero, h y c son vectores enteros, y q y v son parámetros relacionados con los disparos de transición. Se denominan vector de disparo (q) y vector de Parikh (v). Se ha demostrado también que las restricciones de la forma son tan expresivas como las restricciones de la forma , donde l es otro vector entero y μ es la marca del PN. Nótese que bajo algunos supuestos de delimitación, los lenguajes PN pueden describir disyunciones [63].

En [62], que describe los enfoques de SC en los que nos hemos centrado, el problema de aplicación del lenguaje se ve como el problema de aplicación de un conjunto de restricciones . La solución se encuentra en base a la solución de un problema de supervisión que implica un PN transformado y un conjunto transformado de restricciones . Por lo tanto, los métodos para las restricciones discutidas en la subsección anterior son muy importantes para las especificaciones generales.

La equidad es una cuestión importante en las aplicaciones de programación. La inanición se refiere a la situación en la que uno o más procesos pueden tener que esperar indefinidamente. Las restricciones podrían utilizarse para describir las restricciones de imparcialidad. Alternativamente, la imparcialidad podría considerarse en el código de nivel inferior. En este caso, un segmento de código de nivel inferior incluido en la especificación HLL describiría cuál de las transiciones habilitadas por el código SC debe dispararse.

Nótese que los controladores DES, en contraposición a los supervisores, parecen ser de mayor interés en las aplicaciones de programación. Mientras que un supervisor desactiva los eventos que pueden conducir a un comportamiento inaceptable de la planta, un controlador selecciona los eventos que deben ser disparados a continuación, mientras se asegura de que la especificación se mantenga satisfecha. Así, los controladores pueden derivarse fácilmente de los supervisores. En este contexto, la cuestión de si el método de diseño de un supervisor es óptimo o no es importante, siempre y cuando el método sea capaz de encontrar una solución cuando ésta exista. De hecho, los controladores harían cumplir una especificación independientemente de si proceden de supervisores menos restrictivos o de supervisores no óptimos. El trabajo sobre el problema de la obtención de controladores de DES aparece por ejemplo en [12, 23, 24, 38, 56].

6.3 Ejemplo Teniendo en cuenta el ejemplo de la sección 5, la especificación SC de la figura 10 no puede utilizarse como supervisor, ya que no tiene en cuenta la incontrolabilidad parcial de la planta. Al tener en cuenta las transiciones incontrolables de la planta, el paso de SC cambia las desigualdades (1)-(4) a (9)-(12), que pueden implementarse como se muestra en la figura 11.

## Generación de códigos

### Formulación del problema y posibles enfoques

Refiriéndose a la figura 1, observe que el papel de la generación de código es implementar tanto la planta como el supervisor en el código. La implementación de la planta implica las siguientes operaciones.

* Escribir el código asociado con los lugares de la HPN e implementar las declaraciones "if-then-else" asociadas con las condiciones que etiquetan las transiciones.
* Redactar el código para la comunicación con el supervisor.
* Escribir el código para las transiciones que implican múltiples lugares de entrada y salida. Dicho código puede implementar la sincronización, la creación de procesos y la terminación de procesos.

El supervisor se utiliza para restringir la operación de una planta PN. El papel del supervisor es identificar las transiciones que pueden ser disparadas sin violar la especificación dada. Nótese que un supervisor no fuerza los disparos de transición. Sólo indica qué transiciones pueden ser despedidas. Una vez que la transición es habilitada por la planta y el supervisor, puede ser disparada inmediatamente. La implementación del software del supervisor implica escribir el código asociado a los eventos de la planta, como los disparos de transición o las solicitudes de permiso para disparar ciertas transiciones.

La implementación en el software del supervisor podría hacerse de más de una manera. Una posibilidad sería utilizar semáforos u otros mecanismos de sincronización para aplicar la política de supervisión. Otra posibilidad sería aplicar la política de supervisión como tarea de coordinación.

Los semáforos son una opción natural para la representación de los puestos de supervisor [50, 52, 66, 115]. Dado que el código se genera automáticamente, las dificultades que surgen en la programación con semáforos no son pertinentes para esta aplicación. Debe tenerse en cuenta que son necesarias algunas extensiones simples de las operaciones de los semáforos, más allá de la sincronización simultánea P/AND [84]. Algunas extensiones necesarias se muestran en la Figura 12.

Los semáforos permitirían a las tareas decidir de forma autónoma qué transición a disparar y cuándo. Sin embargo, un supervisor puede implicar más que un número de lugares conectados a las transiciones de planta. Por lo tanto, una solución más general es utilizar un proceso de coordinación para poner en práctica el supervisor. En este caso los lugares de supervisión podrían ser modelados por variables enteras. Estas variables se pondrían a prueba cada vez que se tomara la decisión de poner en marcha una transición. En la actualidad, este proyecto aplica el enfoque del proceso de coordinación. En el futuro se podría considerar también un enfoque semafórico .

Los enfoques de los procesos de coordinación y de los semáforos se ilustran en un ejemplo que figura en la subsección 7.2 .

Luego, los detalles de la generación de códigos se consideran en la subsección 7.3 .

|  |
| --- |
|  |
| Figura 12: Operaciones regulares de P y V representadas en a) y b). A continuación, c) corresponde a una P simultánea. Se necesitan extensiones simples para las operaciones indicadas en d-f). Nótense los pesos enteros del arco. |

Ejemplo

Refiriéndose al ejemplo de la sección 6.3, el paso de generación de código consiste en generar el código de programación basado en la planta de PN y el supervisor que se muestra en la figura 11. La implementación de un semáforo podría hacerse de la siguiente manera. Cuatro semáforos S1...S4 son necesarios, uno para cada una de las cuatro plazas de supervisor C1...C4. Las operaciones de tipo P en los semáforos se ejecutarían antes de disparar cualquiera de las operaciones de tipo t1, t5 y t7, y las operaciones de tipo V después de disparar t4, t10 y t11. Por ejemplo, P(S1, S2) se ejecuta antes de disparar t5 y V (S1) y V (S3) después de disparar t11. Obsérvese que la sincronización simultánea P/AND [84] se utiliza para la operación P.

|  |
| --- |
|  |
| Figure 11: Plant and supervisor |

Alternativamente, se podría utilizar un proceso de coordinación para implementar el supervisor en lugar de los semáforos. El coordinador utilizaría cuatro variables, m1...m4, cada una de las cuales corresponde a la marca de los lugares de supervisión C1...C4. En este enfoque, el proceso de transporte notifica al coordinador cada vez que se dispara t4 y el proceso de montaje cada vez que se dispara t10 y t11. El proceso de transporte y montaje solicita permiso para disparar t1, t5 y t7 al coordinador.

El coordinador habilita/inhabilita t1, t5 y t7 basándose en la solución de supervisión que se muestra en la figura 11. Por ejemplo, t1 es habilitado por el coordinador cuando m2 ≥ 1 y m4 ≥ 1. Además, si se concede el permiso para disparar t1, los m2 y m4 se disminuyen. 2. De manera similar, si se anuncia el disparo de t4 o t10, se incrementan los m2 y m4 .

## Un enfoque basado en el coordinador

Aquí presentamos un posible enfoque de generación de código que implica un proceso de coordinación. Recordemos que en este proyecto la especificación describe una serie de procesos, así como las limitaciones en el funcionamiento de los mismos. Los procesos definidos en la especificación se denominan procesos de aplicación. Las PN se utilizan para modelar la secuencia de funcionamiento de los procesos de aplicación, donde un proceso está representado por un testigo PN y las etapas del funcionamiento del proceso están representadas por lugares PN. Además, el funcionamiento de los procesos de aplicación está controlado por un proceso supervisor, que puede inhibir o permitir varias transiciones de PN. El objetivo de la generación de código es obtener los programas que implementan los procesos de aplicación así como el código de coordinación.

Los algoritmos de generación de código descritos aquí se desarrollan bajo los siguientes supuestos.

Aunque estas suposiciones no parecen ser restrictivas para las aplicaciones prácticas, pueden simplificar los algoritmos de generación de código.

* A1. Las PN asociadas a los procesos de aplicación son máquinas de estado. (Sin embargo, su composición paralela no tiene por qué ser una máquina de estados).
* A2. Las PNs asociados a los procesos de aplicación tienen etiquetas distintas. (Sin embargo, los PN de procesos diferentes pueden tener etiquetas comunes y el proceso supervisor no tiene que tener etiquetas distintas).

Como se mencionó en la primera suposición, las PN utilizadas para representar la estructura de un programa son una composición de componentes de la máquina de estados (SM). Cada componente de SM corresponde a las posibles etapas de un proceso. Como ejemplo, consideremos la figura 3. En la figura, al componer las transiciones con la misma etiqueta de las SM (a), (b) y (c), se obtiene la PN (d). Cada componente de los SM representa un tipo de proceso, donde el número de token de un componente de SM es igual al número de procesos de ese tipo. Las transiciones se utilizan para trasladar un proceso de una etapa a otra, para terminar un proceso o para iniciar un proceso. Por ejemplo, cuando se dispara la transición de la etiqueta b, se crea un nuevo proceso del tipo c). Además, cuando se dispara la transición del rótulo d, se termina un proceso del tipo c).

En la generación de códigos se presta especial atención a la sincronización de la transición. La sincronización se lleva a cabo mediante un proceso de coordinación. Cuando un proceso de aplicación está listo para disparar una transición t que se sincroniza con otras transiciones, el proceso solicita al coordinador permiso para disparar t. El permiso se concede cuando todos los demás procesos que intervienen en la sincronización están listos. Cada sincronización implica dos o más procesos, en los que uno de los procesos puede ser el proceso de supervisión.

Las transiciones que no estén sincronizadas no pueden dispararse inmediatamente. Sin embargo, puede ser necesario notificar al proceso supervisor cuando se produzcan. Según su relación con el supervisor, las transiciones se clasifican de la siguiente manera.

- Una transición se controla si el supervisor puede inhibir su disparo. Un proceso no puede disparar una transición controlada sin el permiso del supervisor.

- Se observa una transición si el supervisor supervisa sus disparos para actualizar las variables internas. Un proceso notificará al supervisor después de disparar una transición observada.

Obsérvese que las dos propiedades definidas anteriormente son independientes. Sin embargo, la mayoría de las veces también se observará una transición controlada. En términos de la representación PN de los supervisores, las transiciones controladas tienen uno o más lugares de supervisión como entradas y las transiciones observadas están conectadas a uno o más lugares de supervisión. Sin embargo, una transición con lugares de supervisión de entrada no está necesariamente controlada. Esto sucede cuando para todos los estados alcanzables nunca se da el caso de que la transición se habilite la planta y se inhabilite el supervisor. Además, una transición conectada a los lugares de supervisión puede no ser observada. Esto puede ocurrir cuando los lugares de supervisión están controlados por medio de autoenlaces.

Podría utilizarse un algoritmo basado en la accesibilidad para determinar las transiciones que se controlan y observan. Sin embargo, esto sería bastante costoso desde el punto de vista informático. Al aproximarse demasiado a los conjuntos de transiciones controladas y observadas, el rendimiento del código resultante podría verse afectado en cierta medida, ya que aumenta el grado de comunicación entre los procesos y el supervisor.

Sin embargo, como no se prevé una degradación significativa del rendimiento, se utilizará el algoritmo 7.1 para aproximar en exceso los conjuntos de transición controlada y observada. El algoritmo es correcto porque un supervisor nunca intentaría controlar una transición no controlada ni observar una transición no observable.

La siguiente notación de PN es conveniente para los desarrollos posteriores. Si es un lugar o una transición, decimos que

* es el conjunto de lugares de salida o transiciones de salida de
* es el conjunto de lugares de entrada o transiciones de entrada de .
* denote el número de elementos del conjunto .
* Dado que los PN que representan procesos son máquinas de estado, para todas las transiciones y .

Obsérvese que mientras que las máquinas de estado se representan normalmente por PNs ordinarios (PNs en los que todos los arcos tienen un peso de unidad), aquí permitimos pesos enteros arbitrarios.

Además de las transiciones controladas y observadas, se definen las siguientes clases de transiciones.

* Una transición t de un tipo de proceso es una transición de sincronización si satisface al menos una de las siguientes propiedades.
  + t tiene un arco de entrada de peso mayor que uno.
  + t tiene un lugar de entrada y existe una transición t′ de un tipo de proceso diferente, de manera que t′ tiene un lugar de entrada y t y t′ tienen la misma etiqueta.

Además, si dos transiciones de dos procesos de aplicación tienen la misma etiqueta, una de ellas tiene un lugar de salida pero no de entrada, y ninguna es una transición de sincronización, entonces son transiciones de acción. Obsérvese que las transiciones con un lugar de salida y sin lugar de entrada crean nuevos procesos.

|  |
| --- |
|  |
| Algorithm 7.1 Finding Controlled and Observed Transitions |

Así pues, cuando se produce una transición de acción, debe notificarse al proceso coordinador para que genere los nuevos procesos correspondientes.

A continuación se presenta un esquema de las operaciones realizadas cuando un proceso (testigo) entra en la etapa (lugar) p

1. Se ejecutan las funciones asociadas a p.
2. 3. Se selecciona la siguiente transición t.
3. Si es aplicable, el proceso solicita y espera el permiso para disparar t.
4. Si es aplicable, el proceso notifica al supervisor que t es despedido.
5. La transición t es despedida. Hay dos posibles resultados:
   * El proceso termina si t no tiene un lugar de salida.
   * El proceso continúa con la siguiente etapa p′, donde p′ es un lugar de salida de t.

Obsérvese que si un lugar p tiene varias transiciones de salida, la siguiente transición t puede seleccionarse en función de las condiciones asociadas a las transiciones o en función de las operaciones internas realizadas por las funciones asociadas al lugar p. Si hay que obtener el permiso para disparar t de un supervisor, se realiza una llamada a la función del formulario

RequestToFire(t).

|  |
| --- |
|  |
| Figura 13: Ejemplos de transiciones de sincronización y de acción. t1 es una transición de sincronización ya que tiene un arco de entrada de peso 2. t3, t5 y t9 son también transiciones de sincronización. t6, t7 y t8 son transiciones de acción. |

Después de una llamada de RequestToFire, el proceso tiene que esperar hasta que se conceda el permiso. Ahora bien, la llamada RequestToFire(t) aborda el caso en que el proceso selecciona sólo una transición t. Sin embargo, puede ser que un proceso pueda continuar disparando cualquiera de las n transiciones t1, t2, . . . tn. Un ejemplo en el que puede surgir esta posibilidad es cuando un proceso puede tener que completar n operaciones y el orden en que las realiza no es importante. Entonces, cada una de las operaciones podría estar asociada a un lugar alcanzado por el disparo de una de t1, t2, . . . tn. Así, la función RequestToFire puede ser llamada con más de un parámetro:

RequestToFire(t1, t2, ..., tn)

En una llamada con más de un parámetro, la transición que se dispara es seleccionada por el proceso de supervisión. Normalmente, el supervisor seleccionará la primera transición de la lista que esté habilitada o, si todas las transiciones están deshabilitadas, la primera que se habilite.

Obsérvese que el orden de los parámetros indica el orden de preferencia de las transiciones. Al igual que en el caso de un solo parámetro, el proceso que llama a RequestToFire tendrá que esperar hasta que el supervisor dé permiso para disparar una de las transiciones. Una versión no bloqueante de RequestToFire podría ser

Try(t)

Esta función obtendría el permiso para disparar t si t está habilitada o bien informar que t está deshabilitada.

Aunque esta función proporcionaría una flexibilidad adicional para el código que se escribe manualmente, la función RequestToFire es suficiente para el código generado automáticamente. Utilizando una llamada de la forma RequestToFire(t1, t2, ..., tn, t) en la que la última transición t esté siempre activada, la llamada no se bloqueará. Un ejemplo de una transición t que siempre está habilitada es una transición que implementa un auto-ciclo.

Un lugar no determinante es un lugar p en el que se selecciona la siguiente transición mediante una llamada RequestToFire(t1, t2, ..., tn) en la que están presentes todas las transiciones de salida de p. Obsérvese que un lugar no determinístico es un lugar en el que el proceso permite al supervisor hacer la selección de la siguiente transición. La capacidad de controlar la selección de la siguiente transición es importante para los supervisores de la aplicación de la vida.

|  |
| --- |
|  |
| Algorithm 7.2 Structure of the Application Process |

En el Algoritmo 7.2 se describe la estructura del código de un proceso de aplicación. Una descripción del coordinador se da en el Algoritmo 7.3. Para simplificar, todas las funciones de coordinación se asignan al proceso de supervisión. Así pues, el proceso de supervisión y el proceso de coordinación son iguales. Para simplificar, se han omitido los detalles de aplicación en las descripciones de los algoritmos.

|  |
| --- |
|  |
| Algorithm 7.3 The Supervisor (Coordinator) Process |

Varios pasos de los algoritmos podrían realizarse de manera más eficiente.

Se podrían hacer las siguientes observaciones sobre los algoritmos.

* Un tipo de proceso representa la estructura de un proceso. Esto incluye la estructura del PN, el código asociado a cada lugar y las condiciones asociadas a las transiciones. Nótese que un tipo de proceso no define las etiquetas de la estructura PN.
* Para cualquier proceso, el PN que representa su estructura tiene etiquetas distintas. Sin embargo, dos procesos diferentes pueden compartir etiquetas comunes. Además, el PN asociado a un proceso es una máquina de estados, es decir, una transición puede tener como máximo un lugar de entrada y como máximo un lugar de salida.
* Los algoritmos tienen en cuenta que los PN que representan procesos son máquinas de estados y tienen etiquetas distintas. Una excepción es el algoritmo 7.6, que también admite PN que no tienen etiquetas distintas.

|  |
| --- |
|  |
| Algorithm 7.4 The PerformAction Method of the Supervisor Process |

|  |
| --- |
|  |
| Algorithm 7.5 The UpdateSupervisorMarking Method of the Supervisor Process |

|  |
| --- |
|  |
| Algorithm 7.6 The IsPermissible Method of the Supervisor Process |

|  |
| --- |
|  |
| Algorithm 7.7 The SearchQueue Procedure of IsPermissible |

## Conclusión

Notoriamente difícil [22], el desarrollo de programas concurrentes podría simplificarse mediante el uso de herramientas que generen automáticamente parte del código requerido. Este trabajo propone la aplicación de control de supervisión (SC) para la síntesis automática de programas concurrentes.

SC es de interés porque varios requerimientos de alto nivel pueden ser vistos como especificaciones de control supervisor (SC). Por lo tanto, los métodos SC o métodos similares de áreas de investigación relacionadas tienen que ser aplicados para lograr un alto grado de automatización del proceso de programación.

El problema del SC, en su forma general, es de considerable dificultad. No obstante, se han hecho progresos significativos y ya se dispone de resultados potentes. El proyecto que se describe en este documento tiene por objeto desarrollar un software para la síntesis de programas que aproveche los métodos de SC disponibles.

En este proyecto, basado en una especificación escrita en un lenguaje de especificaciones de alto nivel (HLL), se formula un problema de SC y luego se resuelve utilizando métodos de SC. El resultado del paso de SC se convierte entonces en código de bajo nivel. El proyecto implica el trabajo en el HLL, la traducción de las especificaciones HLL a especificaciones SC, la traducción de las políticas de supervisión a código de bajo nivel, y los métodos SC.

Tres de las características novedosas de este proyecto son las siguientes. Primero, proponemos aplicar los resultados SC desarrollados para los modelos de la red de Petri (PN) a la síntesis de programas. Nótese que al usar los modelos PN es posible recurrir también a los resultados de los autómatas. En segundo lugar, la traducción de una especificación de programación de alto nivel a una especificación SC es también un nuevo tema. En tercer lugar, para el SC proponemos un marco en el que se pueden utilizar múltiples métodos, en función del contexto, en un esfuerzo por explotar los puntos fuertes de cada método.

## References

[1] A Concurrency Tool Suite. http://www.letu.edu/people/marianiordache/acts.

[2] PEP homepage. http://parsys.informatik.uni-oldenburg.de/∼pep.

[3] PNetLab homepage. http://www.prisma.unina.it/automation/frame.htm.

[4] Spectool homepage. http://www.engr.uky.edu/∼holloway/spectool.

[5] Supremica homepage. http://www.supremica.org.

[6] A. V. Aho, R. Sethi, and J. D. Ullman. Compilers: Principles, Techniques, and Tools. Addison-Wesley, Reading, Massachusetts, 1986.

[7] K. Altisen, G. Gosler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In 20th IEEE Real-Time Systems Symposium (RTSS’99), pages 154– 163. IEEE Computer Society, 1999.

[8] J. Ashley and L.E. Holloway. Automated control, observation, and diagnosis of multi-layer condition systems. Studies in Informatics and Control, 16(1), 2007.

[9] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara. Hardware-Software Co-Design of Embedded Systems: The Polis Approach. Kluwer Academic Publishers, 1997.

[10] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, A. Sangiovanni- Vincentelli, E. Sentovich, and K. Suzuki. Synthesis of software programs for embedded control applications. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 18(6):834–849, 1999.

[11] Z. Banaszak and B. Krogh. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. IEEE Transactions on Robotics and Automation, 6(6):724– 734, 1990.

[12] M. Barbeau, M. Frappier, F. Kabaza, and R. St.-Denis. A supervisory control synthesis case study: The antenna control system. In Proceedings of 1997 Allerton Conference on Communication, Control, and Computing, pages 533–542, 1997.

[13] K. Barkaoui and J.-F. Pradat-Peyre. Verification in concurrent programming with Petri nets structural techniques. In High-Assurance Systems Engineering Symposium, pages 124–133, 1998.

[14] E. Best, R. Devillers, and M. Koutny. Petri nets, process algebras and concurrent programming languages. In Reisig, W. and Rozenberg, G., editors, Lectures on Petri Nets II: Applications, volume 1492 of Lecture Notes in Computer Science, pages 1–84. Springer-Verlag, 1998.

[15] E. Best, W. Fraczak, R. Hopkins, H. Klaudel, and E. Pelz. M-nets: an algebra of high-level petri nets, with an application to the semantics of concurrent programming languages. Acta Informatica, 35(10):813–857, 1998 [16] E. Best and R. Hopkins. B(PN)2 - a basic Petri net programming notation. In Bode, A., Reeve, M., and Wolf, G., editors, PARLE, volume 694 of Lecture Notes in Computer Science, pages 379–390. Springer-Verlag, 1993.

[17] E. Best and C. Palamidessi. Linear constraint systems as high-level nets. In Montanary, U. and Sassone, V., editors, CONCUR, volume 1119 of Lecture Notes in Computer Science, pages 498–513. Springer-Verlag, 1996.

[18] J. Billington. Protocol specification using P-graphs, a technique based on coloured Petri nets.

In Reisig, W. and Rozenberg, G., editors, Lectures on Petri Nets II: Applications, volume 1492 of Lecture Notes in Computer Science, pages 293–330. Springer-Verlag, 1998.

[19] R. K. Boel, L. Ben-Naoum, and V. Van Breusegem. On forbidden state problems for a class of controlled Petri nets. IEEE Transactions on Automatic Control, 40(1):1717–1731, 1995.

[20] A. Burns, A. J. Wellings, F. Burns, A. M. Koelmans, M. Koutny, A. Romanovsky, and A. Yakovlev. Towards modelling and verification of concurrent Ada programs using Petri nets. In Pezz, M. and Shatz, M., editors, Workshop Proceedings Software Engineering and Petri Nets, pages 115–134, June 2000.

[21] U. A. Buy and R. H. Sloan. Automatic real-time analysis of reactive systems with the PARTS toolset. Automated Software Engineering, 23(4):227–273, 2001.

[22] D. Callahan. Design considerations for parallel programming. MSDN Magazine, October 2008.

[23] V. Chandra, Z. Huang, and R. Kumar. Automated control syntesis for and assembly line using discrete event system control theory. IEEE Transactions on Systems, Man, and Cybernetics: Part C, 33(2):284–289, 2003.

[24] F. Charbonnier, H. Alla, and R. David. The supervised control of discrete-event dynamic systems. IEEE Transactions on Control Systems Technology, 7:175–187, 2003.

[25] H. Chen. Net structure and control logic synthesis of controlled Petri nets. IEEE Transactions on Automatic Control, 43(10):1446–1450, 1998.

[26] H. Chen. Control synthesis of Petri nets based on s-decreases. Discrete Event Dynamic Systems: Theory and Applications, 10(3):233–250, 2000.

[27] H. Chen and H.-M. Hanish. Control synthesis of timed discrete event systems based on predicate invariance. IEEE Transactions on Systems Man and Cybernetics, 30(5):713–724, 2000.

[28] H. Chen and B. Hu. Monitor-based control of a class of controlled Petri nets. In Proceedings of the 3rd International Conference on Automation, Robotics and Computer Vision, 1994.

[29] M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, and A. Sangiovanni-Vincentelli. A formal specification model for hardware/software codesign. In Proceedings of the International Workshop on Hardware-Software Codesign, 1993.

[30] E. Clarke, O. Grumberg, and D. Peled. Model Checking. MIT Press, 1999.

[31] J. Cortadella, A. Kondratyev, L. Lavagno, M. Massot, S. Moral, C. Passerone, Y. Watanabe, and A. Sangiovanni-Vincentelli. Task generation and compile-time scheduling for mixed datacontrol embedded software. In Proceedings of the Design Automation Conference, pages 489– 494, 2000.

[32] J. Cortadella, A. Kondratyev, L. Lavagno, C. Passerone, and Y. Watanabe. Quasi-static scheduling of independent tasks for reactive systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2005.

[33] J. Cortadella, A. Kondratyev, L. Lavagno, and Y. Watanabe. Quasi-static scheduling for concurrent architectures. In Third International Conference on Application of Concurrency to System Design (ACSD 2003), pages 29–40. IEEE Computer Society, June 2003.

[34] J.-P. Courtiat, J. M. Ayache, and B. Algayres. Petri nets are good for protocols. In ACM, SIGCOMM’84 Tutorials and Symposium, Communications Architectures and Protocols, pages 66–74, 1984.

[35] P. Darondeau and X. Xie. Linear control of live marked graphs. Automatica, 39(3):429–440, 2003.

[36] G. de Jong and B. Lin. A communicating Petri net model for the design of concurrent asynchronous modules. In Proceedings of the 31st Annual Conference on Design Automation (DAC ’94), pages 49–55. ACM Press, 1994.

[37] J. Ezpeleta, J. M. Colom, and J. Mart´ınez. A Petri net based deadlock prevention policy for flexible manufacturing systems. IEEE Transactions on Robotics and Automation, 11(2):173– 184, 1995.

[38] M. Fabian and A. Hellgren. PLC-based implementation of supervisory control for discrete event systems. In Proceedings of the 37th IEEE Conference on Decision and Control, pages 3305–3310, 1998.

[39] H. Fleischhack and B. Grahlmann. A compositional Petri net semantics for sdl. In Desel J.

and Silva M., editors, Application and Theory of Petri Nets, volume 1420 of Lecture Notes in Computer Science, pages 144–164. Springer-Verlag, 1998.

[40] J. Flochova, F. Auxt, M. Radakovic, and O. Jombik. PNDesigner–a tool designed for model based diagnosis and supervisory control of DES. In Proceedings of the 8th International Workshop on Discrete Event Systems, pages 471–472, 2006.

[41] G. Gardey, D. Lime, M. Magnin, and O. H. Roux. Romeo: A tool for analyzing time Petri nets. In Proceedings of the 17th International Conference on Computer Aided Verification, volume 3576 of Lecture Notes in Computer Science, pages 418–423, 2005.

[42] G. Gardey, O. F. Roux, and O. H. Roux. Safety control synthesis of time Petri nets. In Proceedings of the 8th International Workshop on Discrete Event Systems, pages 222–228, 2006.

[43] A. Ghaffari, N. Rezg, and X. Xie. Feedback control logic for forbidden-state problems of marked graphs: application to a real manufacturing system. IEEE Transactions on Automatic Control, 48(1):2–17, 2003.

[44] H. Goldstein. Cure for the multicore blues. IEEE Spectrum, 44(1):41–43, 2007.

[45] B. Grahlmann. The PEP tool. In Grumberg, O., editor, Computer Aided Verification, volume 1254 of Lecture Notes in Computer Science, pages 440–443. Springer-Verlag, 1997.

[46] B. Grahlmann, M. Moeller, and U. Anhalt. A new interface for the PEP tool - parallel finite automata. In Desel, J. and Fleischhack, H. and Oberweis, A. and Sonnenschein, M., editor, Workshop Algorithmen und Werkzeuge f¨ur Petrinetze, volume 22 of AIS, pages 21–26. 1995.

[47] A. Gromyko, M. Pistore, and P. Traverso. A tool for controller synthesis via symbolic model checking. In Proceedings of the 8th International Workshop on Discrete Event Systems, pages 475–476, 2006.

[48] X. Guan and L.E. Holloway. Supervisory control of contradictions in hierarchical task controllers.

In Proceedings of the 37th Annual Allerton Conference on Communication, Control, and Computing, Urbana-Champaign, 1999.

[49] N. Halbwachs. Synchronous Programming of Reactive Systems. Kluwer Academic, 1993.

[50] K.X. He and M.D. Lemmon. On the transformation of maximally permissive marking-based supervisors into monitor supervisors. In Proceedings of the IEEE Conference on Decision and Control, pages 2657–2662, December 2000.

[51] L. E. Holloway, X. Guan, and R. Sundaravadivelu. Automated synthesis and composition of taskblocks for control of manufacturing systems. IEEE Transactions on Systems, Man, and Cybernetics: Part B, 30(5):696–712, 2000.

[52] L. E. Holloway, B. H. Krogh, and A. Giua. A survey of Petri net methods for controlled discrete event systems. Discrete Event Dynamic Systems, 7(2):151–190, 1997.

[53] L.E. Holloway, X. Guan, and L. Zhang. A generalization of state avoidance policies for controlled Petri nets. IEEE Transactions on Automatic Control, 41(6):804–816, 1996.

[54] L.E. Holloway and B.H. Krogh. Synthesis of feedback control logic for a class of controlled Petri nets. IEEE Transactions on Automatic Control, 35(5):514–523, 1990.

[55] P.-A. Hsiung. Formal synthesis and code generation of embedded real-time software. In CODES ’01: Proceedings of the 9th International Symposium on Hardware/Software Codesign, pages 208–213. ACM Press, 2001.

[56] J. Huang and R. Kumar. Nonblocking directed control of discrete event systems. In Proceeding of the 2005 IEEE Conference on Decision and Control, 2005.

[57] M. V. Iordache and P. J. Antsaklis. Software tools for the supervisory control of Petri nets based on place invariants. Technical report isis-2002-003, University of Notre Dame, April 2002.

[58] M. V. Iordache and P. J. Antsaklis. Design of T-liveness enforcing supervisors in Petri nets.

IEEE Transactions on Automatic Control, 48(11):1962–1974, 2003.

[59] M. V. Iordache and P. J. Antsaklis. Synthesis of supervisors enforcing general linear vector constraints in Petri nets. IEEE Transactions on Automatic Control, 48(11):2036–2039, 2003.

[60] M. V. Iordache and P. J. Antsaklis. Decentralized supervision of Petri nets. IEEE Transactions on Automatic Control, 51(2):376–381, 2006.

[61] M. V. Iordache and P. J. Antsaklis. Supervision based on place invariants: A survey. Discrete Event Dynamic Systems, 16:451–492, 2006.

[62] M. V. Iordache and P. J. Antsaklis. Supervisory Control of Concurrent Systems: A Petri Net Structural Approach. Birkh¨auser, 2006.

[63] M. V. Iordache and P. J. Antsaklis. Petri net supervisors for disjunctive constraints. In Proceedings of the 2007 American Control Conference, pages 4951–4956, 2007.

[64] M. V. Iordache and P. J. Antsaklis. Petri nets and programming: A survey. In Proceedings of the 2009 American Control Conference, pages 4994–4999, 2009.

[65] K. °Akesson, M. Fabian, H. Flordal, and R. Malik. Supremica – an integrated environment for verification, synthesis and simulation of discrete event systems. In Proceedings of the 8th International Workshop on Discrete Event Systems, pages 384–385, 2006.

[66] S. R. Kosaraju. Limitations of Dijkstra’s semaphore primitives and Petri nets. Operating Systems Review, 7(4):122–126, 1973.

[67] B.H. Krogh and L.E. Holloway. Synthesis of feedback control logic for manufacturing systems.

Automatica, 27(4):641–651, 1991.

[68] O. Kupferman, N. Piterman, and M.Y. Vardi. Safraless compositional synthesis. In Proceedings of the 18th International Conference on Computer Aided Verification, Lecture Notes in Computer Science. Springer-Verlag, 2006.

[69] O. Kupferman and M. Vardi. Church’s problem revisited. The Bulletin of Symbolic Logic, 5(2):245–263, 1999.

[70] S. Lafortune. Umdes-lib software library. http://www.eecs.umich.edu/umdes/toolboxes.html.

[71] K. Lautenbach and P. S. Thiagarajan. Analysis of a resource allocation problem using Petri nets. In Proceedings of the 1st European Conference on Parallel and Distributed Processing, pages 260–266. Cepadues Editions, 1979.

[72] E. A. Lee. Cyber-physical systems – are computing foundations adequate? In NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap, Austin, TX, October 2006.

[73] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. IEEE Transactions on Computers, 36(1):24–35, 1987.

[74] M. Lemmon and K. He. Supervisory plug-ins for distributed software. In Pezze, M. and Shatz, M., editors, Proceedings of the Workshop on Software Engineering and Petri Nets, pages 155–172. University of Aarhus, Department of Computer Science, 2000.

[75] M. Lemmon, K. He, and S. Shatz. Dynamic reconfiguration of software objects using Petri nets and network unfolding. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, pages 3069–3074, 2000.

[76] J. R. Levine, T. Mason, and D. Brown. Lex and Yacc. O’Reilly & Associates, second edition, 1992.

[77] B. Lin. Software synthesis of process-based concurrent programs. In DAC ’98: Proceedings of the 35th annual conference on Design automation, pages 502–505. ACM Press, 1998.

[78] C. Liu, A. Kondratyev, Y.Watanabe, J. Desel, and A. Sangiovanni-Vincentelli. Schedulability analysis of Petri nets based on structural properties. In IEEE International Conference on Application of Concurrency to System Design, 2006.

[79] J. O. Moody and P. J. Antsaklis. Supervisory Control of Discrete Event Systems Using Petri Nets. Kluwer Academic Publishers, 1998.

[80] J. O. Moody and P. J. Antsaklis. Petri net supervisors for DES with uncontrollable and unobservable transitions. IEEE Transactions on Automatic Control, 45(3):462–476, 2000.

[81] T. Murata. Petri nets: Properties, analysis and applications. In Proceedings of the IEEE, pages 541–580, April 1989.

[82] T. Murata, B. Shenker, and S. M. Shatz. Detection of Ada static deadlocks using Petri net invariants. IEEE Transactions on Software Engineering, 15(3):314–326, 1989.

[83] M. Notomi and T. Murata. Hierarchical reachability graph of bounded petri nets for concurrent-software analysis. IEEE Transactions on Software Engineering, 20(5):325–336, 1994.

[84] G. Nutt. Operating Systems. Addison Wesley, 2003.

[85] J. Park and S. Reveliotis. Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings. IEEE Transactions on Automatic Control, 46(10):1572–1583, 2001.

[86] J. Park and S. Reveliotis. Liveness-enforcing supervision for resource allocation systems with uncontrollable behavior and forbidden states. IEEE Transactions on Robotics and Automation, 18(2):234–240, 2002.

[87] E. Pastor, O. Roig, J. Cortadella, and R. M. Badia. Petri net analysis using boolean manipulation.

In Application and Theory of Petri Nets ’94, volume 815 of Lecture Notes in Computer Science, pages 416–435. Springer-Verlag, 1994.

[88] J. L. Peterson. Petri Net Theory and the Modeling of Systems. Prentice Hall, Englewood Cliffs, NJ, 1981.

[89] J. L. Peterson and A. Silberschatz. Operating Systems Concepts. Addison-Wesley, 1985.

[90] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In Proceedings of the ACM Symposium on Principles of Programming Languages, pages 179–190, 1989.

[91] W. Reisig. Petri Nets, volume 4 of EATCS Monographs on Theoretical Computer Science.

Springer-Verlag, 1985.

[92] S. Reveliotis. Real-Time Management of Resource Allocation Systems: A Discrete Event Systems Approach. Springer-Verlag, 2005.

[93] A. Sathaye. Synthesis of real-time supervisors for controlled time Petri nets. In Proceedings of the 32nd International Conference on Decision and Control, pages 235–236, 1993.

[94] A. Sathaye and B. Krogh. Logical analysis and control of time Petri nets. In Proceedings of the 31st International Conference on Decision and Control, pages 1198–1203, 1992.

[95] M. Sgroi, L. Lavagno, Y. Watanabe, and A. Sangiovanni-Vincentelli. Quasi-static scheduling of embedded software using equal conflict nets. In Donatelli, Susanna and Kleijn, Jetty, editors, 20th International Conference on Application and Theory of Petri Nets 1999 (ICATPN’99), volume 1630 of Lecture Notes in Computer Science, pages 208–227. Springer- Verlag, 1999.

[96] M. Sgroi, L. Lavagno, Y. Watanabe, and A. Sangiovanni-Vincentelli. Synthesis of embedded software using free-choice Petri nets. In Proceedings of the 36th Design Automation Conference (DAC-99), pages 805–810, 1999.

[97] S. Shatz, K. Mai, D. Moorthi, and J. Woodward. A toolkit for automated support of Adatasking analysis. In Proceedings of the 9th International Conference on Distributed Computing Systems, pages 595–602, 1989.

[98] D. Shewa, J. Ashley, and L. Holloway. Spectool 2.4 beta: A research tool for modular modeling, analysis, and synthesis of discrete event systems. In Proceedings of the 8th International Workshop on Discrete Event Systems, pages 477–478, 2006.

[99] C. Stehno. Real-time systems design with PEP. In Katoen, J.-P. and Stevens P., editors, Tools and Algorithms for the Construction and Analysis of Systems, volume 2280 of Lecture Notes in Computer Science, pages 476–480. Springer-Verlag, 2002.

[100] K. Strehl, L. Thiele, D. Ziegenbein, R. Ernst, and J. Teich. Scheduling hardware/software systems using symbolic techniques. In International Workshop on Hardware/Software Codesign, pages 173–177, 1999.

[101] G. Stremersch. Supervision of Petri Nets. Kluwer Academic Publishers, 2001.

[102] G. Stremersch and R. K. Boel. Structuring acyclic Petri nets for reachability analysis and control. Discrete Event Dynamic Systems, 12(1):7–41, 2002.

[103] F.-S. Su and P.-A. Hsiung. Extended quasi-static scheduling for formal synthesis and code generation of embedded software. In CODES ’02: Proceedings of the tenth international symposium on Hardware/software codesign, pages 211–216. ACM Press, 2002.

[104] Z. Suraj. Resource allocation problem. In Proc. of the 3rd Symp. on Math. Foundations of Comput. Science, Zaborow 1980, ICS PAS Reports, pages 83–86, 1980.

[105] T. Suzuki, S. M. Shatz, and T. Murata. A protocol modeling and verification approach based on a specification language and Petri nets. IEEE Transactions on Software Engineering, 16(5):523–536, 1990.

[106] A. Tarafdar and V. K. Garg. Predicate control: synchronization in distributed computations with look-ahead. Journal of Parallel and Distributed Computing, pages 219–237, 2004.

[107] F. Tricas, F. Garcia-Valles, J. M. Colom, and J. Ezpeleta. New methods for deadlock prevention and avoidance in concurrent systems. Actas de las Jornadas de Concurrencia 2000, pages 97–110, June 2000.

[108] S. Tu, S. Shatz, and T. Murata. Applying Petri nets reduction to support Ada-tasking deadlock detection. In Proceedings of the 10th IEEE International Conference on Distributed Computing Systems, pages 96–102, 1990.

[109] S. Vercauteren, D. Verkest, G. de Jong, and B. Lin. Derivation of formal representations from process-based specification and implementation models. In Proceedings of the 10th International Symposium on System Synthesis (ISSS ’97), pages 16–23, 1997.

[110] Y. Wang, T. Kelly, M. Kudlur, S. Mahlke, and S. Lafortune. The application of supervisory control to deadlock avoidance in concurrent software. In Proceedings of the 9th International Workshop on Discrete Event Systems, pages 287–292, 2008.

[111] W. M. Wonham. Supervisory control of discrete event systems and design software.

Department of Electrical and Computer Engineering, University of Toronto, http://www.control.toronto.edu/DES.

[112] K. Xing, B. Hu, and H. Chen. Deadlock avoidance policy for Petri net modeling of flexible manufacturing systems with shared resources. IEEE Transactions on Automatic Control, 41(2):289–295, February 1996.

[113] M. Zhou and M. P. Fanti. Deadlock Resolution in Computer-Integrated Systems. Marcel Dekker, Inc., 2005.

[114] X. Zhu and B. Lin. Compositional software synthesis of communication processes. In Proceedings of the International Conference on Computer Design, pages 646–651, 1999.

[115] W. M. Zuberek. Petri net models of process synchronization mechanisms. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, pages 841–847, 1999.

1. En este clásico problema de sincronización cinco filósofos se sientan alrededor de una mesa circular. Un filósofo puede comer, pensar o tener hambre. Hay una bifurcación entre cada dos filósofos adyacentes. Para comer, cada filósofo necesita los dos tenedores a su izquierda y derecha. Puede tomar un tenedor sólo si no lo ha tomado ya un vecino. Cuando tiene hambre, un filósofo puede empezar a comer sólo cuando tiene ambos tenedores. Cuando termina de comer, deja los tenedores a su derecha y a su izquierda. El problema es encontrar una estrategia que permita a los filósofos pensar y comer sin llegar a un callejón sin salida o al hambre. [↑](#footnote-ref-1)