TESIS DOCTORAL

Técnicas de automatización avanzadas en procesos industriales

Emilio Jiménez Macías

UNIVERSIDAD DE LA RIOJA

TESIS DOCTORAL

Técnicas de automatización avanzadas en procesos industriales

Emilio Jiménez Macías

Universidad de La Rioja
Servicio de Publicaciones
2004

Esta tesis doctoral, dirigida por el doctor D. Emilio Jiménez Tofé, fue leída el 25 de enero de 2002, y obtuvo la calificación de Sobresaliente Cum Laude Unanimidad.

Edita: Universidad de La Rioja
Servicio de Publicaciones

ISBN 84-689-0360-4

Técnicas de Automatización Avanzadas en Procesos Industriales

UNIVERSIDAD DE LA RIOJA

2001

Emilio Jiménez Macías

Trabajo presentado por D. Emilio Jiménez Macías para la obtención del título de Doctor.

Título del Trabajo realizado por el doctorando:

TÉCNICAS DE AUTOMATIZACIÓN AVANZADAS EN PROCESOS INDUSTRIALES.

Doctor Director del Trabajo:
Dr. Emilio Jiménez Tofé

Código Unesco (principal):
3311 Tecnología de la Instrumentación

Programa de Doctorado cursado:
911 Ingeniería, Matemáticas y Computación

Responsable del Programa de Doctorado:
Dr. José Ignacio Extremiana Aldana

Departamento Responsable del Trabajo de Investigación:
Departamentos de Matemáticas y Computación y de Ingeniería Eléctrica

Al director de la tesis, a toda mi familia y a Rosa, por su inestimable apoyo, y a mis amigos por saber disculpar mi ausencia durante estos años.

Índice

pag.

INTRODUCCIÓN

Tema 1 ELEMENTOS DE LA AUTOMATIZACIÓN AVANZADA

Tema 2 METODOLOGÍA Y ARQUITECTURAS PARA AUTOMATIZACIÓN DE PROCESOS COMPLEJOS

Tema 3 IMPLEMENTACIÓN DE LAS AUTOMATIZACIONES EN LOS DISPOSITIVOS INDUSTRIALES DE CONTROL AUTOMÁTICO

Tema 4

METODOLOGÍA DE SIMULACIÓN MEDIANTE LENGUAJES DE PROGRAMACIÓN

Tema 5

IMPLEMENTACIÓN DE TÉCNICAS AVANZADAS EN LA AUTOMATIZACIÓN

Tema 6

APLICACIÓN EN PLANTA INDUSTRIAL DE ÚLTIMA GENERACIÓN

Tema 7

RESULTADOS Y APORTACIONES

Índice Figuras

pag.

Tema 1 ELEMENTOS DE LA AUTOMATIZACIÓN AVANZADA

Tema 2 METODOLOGÍA Y ARQUITECTURAS PARA AUTOMATIZACIÓN DE PROCESOS COMPLEJOS

Tema 3 IMPLEMENTACIÓN DE LAS AUTOMATIZACIONES EN LOS DISPOSITIVOS INDUSTRIALES DE CONTROL AUTOMÁTICO

Tema 4 METODOLOGÍA DE SIMULACIÓN MEDIANTE LENGUAJES DE PROGRAMACIÓN

Tema 5 IMPLEMENTACIÓN DE TÉCNICAS AVANZADAS EN LA AUTOMATIZACIÓN

Tema 6 APLICACIÓN EN PLANTA INDUSTRIAL DE ÚLTIMA GENERACIÓN

INTRODUCCIÓN

1. Presentación del trabajo

El presente Trabajo de Investigación, presentado para la obtención del título de Doctor, y titulado "*Técnicas de Automatización Avanzadas en Procesos Industriales*" constituye un documento original, resultado de la investigación desarrollada por su autor, a veces en colaboración con otros grupos de investigación (única forma eficiente de avanzar de forma efectiva), con el apoyo especial del director de la tesis, que en estos años se ha volcado en ella hasta casi monopolizar su quehacer científico.

Existen varios pilares básicos en los que se asienta el trabajo, pero principalmente se indican los siguientes:

- Procesos Industriales
- SCADAS
- GRAFCET y Redes de Petri
- Simulación de Procesos
- Autómatas Programables (PLCs)
- Supervisión
- Ingeniería Gráfica
- Ingeniería Térmica y de Fluidos

Todos esos pilares son tratados con el fin común de obtener los objetivos que más adelante se indican, y que creemos que han sido conseguidos de una manera muy satisfactoria.

Precisamente, por tratarse de un trabajo enfocado a las aplicaciones que beneficien las prestaciones de los procesos industriales, tanto por aspectos técnicos como metodológicos, y basado en un considerable número de temas y aspectos, como los que se han nombrado, e incluso algunos más, ha sido considerable el esfuerzo realizado para contar con la experiencia de equipos expertos en el tema. A ese respecto hay que agradecer la ayuda recibida por muchos de ellos en las consultas que frecuentemente se les ha realizado.

Dichos pilares debidamente coordinados en la investigación, basada en la experiencia en la Ingeniería Industrial y en la Industria en general tanto del doctorando como del director de la tesis, tanto desde el ámbito profesional como desde los ámbitos docente y investigador. Todo esto además ha permitido prolongar el Trabajo de Investigación a un trabajo de I+D. El Desarrollo ha consistido en la aplicación de algunos de los resultados obtenidos, sobre una planta industrial de última generación (concretamente una planta cerámica), con resultados muy satisfactorios. El hecho de que la Planta Industrial se

encuentre entre las más modernas a nivel mundial, y el interés que ha despertado dicho desarrollo, son un indicativo de que ciertamente se está caminando en un terreno novedoso e innovador.

El trabajo se ha dividido en 7 capítulos, de los cuales el último constituye simplemente un resumen de los resultados obtenidos a lo largo de esta tesis, y que han sido relatados en cada uno de los capítulos e incluso a veces en ciertos apartados en los que consideraba interesante.

El primer capítulo, Elementos de la automatización avanzada, constituye una visión de los aspectos básicos necesarios para poder avanzar eficientemente en el resto del trabajo. Se centra en los autómatas programables, los SCADAs, y las herramientas gráficas de automatización (GRAFCET y redes de Petri). Este capítulo constituye una visión del Estado del Arte de los temas tratados, y evidentemente no presenta aportación alguna ni resultados, sino que puede entenderse como una guía de estudio para los no expertos en automatización de plantas y procesos industriales puedan seguir el resto del trabajo. Los lectores expertos en cada uno de los apartados pueden prescindir de su lectura.

Y los demás capítulos, desde el 2 hasta el 6, presentan el desarrollo de los diversos temas sobre los que se ha realizado esta investigación para mejorar las prestaciónes de los procesos productivos automáticos: el tema 2 centrado en la metodología y las arquitecturas básicas para la automatización de procesos complejos, el 3 en la implementación de forma metodológica y automática sobre los PLCs de los automatismos desarrollados mediante herramientas gráficas, el 4 muestra una metodología similar pero para simular dichas herramientas, y por tanto el sistema y la automatización, mediante lenguajes de programación, el tema 5 muestra la implementación de técnicas para mejorar las prestaciones en procesos flexibles muy cambiantes, e introduce al lector a las técnicas que más se usan de inteligencia artificial en las automatizaciones industriales, mediante un ejemplo, y el tema 6, tras analizar un tipo especial de procesos en el que el producto avanza por sucesivos ciclos interconectados, muestra los resultados de aplicar sobre un ejemplo real de uno de ellos las técnicas anteriores.

2. Objetivos Planteados

La primera reflexión que se debe de tener para comenzar cualquier trabajo, y más aún si lo es de cierta envergadura, debe ir encaminada hacia los objetivos que se pretenden. Con ello conseguimos que sea más sencillo seguir hacia delante en nuestros propósitos sin desviarnos, y también poder evaluar a medida que avanzamos lo cerca o lejos que estamos de nuestra meta, o lo bien que ha resultado todo cuando por fin se da por terminado.

Cuando se comenzó este trabajo de investigación se pretendía realizar una obra en la que la producción científica resultante se pudiese aplicar directamente a la Industria. De esta manera se pretendía, por un lado dar un carácter práctico a la investigación, y por otro abrir el camino de mostrar cómo otras investigaciones pueden traducirse en

aplicaciones prácticas. Cuando, tras haber trabajado en la industria, asistí a los primeros foros de investigación en los que se presentaban avances en automatización, siempre tendía a pensar en los puntos fuertes y débiles de su implementación real en la industria. Ahí empezó a gestarse la idea que ha desembocado en este trabajo, con el punto de vista puesto en dichos objetivos.

Esta ha sido también la razón de que el trabajo haya resultado tan extenso, porque ya partía con la idea no de concluir una investigación, sino de abrirla para que posteriormente se pueda avanzar más sobre ella y sobre el tipo de objetivos comentados; y ciertamente pese a que se ha avanzado mucho, también son muchos los nuevos caminos que han aparecido.

3. Campos de influencia

Puesto que toda tesis debe clasificarse en algún campo, el más apropiado en este caso seguramente sea el anteriormente indicado, 3311, tecnología de la instrumentación, si bien dado el carácter del trabajo, se presentan otros campos afines al mismo.

3304 TECNOLOGIA DE LOS ORDENADORES

330402	CONVERTIDORES ANALOGICO-DIGITALES
330404	UNIDADES CENTRALES DE PROCESO
330407	PERIFERICOS DE ORDENADORES
330408	FIABILIDAD DE LOS ORDENADORES
330409	MANTENIMIENTO DE LOS ORDENADORES
330412	DISPOSITIVOS DE CONTROL
330413	DISPOSITIVOS DE TRANSMISION DE DATOS
330417	SISTEMAS EN TIEMPO REAL
330418	DISPOSITIVOS DE ALMACENAMIENTO

3310 TECNOLOGIA INDUSTRIAL

331001	EQUIPO INDUSTRIAL (ver 3313.12)
331002	MAQUINARIA INDUSTRIAL (ver 3313.12)
331003	PROCESOS INDUSTRIALES
331005	INGENIERIA DE PROCESOS
331006	ESPECIFICACIONES DE PROCESOS
331007	ESTUDIO DE TIEMPOS Y MOVIMIENTOS (ver 5311.09)

3311 TECNOLOGIA DE LA INSTRUMENTACION

331101	TECNOLOGIA DE LA AUTOMATIZACION
331102	INGENIERIA DE CONTROL
331105	EQUIPOS ELECTRICOS DE CONTROL
331114	SERVOMECANISMOS

3328 PROCESOS TECNOLOGICOS

332805	CRISTALIZACION

332808	DESECACION
332819	MEZCLADO

3312 TECNOLOGIA DE MATERIALES

331203	MATERIALES CERAMICOS
331205	PRODUCTOS DE ARCILLA
331211	REFRACTARIOS (ver 3315.17)

4. Aportaciones

Afortunadamente este trabajo ya está dando resultados y aportaciones que compensan el esfuerzo invertido en él. Estas aportaciones pueden dividirse en dos grupos: aportaciones científicas e industriales.

Como aportación industrial destaca la aplicación de los resultados obtenidos al desarrollo de un control real en una planta de última generación, y la satisfacción y expectativas que ha suscitado en los responsables de dicha industria.

Las aportaciones científicas corresponden a la contribución que ha aportado en los congresos y jornadas, tanto nacionales como internacionales, en los que se ha presentado. Y como foro ideal para comentar los resultados se puede indicar las ponencias admitidas para su presentación en noviembre, con el trabajo de tesis ya concluido, que son las siguientes:

Ponencia: **INDUSTRIAL AUTOMATION TECHNIQUES IN MANUFACTURING PROCESSES**
Autores: E. Jiménez Macías, Emilio Jiménez
Congreso: IASTED International Conference on Intelligent Systems and Control, ISC'2001
Celebración: November 19-22, 2001. Tampa, USA.

Ponencia: **FACTORY MODELLING AND CONTROL WITH PETRI NETS IN PROCESSES WITH CYCLIC SUBSYSTEMS**
Autores: E. Jiménez Macías, Emilio Jiménez
Congreso: IASTED International Conference on Intelligent Systems and Control, ISC'2001
Celebración: November 19-22, 2001. Tampa, USA.

Ponencia: **ROBOTIC APPLICATIONS IN ADAPTIVE CONTROL OF FLEXIBLE MANUFACTURING PROCESSES**
Autores: E. Jiménez
Congreso: IASTED International Conference on Robotics and Applications RA'2001
Celebración: November 19-22, 2001. Tampa, USA.

Ponencia: **ROBOTICS, SUPERVISION AND VIRTUAL REALITY**
Autores: E. Jiménez Macías, Félix Sanz, Emilio Jiménez and Arturo Fernández
Congreso: IASTED International Conference on Robotics and Applications RA'2001
Celebración: November 19-22, 2001. Tampa, USA.

Una visión resumida de las aportaciones de los capítulos puede encontrarse, como se ha indicado en el Tema 7.

Tema 1

ELEMENTOS DE LA AUTOMATIZACIÓN AVANZADA

1.1. Introducción

En este capítulo se van a analizar los elementos principales de automatización que serán empleados a lo largo de todo este trabajo. La visión tratará de ser amplia y enfocada tanto desde un punto de vista teórico como práctico. La visión práctica es necesaria puesto que la culminación de este trabajo es la aplicación de los resultados de la investigación sobre las plantas industriales, para permitirles mejorar su funcionamiento y sus prestaciones. La visión teórica es igualmente importante en este capítulo, base para los demás, para disponer de un marco claro y conciso sobre el que asentar los conceptos posteriores.

Por lo tanto, este capítulo, incluye una visión concernienete al estudio del *Estado del Arte* y constantemente hace referencia al material más interesante en los campos que serán posteriormente tratados, por lo que puede ser obviado por el lector experto en Ingeniería Industrial (especialmente en automatizaciones mediante autómatas programables PLC (Programmable Logic Controllers)). Si por el contrario se aventura en esta tesis algún lector totalmente profano a la automatización industrial se recomienda no sólo leer el capítulo con detenimiento sino además trabajar sobre las referencias que se indican, para poder profundizar en el resto de la tesis. Por esta razón las referencias y la bibliografía son especialmente extensas en este capítulo, y se proporcionan en cada uno de los apartados.

1.2. Los Autómatas Programables

1.2.1. Arquitectura y configuración

Un autómata es una máquina industrial susceptible de ser programada (autómata programable industrial API) al estar basada en un sistema de microprocesador dotado de un hardware estándar independiente del proceso a controlar. Se adapta a tal proceso mediante un programa de usuario específico (software), escrito en algún lenguaje de programación y que contiene la secuencia de operaciones a realizar.

El programa, realizado y depurado en una unidad de programación propia o ajena al autómata, se incorpora a la memoria de programa del mismo, para su ejecución por la Unidad Central de Proceso (CPU) del autómata.

La secuencia de operaciones del programa se realiza sobre señales de entrada y salida del proceso, llevadas al bus interno del autómata a través de las correspondientes interfaces de entrada y salida (E/S). El autómata gobierna las señales de salida según el programa de control previamente almacenado en su memoria de programa, a partir del

estado de las señales de entrada. Los tipos de interfaces de E/S son muy variados, según las características de las señales procedentes del proceso o las que se van a aplicar al mismo (señales analógicas de tensión o corriente, pulsos de 0/5 V, 0/24 V, tensiones alternas 110 V, 220 V, tensiones continuas 12/24/48 V, etc). En la mayoría de los APIs, el número (hasta la capacidad permitida por la CPU), tipo y ubicación de las interfaces lo define el usuario, adaptando así el autómata, junto con su programa, a las necesidades de su proceso.

Ejemplos de señales de entrada son las procedentes de elementos digitales, como interruptores, finales de carrera y detectores de proximidad, o analógicos, como tensiones de dinamos tacométricas, tensiones de termopares, etc.

Ejemplos de señales de salida son las órdenes digitales todo o nada o analógicas en tensión o corriente, que se envían a los elementos indicadores y actuadores del proceso, tales como lámparas, contactores, válvulas, etc.

Ha de hacerse constar como característica esencial de los APIs, el disponer de un hardware estándar que posibilita la realización de sistemas de control de acuerdo con las necesidades del usuario. La elección del API (gama baja, media o alta) será función de las necesidades de potencia de cálculo y número y tipo de señales de entrada y de salida.

La configuración del autómata, llamada arquitectura interna, como en todo sistema microprocesador, incluye fundamentalmente los siguientes cuatro bloques básicos: una CPU o unidad central de proceso, una memoria interna de trabajo (RAM), una memoria de programa (RAM, EPROM, EEPROM), y las interfaces de entradas y salidas conectadas al bus interno. A su vez, tanto la CPU como la memoria de programa están conectadas a dicho bus interno.

Las instrucciones de un programa de usuario almacenado en la memoria de un API son ejecutadas correlativamente generando unas órdenes o señales de control a partir de las señales de entrada leídas de la planta. Cuando se detectan cambios en las señales, el autómata reacciona de acuerdo con el programa hasta que obtiene las órdenes de salida necesarias. Esta secuencia se ejecuta continuamente a fin de conseguir el control actualizado del proceso.

Además de ejecutar las instrucciones del programa, el autómata realiza un conjunto de acciones que aseguran su funcionamiento correcto: test de CPU y memoria, comprobación del reloj de guarda, etc. La secuencia o ciclo de operación consta básicamente de las siguientes etapas:

1.- Test del sistema.
2.- Lectura de señales desde la interface de entrada.
3.- Escritura de señales en la interface de salida.
4.- Procesado del programa a fin de obtener las señales de control.

Para reducir los tiempos de acceso a las interfaces de E/S, la lectura y escritura de las entradas y salidas involucradas se realiza a la vez, guardando las entradas leídas en una memoria temporal o imagen de entradas a la que accede la CPU mientras ejecuta el

programa, en tanto que los resultados o señales de control se van guardando en otra memoria temporal o imagen de salidas a medida que se van obteniendo. Al terminar la ejecución del programa los resultados se colocan de una sola vez en la interface de salida.

Aparte de las cuatro etapas descritas anteriormente, el autómata eventualmente puede establecer comunicación con periféricos exteriores, por ejemplo para sacar datos por impresora, comunicación con otros autómatas u ordenadores, conexión con la unidad de programación, etc.

Las anteriores acciones, repitiéndose periódicamente, definen un ciclo de operación que requiere un cierto tiempo (dependiendo del número de entradas y salidas, y de la longitud del programa) para ser ejecutado, de modo que el autómata no puede responder en tiempo real a sucesos que ocurren en el sistema exterior. Este tiempo será determinante cuando con el autómata se pretendan controlar procesos rápidos, con señales de muy corta duración o alta frecuencia de conmutación.

Los retardos aportados por entradas o salidas son debidos, respectivamente, al filtrado de señal que incorporan (filtro pasa bajo), y a los tiempos de respuesta del interruptor (relé, transistor, etc.) o convertidor digital/analógico.

Para las entradas, los retardos típicos oscilan entre 10 ms y 100 ms, aunque hay autómatas que permiten ajustes del tiempo de filtro menores.

Para los tiempos típicos, la frecuencia máxima de señal de entrada queda limitada entre 100 Hz y 10 Hz, de forma que cualquier señal de frecuencia superior, o de periodo T menor que el tiempo de filtro, no podrá ser leída desde las entradas estándar del autómata.

Los anteriores problemas debidos a los retardos pueden reducirse de las siguientes maneras:

- Para las entradas, con elementos de entrada de proceso rápido: filtrado débil asociado a programa de ejecución rápida, entradas detectoras de flancos o entradas de contador rápido.
- Para el tiempo de procesado del programa: escribiendo subprogramas rápidos contenidos en el principal asociados a algún elemento de proceso rápido y activados periódicamente.
- Para las salidas: utilizando elementos semiconductores en sustitución de relés electromecánicos.

En general se dice que un autómata es capaz de controlar en tiempo real un proceso, cuando sus tiempos de respuesta o retardo son muy pequeños comparados con los tiempos de reacción del mismo.

La configuración del autómata es la estructura que tiene su sistema físico (hardware), fundamentalmente la unidad de control, el sistema de E/S y la memoria, de modo que

pueda adaptarse a las características particulares de la aplicación industrial en que vaya a usarse.

Siendo la modularidad una de las características fundamentales de los autómatas, la elección de la configuración adecuada resulta fácil al ofrecer sus fabricantes una amplia variedad de módulos y ampliaciones.

En cuanto a la unidad de control las configuraciones son:

- Unidad de control compacta (control centralizado). Es el caso en el que una única CPU gestiona tanto el programa como las entradas y salidas asociadas, agrupadas en módulos que contienen exclusivamente interfaces E/S. Esta configuración se usa en aplicaciones de poca complejidad, dando lugar a los llamados microautómatas y miniautómatas**.**
- Unidad de control modular (control distribuido). En aplicaciones de mayor complejidad, en lugar de una única CPU, existen varios módulos con tarjetas o unidades de proceso propias e incluso con sus interfaces de E/S. Es lo que se denomina estructura de multiprocesadores o con control distribuido. Cada procesador trabaja sobre subprogramas (partes del programa de usuario) o específicamente sobre otras aplicaciones concretas (regulación, posicionamiento, etc.) con su propio programa de tratamiento. En la estructura de multiprocesadores las unidades de proceso están conectadas a una unidad central (CPU maestra o principal) que gestiona el sistema en conjunto y permite el intercambio de datos entre el resto de las unidades e interfaces. En algunas aplicaciones es interesante duplicar la CPU o algún otro elemento del autómata, configuración de seguridad, de modo que esta redundancia permite un funcionamiento ininterrumpido aún en caso de avería, por conmutación al elemento de reserva.

El sistema de entradas-salidas de un autómata es el conjunto de interfaces E/S que hacen posible la conexión de la CPU con la planta y la identificación de las señales de ésta mediante una tabla de direcciones. Dada la modularidad característica de los autómatas, en casi todos ellos puede ampliarse el número de E/S mediante la conexión a la CPU de módulos de expansión con interfaces de E/S adicionales. En cuanto al sistema de entradas/salidas, las configuraciones pueden ser:

- Sistema de E/S centralizado. Es aquel en el que las interfaces de E/S se comunican con el autómata directamente a través de su bus interno, o a lo sumo mediando un amplificador de bus si se emplea un bastidor de ampliación, pero sin mediar procesadores de comunicación.
- Sistema de E/S distribuido. Es aquel en el que se necesitan procesadores de enlace de E/S conectados sobre el bus interno para la comunicación entre los módulos de E/S y la CPU. Estos procesadores de enlace son los encargados de amplificar, serializar y transmitir las informaciones entre las expansiones y la CPU del autómata base, mediante una línea común. En función de las distancias de conexión y de las prestaciones del enlace distribuido, éste puede ser local o remoto.

En cuanto a la capacidad de almacenamiento (memorias), en general los autómatas disponen de suficiente memoria como para realizar el mando y control de la mayoría de

los procesos industriales, si bien en casos de aplicaciones con gran volumen de información a gestionar puede ser necesaria la instalación de una memoria de masa adicional que, conectada directamente a las unidades de programación y bajo el control de la CPU puede intercambiar datos con la memoria de trabajo.

En resumen, dada la amplia gama de autómatas existente en el mercado y la modularización de sus elementos, es posible en cualquier caso encontrar la configuración adecuada para una determinada aplicación.

1.2.2. Interfaces de Entrada/Salida

Son muchos los automatismos industriales que necesitan de una cadena de realimentación para poder ejecutar un control en lazo cerrado con una regulación precisa y rápida.

La cadena de realimentación se alimenta de las magnitudes de la planta a controlar (entradas), que son captadas mediante sensores o transductores y cuyas salidas han de adaptarse en unos circuitos llamados de interface para su procesamiento por el autómata.

Por otra parte, las débiles señales de control generadas por el autómata han de actuar, generalmente previa amplificación, sobre la parte de potencia de la planta. A los elementos finales que actúan sobre la parte de potencia de la planta se les denomina accionamientos, y a los elementos intermedios que interpretan las señales de control y las amplifican se les denomina preaccionamientos.

En el control de cualquier proceso ha de existir un diálogo entre el operador y la máquina a controlar (diálogo hombre-máquina), y una comunicación entre el sistema de control y la máquina a controlar.

Traducido lo anterior a un autómata, supone que a éste le lleguen un conjunto de señales, de mando y de realimentación que se denominan entradas.

Por otra parte, el operador ha de conocer ciertos datos sobre la evolución del proceso y los accionamientos han de recibir las órdenes precisas para controlarlo, a todo lo cual se denominan salidas.

A todo el conjunto de entradas-salidas (E/S), es a lo se le denomina comunmente "medios de diálogo operador-máquina y máquina-controlador".

Tanto las entradas como las salidas pueden consistir en señales todo-nada (final de carrera, electroválvula, etc.), señales analógicas (velocidades, temperaturas, presiones) y señales digitales (contadores).

Una característica ventajosa y esencial de los autómatas programables, frente a otros controladores digitales, es el disponer de un bloque de interfaces E/S muy potente que les capacita para conectarse directamente con los sensores y accionamientos del

proceso. De ahí que de la adecuada elección de las interfaces E/S se derive una alta fiabilidad y disponibilidad del sistema.

Teniendo en cuenta lo anterior, es frecuente que sistemas de control complejos que incorporan un ordenador central con gran potencia de cálculo, utilicen como elemento de interface con el proceso industrial un autómata programable.

Además de los tipos de interface que intervienen en el proceso industrial propiamente dicho, existen otros tipos de interface dedicados a funciones específicas que incluso incluyen su propia CPU.

Además de las interfaces estándar digitales y analógicas, disponibles para todas las gamas de autómatas, existen otros tipos de interfaces llamadas específicas que, de modo opcional, pueden ser incorporadas al autómata base como tarjetas o módulos en las máquinas de las gamas media y alta.

Tales interfaces específicas hacen posible la conexión con elementos o procesos particulares de la planta, pudiendo realizar funciones muy variadas: manejo de señales particulares (códigos binarios, impulsos, señales analógicas débiles, etc.), regulación (PID, comparadores, control numérico), presentación de sinópticos y control (SCADA), posicionamiento de ejes, contadores rápidos, etc.

Por la función que realizan, las interfaces específicas pueden clasificarse como: de E/S especiales, de E/S inteligentes, y procesadores periféricos inteligentes.

Las interfaces con E/S especiales son interfaces análogas a las estándar pero que tratan señales particulares por su forma o por su aplicación, pero sin ningún control sobre las variables de la planta. El tratamiento de las señales está predeterminado y no es modificable por el usuario que sólo puede actuar sobre los modos de trabajo o algún parámetro de la tarjeta mediante instrucciones de programa o por micro-switchs externos.

Las interfaces con E/S inteligentes permiten diferentes modos de configuración ordenados por programa, e incorporan un control elemental que posibilita, utilizando señales binarias propias de la tarjeta, establecer lazos de regulación ON-OFF sobre variables de la planta, en funcionamiento transparente para la CPU. Desde la CPU y por el programa de usuario se envían las consignas y controles necesarios a estas interfaces. Tal forma de actuar descarga de trabajo a la unidad central y mejora de paso la capacidad de direccionamiento al poder acceder a señales de E/S que no han de aparecer en su memoria imagen.

Finalmente, los procesadores periféricos inteligentes son tarjetas o módulos que disponen de su propio procesador, memoria y puntos auxiliares de E/S. Tales procesadores incorporan de origen un programa o intérprete de programa especializado para la ejecución de una tarea específica, al que sólo se le han de fijar las consignas y los parámetros de aplicación para que, de forma autónoma y sin intervención de la CPU principal ejecute el programa de control.

Los procesadores periféricos, de uso mucho más general que las interfaces con E/S inteligentes, necesitan de mucha más información para definir, además de la configuración del periférico: las condiciones de aplicación y de entorno, las condiciones de control (respuesta en función de la evolución del proceso) y las consignas a seguir. A todos los anteriores valores, que en definitiva no programan sino que parametrizan la tarjeta, se les denomina programa de la interface y son enviados al periférico desde la CPU principal o desde la unidad de programación.

Aunque las tareas que realizan las interfaces específicas podrían realizarse por el programa de usuario desde la CPU principal, su especialización permite evitar o minimizar problemas tales como: a) Parte de las E/S estándar del autómata serían ocupadas para el tratamiento, a veces sin éxito, de señales específicas que por su naturaleza (por ejemplo tiempo de respuesta), pueden requerir un tratamiento especial, b) El aumento de la dificultad de programación, y c) El incremento del tiempo de ciclo del autómata que retardaría las reacciones del mismo ante el proceso y que, en el caso de procesamiento rápido de señales causaría problemas.

1.2.3. Programación

El autómata es una máquina electrónica integrando elementos de hardware que son capaces de comunicarse físicamente con un proceso para: a) Recibir desde el proceso algunas variables (analógicas o digitales) que determinan su estado y que se denominan señales de entrada, y b) Enviar otras variables que modifiquen tal estado en un determinado sentido, y que se denominan señales de salida.

Por su condición de programable, es necesaria la intervención de un operador humano que defina cómo ha de evolucionar el proceso y que intercambie información con el autómata para: a) Establecer mediante una secuencia de instrucciones (programa), cuál ha de ser la ley general de mando. De la ejecución de tal programa se obtienen las señales de salida o de control; y b) Intervenir, esporádica o continuamente sobre el proceso a efectos de informarse de su estado o de modificar su evolución. Al apartado a) se le denomina programación del autómata y a la secuencia de instrucciones programa de la aplicación. Al apartado b) se le llama comúnmente explotación de la aplicación, mediante la cual se pueden modificar ciertos parámetros (consignas, tiempos, módulos de cuenta, etc.), pero no modificar el programa.

Las intervenciones sobre la planta se realizan normalmente mediante el autómata, si bien en casos de fuerza mayor (parada de emergencia por motivos de seguridad), el operador puede actuar directamente sobre el proceso.

El intercambio de información entre autómata y proceso corre a cargo de las interfaces de E/S, en tanto que la comunicación con el operador para programación/explotación requiere de un software que haga de intérprete entre el sistema real y los deseos del usuario.

De este modo puede decirse que este software es "el conjunto de programas que posibilitan la utilización del hardware para el control y la explotación de las aplicaciones".

De acuerdo con la anterior definición, las herramientas de software son clasificables como: a) Sistemas operativos residentes en el propio autómata que tienen la misión de establecer las secuencias de intercambios de información, interpretar y ejecutar las instrucciones del usuario y vigilar el correcto funcionamiento del equipo, y b) Software de edición/depuración de programas, que permite al usuario introducir su propio programa sobre soportes físicos tipo cinta, disco, etc., modificarlo para perfeccionarlo, obtener la documentación que se precise del proceso y, en su caso sacar copias de seguridad.

Según los casos, el software de edición/depuración puede ser residente, es decir está instalado en la máquina o, es instalable sobre un terminal denominado unidad de programación que a su vez puede ser autónoma o dependiente de la CPU. Las misiones de la unidad de programación son fundamentalmente: a) Permitir la edición, depuración y modificación del programa y, b) Servir de interface física entre el usuario y el autómata, a fin de poder transferir programas y realizar la supervisión y el control del equipo.

Las instrucciones u órdenes que el usuario introduce en el programa han de ser entendibles por el autómata, es decir que han de ser codificadas mediante los lenguajes de programación y explotación prefijados por el fabricante.

Por tanto, el lenguaje de programación puede definirse como "el conjunto de símbolos y textos, entendibles por la unidad de programación, que utiliza el usuario para codificar sobre un autómata las leyes de control que desea". Asimismo, el lenguaje de explotación se definiría como "el conjunto de comandos y órdenes que, desde la CPU u otro terminal adecuado, puede enviar el usuario para conocer el estado del proceso, y en su caso para modificar alguna variable".

En esencia, el usuario introduce su secuencia de instrucciones (programa) en la unidad de programación, en un lenguaje que entienden ambos. La unidad de programación compila (convierte) las instrucciones del programa a unos códigos binarios, únicos que entiende el autómata (código máquina del autómata) y los almacena en la memoria. Finalmente el sistema operativo residente interpreta tales códigos binarios para activar los recursos físicos que requiere la ejecución del programa (procesador, interfaces E/S, etc.).

En la tarea de programación del autómata, es decir de establecer el programa a introducir en la unidad de programación, han de seguirse los siguientes pasos:

1. Establecer mediante un diagrama de flujo, una descripción literal o gráfica (GRAFCET, RdP, etc.) que indique qué es lo que se quiere que haga el sistema y en qué orden.
2. Identificar las señales de E/S del autómata.

3. Representar de forma algebraica (instrucciones literales o de textos) o gráfica (símbolos gráficos) un modelo del sistema de control con las funciones que intervienen, con las relaciones entre las mismas y con la secuencia a seguir.
4. Asignar a cada uno de los elementos que figuran en el modelo direcciones de E/S o internas.
5. Codificar la representación del paso 3 en instrucciones o símbolos entendibles por la unidad de programación (lenguaje de programación). Cada instrucción del programa consta de dos partes: el código de operación, que dice qué se ha de hacer y el código de los operandos (identificados por su dirección) que dicen sobre qué variables, o constantes, se ha de operar.
6. Transferir el conjunto de instrucciones escrito en la unidad de programación a la memoria del autómata.
7. Depurar, poner a punto el programa y guardar una copia de seguridad.

En cuanto a los lenguajes de programación a utilizar: literales o gráficos ha de decirse que depende de la aplicación a que se destina e incluso de la costumbre o hábito del programador. No obstante seguidamente se comentan las características fundamentales de ambos:

- Lenguajes literales.- Formados por instrucciones elementales del programa, cada una de las cuales es una secuencia de textos. Las instrucciones disponibles dependen de la complejidad del lenguaje y van desde muy sencillas funciones lógicas (AND, OR, NOR) hasta las estructuras complejas de programación de alto nivel (FOR ... NEXT, DO, WHILE, etc.), o instrucciones de manipulación de textos y valores numéricos, o instrucciones de acceso a bloques secuenciales (TIM, CNT, etc.).
- Lenguajes gráficos.- Tienen su origen en los esquemas eléctricos de relés y utilizan símbolos de contactos y bobinas para las instrucciones básicas y símbolos de bloques lógicos para las extensiones al lenguaje, con una potencia similar a la de los lenguajes literales de alto nivel y con la ventaja de visión de conjunto que proporciona la representación gráfica.

En la automatización de procesos usuales, de no mucha complejidad (cadenas de montaje, control de máquinas, etc.), puede utilizarse indistintamente un programa a base de lista de instrucciones o uno a base de diagrama de contactos, lenguajes básicos para la mayoría de autómatas. Tanto es así que varios fabricantes prevén en su software de programación sobre PC la posibilidad de transcodificación entre ellos con sencillas operaciones de compilación/descompilación.

Para aplicaciones complejas que requieran manipular largas cadenas de caracteres, realizar muchos cálculos, utilizar subrutinas o bloques de programación específicos (regulación PID, posicionamiento de ejes, contaje rápido, etc.), podría ser necesaria la utilización de lenguajes literales de alto nivel que también permiten programar sencillas sentencias booleanas o manejar contadores y temporizadores como listas de instrucciones.

La tendencia actual de los fabricantes en cuanto a los lenguajes de programación se centra en integrar los lenguajes antedichos en un lenguaje mixto que aúne la claridad de los lenguajes gráficos para las funciones combinacionales y secuenciales con la compacidad de los literales para el manejo de textos y los cálculos matemáticos.

Para el logro del mencionado lenguaje mixto se ha de actuar en los siguientes campos: a) Potenciar el uso de estructuras de programación avanzada en los lenguajes gráficos (GRAFCET a menor nivel y RdP a nivel superior) y aumentar el número de las actuales instrucciones de expansión, b) Permitir el uso de instrucciones literales dentro de un programa gráfico, tratándolas como tales instrucciones dentro del programa o como subrutinas accesibles desde él, y c) Desarrollar herramientas de edición con las que el usuario pueda almacenar sus sentencias en un bloque de expansión dentro de la librería disponible.

En definitiva y en lo referente a los lenguajes de programación, se prevé una evolución de los lenguajes gráficos en el sentido de hacerlos más potentes, más abiertos y de más fácil manejo por el usuario que, progresivamente podrá desarrollar sus aplicaciones sobre terminales tipo PC.

Los bloques funcionales, de mayor o menor complejidad, añaden al lenguaje básico instrucciones preprogramadas por el fabricante, de uso general en automatización (contadores, temporizadores, transferencias, registros, etc.) aumentando así la potencia de cálculo del autómata y simplificando su programación.

Tales bloques, que pueden introducirse en programas escritos en lenguajes literales, lenguajes de alto nivel y lenguajes gráficos, se clasifican en dos grupos en función de su forma de operar y su disponibilidad en el programa:

- Bloques secuenciales básicos.- Aquellos que son de uso generalizado en todo tipo de autómatas, incluidos los de la gama baja (contadores, biestables, temporizadores y registros de desplazamiento).
- Bloques de expansión o funciones.- Son los que hacen posible el tratamiento de variables numéricas y el registro de datos, con sentencias aritméticas (comparación, transferencias, etc.), aumentando así la potencia del lenguaje.

Los bloques secuenciales básicos se pueden considerar parte de los lenguajes básicos del autómata, en tanto que los bloques de expansión son extensiones de aquellos.

El usuario ha de adaptar los anteriores bloques funcionales a sus particulares necesidades fijando las condiciones de trabajo: nombre de los registros con los que desea operar (direcciones), valores de temporizaciones en los temporizadores, direcciones de origen y destino en las transferencias, etc..

Los bloques funcionales, en su caso más general hacen intervenir tres tipos de variables asociadas: a) Condiciones de operación (entradas).- Son las que definen la habilitación y control del bloque, b) Operandos de función.- Son aquellos sobre los que actúan las sentencias preprogramadas del bloque funcional, y c) Salidas asociadas cuyo estado depende de la ejecución del bloque.

A su vez, los operandos de función pueden ser: a) Parámetros iniciales que normalmente permanecen inalterados una vez fijados por programa o transferidos desde consola, y b) Datos de operación (constantes o variables expresadas en palabras de 8/16 bits y que muestran el estado de valores internos, E/S, resultados, etc.

Los datos (numéricos o alfanuméricos) que se usan como operandos pueden corresponder a: a) Constantes (números o caracteres ASCII) definidos en el programa. b) Textos preprogramados escritos en alguna unidad de memoria o dispositivo exterior. c) Variables numéricas (caso más usual) en: contadores o temporizadores (valores actuales), registros internos, canales de datos de 8/16 bits de E/S (p.e. resultado binario de una conversión A/D).

A pesar de que el usuario puede definir en su programa los anteriores datos en cualquier base (decimal, BCD, hexadecimal, etc), siendo que los datos internos que maneja el autómata son siempre binarios, han de ser convertidos automáticamente por el intérprete a tal base.

Según los fabricantes, un bloque funcional es considerado como elemento de un diagrama de contactos o como una sentencia literal en lista de instrucciones (con ciertas reglas de sintaxis).

Los programas de autómata para un proceso determinado pueden escribirse según estructuras monotarea y multitarea.

Si se define la tarea como "un conjunto de sentencias ejecutables que describen un tratamiento limitado y concreto sobre ciertas variables de un proceso", una estructura monotarea sobre una aplicación determinada es la que se desarrolla sobre una tarea única incluyendo la totalidad del programa, con todas sus variables de E/S y todas las sentencias de operación. Por el contrario, una estructura multitarea es aquella en que el programa está integrado por subprogramas, independientes o no, dando lugar a tareas aisladas referidas a tratamientos parciales y particulares de la aplicación (comunicaciones, supervisión, etc.).

Cuando la estructura es monotarea, la totalidad del programa (tarea única) se ejecuta periódicamente siguiendo un ciclo único de operación, en tanto que en estructuras multitarea se desarrollan varios ciclos a la vez durante la ejecución, uno por tarea, pudiendo además ejecutarse periódicamente o no las distintas tareas.

En cualquier caso, el ciclo de operación de cualquier tarea (tanto en estructuras mono como multi), recorre la típica secuencia de cuatro pasos: 1)Recogida de entradas. 2)Escrutinio del programa (de la tarea en operación). 3)Actualización de salidas, y 4)Servicio a terminales de explotación y/o periféricos.

Como puede verse, en una estructura multitarea cada tarea constituye una unidad de programación completa, con sus propias E/S, variables internas e instrucciones de control, lo cual permite optimizar la ejecución cuando se dispone de un hardware con varios procesadores adaptados a los distintos tipos de tratamiento de la información (tratamiento de textos, booleano, regulación, etc.). Como contrapartida, este hardware multiprocesador ha de ser coordinado por un gestor de recursos (software) que asegure a cada tarea el acceso a los mismos y evite la conflictividad en su uso compartido.

En efecto, el gestor de recursos es un ente software que puede ser parametrizado por el usuario a fin de fijar las prioridades de las tareas de su programa, y que dependiendo del fabricante puede correr sobre una CPU coordinadora (específica) o sobre la CPU principal.

Del mismo modo que los procesadores periféricos montados en bastidor pueden considerarse como parte de una misma unidad de control, los programas que se ejecutan sobre ellos (con lectura y generación de señales sin intervención de la CPU principal) pueden también considerarse como parte de un tratamiento multitarea.

La clasificación anterior de estructuras de programación (mono y multitarea), fuertemente dependiente de la configuración del hardware de la unidad de control, no ha de confundirse con las metodologías de programación a utilizar.

En efecto, una vez elegida para la aplicación a desarrollar un tipo de estructura mono o multitarea para su programación, cada una de las tareas parciales ha de ser programada en una secuencia de sentencias que puede obedecer a una metodología de programación lineal o estructurada.

En cuanto a la metodología a utilizar se dice que la programación es lineal cuando las sentencias están ordenadas en el mismo orden en que se van a consultar, y en su caso a ejecutar. Por el contrario, se dice que la programación es estructurada cuando la tarea de control está repartida en módulos o subprogramas relativos a distintas funciones y cuya ejecución puede ser necesaria varias veces dentro de un mismo ciclo de ejecución del autómata.

A pesar de que la programación estructurada de una tarea se realiza con mayor eficiencia en autómatas con coprocesadores en su CPU que estén especializados en las funciones de cada subprograma, resulta también posible sobre autómatas con CPU única, que ejecutará los subprogramas o módulos en el orden en que sean llamados por el programa principal. En este sentido ha de hacerse constar que existen módulos pregrabados por el fabricante (para realizar tareas concretas o gobernar interfaces específicas) que pueden ser adaptados por el usuario a su aplicación concreta con sólo parametrizarlos adecuadamente.

En resumen, puede decirse que si bien, tanto los tratamientos monotarea como los multitarea pueden desarrollarse en autómatas con un solo procesador o con varios procesadores, ha de ser el programador quien según la complejidad de la aplicación, la estructure o subdivida de la forma más eficiente posible de acuerdo con los recursos hardware de que disponga.

Finalmente, obsérvese como la programación en lenguajes gráficos (GRAFCET o RdP) también puede ser considerada como una programación estructurada especialmente útil para la programación de los procesos secuenciales.

En cualquier aplicación con autómata programable, tanto durante la fase de concepción, edición y depuración del programa como durante la fase de operación o explotación del sistema, es necesaria una comunicación o diálogo hombre-máquina. En la primera fase

el hombre (programador) carga el programa en la memoria del autómata, verifica su funcionamiento observando la evolución de las variables (monitorización) y en su caso modifica su estado en variables lógicas o su valor en variables alfanuméricas (forzado). En la segunda fase o fase de explotación, sigue siendo conveniente y a menudo imprescindible la comunicación entre el hombre (operador) y la planta, a fin de conocer (monitorizar) a través del autómata los valores de ciertas variables claves para el correcto desarrollo del proceso y su control, variables que en su caso pueden modificarse (forzado).

Las comunicaciones descritas entre el hombre (programador/operador/usuario) y el autómata se realizan mediante dispositivos específicos o mediante la utilización de un entorno software que corre sobre un PC. Los dispositivos específicos, genéricamente denominados "Unidades de Programación y Servicio" proporcionan la comunicación entre el programador y la máquina en la fase de programación y la comunicación entre la planta y el usuario en la fase de observación y control (explotación).

Las anteriores comunicaciones se realizan siempre sobre el autómata, que para ello dispone de los conectores adecuados, en la CPU para la programación y en la CPU o procesadores auxiliares de comunicaciones para la explotación y el servicio.

En general existe una gran variedad de dispositivos conectables a un autómata, bien directamente o vía modem, aportando soluciones a necesidades del proceso muy dispares: unidades específicas de programación o entornos software sobre PC, para la edición y puesta a punto de programas de autómata; unidades de explotación desde un visualizador de baja funcionalidad hasta un terminal gráfico interactivo pasando por visualización de mensajes asociados a la evolución del programa, impresión de textos, intercambio de datos con otros equipos, etc.

1.2.4. Aplicaciones

La finalidad primordial de la automatización industrial es la de gobernar la evolución de un proceso sin la intervención, salvo esporádica, de un operador.

Cuando se trata de procesos de fabricación rígidos con poca variación en el tiempo o de tipo independiente y sin interrelación con tratamientos anteriores o posteriores, la finalidad se logra programando sobre los controles locales de la planta las instrucciones de control que se deseen y cerrando los bucles de control precisos para que los valores de las variables significativas estén dentro del intervalo fijado por las señales de consigna.

La mayoría de los procesos industriales no cumplen las condiciones del punto anterior, sino que han de ser flexibles adaptándose a determinados aspectos y además están fuertemente interrelacionados entre sí no ya sólo por exigencias de factores propios de la producción sino por otros ajenos como p.e. la calidad total, minimización de costes de stocks, ahorro de energía, etc.

Tales necesidades han motivado la aparición de sistemas automatizados de control muy complejos que han de ser dotados de funciones adicionales a las básicas de ejecución de tareas y monitorización del proceso. Cuestiones tales como la gestión de menús de producción, la toma automatizada de decisiones, generación de históricos, gestión de alarmas, emisión de informes, etc. han de ser atendidas en procesos de cierta complejidad.

Las funciones asociadas a los niveles de control de producción y supervisión de planta en un modelo jerárquico de automatización requieren el conocimiento de la realidad de la planta y la capacidad de interactuación sobre ella.

Los sistemas digitales que utiliza la informática industrial son actualmente capaces de implementar sobre la pantalla de un ordenador los paneles de control tradicionales con indicadores luminosos, pulsadores, interruptores y aparatos de medida cableados de forma rígida, con altos costes de instalación y mantenimiento. Con una supervisión inteligente que permita al operador interactuar de forma dinámica con el proceso, y con la ayuda de factores tales como la capacidad de almacenamiento y proceso del ordenador y su facilidad de comunicación con los controladores de planta, tal operador detecta de inmediato las variaciones significativas en el proceso en tanto observa su evolución en el tiempo y sus tendencias.

Para un sistema típico el control directo de la planta corre a cargo de los autómatas programables, en tanto que el ordenador a ellos conectado se encarga de las funciones de diálogo con el operador, tratamiento de la información del proceso y control de la producción. Con tal estructura, el ordenador se limita a la supervisión y control de los elementos de regulación locales instalados en la planta y al tratamiento y presentación de la información, pero no actúa directamente sobre la planta. Aunque eventualmente el ordenador podría ejercer acciones directas de control como lecturas de sensores o activación o desactivación de actuadores, dotado de un hardware adicional conectado a sus buses internos, no es usual tal forma de actuación.

Apoyándose en la estructura de dispositivos locales, el ordenador u ordenadores se conectan a ellos mediante líneas de interconexión digital, tales como buses de campo o redes locales por las que reciben información sobre la evolución del proceso (obtención de datos) y envía comandos u órdenes del tipo arranque, parada, cambios de producción, etc. para su gobierno (control de producción).

A los programas requeridos y, en su caso el hardware adicional necesario, se les denomina genéricamente como sistemas SCADA ("Supervisory Control And Data Acquisition") y pueden ofrecer prestaciones avanzadas como:

- Crear paneles de alarmas, con registro de incidencias que exijan la presencia del operador para darse por enterado de una parada o una situación de alarma.
- Posibilidad de generación de históricos de señales de planta que pueden ser volcados sobre una impresora o sobre una hoja de cálculo para su procesado inmediato.
- Imprimir informes, avisos y documentación varia sobre la evolución del proceso.
- Ejecutar programas modificando la ley de control o cambio total del programa ante determinadas condiciones en el proceso.

- Posibilidad de una programación numérica que posibilite la realización de complejos cálculos matemáticos sobre la CPU del ordenador, más especializada que la del autómata.

Con las anteriores prestaciones es posible desarrollar aplicaciones basadas en el PC que recoja los datos, analice las señales, haga presentaciones en pantalla, envíe datos a impresora o disco, controle actuadores, etc.

En general, un paquete SCADA incluye dos programas: Editor y Ejecutor ("Run-Time"). Con el Editor se generan las aplicaciones antes descritas haciendo uso de los editores, macros, lenguajes y ayudas de que dispone, y con el Ejecutor se compilan a fin de obtener el fichero .EXE de ejecución.

1.2.5. Referencias

Páginas WEB

http://www.ee.umanitoba.ca/tech.archive
TECHNICAL REPORTS sobre AUTOMATA, COMPUTATIONAL INTELLIGENCE, DATA AND SIGNAL COMPRESSION, FORMAL METHODS IN SYSTEM DESIGN, FUZZY SYSTEMS, GENETIC ALGORITHMS, MATHEMATICS, MICROELECTRONICS AND SOFTWARE SYSTEMS, NEURAL NETWORKS, PARALLEL PROCESSING, PETRI NETS, REAL-TIME SYSTEMS, SPECIFICATION AND DESCRIPTION LANGUAGE (SDL)

http://www.informatik.uni-stuttgart.de/ifi/ti/fap98/fap98.html
WORKSHOP ON FORMAL LANGUAGES, AUTOMATA AND PETRI-NETS

http://www.aisa.uvigo.es/software.html
SOFTWARE DE LIBRE DISPOSICIÓN DESARROLLADO EN EL DEPARTAMENTO

http://www.cs.duke.edu/~magda/flap/index.html
FORMAL LANGUAGES & AUTOMATA PACKAGE. Demo sobre una herramienta gráfica usada para aprender los conceptos de los lenguajes del autómata.

Bibliografía y artículos

- Mayol, Albert. “Autómatas Programables”, Colección Prodúctica. Marcombo, S.A. (1987)
- Michel, G. “Autómatas Programables Industriales: Arquitectura y aplicaciones”. Marcombo, S.A. (1990)
- Mandado, Enrique. “Sistemas electrónicos digitales”, Marcombo, S.A. (4ª edición, 1981)
- Taub H. Y Schilling D. “Electrónica digital integrada”, Marcombo, S.A. (1980)
- Blanchard, Michel y otros. “Le GRAFCET, de nouveaux concepts”. ADEPA – Cepadues Editions (1985)

- Varios autores. “GEMMA (Guide d’Étude des Modes de Marches et d’Arrêts)”. ADEPA (Agence pour le Développement de la Productique Appliquée), 13/17 Rue Périer, BP 5492123, Montrouge, Cedex.
- Kuo, Benjamin C., Sistemas Automáticos de Control, CECSA (2ª Edición 1970)
- Ogata, Katsuhiko, Ingeniería de Control Moderna, Prentice-Hall (1980)
- Fröhr, Friedrich y Orttenburger, Introducción al Control Electrónico, Marcombo, S.A. – Siemens (1986)
- Ras, Enrique, Teoría de circuitos, Marcombo, S.A. (1969)
- Ras, Enrique, Análisis de Fourier y Cálculo Operacional Aplicados a la Electrotecnia, Marcombo, S.A. (1979)
- Pizziola, Antonio, Electrónica industrial y servomecanismos, Ediciones Don Bosco (1974)
- SIEMENS. Cat. Núm. ST 54. Autómatas Programables S5-135U Y S5-150U
- IDEC-IZUMI. Users Manual EM228-4. MICRO-1 programmable controller
- TÉLÉMECANIQUE TSX D11 000. Microautómata TSX 17. Instalación
- IDEC-IZUMI. Cat. Núm. EP574-0. FA-3S Series
- Hitachi. Cat. Núm. NJI 008 (X). Operation Manual E-Series
- Toshiba EX250/500. Cat. Núm. EBE-113002. Instruction Manual
- OMRON. Cat. Núm. X50-E1-2. The C-Series solution
- Varios Autores. Transductores y Medidores Electrónicos: Serie Mundo Electrónico Ed. Marcombo (1983)
- Balcells, José; Romeral, José L. Sensores y Transductores Industriales (Monografía docente) Ed. Delegación de Alumnos ETSEIT
- Norton, Harry N. Sensores y Analizadores. Ed. Gustavo Gili (1984)
- Pallás, Ramón. Transductores y Acondicionadores de Señal. Ed. Marcombo (1989)
- Tobey G.E., Graeme J.G., Huelsman L.P. Operational Amplifiers: Design and applications. McGraw-Hill (Serie Burr-Brown), 1971
- Catálogos de autómatas SIEMENS tipos S-100U, S-115U, S-135U y S-150U
- Catálogos de autómatas TÉLÉMECANIQUE tipos TSX-17, TSX-47
- Catálogos de autómatas OMROM
- IDEC-IZUMI. Users Manual EM073-0. FA-2Junior Series
- CIMTEL – TÉLÉMECANIQUE. Los Automatismos Programables Integrados. 1992
- SIEMENS. Cat. Núm. ST 54.1. Autómatas Programables S5-135U y S5-155U
- SIEMENS. Cat. Núm ST 52.3. Autómatas Programables S5-115U y S5-115H
- ALLEN-BRADLEY. Product Guide. Noviembre 1987
- Mayol i Badía. Autómatas Programables. Serie Prodúctica núm. 3. A. Ed. Marcombo, 1987
- Varios autores. Los Automatismos Programables. Ed. Citef, 1991
- SIEMENS. Autómata Programable S5-100U (CPU 100). Edición Octubre 1986
- OMRON. Manual de Programación. SYSMAC Serie C. Febrero 1991
- TOSHIBA. Programming Manual. EX Series EX200B/250/500, 1986
- TÉLÉMECANIQUE. Programación TSX T407. Contactos TSX 67/87. Lenguaje Contactos. Manual 10. 1986
- IDEC-IZUMI. Fa-3S Series. User’s Manual. FP3S-CP11 Standard CPU Marzo 1992. FA-2Junior Series. User’s Manual. Abril 1990
- SIEMENS. “Aparatos de Programación PG-615 y PG-685”. Cat. ST-59. 1991

- Romeral, José L. “Curso de Especialización en Automatización Industrial”. UPC X-0455, 1994
- MITORMAT S.L. Terrassa. “Manual HTERM”. 1993
- IDEC-IZUMI “Micro Programmable Controller Micro3”, User’s Manual Cat. Núm. EM 289-0. Marzo 1993
- Is Your Company Ready Together? The C Series Solution. Cat. X50 E 1-2. Diciembre 1989
- Carracedo Gallardo, Justo. Redes locales en la industria. Ed. Marcombo, 1988
- SIEMENS AG. Telecomunicación digital. Marcombo, S.A. 1988
- Ballcells J., Daura F., Esparza R., Pallás R. Interferencias Electromagnéticas en Sistemas Electrónicos. Marcombo, S.A. 1992
- Dunlop J., Smith D.J. Ingeniería de las Telecomunicaciones. Gustavo Gili. 1988
- Cambell Joe. El libro del RS-232. Ediciones Anaya Multimedia S.A. 1985
- PROFIBUS Nutzerorganisation e.V. Geschäftstelle Pelzstraβe, 5. D-5305 Alfter, Alemania
- THE BITBUS™ INTERCONNECT SERIAL CONTROL BUS. Embeded Microcontrollers Databook. Varios autores INTEL, 1987
- González, Ignacio. Los PCs en la Industria. Automática e Instrumentación, nº 226, pág. 76, 1992
- Soriano, Antonio. Autómatas programables frente a computadores personales en sistemas de control. Automática e Instrumentación, nº 196, pág. 811, 1989
- Paredes, Pedro. El PC en aplicaciones industriales. Automática e Instrumentación, nº 231, pág. 73, 1993
- Dexter, Arthur L. Microcomputers Bus Structures and Bus Interface Design. Ed. Marcel Dekker Inc., 1986/Electrical Engineering and Electronics
- Desmond, Michael. The VESA Local Bus PC World. 1993
- INTEL CORP. PUB. Nº 241761.001. The PCI Local Bus: A Technical Overview
- De La Torre, Miguel. El Bus VME. Conceptos básicos. Mundo Electrónico, nº 224-225, pág. 52, 1992
- PEP Modular Computers. VME For Everyone. Autobahn Technology
- Mandado E., Marcos J., Pérez S.A. Controladores lógicos y autómatas programables. Marcombo S.A. 1990
- Varios autores. The TTL Data Book for Design Engineers (Vol 1). Texas Instruments, 1984

1.3. Los SCADAs

Un sistema SCADA podría definirse como "una aplicación software diseñada especialmente para funcionar sobre ordenadores de control de producción", comunicada con la planta mediante interconexión digital con los reguladores locales básicos y, con el usuario mediante interfaces gráficas de alto nivel como pantallas táctiles, lápices ópticos, ratones, etc.

Desde la pantalla del ordenador, que es configurada por el usuario y es fácilmente modificable, el sistema permite la comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, sistemas de dosificación, máquinas de control numérico, etc.) para controlar el proceso de forma automática. Además, de

toda la información generada en el proceso productivo, se envía a cada usuario de la empresa (supervisión, ingeniería, control de calidad, mantenimiento, etc.) la que necesita.

A fin de que la instalación de un sistema SCADA sea perfectamente aprovechada debe cumplir ciertas condiciones: a) Ha de ser un sistema de arquitectura abierta, capaz de ampliarse y acomodarse a necesidades cambiantes. b) Ha de permitir una fácil comunicación y de forma transparente para el operador, tanto con el equipo de planta (drivers de comunicación con API, CN, PID, etc.) como con el resto de la empresa (gestión y acceso a redes locales), y c) Ha de utilizar programas fáciles de instalar, con no demasiadas exigencias de hardware, sencillos de utilizar, y disponiendo de interfaces afines al usuario, tales como imágenes, sonido, pantallas táctiles, etc.

Cumpliendo tales condiciones, el sistema SCADA llega a ser una herramienta fundamental en la organización del proceso productivo permitiendo el establecimiento de estrategias avanzadas de gestión con respuestas inmediatas de la producción.

Como soporte hardware de los programas SCADA se utilizan normalmente ordenadores convencionales (miniordenadores PC, ordenadores portátiles y estaciones de trabajo).

Aunque pueden emplearse arquitecturas económicas basadas en ordenadores PC con sistema operativo DOS/Windows y software adicional con funcionalidades del tipo interrupciones, comunicación en red, etc., para mejorar sus prestaciones, la utilización de ordenadores con sistemas operativos más completos (VAX/VMS, Unix, Windows NT, etc.) y arquitecturas cliente-servidor aptas para compartir recursos informáticos (datos y aplicaciones) permiten ofrecer programas capaces de atender simultáneamente a varios servicios. Por ejemplo, un operador puede estar viendo informes de incidencias desde un ordenador industrial con sistema operativo Unix en tanto que otro está modificando la evolución del proceso desde una estación de trabajo y un tercero monitorizando la situación en planta desde un PC. Estas arquitecturas cliente-servidor se utilizan en grandes aplicaciones a fin de repartir los datos procesados entre distintos ordenadores. Por ejemplo se podrían establecer nudos separados a modo de servidores para la demanda de tareas tales como monitorización y procesado de alarmas, comunicación con los dispositivos de E/S de campo, registro y almacenado de datos para históricos, análisis de tendencias, etc.

Un factor esencial a tener en cuenta para la elección del sistema es la capacidad del sistema operativo sobre el que corre la aplicación para soportar multitarea y/o multiusuario. La capacidad multiusuario es de utilidad cuando en aplicaciones complejas se han de disponer en lugares separados funciones específicas en cada uno de ellos. Tales sistemas usualmente se articulan mediante redes de área local.

Para aplicaciones de media y baja complejidad, actualmente es habitual la utilización de ordenadores PC, con una CPU tanto más potente cuanto menores sean los tiempos de respuesta exigidos y configuraciones tanto más robustas cuanto más agresivas sean las condiciones ambientales de la aplicación. A pesar de leves inconvenientes, Windows se mantiene como el sistema operativo típico de soporte con un paquete software SCADA que corre sobre él en multitarea y monousuario. Windows aprovecha las ventajas de un

entorno familiar multitarea con un intercambio de datos entre aplicaciones muy sencillo (DDE, OLE), una potente interfaz de usuario (GUI) y características de sistema abierto para incorporar fácilmente nuevos software de interfaz audiovisual o multimedia.

Mediante interfaces serie estándar (RS-232, RS-422 o RS-485) y utilizando los protocolos adecuados, ya incluidos en el propio SCADA, se realiza la comunicación con los elementos de campo.

También es importante saber el número de pantallas gráficas de representación (sinópticos) que el sistema puede soportar, así como el número máximo de variables a manipular.

Finalmente son también datos a considerar la capacidad de intercambio de datos con otros entornos como dBase o Excel para integrar sus funciones dentro de la aplicación (cálculos estadísticos, gráficos, presentaciones, etc.) y, la posibilidad de programación de funciones complejas incluyendo en el SCADA ficheros y rutinas escritos en lenguajes de propósito general (C, Pascal, Basic, etc.).

1.3.1. Estructura y configuración

Para la selección de un paquete SCADA, como en la mayoría de aplicaciones informáticas, han de distinguirse las dos posibilidades siguientes:

- Puede encargarse a una empresa especializada el desarrollo de un software (y eventualmente incluso el hardware), específicamente orientado a una aplicación concreta. En el encargo se definen las especificaciones del cliente: pantallas, sinópticos, señales de control, históricos, informes, gráficos, etc. que la empresa programa y compila en una aplicación cerrada, como un "traje a medida". La ventaja de esta posibilidad es que el programa responde perfectamente a los deseos del cliente (usuario), pero como inconveniente la dependencia del programador y la dificultad de obtener los protocolos de comunicación con los actuales elementos de campo y los que en el futuro puedan instalarse.
- La otra posibilidad consiste en la utilización de un paquete genérico comercial que, el usuario ha de parametrizar adecuadamente para adaptarlo a su aplicación particular. Con esta solución, el usuario puede desarrollar su aplicación ayudándose de los editores gráficos y funcionales y de los drivers de comunicación suministrados por el proveedor. Para tal desarrollo, el usuario dibuja los sinópticos de planta indicando las variables y textos que quiere visualizar, define las relaciones entre variables, configura las comunicaciones, etc., es decir, parametriza el paquete de acuerdo con sus necesidades actuales, si bien este permanece abierto a modificaciones o ampliaciones posteriores hasta donde lo permita la licencia del suministrador del paquete o su capacidad.

Esta última posibilidad es la más utilizada en aplicaciones de media y baja complejidad dadas sus características de modularidad, escalabilidad y autonomía.

No obstante, sea cual sea la solución adoptada, el sistema SCADA ha de realizar las tres funciones principales siguientes: a) Recoger la información que ha de procesar y

almacenar (adquisición de datos). b) Visualización de la evolución del proceso (monitorización) y c) Modificar, si es necesario, la evolución del proceso actuando directamente sobre él o sobre los reguladores autónomos básicos tales como alarmas, consignas, menús, etc. (supervisión y control).

Para el desarrollo de las tres funciones expuestas el sistema dispone de los siguientes bloques software que se irán viendo.

El módulo de configuración define el entorno de trabajo de su sistema SCADA y lo adapta a su aplicación particular. En efecto, con él define las pantallas de texto o gráficas que quiere utilizar, para lo cual las genera desde el mismo sistema SCADA o las aprovecha de otra aplicación anterior, apoyándose en el editor gráfico que incorpora y con el que puede dibujar a nivel de pixel cualquier figura o utilizar elementos ya disponibles, tales como círculos, cuadrados, líneas, etc. con las funciones clásicas de mover, copiar, desplazar, etc. Una vez definidas las pantallas, se establece la relación entre ellas determinando el orden de presentación, los enlaces y su accesibilidad a diferentes operarios. Teniendo cada pantalla asociadas sus propiedades configurables, el mantenimiento de las mismas resulta sencillo. Igualmente, en esta fase de configuración se seleccionan los drivers de comunicación para el enlace con los elementos de campo y su conexión o no en red. En ciertos sistemas es también en esta fase de configuración donde se definen las variables a visualizar, procesar o controlar, identificadas por nombres o etiquetas para su posterior referenciado y programación.

1.3.2. La supervisión

El ordenador habilita las funciones de supervisión de la planta al operador mediante una ventana abierta a la planta desde el teclado, el ratón y el monitor. El proceso a supervisar es representado por sinópticos gráficos, almacenados en el ordenador de proceso y generados previamente en la fase de configuración. Los cambios que se producen en la planta a lo largo del tiempo pueden ser contemplados en el gráfico mediante zonas dinámicas que varían con tales cambios. En efecto, los sinópticos presentan zonas activas cambiantes en forma y color, siguiendo la evolución del proceso en la planta o las acciones del operador. Así p.e. la pantalla podría configurarse de modo que muestre las tres áreas siguientes: a) proceso global, b) partes significativas del proceso y c) zona con esquema de asignación de teclas para el mando de las posibles acciones. Cada una de las zonas pueden asimismo incluir valores numéricos o alfanuméricos de variables según la evolución de la planta.

Han de tenerse en cuenta ciertas consideraciones o consejos que ayudan al diseño de pantallas:

- La apariencia de las pantallas ha de ser tal que muestre zonas diferenciadas para presentar la planta (sinópticos), las botoneras y entradas de mando (control) y, los mensajes de salida del sistema (estados, alarmas).
- El proceso se debe representar preferentemente con sinópticos desarrollados de izquierda a derecha siguiendo el hábito de lectura y escritura.
- Es conveniente que las señales de control estén agrupadas por funciones, y que la información sobre cada elemento gráfico se coloque sobre él.

- La utilización de colores hace más cómoda la interpretación rápida de la información, si bien no han de utilizarse demasiado número de ellos.
- La utilización de colores tales como el rojo (alarma, peligro) o verde (funcionamiento normal) es recomendable.
- Como complemento a los colores y en previsión de bajas iluminaciones, alto brillo, etc., es aconsejable reforzar las señales de alarma y peligro acompañándolas de símbolos, intermitencias, gráficos dinámicos, etc., si bien cuando se utilicen intermitencias, éstas no deben afectar al mensaje a fin de no dificultar su lectura.

Para procesos complejos puede haber necesidad de definir varias pantallas dentro de la misma aplicación, disponiendo cada una de ellas de sus sinópticos, zonas activas y variables asociadas, y representando distintas secuencias como parte del proceso global. El cambio de pantallas se produce automáticamente según va evolucionando el proceso o bien a petición del operador. Ante una situación anormal, el sistema además de reaccionar ante tal situación en la forma preprogramada, puede sugerir al operador en forma de texto una orientación sobre qué acciones correctoras debe ejercer.

Las actuaciones del operador pueden ser, bien sobre variables intermedias en el ordenador o autómata, o bien sobre variables directas de la planta, posicionando el ratón sobre alguna zona activa y modificando el valor de la variable seleccionada. Determinadas acciones de mando pueden estar reservadas sólo a operadores autorizados que para ejercerlas han de activar previamente su código personal.

1.3.3. Elementos del SCADA

Sobre cada pantalla o zona activa es posible programar relaciones entre variables del ordenador o autómata, que se ejecutarán continuamente en tanto la pantalla esté activa. Tales programaciones se realizan mediante bloques escritos en lenguajes de alto nivel (Pascal, C, etc.), o parametrizando macroinstrucciones suministradas por el fabricante, y pueden llevar asociada una plantilla de tiempos para definir su frecuencia de ejecución.

Usualmente el sistema SCADA utiliza a los dispositivos de campo (generalmente autómatas) como controladores directos de planta, reservándose para sí la supervisión (control del proceso, análisis de tendencias, generación de históricos, gestión de alarmas, etc.).

El programa de mando que el sistema SCADA ejecuta de forma automática, relaciona las variables para conseguir:

- Acciones de mando automático, previamente programadas, función de las señales de entrada, salida o sus combinaciones.
- Maniobras o secuencias de acciones de mando.
- Animaciones sobre figuras y dibujos que relacionen su color, forma, tamaño, parpadeo, etc. con el valor actual de ciertas variables.
- Procedimientos de arranque/parada del proceso (en frío, en caliente, condicionado al valor de ciertas variables, etc.).
- Gestión de recetas modificando los parámetros del proceso (estado de variables, conteo, consignas de tiempo, etc.), de una forma preprogramada en función del tiempo o dinámicamente a la vista de la evolución de la planta.

Los comandos (maniobras y secuencias de mando) son las más importantes de las acciones anteriores, puesto que implementan la comunicación hombre-máquina con que el usuario puede controlar el proceso.

Para la entrada de comandos se deberán tener en cuenta de modo general las siguientes consideraciones:

- Para ordenar una función (cambiar un estado, introducir un valor, etc.), el usuario ha de tener posibilidad de conocer el estado o valor anterior y una explicación de lo que hace la función.
- Cuando en la entrada a un comando el rango numérico o alfanumérico de entrada esté limitado, será conveniente indicar tal rango en la explicación que acompañe al comando. P.e. "introduzca número de piezas (100-110 u.)".
- En el caso de que la explicación que acompañe al proceso sea demasiado larga, puede optarse por: a) Mostrar la explicación en una ventana que se abra específicamente para el comando. b) Recurrir a mensajes giratorios y c) Hacer uso de abreviaturas, símbolos, etc.
- A efectos de comprobar que un estado o valor ha sido modificado, la planta ha de devolver (feed-back) al usuario el nuevo valor actual.
- Si dentro de las alternativas posibles en un comando hay una de utilización frecuente, ésta ha de mostrarse como alternativa por defecto, que posteriormente el usuario podrá o no modificar.
- La aceptación de comandos que supongan acciones de mucha responsabilidad (cambios de producción, detención total, etc.), necesitarán en general de un código (password) de identificación del operador.
- El usuario, desde cualquier pantalla, ha de poder acceder fácilmente a las paradas de alarma, seguridad o emergencia que habrán de aparecer en gran tamaño y/o en símbolos gráficos.

En el módulo de proceso existe un caso particular de relaciones programadas que constituye lo que se denomina gestión de alarmas, pudiéndose definir intervalos de variación de las variables (lógicas o numéricas), fuera de los cuales se dan condiciones de alarma, y en caso de que varias aparezcan al mismo tiempo, su orden de prioridad. En caso de detectarse una alarma, el sistema, de forma preprogramada, reacciona advirtiendo al operador, con señales acústicas, con textos parpadeantes, con cambios de color, etc. El operador, ante tal advertencia, puede darse por enterado de la anomalía, modificar alguna variable del proceso o dirigirse a una pantalla auxiliar específica que dé instrucciones de cómo tratar tal alarma.

Las alarmas ocurridas quedan registradas con el estado que las produjo y codificadas como: a) Alarma activa no reconocida, es el caso en que ante una alarma el operador no ha pulsado el botón de "enterado" y por tanto no ha efectuado ninguna corrección. b) Alarma activa reconocida, el operador ha pulsado el botón de "enterado", aunque la alarma permanece activa por no haberse efectuado aun ninguna corrección. y c) Alarma inactiva, el operador ha efectuado las correcciones pertinentes y la alarma se ha eliminado.

En el registro de alarmas quedan reflejados los datos correspondientes a la hora en que se produjeron las alarmas, la hora en que se desactivaron y si el operador se dio o no por enterado de ella.

1.3.4. Aplicaciones

La gestión y archivo de datos se realiza en bloque del SCADA que los almacena y procesa, codificados de tal modo que puedan ser enviados a impresoras o registradores (hardware) y a hojas de cálculo o bases de datos (software) del propio sistema.

Los datos de planta seleccionados pueden ser capturados con una periodicidad fijada y almacenados en formatos adecuados para ser vertidos en forma de históricos, estadísticas, análisis de calidad, etc. a periféricos gráficos o alfanuméricos. Lo anterior es posible gracias a un intercambio dinámico de información entre el SCADA y las aplicaciones que corren bajo el mismo sistema operativo.

Un ejemplo de lo anterior es el protocolo DDE de Windows que hace posible el intercambio de datos en tiempo real, aunque con las limitaciones propias de Windows. A tal fin, el SCADA opera como un servidor DDE cargando en memoria variables de planta que posteriormente serán usadas en diversas aplicaciones.

Los datos, después de procesados, pueden presentarse como histogramas, gráficas analógicas, gráficos tridimensionales, etc., formando históricos y resúmenes para el posterior análisis del funcionamiento de la totalidad de la planta que, permitirá conocer la influencia de cada elemento en el proceso y su intensidad.

Asimismo, los datos que ha procesado el módulo de control pueden ser enviados de inmediato a ficheros auxiliares donde quedan almacenados hasta ser llamados para su tratamiento posterior.

El bloque de comunicaciones realiza la transferencia de información entre el hardware del SCADA y la planta, y entre aquél y todos los demás elementos de gestión.

En el bloque de comunicaciones se encuentran los "drivers" de conexión con todos los elementos digitales conectados, es decir, los programas software a los que se les encomienda la iniciación de los enlaces, la creación de los formatos, la ordenación de las transferencias, etc. En resumen, la gestión de los protocolos de comunicación, que pueden ser abiertos como FieldBus, ModBus, Map, etc., o específicos del fabricante, caso éste en el que frecuentemente requieren una licencia del mismo para incluirlos en la aplicación.

Tanto el protocolo como los parámetros de la aplicación (puertos, velocidad, etc.) son activados de forma automática cuando el usuario elige el modelo de dispositivo E/S de campo: autómatas, lectores de barras, reguladores PID, analizadores, etc., es decir durante la configuración.

En aplicaciones complejas, en los SCADA distribuidos en arquitecturas del tipo cliente-servidor, los bloques de comunicaciones son asimismo los encargados de los enlaces

(generalmente establecidos sobre una red local) entre los distintos ordenadores de proceso que soportan la aplicación.

En el entorno Windows, los módulos Net-DDE aportan todas las ventajas de los protocolos DDE a los sistemas en red, de modo que pueden (sin necesidad de servidores) establecer comunicaciones punto a punto permitiendo la conectividad del software entre aplicaciones que corren sobre diversas plataformas estándar.

1.3.5. Referencias

Páginas WEB

http://www.csi.ull.es/docencia/asignaturas/332.html
TÉCNICAS DE SIMULACIÓN

http://www.keytelemetering.com/
Building SCADA Systems, Remote Terminals (RTU), Data Radio Modems, Voice Reporter, Solar Power

http://www.advancedscadaandtelemetering.com
Advanced SCADA and Telemetering

http://www.scadaengine.com/
SCADA Gateway and ActiveX component for developing Man Machine Interfaces. Drivers for Modbus, Carrier, BACnet and Atlas systems.

http://www.autosoln.com/
SCADA-Automation Solutions. Monitoring, data acquisition and remote control by integrating various HMI's, RTU's, and PLC's with OPC Server...

Bibliografía y artículos

- FEATURES - Tech Update - SCADA Evolves Towards MES - Today's SCADA systems are evolving beyond supervisory control and data acquisition with greater functionalities and integration capabilities. (2001)
- Acebes, L. F., Diseño de un scada utilizando un simulador OPC / L. F. Acebes, J. M. Zamarreño (2001)
- Clements-Jewery, K., The PLC workbook : programmable logic controllers made easy / K. Clements-Jewery, W. Jeffcoat (1996)
- Ventajas del PLC en control de procesos. Industria internacional; 2001; Nº 398 ; pp. 28
  SIEMENS. PC Industrial. Los Sistemas Operativos. Sinopsis. Edición 8/90
- NATIONAL INSTRUMENTS, Seminario de LabWindows/CVI: Ref. 350181A-01. 1994
- Lamaison Urioste, Rafael. El PC Industrial. Curso de Doctorado UPC-DDE

- NATIONAL INSTRUMENTS, Instrumentation Reference and Catalogue. Process Monitoring and Control. 1995
- Frau, Joan I. Sistemas de Adquisición de Datos Automática e Instrumentación, nº 242, pág 70, 1994
- Digitronics SIXNET. About SIXTRAK SYSTEM. 1995
- Aiza, Jordi. Software para el Control de la Calidad y el Mantenimiento. Automática e Instrumentación, nº 250, pág 640, 1995
- Campanera Radua, Jaume. Equipos y Software para control de calidad automatizado. Automática e Instrumentación, nº 205, pág. 167. 1990
- Jubera, J. A. Mantenimiento, Gestión e Informática. Mantenimiento y Electrónica Industrial, nº 14, pág. 13. 1992
- BURR-BROWN. The Handbook of Personal Computer Instrumentation. 1990
- MATSUSHITA AUTOMATION CONTROLS ESPAÑA S.A. PROGRAMMABLE CONTROLLER FP1. Technical, 1992

1.4. Herramientas de automatización: GRAFCET

El GRAFCET, cuyo nombre deriva de GRÁfico Funcional de Control de Etapas y Transiciones (en francés GRAphe Fonctionnel de Commande Etapes-Transitions), nació como resultado de los trabajos de la AFCET (Association Française pour la Cybernétique Economique et Technique), iniciados en la década de los setenta. Por ello veremos que en este campo gran parte de la producción científica proviene de Francia. El propósito inicial fue la puesta a punto de un método de descripción de procesos, independientemente de la tecnología de los mismos, mediante un gráfico funcional interpretable fácilmente por personas no especialistas en automatización. Tal gráfico funcional permite normalizar la forma de descripción del proceso para técnicos en distintas áreas, p.e. el ingeniero de organización o de producción, que ha de definir las necesidades del automatismo; el de diseño, que debe implementar el sistema de control y los accionamientos; y el técnico de mantenimiento, que debe cuidar de su funcionalismo o introducir modificaciones en fase de explotación.

La colaboración iniciada en 1977 entre AFCET Y ADEPA (Agence pour le Développement de la Productique Appliquée), dio lugar a una serie de útiles metodológicos, como el GEMMA (Guide d'Étude des Modes de Marche et Arrêt), para apoyar el GRAFCET como método no sólo descriptivo, sino como herramienta de diseño. En 1982 el trabajo fue retomado por un grupo de trabajo de AENOR (organismo encargado de la normalización en Francia), compuesto por miembros de UTE, CNOMO, UNM y de otros organismos relacionados con la industria, la automatización y la enseñanza, culminando con la publicación de la Norma NF C03-1904. Dicha norma fue también adoptada por IEC en 1988, con la denominación IEC-848 y título "Établissement des diagrammes fonctionnels pour systèmes de commande".

Actualmente, varios fabricantes de autómatas programables incorporan instrucciones de programación que permiten introducir directamente el grafo de GRAFCET. En su defecto, existe software capaz de compilar un grafo GRAFCET al lenguaje del autómata, permitiendo en ambos casos una gran flexibilidad y rapidez de diseño, con notables ventajas en las fases de verificación, explotación o posible modificación del automatismo. No obstante lo anterior no ha de confundirse el GRAFCET con un lenguaje de programación.

La combinación del gráfico funcional con los métodos del álgebra de Boole, se ha convertido en una potente herramienta de diseño de sistemas lógicos, que con unas reglas bastante simples, permite ir más allá de la simple descripción e interpretación gráfica de un proceso

1.4.1. Los principios del GRAFCET

Los principios en los que se basa la aplicación del GRAFCET, derivados de los que inspiraron su creación son los siguientes:

- Ha de definirse el funcionamiento del automatismo con total independencia de los componentes que han de intervenir en el mismo. Ello supone centrar el interés no tanto en la estructura física o en la tecnología a utilizar en el automatismo, sino en la "función" que ha de realizar.
- Un sistema automático se divide en dos partes: parte de control (PC) y parte operativa (PO). La parte de control comprende todo aquello que interviene en la automatización del proceso y la parte operativa incluye el resto del sistema. El conjunto PC+PO se relaciona con el medio exterior a través de un diálogo con el operador (diálogo hombre-máquina) y a través de comunicaciones con otros automatismos que operen en el mismo contexto.
- Cada fase de un proceso es la "operación" (que se denomina "etapa" en el lenguaje de GRAFCET), entendida como cualquier acción realizada por el automatismo. Hay que notar que en una primera aproximación puede dividirse la totalidad del proceso en unas pocas operaciones relativamente complejas, tales como cilindrar, cortar, taladrar, roscar, cambiar herramienta, etc., denominadas "macroetapas". Estas macroetapas pueden ser subdivididas a su vez en operaciones más elementales a medida que se progresa en el nivel de detalle. Así, p.e., una operación de roscado puede subdividirse en otras más elementales como: impulsar pieza, bloquear pieza, aproximación de herramienta, etc.
- Inicialmente ha de dividirse un proceso en macroetapas y éstas en etapas elementales, hasta conseguir que las acciones a realizar en cada una de ellas solamente dependan de relaciones combinacionales entre entradas y salidas. Cada una de estas etapas elementales llevará asociada una variable de estado.
- Ha de realizarse un gráfico de evolución que defina la sucesión de operaciones (secuencia de etapas) y las condiciones lógicas para pasar de una a otra (denominadas condiciones de transición en el lenguaje de GRAFCET). Como resultado de esta fase se obtienen las ecuaciones lógicas de las variables de estado y, por tanto, queda resuelta la parte secuencial del automatismo.
- Para cada operación elemental (etapa), se han de establecer las relaciones lógicas entre entradas y salidas, utilizando eventualmente otras variables internas combinacionales.
- Finalmente, implementar el sistema utilizando tantos biestables como variables de estado y cableando o programando las relaciones lógicas obtenidas.

Es importante resaltar que el GRAFCET no sólo es útil como herramienta de diseño, sino también en las fases de especificación y posteriormente en la fase de explotación y mantenimiento, y que el método está basado en la observación del número de estados que debe memorizar el sistema, para poder fijar su comportamiento posterior, partiendo de cualquier estado inicial, para identificar las etapas y, en consecuencia, las variables de estado. El número de estados distintos en un proceso no puede ser infinito, sino que se repiten de forma más o menos cíclica una serie de estados equivalentes y, por tanto, el número de etapas es finito; de lo contrario, nos encontraríamos ante un sistema de comportamiento aleatorio.

Aparece aquí el concepto de estados equivalentes, que se definen de la siguiente forma: Dos estados son equivalentes si la evolución posterior del sistema a partir de

ellos y para cualquier combinación de entradas es la misma. En el GRAFCET los estados equivalentes se asocian a una única etapa y en el modelo algebraico quedarán representados por una misma variable de estado.

Al contrario de lo que ocurre con otros métodos, el método basado en GRAFCET no pretende minimizar el número de variables de estado, por lo que puede no resultar óptimo desde el punto de vista de minimizar el hardware. Sin embargo, el coste y volumen de un sistema dependen cada vez menos del número de variables empleadas, sobre todo si se emplean autómatas programables y, en cambio, adquieren cada vez más importancia otros aspectos como el propio coste de diseño, tiempo de desarrollo de software, fiabilidad y facilidad de test y mantenimiento, y aspectos que permite optimizar el método propuesto.

1.4.2. Las reglas de evolución

Según lo anterior, el GRAFCET es un modelo para la representación gráfica del funcionamiento de un sistema automático. Tal modelo está definido en base a los siguientes elementos y reglas de evolución:

Elementos
Constituyen los símbolos con los que se dibuja el gráfico funcional. Los símbolos básicos son:
- Las etapas, que simbolizan cada uno de los estados del sistema. Cada etapa debe corresponder a una situación tal que las salidas dependan solamente de las entradas o, dicho de otro modo, la relación de entradas y salidas dentro de una etapa es puramente combinacional. El símbolo que se emplea para representar una etapa es un cuadrado con un número o símbolo en su interior que la identifica y eventualmente una etiqueta.
Las etapas iniciales son aquellas en las que se posiciona el sistema al iniciarse el proceso por primera vez y se simbolizan por un cuadrado con doble línea.
- Las líneas de evolución, que son las que unen entre sí las etapas que representan actividades consecutivas. Tales líneas se suponen siempre orientadas de arriba hacia abajo, a menos que se represente una flecha en sentido contrario.
- Las transiciones, que simbolizan las condiciones lógicas necesarias para que finalice la actividad de una etapa y se inicie la de la etapa o etapas inmediatamente siguientes. Estas condiciones lógicas se obtendrán por una combinación de variables denominadas receptividades. Gráficamente se simbolizan las transiciones por un trazo perpendicular a las líneas de evolución.
- Los reenvíos son símbolos en forma de flecha indicando la procedencia o destino de las líneas de evolución. Las flechas de reenvío hacen posible fraccionar un gráfico demasiado extenso o evitar líneas de evolución con excesivos cruces.
- El cruce de dos líneas de evolución no supone que estén unidas. Las reglas para cruces y bifurcaciones se contemplan mas adelante en las estructuras del GRAFCET.
- Al recorrer el gráfico de evolución, por cualquier camino posible, siempre deben alternarse una etapa y una transición.
La regla básica de sintaxis del GRAFCET es que entre dos etapas debe existir una y sólo una condición de transición, entendiendo que dicha condición puede venir expresada por

una función lógica combinacional todo lo compleja que sea necesario, siempre que su resultado sea un bit (1=condición verdadera, 0=condición falsa).
- Ha de tenerse presente que el gráfico funcional simboliza en forma estática un conjunto de situaciones posibles. No obstante, es posible representar la situación dinámica en un instante dado, indicando qué etapa o etapas están activas y cuáles están inactivas. El simbolismo utilizado para ello consiste en marcar con un punto las etapas activas.
- Finalmente, ha de tenerse en cuenta que los números de las etapas nada tienen que ver con su orden de ejecución, solamente tienen carácter de identificación (puede ser una etiqueta en vez de un número). Por tanto, se pueden numerar las etapas de la forma que se desee, sin que ello tenga ninguna repercusión desde el punto de vista funcional.

Mensajes
Son mensajes en forma de textos, símbolos o ecuaciones lógicas asociados a las etapas o a las transiciones y que indican la actividad desarrollada o las relaciones entre variables del sistema que han de cumplirse. Existen dos tipos de mensajes:
- Mensajes de acción, que van asociados a cada etapa, e indican que actividad se ha de desarrollar en dicha etapa cuando esté activa (ver reglas de evolución). Tales mensajes pueden ser en forma de texto o en forma de ecuaciones lógicas indicando la relación salidas-entradas.
- Mensajes de receptividad asociados en este caso a cada transición. Tales mensajes indican las condiciones lógicas necesarias y suficientes para el paso de cada etapa a la siguiente o siguientes.

Reglas
Las reglas de evolución son las que permiten definir e interpretar de forma unívoca el comportamiento dinámico del sistema. Unas hacen referencia a las etapas y otras a las transiciones, por lo que algunas pueden resultar redundantes entre sí. A continuación se expone una lista de las fundamentales:
- Cada etapa lleva asociada una variable de estado X1 de tipo bit.
- Se distinguen dos estados posibles de una etapa: activa o inactiva. Se dice que una etapa está activa si su variable de estado vale 1 e inactiva si vale 0.
- Se denomina arranque en frío a la inicialización de un proceso automático sin guardar memoria de situaciones anteriores. La orden de arranque en frío puede darla un operador humano o proceder de un sistema automático jerárquicamente superior.
Recibida la orden de un arranque en frío se activan todas las etapas iniciales y se desactivan todas las demás.
- Se denomina arranque en caliente a la reinicialización de un automatismo cuando éste guarde memoria de situaciones anteriores. Tal situación suele corresponder a un rearranque sin pérdida del contexto anterior, es decir, manteniendo memorizadas las variables de estado del proceso.
En caso de un arranque en caliente pueden activarse las etapas iniciales o mantener el estado anterior al arranque en caliente (contexto). Esta decisión es generalmente tomada por una parte específica del automatismo destinado a ejecutar lo que se llama una tarea previa.
- En la evolución normal de un proceso, una etapa no inicial se activará cuando esté activada la etapa anterior y se cumplan las condiciones de transición entre ambas.

- Cualquier etapa se desactivará cuando se cumplan las condiciones de transición a la siguiente o siguientes y dicha transición se haya efectuado. En el gráfico de las figuras, si se cumple la condición de transición T1/2, se activará la etapa 2 y se desactivará la etapa 1.
- Las cuatro situaciones posibles en que puede hallarse una transición son:

  No validada: La etapa o etapas inmediatamente anteriores o siguientes no están activas.

  Validada: La etapa o etapas inmediatamente anteriores si que están activas, pero no se cumple la condición lógica de transición.

  Franqueable: La etapa o etapas inmediatamente anteriores están activas y además si se cumple la condición lógica de transición. Se trata únicamente de una situación transitoria, ya que dicha transición será automáticamente franqueada, según se ve más adelante.

  Franqueada: Se ha activado la etapa o etapas inmediatamente siguientes y se han desactivado la etapa o etapas inmediatamente anteriores.

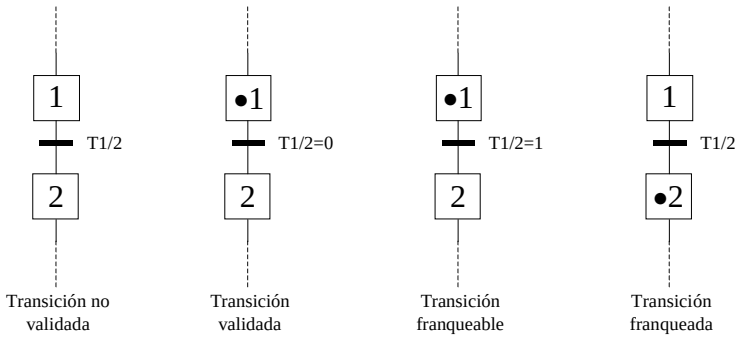


Figura 1.1: Situaciones posibles para una transición

- Sólo es posible franquear una transición si ésta está previamente validada.
- Cualquier transición franqueable será inmediatamente franqueada.
- Cuando existan varias transiciones franqueables simultáneamente, todas ellas serán franqueadas simultáneamente.
- Cuando se franquea una transición, automáticamente se desactivan todas las etapas inmediatamente anteriores.
- Durante el funcionamiento de un automatismo, si una etapa debe ser simultáneamente activada y desactivada, dicha etapa permanecerá activada. Esta regla no es sino un convencionalismo para resolver casos de indeterminación, pero es difícil de llevarse a la práctica dado que, en automatismos programables, p.e. la respuesta de un SET y un RESET simultáneos depende del orden de programación, pero en automatismos cableados puede depender de una “carrera crítica” en la que ya intervienen los tiempos de respuesta de los componentes. Por tanto, es aconsejable evitar que una etapa pueda ser activada y desactivada simultaneamente. Por todo lo anterior y como se verá enseguida, han de seguirse ciertas reglas “de coherencia” no explicitadas por el GRAFCET.
- Los gráficos de evolución de cualquier proceso, expresado en GRAFCET, deben ser siempre cerrados, sin dejar ningún camino abierto. En efecto, un camino abierto

supondría que el proceso no puede continuar. No obstante pueden darse situaciones en las que la salida sea inicializar el proceso mediante alguna señal externa.

Principios de implementación

Además de las reglas de evolución ya vistas, hay otra serie de reglas relativas a la forma de expresar el diagrama funcional y a su forma de interpretarlas que se tratarán en el apartado de estructuras. Sin embargo, se contemplan a continuación unos cuantos principios que, aunque no específicos del GRAFCET, son genéricos para cualquier automatismo secuencial:

- Se denomina evento a cualquier situación en la que se produzca el cambio de al menos una de las variables que intervienen en el sistema. De este modo, un evento corresponde siempre a un flanco de subida o de bajada de una variable lógica.
- Dos eventos pueden estar entre sí correlacionados o no correlacionados. Se dice que están correlacionados cuando:
    - Están asociados a una misma variable lógica. Por ejemplo: el flanco de subida de una variable B y el flanco de bajada de su complementaria, B, están correlacionados.
    - Están asociados a dos variables lógicas que tengan una intersección común. Por ejemplo, las variables Z y W tales que Z = B + D y W = E * B, también están correlacionadas puesto que un flanco de subida de B puede provocar un flanco de subida simultáneo de Z y W.
- Se admite que formalmente dos eventos externos no correlacionados nunca pueden producirse simultáneamente. Es decir: Siempre habrá una pequeña diferencia de tiempo entre ambos que hará que no sean simultáneos.

Es de suma importancia el tener en cuenta que el modelo de GRAFCET impone la simultaneidad de los eventos producidos al franquearse una transición. Así, en las figuras 1.1 ha de suponerse que el flanco de subida T1/2, el flanco de bajada de X1 y el flanco de subida de X2, son eventos correlacionados y, por tanto, simultáneos. No obstante, debido a los retardos propios de los componentes o a la ejecución mediante un controlador programable con procesador único, es posible que tecnológicamente dicha simultaneidad no se produzca, Lo anterior puede condicionar a que la implementación práctica de un controlador resulte dependiente del tipo de componentes utilizados en la implementación del sistema o del orden de programación en el caso de autómatas.

De cualquier modo, en el apartado estructuras del GRAFCET, se trata de cómo evitar tales inconvenientes añadiendo ciertas condiciones lógicas adicionales que hagan que la evolución del sistema secuencial sea independiente de la simultaneidad de dos eventos.

1.4.3. Estructuras básicas

Existen tres tipos de estructuras básicas en GRAFCET, de las cuales se pueden derivar todas las demás. Tales estructuras básicas son:

- Secuencia lineal.
- Convergencia o divergencia en “O” (subprocesos alternativos).
- Convergencia y divergencia en “Y” (subprocesos simultáneos).

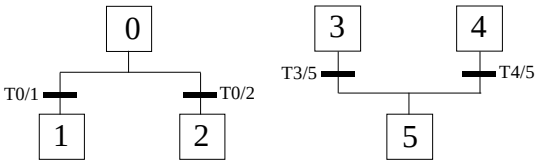


Figura 1.2: Divergencia y convergencia en O

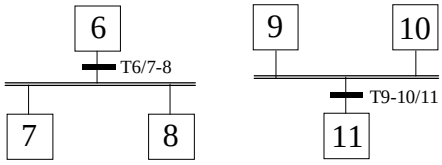


Figura 1.3: Divergencia y convergencia en Y

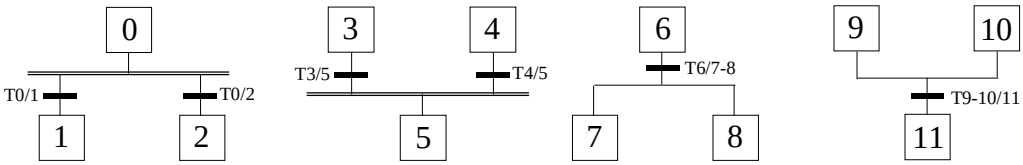


Figura 1.4: Dispociciones incorrectas (no permitidas) en GRAFCET

En lógica combinacional se demuestra que cualquier función lógica puede expresarse mediante combinación de las operaciones “Y”, “O” y “NO”. De acuerdo con ella, paralelamente puede decirse que los sistemas secuenciales, de cualquier complejidad, siempre pueden expresarse en GRAFCET mediante gráficos que sólo incluyan como estructuras básicas las tres citadas anteriormente.

En la práctica se comienza por describir los procesos mediante gráficos funcionales muy genéricos (macroetapas), con poco nivel de detalle, que casi siempre serán de tipo lineal, pero al ir avanzando en el nivel de detalle aparecen las bifurcaciones (convergencias y divergencias en O y en Y).

Seguidamente se contemplan las formas puras de cada una de las estructuras básicas mencionadas, si bien, en un proceso real tales estructuras aparecerán entremezcladas de tal forma que, en el interior de estructuras en “O” aparecerán tramos lineales u otras en “Y” o viceversa. Sin embargo, casi siempre se puede recuperar la estructura en “Y” o en “O” pura, haciendo uso del concepto ya expuesto de macroetapa.

Secuencia lineal

La estructura más simple posible es la secuencia lineal, consistiendo en una sucesión de etapas unidas consecutivamente por las líneas de evolución y condiciones de transición**.**
Las propiedades que cumple la estructura secuencia lineal son las siguientes:
- En un tramo de secuencia lineal, solamente una etapa debe estar activa en un instante determinado. En rigor, si bien las reglas del GRAFCET no impiden formalmente la posibilidad de que en una secuencia lineal pueda existir más de una etapa activa, tal situación suele denotar una incoherencia de diseño. En efecto, la implicación práctica de que se activen dos etapas simultáneamente supone que deben ejecutarse dos grupos de acciones simultáneamente y esto puede representarse de una forma más apropiada mediante bifurcaciones en “Y”, como podrá verse seguidamente.

De otra parte, si en una estructura lineal progresan varias etapas activas a la vez pueden "darse caza" y ello podría provocar condiciones contradictorias de que una etapa deba activarse y desactivarse a la vez.
- Cada etapa se activa cuando se encuentre activada la anterior y además se cumplan las condiciones de transición entre ambas.
- La activación de una etapa implica automáticamente la desactivación de la etapa anterior.
- Una secuencia lineal puede estar integrada en una estructura más compleja.
Es usual que una estructura lineal aparezca casi siempre a nivel de descripción genérica con macroetapas y también como parte de estructuras más complejas.

Divergencia y convergencia en "O"
La divergencia y convergencia en "O", a las que se denominan conjuntamente bifurcación en "O", forman una estructura integrada por los siguientes elementos:
- Una divergencia en "O", como comienzo de varios caminos o subprocesos alternativos posibles.
- Una serie de caminos alternativos, cada uno con una macroestructura lineal, aunque pueden contener otras estructuras más complejas.
- Una o más confluencias (convergencias) en "O" de tales caminos alternativos, de modo que la macroestructura ha de ser globalmente cerrada.
Tal tipo de estructura se prevé para representar procesos alternativos que han de ejecutarse dependiendo de ciertas condiciones lógicas. Por ejemplo en una embotelladora, si el llenado es correcto poner el tapón y meter en la caja, si no, retirar para su rellenado. Informáticamente hablando, la bifurcación en "O" corresponde a una estructura del tipo "IF...THEN...ELSE".

Dependiendo de cuáles sean las condiciones de transición que se cumplan a partir de la etapa previa a la bifurcación, se seguirá uno u otro camino o subproceso a partir de la misma.

Si bien no es necesario que los subprocesos que parten de una misma divergencia deban confluir en una misma convergencia, lo que si ha de ocurrir en todo proceso es que parta de una divergencia y termine en una convergencia en algún lugar del ciclo, para que se cumpla que el gráfico de fluencia visto globalmente sea cerrado.

Las propiedades básicas a cumplir por la estructura de bifurcación en "O" son las siguientes:
- Partiendo del punto de divergencia el proceso puede evolucionar por uno de los distintos caminos alternativos, cada uno de los cuales tiene su propia condición de transición.
- Las condiciones de transición de los distintos caminos de divergencia han de ser excluyentes entre sí (intersección nula), de forma tal que el proceso sólo podrá progresar en cada caso por uno de ellos.

En rigor, las reglas del GRAFCET no imponen esta restricción, pero su incumplimiento da lugar a una incoherencia. En efecto, si las condiciones no son excluyentes entre sí, existiría la posibilidad de poder iniciarse simultáneamente varios procesos en caso de

cumplirse dos o más condiciones de transición simultáneamente. Si esta situación es la deseada, debe resolverse utilizando una estructura de bifurcación en "Y", que se expondrá más adelante. En cambio, si la situación es accidental, ello pondría de manifiesto una falta de especificación ante tal contingencia, que deberá ser resuelta o bien imponiendo condiciones adicionales para evitar la simultaneidad o especificando claramente cuándo el proceso ha de ser exclusivo y cuándo simultáneo.

Abundando en lo anterior, en los automatismos reales, donde no puede garantizarse la simultaneidad de eventos debido a los tiempos de respuesta, el incumplimiento de tal restricción puede ocasionar respuestas aleatorias, debido a lo que se conoce como "carreras críticas". En resumen, a fin de evitar los problemas descritos, es aconsejable imponerse dicha restricción en las bifurcaciones en "O".
- A nivel de gráfico global, los distintos caminos iniciados como divergencia en "O" deben confluir en uno o más puntos de convergencia en "O". Es decir: la estructura debe ser globalmente cerrada y no pueden existir caminos abiertos, que denotarían situaciones sin posible salida.
También se excluye, como se verá enseguida, que los caminos de una divergencia en "O" puedan concurrir en una convergencia en "Y", puesto que ello provocaría un bloqueo del sistema en el punto de convergencia ante la imposibilidad de finalizar simultáneamente todos los caminos, habiendo iniciado uno sólo.

En la obtención del esquema lógico de la parte secuencial, las únicas etapas que merecen comentario son las que se encuentran inmediatamente antes o después de la divergencia y convergencia, ya que las demás son simplemente parte de una estructura lineal ya conocida.

Divergencia y convergencia en "Y"
La divergencia y convergencia en "Y", a las que se denominan conjuntamente bifurcación en "Y", forman una estructura integrada por los siguientes elementos:
- Una divergencia en "Y" en la que comienzan varios caminos o subprocesos que han de iniciarse simultáneamente cuando se cumpla una determinada condición de transición común.
- Una serie de caminos simultáneos, cada uno de ellos con una macroestructura lineal, aunque pueden contener otras estructuras más complejas.
- Una o más confluencias en "Y" de tales caminos, de modo que la macroestructura debe ser globalmente cerrada.

Esta estructura está prevista para representar procesos que se inician simultáneamente, se ejecutan de forma independiente con distintos tiempos, y condicionan la continuación del proceso a que hayan terminado todos ellos.

Al igual que se vio para las bifurcaciones en "O", no es necesario que los subprocesos simultáneos que parten de una misma divergencia deban concluir en la misma convergencia. Lo que también aquí es imprescindible es que el gráfico, visto globalmente, sea cerrado.

Las propiedades que cumplen las convergencias y divergencias en "Y" son las siguientes:
- Desde el punto de divergencia el proceso evoluciona por varios caminos a la vez, ejecutando varias tareas simultáneamente.
- Para el inicio de las tareas simultáneas la condición de transición es única e igual para todas ellas.
- En el gráfico global, los diferentes caminos iniciados como divergencia en "Y" han de confluir en uno o más puntos de convergencia en "Y". Es decir, la estructura debe ser globalmente cerrada y sin que puedan existir caminos abiertos, que denotarían situaciones sin posible salida. También se excluye que los caminos de una divergencia en "Y" puedan concurrir en una convergencia en "O". En rigor las reglas del GRAFCET no prohiben explícitamente esta situación, pero en caso de cerrar una divergencia en "Y" con una convergencia en "O" podrían activarse varias etapas consecutivas de una estructura lineal que estuviera a continuación, lo que no cumple con la primera regla del apartado Secuencia lineal.
- En una convergencia en "Y" ya se impone naturalmente una condición de transición: en efecto, todas las tareas que confluyan en ella han de haber terminado para que el proceso continúe. De modo que, a la hora de comprobar la regla del apartado *Las reglas de evolución*, puede considerarse a todos los efectos equivalente una convergencia "Y" a una transición. Ello no impide que puedan existir otras condiciones adicionales, aparte de la propia de convergencia.

Detalles dignos de comentario son:
- La etapa previa a una divergencia "Y" no ha de desactivarse hasta que estén activadas todas las etapas siguientes.
- La activación de cualquier etapa inmediatamente después de una divergencia depende de que esté activa la etapa inmediatamente anterior y de la condición de transición común.
- La activación de la etapa siguiente a una convergencia "Y" depende de que estén activas todas las etapas previas y en su caso de alguna condición adicional.

1.4.4. Macroetapas

Al aplicar las técnicas del GRAFCET a la representación de procesos complejos, se comienza por representar un diagrama con las tareas principales a ejecutar en el proceso, definiendo grandes bloques de acciones llamados macroetapas y sin desarrollar en principio los detalles del proceso.

El símbolo empleado para representar una macroetapa es un cuadrado dividido en tres partes (vertical u horizontalmente). En una de las partes puede colocarse un número, en otra la identificación de la macroetapa y en la tercera la etiqueta.

Por tanto las macroetapas representan "tareas" y equivalen a lo que en algunos lenguajes se definen abreviadamente como "macros". Desde un punto de vista formal, una macroetapa no es más que un conjunto de etapas agrupadas que se definen

posteriormente de forma individual, en lo que se denomina representación en detalle o expansión de la macroetapa.

En esencia, la finalidad de la macroetapa es la de permitir una aproximación progresiva y estructurada al proceso en todas las fases (diseño, explotación y mantenimiento) de un automatismo. Se puede partir de una definición en líneas generales (macroetapas) del proceso y posteriormente descomponer cada macroetapa en las acciones simples correspondientes.

Cuando se introducen macroetapas en un gráfico funcional, han de tenerse en cuenta las siguientes reglas básicas:
- La expansión de una macro debe tener una única etapa inicial y una única etapa final, sin que ello condicione a la macro a una estructura lineal. En efecto, dentro de la macro pueden existir estructuras de cualquier complejidad e incluso otras macros “anidadas”.
- El franqueo de la transición inmediatamente anterior a la macro, activa la etapa de entrada de la misma y, la activación de la etapa de salida de la macro “valida” la transición siguiente a la misma.

Por motivos de claridad o de estructuración se utilizan a veces macroetapas anidadas. Es decir, la expansión de una macroetapa puede, a su vez, utilizar otras macroetapas, fragmentando así el problema global en “tareas” que normalmente se hacen corresponder a partes del proceso tecnológicamente completas (corte, cilindrado, taladrado, roscado, cambio de herramienta, traslado de piezas, pintura, tratamiento térmico, etc.)

Es muy importante observar que en la definición de AFCET se excluye explícitamente que una misma expansión pueda ser llamada desde dos macroetapas distintas del gráfico funcional, tal como se ha representado p.e. en la Figura 1.6. En otras palabras: excluye la utilización del concepto de macroetapa como sinónimo de “subrutinas”.

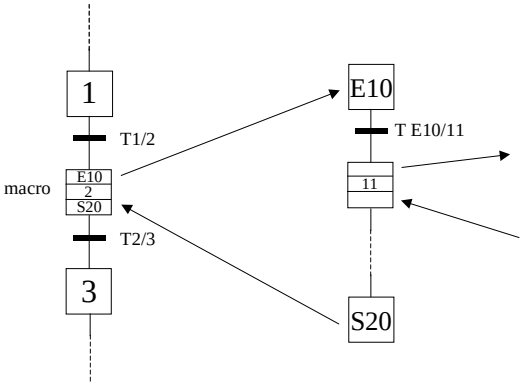


Figura 1.5: Macroetapa

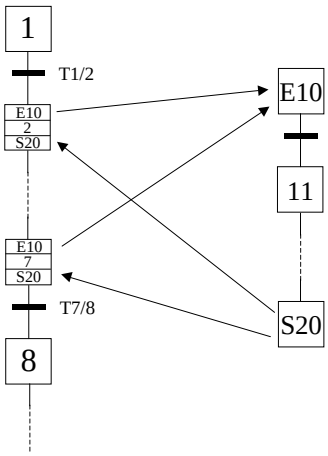


Figura 1.6: Macroetapa empleada como subrutina

Tal restricción se introduce a fin de evitar conflictos de acceso en el caso de que la misma expansión fuese llamada por dos macroetapas activas simultáneamente. No

obstante, si se evita explícitamente dicha posibilidad puede pasarse por alto la restricción y utilizar la misma expansión para desarrollar varias macroetapas distintas. En automatismos programables, la utilidad de las macroetapas se potencia notablemente cuando no se impone la restricción antedicha. De esta forma, las macroetapas pueden definirse como auténticas "subrutinas" o "procedimientos" que pueden ser llamados desde cualquier punto del programa, con la única condición de que no sean llamadas mientras se están ejecutando. Lo anterior exige ciertas precauciones en su inicialización y utilización, pero permite en cambio una programación más estructurada de las tareas de un proceso y acorta la longitud del programa. Hay ya autómatas que disponen de lenguajes de programación estructurados, capaces de interpretar macroetapas con carácter de subrutina, incluso con varios niveles de anidamiento.

1.4.5. Programación de alto nivel

Es interesante establecer una comparación entre los gráficos de flujo tradicionales utilizados en informática y el GRAFCET poniendo de manifiesto sus diferencias fundamentales:

- Los gráficos de flujo son una sucesión de tareas ejecutadas secuencialmente a la velocidad del procesador, en tanto que un GRAFCET es una sucesión de tareas, eventualmente controladas por un procesador, pero ejecutadas al ritmo impuesta por el proceso. Es decir que, en general, durante el tiempo de actividad de cada etapa GRAFCET el procesador ejecuta múltiples barridos del gráfico funcional completo.
- Los gráficos de flujo representan, en general, procesos monotarea, en tanto que en GRAFCET es perfectamente lícito representar tareas simultáneas (divergencia y convergencia en Y). Es decir, no existe una estructura en gráficos de flujo para representar tareas simultáneas.

Haciendo hincapié en lo anterior, es incluso posible que un mismo procesador ejecute a la vez varios gráficos funcionales, relacionados entre sí o completamente disjuntos.

- De lo anterior, un bucle en un diagrama de flujo implica que sólo se está ejecutando la parte de programa interior al bucle, hasta que se cumpla la condición que permita salir de él, en tanto que en GRAFCET se explora la totalidad del programa, independientemente de que se cumplan o no las condiciones de transición.
- Un GRAFCET debe separar las acciones combinacionales de las secuenciales, en tanto que en un diagrama de flujo no hay tal distinción.
- De un diagrama de flujo no es posible deducir el programa de forma unívoca, por caracer de información suficiente, en tanto que el GRAFCET permite una "compilación" directa a programa máquina.

No obstante las anteriores diferencias, siempre a tener en cuenta, es interesante desarrollar en GRAFCET algunas de las estructuras usuales en programación estructurada, ya que como se ha visto anteriormente, el GRAFCET permite representar cualquier estructura lógica secuencial partiendo de las tres estructuras básicas ya comentadas. En efecto, pueden desarrollarse diversas estructuras de saltos y bucles basándose casi siempre en la estructura simple de divergencia y convergencia en "O". A modo de ejemplo, pueden obtenerse saltos condicionales, ya sea hacia etapas posteriores o hacia etapas anteriores, o bucles con estructuras típicas como "WHILE...DO", "REPEAT UNTIL", "FOR NEXT", etc., como puede verse en las figuras1.7.

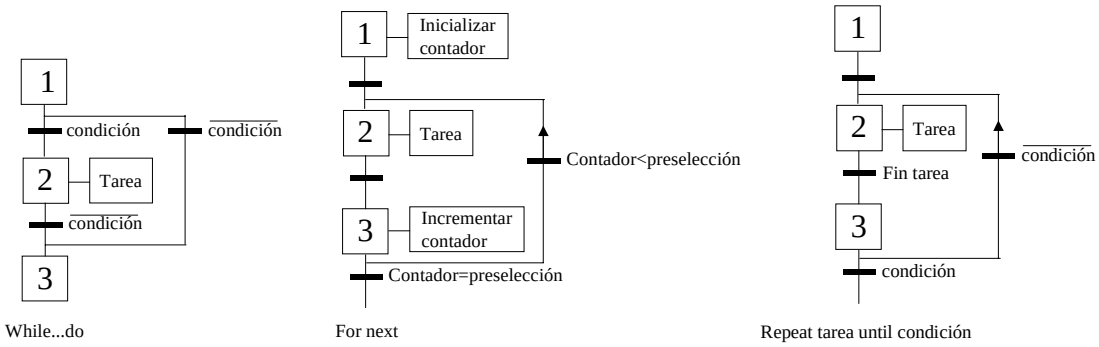


Figura 1.7: Representación mediante GRAFCET de estructuras de programación de lenguajes de alto nivel

La estructura “IF...THEN...ELSE...ELSE...” o las de tipo “CASE” son equivalentes a una bifurcación en “O”, con una rama para el “IF” y una para cada “ELSE”.

También a modo de otra estructura ha de añadirse el concepto de subrutina o procedimiento, que en GRAFCET se obtiene con el concepto extendido, ya visto, de macroetapa y representación en detalle. Ya se vió que, si bien la norma del GRAFCET no admite utilizar el concepto de macroetapa como sinónimo de subrutina, en cambio no impide que existan varias etapas iniciales ni que dos gráficos funcionales puedan tener condiciones de transición cruzadas.

1.4.6. Inicialización del sistema

Al plantear el problema de automatización de un proceso, uno de los problemas que ha de resolverse es el de inicializar dicho proceso en el momento de arranque inicial (arranque en frío) y del mismo modo el de establecer cómo debe rearrancar en condiciones anómalas como pueda ser la pérdida de tensión del sistema de control con salvaguarda de datos (arranque en caliente). Tal problema se conoce por el nombre genérico de “tratamiento preliminar”. A fin de establecer el comportamiento del sistema de control en estos casos singulares se habilitan ciertas variables internas capaces de detectar el estado de servicio del propio sistema, que son conocidas como variables de sistema. Por tanto una estructura completa de un GRAFCET estará integrada por tres grandes bloques: tratamiento preliminar, tratamiento secuencial y tratamiento combinacional o posterior.

En cuanto al tratamiento preliminar ha de decirse que se ejecuta solamente en el primer ciclo después de un arranque, distinguiendo entre arranque en frío o arranque en caliente y permite reiniciar el proceso partiendo de sus etapas iniciales (caso más frecuente en el arranque en frío) o bien posicionarlo en otras etapas intermedias a fin de proseguir un proceso interrumpido por condiciones anómalas (caso de arranque en caliente, con memoria de datos que permita continuar el proceso). En este último caso tanto las

variables de sistema como los datos memorizados suelen emplearse para decidir el preposicionamiento del sistema.

El siguiente bloque de tratamiento secuencial se ejecuta, en general, cíclicamente en condiciones normales de funcionamiento y recorre las distintas fases de evolución que ha de seguir el proceso.

El bloque final de tratamiento combinacional queda también incluido dentro del ciclo normal de ejecución y contiene todas las acciones a ejecutar en cada una de las etapas del proceso. La posibilidad de condiciones anómalas de funcionamiento es otro aspecto a considerar. En efecto, ante tales situaciones se desea generalmente que el sistema de control reaccione de distintas maneras, en función de la gravedad de la anomalía. Para incluir estas condiciones en el GRAFCET, cada etapa debería tener al menos dos salidas (divergencia en O), una de ellas para la evolución normal y otra para responder a condiciones de alarma (aviso, parada, etc.). Sin embargo, si se quisiera representar esto resultaría un gráfico funcional excesivamente complicado, optando por tanto por representar en el gráfico funcional sólo la evolución normal del proceso y dando por supuesto que dicho funcionamiento normal puede ser “interrumpido” en cualquier etapa por una condición genérica de alarma.

1.4.7. Ejemplo

A modo de ejemplo se propone un simple ascensor de tres plantas. Antes de nada quisiera indicar que la forma en la que se va a resolver no es la más adecuada para modelizar un ascensor en general. Para un número de plantas mayor que el de este problema, o cuando existe alguna dificultad adicional (memorias, comportamiento inteligente, etc.) se debe realizar un grafcet de gestión de las maniobras y varios sub-grafcet o macroetapas que las vayan realizando. La razón de emplear este modelo como ejemplo se debe a la facilidad de descripción por ser un elemento muy popular: se necesita mucho menos esfuerzo (y sobre todo espacio) para definir cómo es el sistema a tratar.

El ascensor es el que se puede ver en la figura 1.8. Consta de tres plantas a las que se puede llamar desde el interior de la cabina o desde las propias plantas. Se ha supuesto que se trata del mismo pulsador (el de la cabina y el de la planta), es decir, que la señal que se emplea es un función lógica O de ambas. Las señales que la automatización (el GRAFCET) proporciona al sistema a controlar (el ascensor) son las órdenes de subir o bajar (las salidas del GRAFCET). Cuando no se manda ni subir ni bajar el ascensor queda bloqueado en la planta. Se dispone como entradas a la automatización de sensores de posición de las plantas (D1, D2 y D3) además de los ya mencionados correspondientes a los pulsadores (P1, P2, P3). Cuando el ascensor se detiene en una planta debe permanecer un cierto tiempo de espera hasta atender a las llamadas que se le realicen. El esquema está indicado en la figura 1.8.

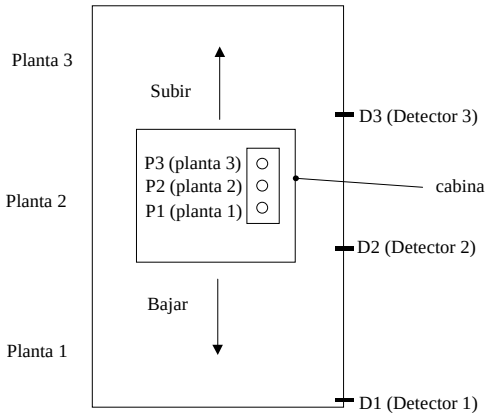


Figura 1.8: Esquema de ascensor simple

En la figura 1.9 se proporciona la solución al problema, en la que se ha intentado que aparezcan el mayor número de elementos posibles de los previamente mencionados. Para reinicializar el sistema se le manda bajar hasta la planta baja, en cuyo caso ya se conoce el estado del sistema; simultáneamente se realiza el chequeo del sistema. Esta primera etapa se realiza mediante una divergencia-convergencia en Y. Cuando ambas operaciones (chequeo y llegada al estado inicial) han concluido comienza el funcionamiento del ascensor atendiendo a las llamadas. Dicho funcionamiento consiste en una serie de divergencias-convergencias en O, en donde las convergencias estan implícitas en los saltos que se indican mediante los círculos; dichos saltos son equivalentes a la unión en la etapa con el resto de transiciones de entrada mediante una convergencia en O.

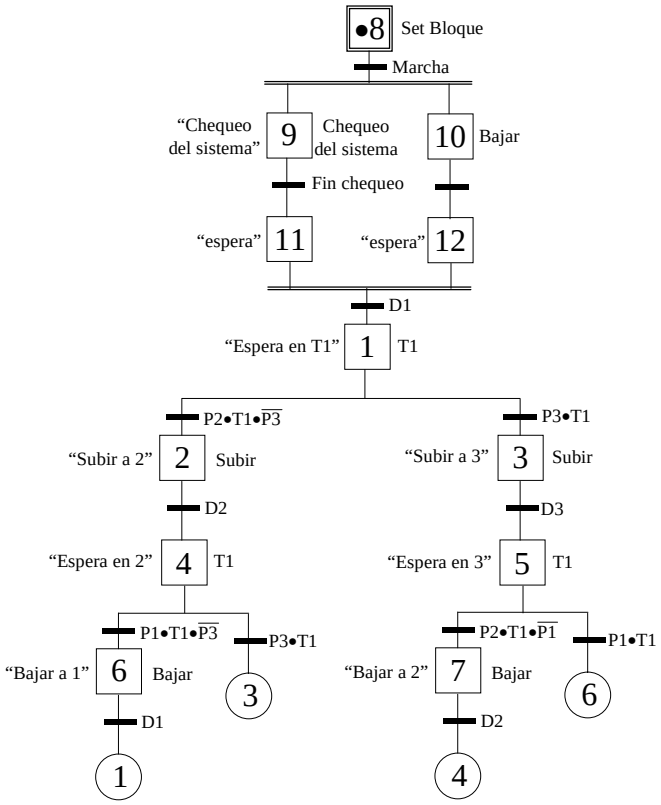


Figura 1.9: GRAFCET correspondiente a la automatización del ascensor

1.4.8. Referencias

a) Páginas WEB

http://www.lurpa.ens-cachan.fr/grafcet.html

Presentación de Grafcet
http://www.lurpa.ens-cachan.fr/cgi-bin/grafcet/redirection?http://www.eerie.fr/~chapurla/enseignements.html

Soporte de curso
http://www.lurpa.ens-cachan.fr/cgi-bin/grafcet/redirection?http://www.lisi.ensma.fr/~manu/GRAFCET0.HTM

Automatismos y Grafcet.
http://www.lurpa.ens-cachan.fr/cgi-bin/grafcet/redirection?http://www-ipst.u-strasbg.fr/pat/autom/index.htm

El modelo Grafcet
http://www.lurpa.ens-cachan.fr/cgi-bin/grafcet/redirection?http://www.gsip.cran.u-nancy.fr/~herve/grafcet/

El Grafcet
http://www.lurpa.ens-cachan.fr/cgi-bin/grafcet/redirection?http://perso.wanadoo.fr/papanicola/s_i/Auto/logique/Grafcet/grafcet.htm

Los Automatismos
http://www.lurpa.ens-cachan.fr/cgi-bin/grafcet/redirection?http://artemmis.univ-mrs.fr/colleges/pcl2/grafcet.htm

Lenguage gráfico de funciones secuenciales
http://www.lurpa.ens-cachan.fr/cgi-bin/grafcet/redirection?http://rolf.ece.curtin.edu.au/~clive/plc/6SFC.HTM

Curso de Automática Grafcet
http://www.lurpa.ens-cachan.fr/cgi-bin/grafcet/redirection?http://www.lab.ens2m.fr/cours%20automatique/INTRO.HTM#intro automatique

b) Tesis Doctorales

Synthese de la commande des systemes a evenements discrets par GRAFCET
C. NDJAB HAGBEBELL, Tesis de la Universidad de Reims Champagne Ardenne, 1999

Un langage de specification pour la conception structuree de la commande des systemes a evenements discrets
J.J. DUMERY, Tesis de l'Ecole Centrale de París, 1999

Elaboration d'un langage de specification comportementale de systemes automatises
L. GUILLEMAUD, Tesis Doctoral de la Universidad de Rennes I, 1998

Ingenierie inverse en genie automatique: une methodologie d'elaboration d'une representation structuree d'une commande preexistante
F. LOUNI, Tesis Doctoral de l'Ecole Nationale Supérieure d'Arts et Métiers, 1998

De la verification de cahiers des charges de systemes a evenements discrets a la validation des specifications decrites en grafcet
S. LAMPERIERE-COUFFIN, Tesis Doctoral de l'Ecole Normale Supérieure de Cachan, 1998

Modelisation du grafcet temporise et verification de proprietes temporelles
D. L'HER, Tesis Doctoral de la Universidad de Rennes I, 1997

Control and supervision of event-driven systems
H. YAZDI, Tesis Doctoral de la Universidad Técnica de Dinamarca (DTU), 1997

Du TTM/RTTL pour la validation des systemes commandes par grafcet
P. DE LOOR, Tesis Doctoral de la Universidad de Reims Champagne Ardenne, 1996

Le modele GRAFCET: reflexion et integration dans une plate-forme multiformalisme synchrone
D. GAFFE, Tesis Doctoral de la Universidad de Nice-Sophia Antipolis, 1996

Prototype implementation of the PLC standard IEC 1131-3
S. JOHANSSON, M. OHMAN, Tesis ISSN 0280-5316 ISRN LUFTFD2/TFRT—5547—SE, Lund Institue of Technology, Lund, Suiza, 1995

D'une theorie des systemes sequentiels a la notion de codeur: validation globale du GRAFCET
M. GUILLAUME, Tesis Doctoral de la Universidad de Rennes I, 1995

Analyse de GRAFCETS par generation logique de l'automate equivalent
J.M. ROUSSEL, Tesis Doctoral de l'Ecole Normale Supérieure de Cachan, 1994

ACSY-R: un modele pour l'analyse, la specification et la simulation de la commande de systemes discrets complexes repartis
V. CHAPURLAT, Tesis Doctoral de la Universidad Motpellier II, 1994

Contribution a la formalisation des modeles et methodes de conception des systemes de production
J.J. LESAGE, Suficiencia Investigadora de la Universidad de Nancy, especialidad automática y producción automática, 1994

Apport de la methodologie synchrone pour la definition et l'utilisation du langage GRAFCET
P. LE PARC, Tesis Doctoral de la Universidad de Rennes I, 1994

Utilisation du GRAFCET et des reseaux bayesiens pour le diagnostic de fonctionnement : application au systeme cryogenique de l'experience tore supra
O. BADIE, Tesis Doctoral en informática de la Universidad de París 6, 1994

Performances temporelles d'automates programmables et conception d'un coprocesseur d'acceleration
G. RUDELOU, Tesis Doctoral de la Universidad de Montpellier 2, 1993

Interaction produit-procede. extension de la modelisation de la commande avec suivi automatique du produit
F. PEREYROL, Tesis Doctoral de la Universidad de Montpellier 2, 1993

Contribution a la specification et a la mise en oeuvre de la commande temps-reel de cellules flexibles d'assemblage
M. SOUAM, Tesis Doctoral de la Universidad de Lille I, 1993

Interaktiv grafisk editering och simulering av GRAFCET (interactive graphical editing and simulation of GRAFCET)
S. JONASSON, Tesis TFRT-5456, Lund Institute of Technology, Lund, Suiza, 1992

Modelisation de protocoles: application a la couche liaison de donnees de f.i.p.
G. GARNIER, Tesis Doctoral de la Universidad de Nancy 1, 1992

Contribution a la didactique des disciplines technologiques: acquisition et utilisation d'un langage d'automatisme
J. GINESTIE, Tesis Doctoral, 1992

Contribution a la conception des systemes de controle reparti
D. FOURMAUX, Tesis Doctoral de LILLE 1, 1991

Modele structure de specification, de conception et de mise au point de systemes a evenements discrets
T. GIACCONE, Tesis Doctoral de Montpellier 2, 1991

Conception d'un systeme de gestion d'interface utilisateur dans l'environnement cim
J. WANG, Tesis Doctoral de Valenciennes, 1991

Une contribution au genie automatique: le prototypage des machines et systemes automatises de production
H. PANETTO, Tesis Doctoral en producción automática de la Universidad de Nancy 1, 1991

Contribution a l'application du dialogue homme-machine. applications industrielles a un reseau d'automates
V. CARRE, Tesis Doctoral de Reims, 1990

Traitement de l'information et efficacite economique des automatismes : la place du GRAFCET
P. PERNOT, Tesis Doctoral de Grenoble 2, 1990

Automatisation d'un reacteur pilote modulaire polyvalent sous pression: application a la carboxylation du resorcinol
J. DALIL-ESSAKALI, Tesis Doctoral de París 6, 1990

Modelisation et simulation de defaillances dans les systemes de production discrets
C. PERSEGOL, Tesis Doctoral en componentes, señales y sistemas de la Universidad de Montpellier 2, 1990

c) Normativa

- Norme française nf c 03-190 + r1 : diagramme fonctionnel "grafcet" pour la description des systemes logiques de commande
  Union Technique de l'Electricité
  Paris : UTE Editions, 1995

- Project of amendment to IEC 848 : Preparation of function charts for control systems
  International Electronical Commission
  Geneve : IEC Editions, 1994

- Projet de modification a la CEI 848 : Etablissement de diagrammes fonctionnels pour systèmes de commande
  Commission Electronique Internationale
  Geneve : IEC Editions, 1994

- Fascicule de documentation UTE C 03-191 : Etablissement des diagrammes fonctionnels pour systèmes de commande
  Union Technique de l'Electricité
  Diagramme fonctionnel GRAFCET, extension des concepts de base

- Fascicule de documentation UTE C 03-190 : Etablissement des diagrammes fonctionnels pour systèmes de commande, Diagramme fonctionnel GRAFCET
  Union Technique de l'Electricité
  Paris : UTE Editions, 1990

d) Libros

- 7 Facettes du GRAFCET - approches pratiques de la conception a l'exploitation
  A. BODART, V. CARRE-MENETRIER P. DE LOOR, J.B. DELUCHE, J. DUPONT, D. GENDREAU, J. HANCQ, A. KRIL, J. NIDO
  Toulouse : CEPADUES Editions, 2000 - 168p.
- LE GEMMA : Modes de marches et d'arrêts, GRAFCET de coordination de tâches, Conception des systèmes automatisés de production sûrs
  S. MORENO, E. PEULOT
  Paris : Editions Casteilla, 1997 - 256 p.
- LE GRAFCET : Conception-Implantation dans les Automates Programmables Industriels
  S. MORENO, E. PEULOT
  Paris : Editions Casteilla, 1996 - 252 p.

- GRAFCET step by step
  P. BARACOS
  FAMIC INC, 1992
- Petri nets and GRAFCET: tools for modelling discrete event systems
  R. DAVID, H. ALLA
  New York : PRENTICE HALL Editions, 1992
- Le GRAFCET
  N. BOUTEILLE, P. BRARD, G. COLOMBARI, N. COTAINA, D. RICHET
  Toulouse : CEPADUES Editions, 1992 - 144p.
- Du GRAFCET aux reseaux de petri
  R. DAVID, H. ALLA
  Paris : HERMES Editions, 2e ed. 1992 - 423 p.
- Le GRAFCET et le gemma sur les automates programmables
  E. CARBONNEAU
  Sept-Iles, Québec : ECS Editions, 1991

e) Ponencias

- A case tool for the synthesis of optimal control implementation of GRAFCET
  V. CARRE-MENETRIER, C. NDJAB HAGBEBELL, F. GELLOT ET J. ZAYTOON
  1999 IEEE International Conference on Systems, Man and Cybernetics, Tokio-Japón, 1999, pp 796-801
- GRAFCET-SFC (sequential function chart) and its extensions as a formal base for modeling the behavior of complex industrial systems
  J.P. FRACHET, M. BERTRAND
  SCI '99/ISAS '99 Orlando, Florida USA
- Une methode d'analyse par objectifs pour la conception de la commande des sap
  H. GUEGUEN , M.A. LEFEBVRE, N. BOUTEILLE
  Actes du 2ième Congrès Modélisation des Systèmes Réactifs (MSR'99), pp 173-182, Cachan, 1999
- SAGITAL : un environnement d'aide a la conception de GRAFCETS base sur des meta-modeles
  J.M. FAURE, F. COUFFIN, S. LAMPERIERE-COUFFIN
  Actes du 2ième Congrès Modélisation des Systèmes Réactifs (MSR'99), pp 183-192, Cachan, 1999
- Obtention des situations stables par calcul anticipe des transitions franchissables dans un GRAFCET
  P. DELANCHY
  Actes du 2ième Congrès Modélisation des Systèmes Réactifs (MSR'99), pp 193-202, Cachan, 1999
- Semantique fonctionnelle et stabilite du GRAFCET sous l'hypothese du synchronisme fort
  J.F. HERY, J.C. LALEUF
  Actes du 2ième Congrès Modélisation des Systèmes Réactifs (MSR'99), pp 263-274, Cachan, 1999
- Synthese d'une implantation optimale de la commande a partir du GRAFCET
  C. NDJAB, J. ZAYTOON
  Actes du 2ième Congrès Modélisation des Systèmes Réactifs (MSR'99), pp 193-202, Cachan, 1999
- Un environnement de developpement pour la conception de systemes automatises de production
  P. LE PARC, R. QUERREC, P. CHEVAILLIER, J. TISSEAU, L. MARCE
  Actes du 2ième Congrès Modélisation des Systèmes Réactifs (MSR'99), pp 407-416, Cachan, 1999
- Proving sequential function chart programs using automata
  D. L'HER, J.L. SCHARBARG, P. LE PARC, L. MARCE
  International Workshop on Implementing Automata WIA'98, Rouen, 1998

- Specification and verification of the korso production cell
  D. L'HER, J.L. SCHARBARG, P. LE PARC, J. VAREILLE, L. MARCE
  9th symposium on INformation COntrol in Manufacturing, INCOM'98 - Advances in Industrial Engineering. Nancy - Metz, Francia. 1998
- The specification of a robot control system with GRAFCET
  G. FRENSEL, P. M. BRUIJN
  UKACC CONTROL '98, IEE International Conference on Control'98, University of Wales, Swansea, UK, 1998
- Petri Net analysis of batch recipes structured with grafchart
  C. JOHNSSON, K.E. ARZEN
  FOCAPO'98, Snowbird, USA, 1998
- Fault detection and isolation on hybrid systems by SFC
  R. FERREIRO GARCIA, X.C. PARDO MARTINEZ, J. VIDAL PAZ
  9th symposium on INformation COntrol in Manufacturing, INCOM'98 - Advances in Industrial Engineering, volume III pp. 569-574. Nancy - Metz, Francia. 1998
- GRAFCET: a north american perspective
  P. BARACOS
  9th symposium on INformation COntrol in Manufacturing, INCOM'98 - Advances in Industrial Engineering, volume II pp. 41-46. Nancy - Metz, Francia. 1998
- From GRAFCET to hybrid automata
  G. FRENSEL, P. M. BRUIJN
  9th symposium on INformation COntrol in Manufacturing, INCOM'98 - Advances in Industrial Engineering, volume II pp. 47-52. Nancy - Metz, Francia. 1998
- Grafchart and its relations to grafcet and petri nets
  C. JOHNSSON, K. E. ARZEN
  9th symposium on INformation COntrol in Manufacturing, INCOM'98 - Advances in Industrial Engineering, volume II pp. 53-58. Nancy - Metz, Francia. 1998
- Implementation of hierarchical sequential functional charts for programmable logic controllers
  P. CHIACCHIO
  9th symposium on INformation COntrol in Manufacturing, INCOM'98 - Advances in Industrial Engineering, volume II pp. 59-64. Nancy - Metz, Francia. 1998
- Validation of a SFC software specification by using hybrid automata
  G. HASSAPIS, I. KOTINI, Z. DOULGERI
  9th symposium on INformation COntrol in Manufacturing, INCOM'98 - Advances in Industrial Engineering, volume II pp. 65-70. Nancy - Metz, Francia. 1998
- Grafchart and batch-recipe structures
  C. JOHNSSON, K. E. ARZEN
  WBF'98, Baltimore, USA, 1998
- Reactivite et determinisme du comportement temporel du GRAFCET
  J.J. LESAGE, J.M. ROUSSEL, J.M. FAURE, P. LHOSTE, J. ZAYTOON
  Actes du Congrès ADPM'98 : 3ème conférence internationale sur l'Automatisation Des Processus Mixtes, pp. 99-106, Reims, Francia. 1998
- On batch recipe structures using high-level grafchart
  C. JOHNSSON, K. E. ARZEN
  Actes du Congrès ADPM'98 : 3ème conférence internationale sur l'Automatisation Des Processus Mixtes, pp. 107-114, Reims, Francia. 1998
- Methode et outils pour l'analyse et la modelisation du comportement d'un systeme hybride complexe
  C. DELTHEIL, J.P. FRACHET, D. LEANDRI
  Actes du Congrès ADPM'98 : 3ème conférence internationale sur l'Automatisation Des Processus Mixtes, pp. 115-121, Reims, Francia. 1998
- Requirements for extending GRAFCET to hybrid specifications
  L. GUILLEMAUD, J.M. GRAVE, H. GUEGUEN
  Actes du Congrès ADPM'98 : 3ème conférence internationale sur l'Automatisation Des Processus Mixtes, pp. 209-215, Reims, Francia. 1998

- Modelisation et verification de la cellule korso par une boite a outils GRAFCET
  D. L'HER, P. LE PARC, J.L. SCHARBARG, L. MARCE
  Actes du Congrès ADPM'98 : 3ème conférence internationale sur l'Automatisation Des Processus Mixtes, pp. 216-223, Reims, Francia. 1998
- Preuve de proprietes dynamiques de GRAFCETS a partir de leur description structurelle
  S. LAMPERIERE, J.M. FAURE, F. COUFFIN
  Actes du Congrès ADPM'98 : 3ème conférence internationale sur l'Automatisation Des Processus Mixtes, pp. 224-231, Reims, Francia. 1998
- Automates microprogrammes sensibles a des evenements mixtes generalises
  J. FLORINE
  Actes du Congrès ADPM'98 : 3ème conférence internationale sur l'Automatisation Des Processus Mixtes, pp. 232-239, Reims, Francia. 1998
- The hyperfinite signal : a new concept for modelling dynamic systems
  J.P. FRACHET
  Internal conference on Computing Anticipating Systems CASYS '97 HEC Liège Belgium
- Grafchart: a Petri Net/GRAFCET based graphical language for real-time sequential control applications
  K.E. ARZEN, C.JOHNSSON,
  SNART'97, Lund, Suiza, 1997
- On the synthesis of grafcet using the supervisory control theory
  J. ZAYTOON, C. NDJAB, V. CARRE-MENETRIER
  IFAC Conference on Control and Industrial Systems, Belfort, France, pp 391-396
- Implementation aspects of the plc standard iec 1131-3
  M. OHMAN, S. JOHANSSON, K.E. ARZEN
  CACSD'97, IFAC, pp. 15-20, Belgium, 1997
- AGGLAE: un outil d'aide a la validation des GRAFCETS de specification
  J.M. ROUSSEL, J.J. LESAGE
  Actes du congrès ELISA'97, Nancy, 1997
- On the recent advances in GRAFCET
  J. ZAYTOON, V. CARRE-MENETRIER, M. NICLET, P. DE LOOR
  IFAC Workshop on Manufacturing systems: Modelling, Management and Control MIM'97, pp. 419-424, Viena, Austria, 1997
- On the supremal controllable grafcet of a given GRAFCET
  J. ZAYTOON, C. NDJAB, J.M. ROUSSEL
  Proceedings of Congress "2nd Mathmod Vienna", pp. 371-376, Vienne, Austria, 1997

f) Artículos en revistas

- Demarche progressive d'implantation d'une commande sûre dans un a.p.i. a partir d'une specification GRAFCET
  V. CARRE-MENETRIER, J. ZAYTOON
  Revue R.E.E. 2000, pp 69-76
- Methods and tools for the synthesis of an optimal control implementation for GRAFCET
  V. CARRE-MENETRIER, C. NDJAB HAGBEBELL, J. ZAYTOON
  JESA, Ed HERMES, Volume 33 - n°8-9/1999, pages 1073 à 1092
- GRAFCET et graphe d'etats : comportement, raffinement, verification et validation
  J. ZAYTOON, V. CARRE-MENETRIER
  JESA, Ed HERMES, Volume 33 - n°7/1999, pages 751 à 782
- GRAFCET et graphes d'etats : synthese hors ligne de la commande
  Janan ZAYTOON, C. NDJAB HAGBEBELL, V. CARRE-MENETRIER
  JESA, Ed HERMES, Volume 33 - n°7/1999, pages 783 à 814

- GRAFCET revisited with a synchronous data-flow language
  P. LE PARC, D. L'HER, J.L. SCHARBARG ET L. MARCE

Revue IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Human - Vol. 29, No. 3 - 1999

- Implementation aspects of the plc standard iec 1131-3
  M. OHMAN, S. JOHANSSON, K.E. ARZEN
  IFAC - Control Engineering Practice 123, Vol 6 Num 4 , pp. 547-555, 1998
- L'enseignement du GRAFCET: pas si facile !
  D. GENDREAU
  Revue Technologies, numéro 94, pp. 11-18, 1998
- Le GRAFCET revisite a l'aide d'un langage synchrone flot de donnees
  P. LE PARC, D. L'HER, J.L. SCHARBARG, L. MARCE
  Revue Technique et Science informatiques Volume 17, numero 1, 1998 pp. 63-86
- Un systeme expert base sur le grafcet pour le controle de procede
  O.M. BADIE, J.Y. JOURNEAUX, B. GRAVIL
  JESA-AFCET/CNRS, Ed. HERMES, Vol. 31-N°8, pp. 1259-1274, 1997
- Contribution to the GRAFCET formalisation: a static meta-model proposition
  F. COUFFIN, S. LAMPERIERE, J.M. FAURE
  JESA-AFCET/CNRS, Ed HERMES, Vol. 31-N°4, pp. 645-667, 1997
- Comportement temporel du GRAFCET
  P. LHOSTE, J.M. FAURE, J.J. LESAGE, J. ZAYTOON
  JESA-AFCET/CNRS, Ed HERMES, Vol. 31-N°4, pp. 695-711, 1997
- Verification et validation du GRAFCET
  J. ZAYTOON, J.J. LESAGE, L. MARCE, J.M. FAURE, P. LHOSTE
  JESA-AFCET/CNRS, Ed HERMES, Vol. 31-N°4, pp. 713-740, 1997
- Modelling discrete event systems bahaviour using the hyperfinite signal
  J.P. FRACHET, S. LAMPERIERE, J.M. FAURE
  JESA-AFCET/CNRS, Ed HERMES, Vol. 31-N°3, pp. 453-470, 1997
- GRAFCET: from theory to implementation
  E. BIEREL, O. DOUCHIN, P. LHOSTE
  JESA-AFCET/CNRS, Ed HERMES, Vol. 31-N°3, pp. 543-559, 1997
- Abstractions and heuristics for the validation of grafcet controlled systems
  P. DE LOOR, J. ZAYTOON, G. VILLERMAIN-LECOLIER
  JESA-AFCET/CNRS, Ed HERMES, Vol. 31-N°3, pp. 561-580, 1997
- Formal semantics for reactive GRAFCET
  F. CASSEZ
  JESA-AFCET/CNRS, Ed HERMES, Vol. 31-N°3, pp. 581-603, 1997
- N'attendez pas trop de la programmation en cei 1131-3 ...
  J.F. PEYRUCAT
  MESURES n°690, pp. 65-68, 1996
- Le systeme sacaloo et l'algorithme d'interpretation du GRAFCET
  K. ABDERRAHIM, M. TAHAR BHIRI, M. KSOURI
  JESA-AFCET/CNRS, Ed HERMES, Vol. 30-N°9, pp. 1253-1272, 1996
- Specification and design of logic controllers for automated manufacturing systems
  J. ZAYTOON
  Robotics and Computer Integrated Engineering, vol. 12, no. 4, P. 353-366, 1996
- Un modele pour la specification, la conception et la validation des systemes de controle/commande repartis : le modele ACSY-R
  V. CHAPURLAT, F. PRUNET
  JESA-AFCET/CNRS, Ed HERMES, Vol. 30-N°1, pp. 25-62, 1996

1.5. Herramientas de automatización: RdP

A lo largo de todo este trabajo se emplean constantemente las RdP como una herramienta para el modelado y el diseño de automatizaciones. Si bien poseen una extensa base teórica que las avala en todos los campos en los que son empleadas, entre ellos la automatización industrial, y es considerable la producción científica que se produce en torno suyo, en este caso serán empleadas como herramientas para la investigación sobre los procesos industriales automáticos, no como el objeto de la investigación.

Aún así se ha considerado interesante incluir las definiciones más interesantes, una clasificación de los tipos y las herramientas comerciales que pueden encontrarse para cada tipo, para poder recurrir a estos apartados o a sus referencias en cualquier momento en que se necesite.

1.5.1. Definiciones

Uno de los términos principales en este trabajo es el de *autómata programable*. La definición de *autómata* según el diccionario de la Real Academia Española es:

> Del latín automata, y éste del griego aütõmatoj, espontáneo.
> 1. [m.] Instrumento o aparato que encierra dentro de sí el mecanismo que le imprime determinados movimientos.
> 2. [m.] Máquina que imita la figura y los movimientos de un ser animado.

Y la definición de *programar*, en sus acepciones 2ª, 3ª y 5ª es:

> 2. [tr.]Idear y ordenar las acciones necesarias para realizar un proyecto. Ú. t. c. prnl.
> 3. [tr.]Preparar ciertas máquinas por anticipado para que empiecen a funcionar en el momento previsto.
> 5. [tr.]Inform. Elaborar programas para los ordenadores

De ambas definiciones se obtiene una idea aproximada de lo que es un autómata programable: es un dispositivo lógico en el que se puede programar la respuesta de las salidas en función de la secuencia de entradas que se ha recibido (o en función de las entradas que se está recibiendo y del estado interno). Su traducción al inglés es Programmable Logic Device (PLC), dispositivo lógico programable, que da una idea de la definición que se acaba de dar.

Pero *autómata* también se puede definir desde un punto de vista matemático. En ese caso su definición es:

Autómata es un sistema discreto tal que:

- Recibe un número finito de símbolos. El conjunto de los símbolos que recibe se denomina *alfabeto de entrada, E.*
- Emite un número finito de símbolos. El conjunto de los símbolos que emite se denomina *alfabeto de salida, S.*

Si el autómata es tal que el conjunto de los estados, *Q*, es finito, se dice que el autómata es de estados finito o simplemente, que es un autómata finito.

En un resumen, un autómata finito (AF) es una quíntupla $\langle E, S, Q, \lambda, \delta \rangle$ en la que:

1) *$E = \{E_i\}$ es el alfabeto de la entrada (conjunto finito).*
2) *$S = \{S_j\}$ es el alfabeto de salida (conjunto finito).*
3) *$Q = \{Q_K\}$ es el conjunto finito de estados (internos) del autómata.*
4) δ es la función de transición, $\delta = Q \times E \rightarrow Q$.
5) λ es la función de lectura o salida, $\lambda = Q \times E \rightarrow S$.

Esta definición que se acaba de ver, la de *autómata*, es importante debido a que se emplea a lo largo del trabajo en sus dos acepciones (en realidad se pueden considerar más como dos puntos de vista). Casi siempre nos estaremos refiriendo a *autómatas programables*, con la primera de las acepciones mostradas, puesto que ésta es una tesis fundamentalmente técnica. Sin embargo este trabajo también desarrolla y emplea una base científica, en la que utiliza el término *autómata* en su sentido matemático, la acepción mostrada en segundo lugar.

A continuación se van a exponer una serie de definiciones relativas a las RdP que pueden ser útiles para entender mejor esta tesis. Esas definiciones pueden obtenerse muy ampliadas en las referncias, pero aun así se incluyen en este apartado introductorio para relacionarlas con su aplicación industrial, especialmente desde el punto de vista de su utilización a lo largo de este trabajo. Por lo tanto a continuación se irá alternando una serie de definiciones formales expresadas en terminología matemática, con algunas definiciones más cercanas al punto de vista industrial, a modo de explicaciones, y que harán mucho más entendible el resto del trabajo a los lectores no expertos en RdP. Muchas de las definiciones no provienen de una terminología estandarizada, por no existir ésta todavía, por lo que pueden diferir según las referencias, pero afortunadamente eso es algo poco habitual.

Un grafo, como su nombre sugiere, es la representación gráfica de los datos de una situación particular. Los datos contienen, en algunos casos, relaciones entre ellos, no necesariamente en forma jerarquizada (por ejemplo las ciudades de un mapa conectadas por las carreteras). La estructura de datos que refleja esta relación recibe el nombre de grafo. Se suelen usar muchos nombres al referirnos a los elementos de una estructura de datos. Algunos de ellos son “elemento”, “ítem”, “asociación de ítems”, “registro”, “nodo” y “objeto”. El nombre que se utiliza depende del tipo de estructura, el contexto en que usamos esa estructura y quién la utiliza. En la mayoría de los textos de estructura de datos se utiliza el termino “registro” al hacer referencia a archivos y “nodo” cuando se usan listas enlazadas, árboles y grafos.

Un grafo también es una terna G = (V,A,j), en donde V y A son conjuntos finitos, y j es una aplicación que hace corresponder a cada elemento de A un par de elementos de V. Los elementos de V y de A se llaman, respectivamente, "vértices" y "aristas" de G, y j asocia entonces a cada arista con sus dos vértices. Esta definición da lugar a una representación gráfica, en donde cada vértice es un punto del plano, y cada arista es una línea que une a sus dos vértices. Si el dibujo puede efectuarse sin que haya superposición de líneas, se dice que G es un grafo plano.

Un GR es un grafo en el que la condición lógica que provoca una transición entre dos estados es cualquier función lógica de las entradas. Un GR permite modelar autómatas finitos (AF).

Una red de Petri es un grafo orientado en el que intervienen dos clases de nudos, los *lugares* (representados por circunferencias) y las *transiciones* (representadas por pequeños segmentos rectilíneos), unidos alternativamente por *arcos.* Un arco une un lugar con una transición, o viceversa, pero nunca puede unir dos transiciones o dos lugares. Un lugar puede contener un número positivo o nulo de *marcas.* Una marca es representada por un punto en el interior del círculo que representa al lugar. El conjunto de marcas asociadas a cada uno de los lugares en un instante dado constituye lo que se denomina un *marcado* de la RdP.

Para describir funcionalmente los sistemas concurrentes, a los lugares se les asocian *acciones o salidas* del sistema que se desea modelar, y a las transiciones se les asocian los eventos (funciones lógicas de las variables de entrada del sistema) y acciones o salidas.

El disparo de una transición supone quitar un cierto número de marcas a cada uno de los lugares de entrada y añadir otro cierto número de marcas a cada uno de los lugares de salida.

En un grafo reducido un sistema se representa mediante el conjunto de estados totales en los que tal sistema se pueda encontrar. En una RdP el estado está representado por el marcado. Como el marcado puede contener varios lugares marcados simultáneamente, el estado está definido por un conjunto de subestados (estados locales o parciales) del sistema. De la anterior consideración, se deduce que una RdP está mejor adaptada que los GR para la descripción de evoluciones simultáneas.

Un lugar con varios arcos de entrada y/o de salida se *denomina nudo O.* Un grafo reducido está formado a base *de nudos O.* Existen dos casos particulares de nudos *O.*

- La selección (un arco de entrada y varios de salida).
- La atribución (varios arcos de entrada y uno de salida).

Una transición con varios arcos de entrada y/o salida se denomina nudo Y. Ha de observarse que estos nudos no existen en los grafos reducidos. Tales nudos Y son los que permiten la creación y extinción de las evoluciones simultáneas. Dos casos particulares de nudos Y son:

- La distribución (un arco de entrada y varios de salida).
- La conjunción (varios arcos de entrada y uno de salida).

A una RdP que no contenga nudos Y se la denomina máquina o grafo de estados. Véase como, según la interpretación dada, esta definición no coincide con la de grafo reducido, ya que en un grafo de estado podemos tener simultáneamente varias marcas. Para que un grafo de estados coincida con un grado reducido es necesario que su marcado se limite a una sola marca.

Se dice que una transición es viva, para un marcado $M_{o,}$ cuando para todo marcado M alcanzable a partir del marcado inicial M_o existe un marcado M´ sucesor de M a partir del cual puede dispararse esa transición. Una RdP es viva, para un marcado dado, cuando todas sus transiciones son vivas para ese marcado.

A partir de un marcado inicial dado, una RdP es binaria cuando cualquier marcado alcanzable es tal que ningún lugar posee más de una marca. En una RdP binaria cualquier lugar estará marcado con una marca o no estará marcado.

Para transformar una RdP en un GR la idea básica consiste en obtener todos los marcados posibles y las transiciones entre éstos.

Una red de Petri generalizada es una cuádrupla R = $\langle P, T, \alpha, \beta \rangle$ tal que

P es un conjunto finito y no vacío de lugares.
T es un conjunto finito y no vacio de transiciones.
$P \cap T = \varnothing$; es decir, lugares y transición son conjuntos disjuntos.
$\alpha. : PxT \rightarrow$ N es la función de incidencia previa.
$\beta. : PxT \rightarrow$ N es la función de incidencia posterior.

Gráficamente una RdP se representa por un grafo bipartido orientado. Los lugares se representan por circunferencias y las transiciones por barras. Existe un arco que va del lugar p_i a la transición t_j sii $\alpha(p_i, t_j) \neq 0$. Análogamente, existe un arco que va desde la transición t_k al lugar p_i sii $\beta(t_k, p_i) \neq 0$. Cada arco se etiqueta con un entero natural, $\alpha(p, t)$ ó $\beta(t, p)$, denominado peso de arco. Como convención se admite que un arco no etiquetado posee un peso unitario. A efectos de claridad y legibilidad, los arcos con peso mayor que la unidad se dibujan usualmente con dos o más trazos paralelos. o con un trazo grueso.

Matricialmente una RdP se representa mediante de dos matrices. Sea $|P|$ = n (número de lugares de la red) y sea $|T|$ = m (número de transiciones de la red). Se denomina matriz de incidencia previa a la matriz $C^- = \left[C_{ij}^-\right]_{n x m}$, en la que $c_{ij}^- = \alpha(p_i, t_j)$. Se denomina matriz de incidencia posterior a la matriz $C+ = \left[C_{ij}^+\right]_{n x m}$, en la que $c_{ij}^+ = \beta(t_j, p_i)$.

Una red es ordinaria cuando sus funciones de incidencia sólo pueden tomar los valores 0 y 1.

$\alpha(p,t) \in \{0,1\}$
$\beta(t,p) \in \{0,1\}$

Una red es pura cuando ninguna transición contiene un lugar que sea simultáneamente de entrada y de salida:

$$\forall t_j \in T \quad \forall p_i \in P \quad \alpha(p_i, t_j)\beta(t_j, p_i) = 0$$

Matricialmente la representación de una red pura se simplifica definiendo una única matriz, C, denominada matriz de incidencia:

$$C = C^{+} - C^{-} \Rightarrow C_{ij} = \beta(t_j, p_i) - \alpha(t_i, p_j)$$

Asi, en la matriz C un elemento positivo indica incidencia posterior y uno negativo incidencia previa. Un elemento nulo en C indica que la transición y lugar correspondientes no están conectados directamente a través de un arco.

Sea una red $R = \langle R, T, \alpha, \beta, \rangle$, $t \in T$ y $p \in P$. Se definen los siguientes conjuntos:

a) Conjunto de lugares de entrada de t:

$${}^{\bullet}t = \{p \in P | \alpha(p,t)\} > 0$$

b) Conjunto de lugares de salida de t:

$$t^{\bullet} = \{p \in P | \beta(t,p)\} > 0$$

c) Conjunto de transiciones de entrada de p:

$${}^{\bullet}p = \{t \in T | \beta(t,p)\} > 0$$

d) Conjunto de transiciones de salida de p:

$$p^{\bullet} = \{t \in T | \alpha(p,t)\} > 0$$

De acuerdo con esta anotación una red es pura sii $\forall t \in T {}^{\bullet}t \cap t^{\bullet} = \varnothing$

$$P \subseteq P \quad y \quad T \subseteq T.$$

Una subred de ***R***=$\langle R,T,\alpha,\beta,\rangle$ es una red $\overline{R} = \langle \overline{P}, \overline{T}, \overline{\alpha}, \overline{\beta} \rangle$ tal que $\overline{P} \subseteq P$ y $\overline{T} \subseteq T$. $\overline{\alpha}$ y $\overline{\beta}$ son restricciones de α y β sobre $\overline{P}x\overline{T}$.

Una transición $t \in T$ está sensibilizada por el marcado *M* cuando cada uno de sus lugares de entrada posee al menos $\alpha(p,t)$ marcas. Es decir, ha de cumplirse que $\forall p \in {}^{\bullet}t \quad M(p) \geq \alpha(p,t)$.

Disparar una transición sensibilizada *t* es la operación que consiste en eliminar $\alpha(p,t)$ marcas a cada lugar $p \in {}^{\bullet}t$ y añadir $\beta(t,p)$marcas a cada lugar $p \in {}^{\bullet}t$. A esto se le denomina regla de evolución del marcado.

$M_i \rightarrow M_j$ simboliza que *t* está sensibilizada por M_i, y que al disparar ***t* al** partir de M_i se alcanza M_j. Es decir, al disparar *t* se obtiene:

$$M_j(p) = M_i(p) + \beta(t,p) - \alpha(p,t) \quad \forall p \in P$$

Un marcado *M* es alcanzable a partir de un marcado inicial $M_\circ$ sii existe una secuencia de disparos aplicable a partir de $M_\circ$ que transforma $M_\circ$ en $M : M_\circ \xrightarrow{\sigma} M$.

La ecuación de estado de una red de Petri se define de la siguiente manera:

$$M_k = M_\circ + C \cdot \sum_{j=1}^{K} U_j = M_o + C \cdot \overline{\sigma} \quad \text{donde} \quad M_\circ \xrightarrow{\sigma} M_K .$$

1) $M_K = M_{K-1} + C \cdot U_K$

2) $\left(M_\circ \xrightarrow{\sigma} M_K \right) \Rightarrow M_K = M_\circ + C \cdot \sigma$

La primera ecuación adquiere la forma de la ecuación de estado de un sistema dinámico lineal e invariante, discretizado en el tiempo, por lo que frecuentemente es conocida como ecuación de estado asociada a la red de Petri. La segunda ecuación integra la evolución desde M_0 hasta M_k.

Un evento o acontecimiento, E_i, se define como:

- El cambio de un estado lógico de una condición externa.
- La unión de un conjunto de eventos.

Interpretación asociada a las RdP. Por convenio asociaremos:

- eventos, condiciones externas y salidas impulsionales a las transiciones,
- salidas a nivel a los lugares.

La activación de una salida puede estar ligada a condiciones externas. En tal caso hablaremos de salidas condicionales.

Una RdP interpretada (RdPI) se define por:

- Una RdP marcada $\langle R, M_\circ \rangle$;
- Una aplicación T en E que asocia a cada transición un evento, E_i;
- Una aplicación de T en C que asocia a cada transición una condición externa, C_i;
- Una aplicación de T en Z que asocia a las transiciones salidas impulsionales;
- Una aplicación de PxC en Y que asocia a los lugares les asocia salidas a nivel, eventualmente condicionadas.

Reglas de evolución del marcado de una RdPI:

Regla 1: El disparo de una transición t_i sensibilizada sólo se realiza si se verifica C_i, E_i. En tales condiciones se dice que t_i es receptiva a la condición y al evento (C_i, E_i).

Regla 2: Cuando una transición es disparada, todas las salidas impulsionales asociadas a t_i son generadas.

Regla 3: Con un marcado dado, las salidas a nivel asociadas a los lugares marcados son generadas cuando se verifican las condiciones externas que en su caso les pueden estar asociadas. Si Y_j es una salida asociada a un lugar p_i condicionada por C_{ij}, se dice que la salida es sensible a C_{ij}, si p_i está marcado.

Un grafo de estados (GE) o máquina de estados (ME) es una RdP tal que:

$$\forall t \in T \quad \left| {}^\bullet t \right| = 1 \quad y \quad \left| t^\bullet \right| = 1 ,$$

es decir, tal que en ella cualquier transición tiene un lugar de entrada y uno de salida. Un grafo de estados no posee nudos Y.

Un grafo marcado (GM) o grafo de sincronización es una RdP tal que:

$$\forall p \in P \quad \left|{}^{\bullet} p\right| = 1 \quad y \quad \left|p^{\bullet}\right| = 1,$$

de modo que en ella cualquier lugar tiene como máximo una transición de entrada y una transición de salida. Un grafo marcado no posee nudos O.

Una RdP libre elección (RLE) es una red tal que:

$$\forall p \in P, \quad si \left|p^{\bullet}\right| = 1, \text{ entonces } \forall t_K \in p^{\bullet}, \quad \left|{}^{\bullet} t_K\right| = 1,$$

es decir, si dos transiciones $t_i \; y \; t_j$ tienen un lugar de entrada *p* en común, se deduce que *p* es el único lugar de entrada de $t_i \; y \; de \; t_j$.

Una red de Petri simple (RS) es una RdP en la que toda transición tiene como máximo un lugar de entrada compartido con otras transiciones.

La primera extensión de las RdP ordinarias (RdP) son las RdP generalizadas (RdPG). Con respecto a las redes ordinarias, las RdPG son RdP "mejoradas" introduciendo el concepto de peso de un arco. La RdPG es tal que el peso de los arcos que van de la transición *a* a $p_2 \; y \; de \; p_3$ a *a* es *k*: $\beta(a, p_2) = \alpha(p_3, a) = k$.

Una red de Petri con capacidad limitada (RdPC) es una quíntupla $\langle P, T, \alpha, \beta, \gamma \rangle$, donde R=$\langle P, T, \alpha, \beta, \rangle$, es una RdPG y γ es una función que asocia a cada lugar su capacidad; es decir, su valor indica el máximo número de marcas que puede contener $(\gamma : P \rightarrow \mathrm{N} \cup \{\infty\})$. $\gamma(p)$ es la capacidad del lugar *p*.
Teniendo en cuenta que el disparo de una transición *t* conduce del marcado M_i al marcado M_j definido por:

$$M_i(p) = M_i(p) + \beta(t, p) - \alpha(p, t) \quad \forall p \in P,$$

resulta que el marcado de una RdPC evoluciona de acuerdo con la siguiente regla de sensibilización de transiciones: una transición *t* está sensibilizada cuando:

$$\forall p \in t^{\bullet} \quad M(p') + \beta(t, p') - \alpha(p', t) \leq \gamma(p'). \; \forall p \in {}^{\bullet} t \quad M(p) \geq \alpha(p, t).$$

Un lugar $\overline{p}$ es complementario del lugar *p* cuando

$$\forall t \in T \qquad \alpha(p, t) = \beta(t, p) \quad y \quad \beta(t, p) = \alpha(p, t)$$

$$\forall t \in T \; \alpha(p, t) = \beta(t, \overline{p}) \; \text{y} \; \beta(t, p) = \alpha(\overline{p}, t),$$

es decir, el disparo de cualquier transición de la red quita o añade tantas marcas de *p* como añade o quita de $\overline{p}$.

Una red de Petri con arcos inhibidores (RdPAI) es una red a la que se añaden unos arcos que sólo parten de lugares y van a transiciones, denominados arcos inhibidores, *I(p,t)*. La regla de disparo de una transición en una RdPAI exige que estén desmarcados todos los lugares que se encuentren unidos a la transición mediante un arco inhibidor. Desde

desde un punto de vista gráfico, un arco inhibidor se distingue de un arco normal por una pequeña circunferencia situada en el extremo que incide sobre la transición.

Las redes de Petri coloreadas se introducen a fin de condensar la descripción (y el análisis) de sistemas en los que se identifican diversos subsistemas con estructura, y comportamientos similares, pero que trabajan en paralelo.

Una RdP temporizada (RdPT) es un par $\langle R, Z \rangle$ tal que R=$\langle P, T, \alpha, \beta \rangle$ y Z es una función que asigna un número real no negativo, z_i, a cada transición de la red: $Z: T \rightarrow R^+$

$z_i = Z(t)$ es llamado tiempo de disparo de la transición t

La regla de evolución del marcado es idéntica a la de una RdPG, con la única diferencia de que el disparo de t dura z_i unidades de tiempo.

Una RdPT es un par $\langle R, Z \rangle$ tal que R=$\langle P, T, \alpha, \beta \rangle$ y Z es una función que asigna un número real no negativo, z_i, a cada lugar de la red:

$Z: P \rightarrow R^+$.

Una marca en una RdPT puede encontrarse en dos estados: dispuesta o indispuesta. Las reglas de evolución del marcado son idénticas a las de las RdP cuando las marcas están dispuestas. Si las marcas están indispuestas, es como si no existieran en lo que se refiere a la evolución de la red. Al pasar a un lugar una marca, entra en estado indispuesto y pasa a estado dispuesto después de z_i=Z(p_i) unidades de tiempo. Esta clase de modelos se usualmente empleado para la evaluación de prestaciones.

Un lugar p_i está implícito en la red marcada $\langle R, M_o \rangle$ cuando para todo marcado alcanzable se verifica que:

1) $M(p_i) = \sum_{\substack{j=i \\ i \neq j}}^{n} \lambda_j M(p_j) + \mu \quad \lambda_j \in Q^+, \mu \in Q$

$$\forall p_j \in P - \{p_i\} \quad M(p_j) \geq \alpha(p_j, t_K) \Rightarrow M(p_i) \geq \alpha(p_i, t_k).$$

2) $\forall p_j \in P - \{p_i\} \; M(p_j) \geq \alpha(p_j, t_k) \Rightarrow M(p_i) \geq \alpha(p_i, t_k)$

Todo conjunto de lugares $\prod_i$ = { $p_j | \lambda_j > 0$} se denomina conjunto de lugares implicantes de p_i.

Propiedad de vivacidad. Se dice que una transición t es viva, para un marcado inicial dado M_0, sii existe una secuencia de disparos a partir de cualquier marcado M, sucesor de M_0, que comprenda a t:

Una RdP marcada es viva para M_0 sii todas sus transiciones son vivas para M_0.

La importancia de la propiedad de vivacidad estriba en su capacidad para caracterizar el bloqueo de un sistema. En efecto, si una RdP es viva, el sistema no puede bloquearse, ya que que todas las transiciones pueden llegar a dispararse. Evidentemente, la

proposición contraria no es cierta. También puede suceder que una RdP marcada no viva no se bloquee. Para caracterizar esta última situación se define la noción de RdP marcada parcialmente viva.

Se dice que una RdP marcada es parcialmente viva para M_0 si, tomando como punto de partida cualquier marcado alcanzable a partir de M_0, existe al menos una transición disparable y otra transición no viva.

Toda RdP marcada parcialmente viva, tiene la posibilidad de evolución global, independientemente de que existan transiciones que no puedan ser disparadas.

Una RdP es estructuralmente viva si existe un M_0 finito para el cual la RdP marcada es viva.

Se dice que una RdP posee un comportamiento globalmente cíclico para M_0 cuando existe una secuencia de disparos que permite alcanzar el marcado inicial M_0 a partir de cualquier marcado *M* alcanzable a partir de M_0:

$$\forall M \in M(R, M_0),\ \exists \sigma \quad tal\ que\ M^{\sigma} \rightarrow M_0 .$$

Un lugar p es *k-limitado para* M_0 cuando existe un número entero k tal que $M(p) \leq k$ para cualquier marcado $M \in M(R, M_0)$. Se denomina límite de lugar p al menor entero k que verifica la desigualdad anterior.

Una RdP marcada es *k-limitada para* M_0 si todos sus lugares son *k-limitados* para M_0: $\forall p \in P\ y\ \forall M \in M(R, M_0), M(p) \leq k$.

La propiedad de limitación determina la finitud del número de estados del sistema representado por una RdP. Desde un punto de vista práctico, esta propiedad ha de verificarse, dado que los lugares se realizarán con memorias de capacidad finita.

Una RdP es estructuralmente limitada cuando es limitada para cualquier marcado inicial y finito.

Se dice que en una RdP existe conflicto estructural cuando un lugar posee más de una transición de salida.

Se dice que dos transiciones, $t_i\ y\ t_j$, están en conflicto efectivo para M_0 sii:

-existe $M \in M(R, M_0)$ que sensibiliza a $t_i\ y\ t_j$;

-al disparar t_i (t_j) el marcado obtenido no sensibiliza la transición $t_j(t_i)$.

Se dice que dos lugares de una RdP están en exclusión mutua para M_0 cuando no pueden estar marcados simultáneamente en los marcados alcanzables a partir de M_0.

El avance sincrónico de la transición t_i con respecto a la transición t_j en la red $\langle R, M_0 \rangle$ es el valor máximo que, considerando todas las secuencias de disparo posibles, puede tomar la diferencia entre el número de disparos de t_i y el de t_j.

Un cerrojo es un subconjunto de lugares de una RdP tal que el conjunto de sus transiciones de entrada está contenido en el conjunto se sus transiciones de salida. Es decir, todo cerrojo T=$\{p_i\} \subseteq P$ verifica $\{{}^\bullet p_i\} \subseteq \{p^\bullet{}_i\}$, lo cual se expresa como $T \subseteq T^\bullet$.

A su vez, una trampa es un subconjunto de lugares de una RdP tal que el conjunto de sus transiciones de salida está contenido en el conjunto de sus transiciones de entrada.

Consideramos que $R, \theta \subseteq P$ es una trampa sii $\theta^\bullet \subseteq^\bullet \theta$. En las definiciones de cerrojos y trampas, sólo se considera la estructura de la RdP, haciéndose abstracción del marcado inicial.

1.5.2. Clasificación

Clasificación de las Redes de Petri (RdP)
Nivel 1
Redes de Petri caracterizadas por lugares que pueden representar valores booleanos, i.e., un lugar está ocupado por al menos una marca.

- Sistemas Condición/Evento (C/E)
- Sistemas de Redes Elementales (EN)
- Sistemas 1-safe
    - Máquinas de Estado (SM)

Nivel 2
Redes de Petri caracterizadas por Lugares que pueden representar valores enteros, i.e., un lugar es marcado por un número de marcas.

- Redes Lugar/Transición (P/T)
    - (Ordinarias) Redes de Petri (RdP)
        - Sistemas de Redes 1-safe
        - Redes de Libre Elección
            - Sistemas S
                - Máquinas de Estado (SM)
            - Sistemas T
                - Grafos Marcados (MG)
            - Estructuras de Extensiones Libre Elección

Nivel 3
Redes de Petri caracterizadas por Lugares que pueden representar valores de altas prestaciones, i.e., un lugar está marcado por un conjunto múltiple de marcas estructurales.

- Redes de Petri de Altas Prestaciones con Tipos de Datos Abstractos (HL+ADT)
    - Redes de Petri Algebraicas (ALG)
    - Redes OBJSA
- Redes de Relación con el Entorno (ER)
- Redes de Producción (Prod)
- Redes Bien Formadas (Coloreadas) (WN)
    - Redes Regulares (RN)
- Redes de Petri Tradicionales de Altas Prestaciones
    - Redes Predicado/Transición
    - Redes de Petri Coloreadas
- Redes de Petri Continuas e Híbridas
    - Redes de Petri Continuas
    - Redes de Petri Híbridas

SISTEMAS CONDICIÓN/EVENTO (C/E)

Definición y Reglas de Transición

De acuerdo con [1], un Sistema C/E es una cuádrupla $\Sigma=(B,E;F,C)$ donde:

1) (B, E; F) es una red simple (B es el conjunto de condiciones, E el conjunto de eventos).
2) Cada caso $c \in C$ es un conjunto de condiciones ($c \subseteq B$) y $C \subseteq P(B)$ es la clase del marcado total.
3) Cada evento tiene concesión (una posibilidad de ocurrir) en algún caso:
   $\forall e \in E\ \exists c \in C$ tal que $(\bullet e \subseteq c) \wedge (e \bullet \cap c = \emptyset)$

Cada caso es alcanzable por cada otro caso en un número finito de (directo e inverso) pasos:

C es una clase equivalente de la relación de alcanzabilidad completa R ;

$R = (r \cup r')^* \subseteq K \times K$ donde $r \subseteq K \times K$ es la relación de alcanzabilidad en un paso definida como:

$(K,K') \in r \Leftrightarrow \exists G \subseteq E$ tal que

(i) $\forall e,e' \in G:[e \neq e' \Rightarrow (\bullet e \cap \bullet e') = (e \bullet \cap e' \bullet) = \emptyset]$

(ii) $K - K' = \bullet G$

(iii) $K' - K = G \bullet$

$(K, K') \in r$ está denotado por $k[G\rangle K'$

Propiedades Estructurales

- **pureza:** de acuerdo con el punto 3 de la definición
- **simplicidad:** un Sistema C/E tiene que ser simple por definición
- **viveza-1:** de acuerdo con el punto 3 cada evento es vivo-1
- **alcanzabilidad directa e inversa:**

Las Transiciones en un Sistema C/E pueden ocurrir, individualmente o en concurrencia, directa e inversa, y el resultado completo de la clase de alcanzabilidad del sistema es mayor que la unión de las clases de alcanzabilidad directa e inversa. La alcanzabilidad completa es un relación de equivalencia, que en cualquier caso puede ser cambiada a representar la clase de alcanzabilidad. Por el contrario, si el sistema de redes evoluciona sólo directamente, la acanzabilidad no es más que una relación de equivalencia y la elección de un caso distinguido al principio cambia la clase de alcanzabilidad.

Herramientas: MOBY, POSES, HyperNet, WinPetri

Referencias: [2]; [3]; [1]

SISTEMAS DE REDES ELEMENTALES (EN)

Definición

(de acuerdo con [1])

Un Sistema de Redes Elementales (EN) es una cuádrupla $N = (B, E; F, C_{in})$ donde

(B, E; F) es una red finita denominada la red subyacente de N y $C_{in} \subseteq B$ el caso inicial de N.

Reglas de Transición

$u \subseteq E$ es llamado un paso permitido en el caso de que $c \subseteq B$ (denotado por $c\ [u\rangle$) sii

(i) u es un conjunto independiente : Ind(u)

(ii) $\bullet u \subseteq c$
(iii) $u\bullet \cap c = \varnothing$

$c[u\rangle c'$ denota la relación elemental transitoria en P(B)xP(E)xP(B) definida como sigue:
$c[u\rangle c' = \{(c, u, c') \mid c[u\rangle \text{ and } c' = (c - \bullet u) \cup u\bullet\}$
El estado espacio (conjunto de casos C_N) de N es generado por un caso inicial C_{in} y la regla de transición:
C_N es el menor subconjunto de P (B) que satisface:

(i) $C_{in} \in C_N$
(ii) si $c \in C_N$, $u \subseteq E$ y $c' \subseteq B$ tal que $c\,[u\rangle c'$ entonces $c' \in C_N$

Propiedades
Transiciones con condiciones indirectas no están permitidas (muertas).
Referencias: [4]; [5]; [1].

SISTEMAS REDES 1-SAFE

Los sistemas 1-safe son definidos como una subclase de los sistemas P/T.

Definición
(de acuerdo con [1])
Un sistema 1-safe es un sistema P/T $\sum = (S, T; F, K, W, M_o)$ donde

1) $\forall s \in S: K(s) = \infty$
2) $\forall s \in S, \forall M \in [M_o\rangle : M(s) \leq 1$
3) $\forall f \in F: W(f) = 1$

$\sum$ puede denotarse simplemente por $(S, T; F, M_o)$ donde marcados de $[M_o\rangle$ son subconjuntos de S.

Regla de Transición
La regla de transición es dada de acuerdo a los sistemas P/T donde la que permite condiciones laterales (regla de transición segura) es usada más frecuentemente.

Propiedad
Los Sistemas Libre Contacto 1-safe y los Sistemas Libre Contacto EN coinciden.
Herramientas: EASE
Referencias:[1]

MÁQUINA DE ESTADO

Definición
De acuerdo con [6], una Máquina de Estado es una Red de Petri donde cada transición tiene sólo un lugar de entrada y un lugar de salida:
$\forall t \in T: \; |\bullet t| = |t \bullet| = 1$
Referencias: [6]; [7]

SISTEMAS LUGAR/TRANSICIÓN (P/T)

Definición

De acuerdo con [1] un Sistema Lugar/Transición es un séxtuplo $\sum = (S, T; F, K, W, M_o)$ donde

1) (S, T; F) es una red donde S es el conjunto de lugares y T el conjunto de transiciones
2) $K:S \rightarrow N^+ \cup\{\infty\}$ es una función capacidad que expresa el máximo número de marcas que cada lugar puede contener
3) $W:F \rightarrow N^+$ es una función peso que expresa cuantas marcas fluyen a través de ellos en cada ocurrencia de las transiciones envolventes
4) $M_o: S \rightarrow N$ es una función de marcado inicial que satisface:

$\forall s \in S: M_o(S) \leq k(s)$

Regla de Transición (semánticas intercaladas)

Una transición t está sensibilizada con un marcado M sii

$\forall s \in S: W(s,t) \leq M(s) - W(t,s)$

Si t ocurre produce un nuevo marcado $M'(s) = W(s,t) + W(t,s)\ \forall s \in S$ denotado por $M[t\rangle M'$.

La Clase de Alcanzabilidad Directa, denotada por $[M_o\rangle$ es el menor conjunto de marcados de $\sum$ tal que:

- $M_o \in [M_o\rangle$

si $M_1 \in [Mo\rangle$ y $M_1\ [t\rangle M_2$ para algún $t \in T$, entonces $M_2 \in [M_o\rangle$

Otra regla de transición dada por [8] y [9] difiere de la dada en las permitidas por las "condiciones laterales":

Una transición $t \in T$ permite un marcado M sii

$\forall s \in S: W(s,t) \leq M(s) \leq K(s) - W(t,s)+W(s,t)$

Comportamiento concurrente (semánticas de paso)

La definición acorde con [10] que corresponde a la definición de "paso simple" en [11]. Esta definición se puede extender a una "bolsa" de transiciones, informalmente, a un conjunto con multiplicidad (o multiconjunto) correspondientes a una noción de "paso" en [3].

Herramientas: CPN/AMI CPN/AMI, MOBY, PENECA, PESIM, PNS, TORAS, GRAF, CAPSNET, FORSEE, INA, PAPETRI, PEP, WINPETRI**.**

Referencias: [12]; [1]; [8]; [9]; [10]; [3]; [11]; [13]

REDES DE PETRI ORDINARIAS (PN)

Definición

De acuerdo con [1] un Sistema P/T $\sum=(S, T; F, K, W, M_o)$ tal que:

$\forall s \in S$: K(s) = ∞ y $\forall f \in F$: W(f)=l, denotado por Σ= (S, T; F, M_o) es conocido como Redes de Petri (ordinarias).

Regla de Transición

Una transición t es permitida bajo un cierto marcado M (M[t⟩) sii

$\forall s \in \bullet t$: M(s)>0

Si ocurre t cambia el marcado M a un nuevo marcado M′

(M[t⟩M′) tal que:

$\forall s \in S$

$$M'(s) = \begin{cases} M(s) - 1 & \text{si } (s \in \bullet t) \wedge (s \notin t\bullet) \\ M(s) + 1 & \text{si } (s \in t\bullet) \wedge (s \notin \bullet t) \\ M(s) & \text{si no} \end{cases}$$

Herramientas: ANARCO, INA, MetaDesign, NETMAN, PAPETRI, PN^3-EDITOR, XPETRI.

Referencias: [1]; [3]

SISTEMAS DE REDES 1-SAFE

Los Sistemas 1-safe son definidos como una subclase de los Sistemas P/T.

Definición

(de acuerdo con [1])

Un sistema 1-safe es un sistema P/T Σ=(S, T; F, K, W, M_o) donde

1) $\forall s \in S$: K(s) = ∞
2) $\forall s \in S$, $\forall M \in [M_o\rangle$: M(s) $\leq$1
3) $\forall f \in F$:W(f) = 1

Σ puede denotarse simplemente por (S, T; F, M_o) donde los marcados de [M_o) son subconjuntos de S.

Regla de Transición

La regla de transición es dada de acuerdo con los sistemas P/T donde la que permite condiciones laterales (regla de transición segura) es la más frecuentemente usada.

Propiedad

El contacto libre del sistema 1-safe y el contacto libre del sistema EN coincide.

Herramientas: EASE

Referencias: [1]

SISTEMAS LIBRE ELECCIÓN (FC)

Una Red Libre Elección es una Red de Petri ordinaria tal que cada arco procedente de un lugar es también un único arco de salida o un único arco de entrada.

Definición

(de acuerdo con [14])

Un Sistema Ordinario de Redes de Petri Σ=(S, T; F, M_o) es llamado un Sistema Libre elección sii la red subyacente es libre elección, i.e.

$\forall t\, t' \in T, t \neq t'$: $\bullet t \cap \bullet t' \neq \emptyset \Rightarrow |t\bullet| = 1 = |\bullet t'|$

equivalentemente : $\forall s, s' \in S, s \neq s'$: $s \bullet \cap s' \bullet \neq \emptyset \Rightarrow |s\bullet| = 1 = |s'\bullet|$

equivalentemente : $\forall s \in S, \forall t \in T: (s, t) \in F \Rightarrow s\bullet = \{t\} \vee \{s\} = \bullet t$

La Regla de Transición es dada de acuerdo con las Redes de Petri (ordinarias).

Extensiones

Las Extensiones de las Redes FC presentan restricciones estructurales más débiles.

Herramientas: QPN Tool

Referencias: [5]; [14]

SISTEMAS S

Definición

(de acuerdo con [5])

$\sum = (S, T; F, M_o)$ es un Sistema S sii su red subyacente es una red S, i.e.

$\forall t \in T: |\bullet t| \leq 1 \wedge |t\bullet| \leq 1$

GRAFOS MARCADOS

Definición

Un Grafo Marcado es puramente un Sistema de Red de Petri (ordinaria) donde cada lugar tiene sólo una transición de entrada y una transición de salida:

$\forall s \in S: |\bullet s| = |s\bullet| = 1$

Herramientas: PESIM

Referencias: [15]

EXTENSIONES ESTRUCTURALES LIBRE ELECCIÓN - INTRODUCCIÓN

- Estructura libre Elección (FC): si dos transiciones comparten un lugar de entrada común entonces no tienen otros lugares de entrada;
- Estructuras Extendidas Libre Elección (EFC): si dos transiciones tienen un lugar de entrada común no tiene otros lugares de entrada;
- Estructura Comportamentales Libre Elección (BFC): si dos transiciones tienen un lugar de entrada común, entonces, si una marca permite a una de ellas también permite a la otra;
- Estructura Asimétrica Elección (AC): si dos lugares tienen una transición de entrada común entonces el conjunto de transiciones de entrada de uno de ellos es un conjunto de transiciones de entrada del otro;
- El Comportamental Asimétrica Elección (BAC) es una ligera extensión de la AC;
- Las Redes Reducidas Asimétrica Elección (RAC) permiten toda estructura FC que no tenga más de dos lugares de entrada para cada transición y sólo un tipo de Estructura no-FC;

EXTENSIONES DE REDES LIBRE ELECCIÓN

Las siguientes definiciones conciernen sólo a la estructura de red y no afectan a la semántica de la red. (Definiciones acordes con [14])

Redes Extendidas Libre Elección (EFC)

Un Sistema de Redes de Petri ordinario $\sum=(S, T; F, M_o)$ es un Sistema Extendido Libre Elección sii

$\forall t\ t' \in T: {}^\bullet t \cap {}^\bullet t' \neq \varnothing \Rightarrow {}^\bullet t = {}^\bullet t'$

equivalentemente: $\forall s,\ s' \in S: s^\bullet \cap s'^\bullet \neq \varnothing \Rightarrow s^\bullet = s'^\bullet$

Redes Comportamentales Libre Elección (BFC)

Un Sistema de Redes de Petri ordinario $\sum=(S, T; F, M_o)$ es un Sistema Comportamental Libre Elección (BFC) sii

$\forall t\ t' \in T: {}^\bullet t \cap {}^\bullet t' \neq \varnothing \Rightarrow \forall M \in [M_o\rangle M$ permite $t \Leftrightarrow M$ permite t'

Redes Asimétrica Elección (AC)

Un sistema de Redes de Petri $\sum=(S, T; F, M_o)$ es un Sistema Asimétrica Elección sii

$\forall s, s' \in S: s^\bullet \cap s'^\bullet \neq \varnothing \Rightarrow (s^\bullet \subseteq s'^\bullet) \vee (s'^\bullet \subseteq s^\bullet)$

Redes Comportamentales Asimétrica Elección (BAC)

Un Sistema de Redes de Petri ordinario $\sum=(S, T; F, M_o)$ es un Sistema Comportamental Asimétrica Elección sii $\forall t,\ t' \in T$:

$({}^\bullet t)^\bullet \cap ({}^\bullet t')^\bullet = \varnothing \ \vee\ (\forall M \in [M_o\rangle$: M permite $t \Rightarrow$ M permite $t')$

$\vee\ (\forall M \in [M_o\rangle$: M permite $t' \Rightarrow$ M permite $t)$

Redes Reducidas Asimétrica Elección (RAC)

Un Sistema Asimétrica Elección es denominado Reducida Asimétrica Elección (RAC) sii

$\forall p, q \in S:\ p^\bullet \cap q^\bullet = \varnothing \ \vee\ (|p^\bullet| = 1 \ \wedge\ |q^\bullet| \leq 2 \ \wedge\ {}^\bullet(q^\bullet) = \{p,q\})$

$\vee\ (|q^\bullet| = 1 \ \wedge\ |p^\bullet| \leq 2 \ \wedge\ {}^\bullet(p^\bullet) = \{p, q\})$

La propiedad RAC permite toda estructura libre elección que no tenga más de dos lugares de salida por cada transición y sólo un tipo de estructura no-FC, concretamente la más simple estructura AC: Cada sistema BAC puede ser simulado por un sistema AC y cada sistema AC puede ser simulado por un sistema RAC.

Referencias: [5]; [14]

REDES DE PETRI DE ALTAS PRESTACIONES + ADT

- Las Redes de Petri de Altas Prestaciones + ADT son redes de nivel 3, que son definidas sobre una especificación algebraica.
- Las Redes de Petri Algebraicas (ALG) son Redes de Petri de Altas Prestaciones, que consisten en un esquema de red y un álgebra SPEC.
- Las Redes OBJSA consisten en un esquema de red descomponible y una especificación algebraica OBJ.

REDES ALGEBRAICAS

Esquemas de Redes Algebraicas

Un esquema de red algebraica es un par consistente en una especificación algebraica y una red con lugares tipificados, con transiciones que tiene predicado (un conjunto de ecuaciones) y con arcos marcados con términos sobre la especificación.

Un esquema de red algebraica consiste en una red coloreada básica donde colores y reglas de disparo son representados por la interpretación de términos sobre las especificaciones algebraicas dadas.

Definición

(de acuerdo con [16])

Un esquema de red algebraica es un óctuplo N=(SPEC,X,P,T,Pre,Post,eqns,sort) donde:

1. SPEC=(SO,OP,EQ) es una especificación algebraica de tipos de datos abstractos
2. X es una familia de variables tipo SO
3. P y T son conjuntos de lugares y transiciones
4. Pre y Post son funciones que asignan a un conjunto de términos con variables para cada par lugar/transición
5. eqns es una función que asocia con cada transición un conjunto de ecuaciones
6. sort es una función que asigna a cada lugar un tipo SO

Redes Algebraicas de Altas Prestaciones

Redes Algebraicas de altas prestaciones son parejas (esquema de red algebraica, álgebra).

Definición

Una red algebraica de altas prestaciones (AHL net) es un duplo (N,A) donde N es un esquema de red algebraica y A es un álgebra SPEC.

Regla de Transición

La semántica de una red algebraica de altas prestaciones es aquella de redes coloreadas asociadas donde los términos y las ecuaciones son interpretados en el álgebra dada.

Herramientas: PAPETRI, SANDS-COOPN**.**

Referencias: [17]; [18]; [19]; [16]

REDES DE PETRI COLOREADAS (CPN)

Definición

(de acuerdo con [20])

Una Red de Petri Coloreada (CPN) es un nueve-uplo CPN = ($\sum$, P, T, A, N, C, G, E, I) donde

1) $\sum$ es un conjunto finito de tipos llamados Lugares coloreados que son finitos y no vacios. El conjunto de conjuntos coloreados determinan los tipos, operaciones y funciones que pueden ser usados en las inscripciones de red.
2) P es un conjunto finito de lugares
3) T es un conjunto finito de transiciones
4) A es un conjunto finito de arcos tal que $P \cap T = P \cap A = T \cap A = \varnothing$
5) $N:A \rightarrow PxT \cup TxP$ es una función nodo
6) $C:P \rightarrow \sum$ es una función color
7) G es una función guarda. Es definida desde T en expresiones tal que:
   $\forall t \in T:[Type(G(t)) = Bool \ \wedge \ Type\ (Var(G(t))) \subseteq \sum]$

8) E es una función expresión de arco. Es definida desde A en expresiones tal que
$\forall a \in A:[Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$
donde p(a) es el lugar de N(a) y C_{MS} denota el conjunto de todos los conjuntos múltiples sobre C
9) I es una función inicialización. Está definida desde P en expresiones tal que
$\forall p \in P:[Type(I(p)) = C(p)_{MS} \wedge Var(I(p)) = \varnothing]$

Regla de Transición

Un conjunto Y está permitido en un marcado M sii
$\forall p \in P:[\sum_{(t,b) \in Y} E(p,t)(b\rangle \leq M(p)]$
donde un conjunto Y es una distribución no vacía: $\forall t \in T: Y(t) \in B(t)_{MS}$
donde $B(t) = \{(c_1.....c_n) \in BT(t) \mid G(t)\langle v_1=c_1.....v_n=c_n \rangle$
donde BT(t) es el tipo binding: $BT(t)=Type(v_1)xType(v2)x.....xType(v_n)$ para una transición $t \in T$ con variables $v_1, v_2, ...v_n$

Herramientas

Alpha/Sim, ANARCO, CPN/AMI, DESIGN/CPN, DESIGN/OA, EXSPECT, FORSEE, INA, LOOPN, PAPETRI, POSES, PROMPT, PNTBLSIM, TemPRO, VisualSIMNET, VOLTAIRE.

References: [21]; [20]; Coloured Petri Nets at University of Aarhus

SISTEMAS REDES OBJSA

Los Sistemas de Redes OBJSA son Redes de Petri de Altas Prestaciones en las cuales 1) la red puede descomponerse en Componentes Máquinas de Estados y
2) los dominios a los que pertenecen las marcas individuales son definidos como tipos de datos abstractos usados en el lenguaje OBJ 2.

Definición

(de acuerdo con [22])
Un Sistema de Redes OBJSA es una pareja C=(N, A) donde N es una Red extendida SA y A es una especificación algebraica OBJ.
Hay las siguientes relaciones entre N y A:

1. cada subred elemental de N es asociada con un objeto OBJ de A, que especifica el dominio de marcado de la subred
2. cada transición está asociada con un módulo OBJ que introduce:
   - variables asociadas con los arcos de entrada
   - un predicado que especifica una condición adicional para la transición permitida
   - operadores asociados con los arcos de salida que especifican los cambios de los datos del marcado
3. el marcado inicial consiste en una suma de términos algebraicos (término base) que han sido definidos en 1

Regla de Transición

Una transición está permitida cuando:
(i) los lugares de entrada contienen al menos tantas marcas como las especificadas por el etiquetado de los arcos de las transiciones de entrada; y
(ii) existe una relación de esos marcados con las variables libres ocurridas en los arcos de las transiciones de entrada tal que el predicado inscrito en la transición es satisfecho.

Este suceso remueve los marcados envolventes de los lugares de entrada y suma a cada lugar de salida los marcados obtenidos de acuerdo con los etiquetados de los arcos de las transiciones de salida, bajo la misma asociación

Herramientas: ONE.

Referencias: [23]; [22]

REDES ER

Redes Entorno/Relación (ER) son redes de Petri de altas prestaciones donde los marcados son desarrollados, i.e. funciones asociando valores a variables.

Definición

(de acuerdo con [24])

Una Red ER es una Red de Petri donde

1) Los marcados son desarrollados en ID (conjunto de identificadores) y V (conjunto de valores), i.e. (posibilidad vacía) funciones: $ID \rightarrow V$

Deja $ENV = V^{ID}$ ser el conjunto de todo entorno .

2) Cada transición está asociada con una acción. Una acción es una relación $\alpha(t) \subseteq ENV^{K(t)} \times ENV^{h(t)}$ donde $K(t) = |\bullet t|$ y $h(t) = |t\bullet|$

La proyección de $\alpha(t)$ en $ENV^{K(t)}$ es denotada por $\Pi(t)$ y es llamada el predicado de la transición t.

3) Un marcado m es una asignación de conjuntos múltiples de entornos con lugares.

Regla de transición

- Una transición t está permitida en un marcado m sii

para cada lugar de entrada p_i de t existe al menos un marcado en v_i tal que

$(env_{1, ..., } env_{K(t)}\rangle \in \Pi(t)$

$(env_{1, ..., } env_{K(t)}\rangle$ es denominado el tuplo permitido para la transición t.

- Un disparo es un triple $x = (enab, t, prod\rangle$ tal que $(enab, prod\rangle \in \alpha(t)$

enab es denominada la tuple entrada, mientras *prod* es denominada la tuple salida.

- La ocurrencia del disparo $(enab, t, prod\rangle$ en el marcado m consiste en la producción de nuevos marcados m' para la red, obtenidos del marcado m por el movimiento permitido del tuple *enab* de los lugares de entrada de la transición t y almacenado del tuple *prod* en los lugares de salida de la transición t.
- Dando un marcado inicial $m_{0,}$ una secuencia de disparos *s* es una secuencia finita $(enab_{1,1,,}prod_1\rangle,.....,\langle enab_{n,n,}prod_n\rangle)$ tal que la transición t_1 es permitida en el marcado m_0 por el tuple $enab_1$, cada transición $t_{i,}$ $2 \leq i \leq n$ está permitida en el marcado m_{i-l} producido por el disparo $(enab_{i-l,\ i-l,}\ prod_{\ i-l}\rangle$ y su disparo produce el marcado $m_{i.}$

Herramientas: CABERNET.

Referencias: [24]

REDES PRODUCTO

Las redes Producto resultan del aumento de las Redes de Petri con los arcos inhibidores y los arcos de borrado con las etiquetas de arco, inscripciones de transiciones y marcados individuales.

Definición

Una Red Producto consiste en

1. un preámbulo en el que los conjuntos usados y las funciones son definidos
2. una red sin etiquetado con un conjunto de lugares S, un conjunto de transiciones T y un conjunto de arcos comprendiendo arcos inhibidores y arcos de borrado
3. un etiquetado con
    - etiquetas de arco para cada arco, que son n-tuples de términos, las cuales pueden ser precedidas por multiplicidades
    - inscripciones de transición (opcional), que son predicados en las variables destino de la transición
    - dominios de lugares (define el marcado que puede ser presentado en los lugares), que son producto de los conjuntos o subconjuntos de productos

Marcado

Un Marcado M de las Redes Productos es una familia de mapeados, cada uno colocando (uno o más) elementos específicos del dominio apropiado a cada lugar de la red.

Interpretación

Una interpretación es la asignación a cada variable x del valor del rango especificado de valores.

Una interpretación permisible de las variables destino de la transición t es una interpretación que satisface:

1. A las variables destino se les asigna sólo números naturales o esos valores que son elementos del conjunto definido en el preámbulo.
2. Las interpretaciones de las variables de destino de la transición t son extensibles a todos los términos que aparecen en las etiquetas de los arcos de entrada y salida y, posiblemente, en las inscripciones de las transiciones.
3. La extensión de la interpretación sobre las etiquetas de los arcos de entrada y salida debe resultar en el marcado acorde con el dominio de los lugares respectivos

Cada interpretación permisible de las variables de destino de las transiciones son asignadas a un marcado umbral que es definido a través de la extensión de la interpretación en las etiquetas de los arcos de entrada.

Regla de Transición

Una transición t está permitida bajo un marcado M y es una interpretación permisible de una variable destino si sostiene lo siguiente:

- En los lugares de entrada de t debe haber al menos los marcados del marcado umbral.
- La inscripción de la transición –si presenta- debe ser verdad bajo la interpretación dada.
- Ningún marcado del respectivo conjunto inhibidor puede residir en el lugar inhibidor de t.

Si ocurre la transición t genera el marcado sucesor que puede ser descrito en tres pasos:

1. El marcado de los lugares de entrada está reducido al marcado que es descrito por la extensión de la interpretación de t sobre las etiquetas de sus respectivos arcos de entrada; i.e., el marcado es decrementado por el marcado umbral.

2. Todos los marcados que son elementos de los respectivos arcos de borrado, están sacados de los lugares de borrado.
3. El marcado de los lugares de salida está incrementado por el marcado que es descrito por la extensión de la interpretación de t sobre los respectivos arcos de salida.

Herramientas: Product Net Machine.
Referencias: [25]

REDES BIEN FORMADAS

Redes Bien Formadas (Coloreadas)(WN) son formalmente definidas como una extensión de las Redes Regulares y tienen el mismo poder de modelado como las Redes de Petri Coloreadas generales, i.e., algún CPN puede traducirse en un modelo equivalente WN con la misma estructura subyacente; sólo la expresión de las funciones de color y de la composición de las clases de colores es reescrita de un modo más explícito (y paramétrico), en términos de las construcciones básicas provistas por los formalismos de WN.

Definición

(de acuerdo con [26])

Una red bien formada (coloreada) WN = (P, T, C, J, W^-, W^+, W^h, Φ,Π,M_O) está hecha de

1) P: un conjunto de lugares
2) T: un conjunto de transiciones
3) C: la familia de clases básicas de desunión C = {$C_{1,.....}C_n$}, = I{1,....n} es el conjunto ordenado de índices
4) J:P$\cup\rightarrow$Bag(I), donde Bag (I) es el multiconjunto en I, C(r) = $C_{J(r)}$ denota el dominio del color del nodo r
5) W^-,W^+,W^h,W^-, (p,t), W^+ (p,t), W^h (p,t)$\in$[$C_{J(t)} \rightarrow$Bag($C_{J(p)}$)] la entrada, salida y funciones inhibidoras son expresiones de arco;
6) Φ(t):$C_{J(t)} \rightarrow${verdad, falso} es un predicado estándar asociado con la transición t; por defecto $\forall$t$\in$T:Φ(t) = verdad
7) Π:T $\rightarrow$ N la función prioritaria; por defecto $\forall$t$\in$T:Π(t) = 0
8) M_0:M_0 (p)$\in$ Bag(C(p)) es el marcado inicial de p

Regla de Transición

Una instancia de transición [t,c] (donde c $\in$ C(t)) es posible en un marcado M sii

1) $\forall$p$\in$ P, W^-(p,t)(c) $\leq$ M(p) $\wedge$ W^h(p,t) >M(p), y Φ (t)(c), que es la expresión de la regla de disparo en las redes sin prioridades,
2) $\forall$t′ con $\prod$(t′) > $\prod$(t),$\forall$c′$\in$ C(t′,)$\exists$p$\in$ P tal que cada

W^-(p,t′) (c′) > M(p) $\vee$ W^h (p,t)(c) $\leq$M(p) or $\neg$ Φ (t′)(c′)

que significa que la transición prioritaria mayor es posible.

El disparo de los casos de transición [t,c] trae un nuevo marcado M′= M[t,c$\rangle$

Definido por: $\forall$p$\in$ P, M′(p) = M(p) + W^+(p,t)(c) – W^-(p,t) (c)

Referencias: [27]; [28]; [26]

REDES REGULARES

Una Red Regular (RN) es una Red de Petri Coloreada en la cual los dominios de color de los lugares y transiciones son hechos por un Producto Cartesiano de clases de objetos básicos; cada clase aparece no más que una vez en el producto.

Definición

Una Red Regular RN={P, T, C, I^-, I^+, $m_0\rangle$ es definida por:

- P el conjunto de lugares
- T el conjunto de transiciones
- C el conjunto de clases de objetos: C={$C_{1,.....,}C_n$}, con $C_i \cap C_J = \varnothing$
- I^- y I^+ las matrices de entrada y salida definidas en P x T, cuyos elementos son funciones coloreadas estándar definidas debajo.
- m_0 el marcado inicial, que debe ser simétrico, i.e. debe verificar la siguiente propiedad:

$\forall p \in P, \forall c, c' \in C(p), m_0(p,c) = m_0(p,c')$

donde m (p,c) denota el número de marcas de color c en p.

Una función estándar de color de una Red Regular $\prod^n_{i=1} = \langle a_i.S_{i+} b_i .X_i\rangle : CxC \rightarrow N$ es definida por: $\prod^n_{i=1} = \langle a_i.S_{i+} b_i .X_i\rangle((c_1',....., c_n').(c_1,....., c_n)) = \prod^n_{i=1} = \langle a_i.S_{i+} b_i .X_i\rangle(c'_i .(c_1,.....,c_n))$

con $X_i:C_i$ xC $\rightarrow$N tal que $X_i(c'_i.(c_1,...,c_n))$ = Id (c_i,c'_i) un etiquetado de arco X_i distinguiendo exactamente un objeto de clase C_i

S_i: C_i x C $\rightarrow$N tal que $S_i(c'_i.(c_1,...,c_n))$ =1 un etiquetado de arco S_i significa que todo objeto de clase C_i juega una parte similar

Las dos funciones están combinadas por:

$a_i.S_i + b_i.X_i : C_i$xC $\rightarrow$N tal que

$a_i.S_i + b_i.X_i(c'_i.(c_1,...,c_n))$ = (if $c_i = c'_i$ then (a_i+b_i) else a_i)

Referencias: Una categoría regular de Redes de Petri de otro nivel: Definición, propiedades y reducciones.

REDES DE PETRI TRADICIONALES DE ALTAS PRESTACIONES

Los dos modelos de Redes Predicado/Transición y Redes de Petri Coloreadas son muy similares y las principales diferencias conciernen a los métodos de cálculo e interpretación, lugares y transiciones invariantes.

Sin embargo están definidos y presentados de dos formas distintas:

- Las Redes Predicado/Transición son definidas usando la notación y conceptos de varias clases de álgebras y
- Las Redes de Petri Coloreadas están definidas usando tipos, variables y expresiones – tal y como conocemos en lenguajes de programación (funcionales).

Referencias: [29]

REDES DE PETRI CONTINUAS E HÍBRIDAS

Son redes en las que los marcados dejan de ser discretos y pueden tomar valores continuos.

Redes de Petri continuas

Definición

Según [30], una red de Petri continua autónoma es una quíntupla Q=<*P*,*T*,*Pre*,*Post*,M_0> tal que:

P={P1, P2, ..., Pn}es un conjunto finito y no vacío de lugares.

T={T1, T2, ..., Tm}es un conjunto finito y no vacío de transiciones.

$P \cap T = \varnothing$, es decir, P y T son conjuntos disjuntos.

Pre:P x T $\rightarrow R^+$ es la función de incidencia previa.

Post:P x T $\rightarrow R^+$ es la función de incidencia posterior.

M0: P$\rightarrow R^+$ es el marcado inicial.

En una RdP continua, a partir de un marcado M, una secuencia de disparos S implica una trayectoria correspondiente a un vector de sucesivos marcados. El vector característico $\underline{S}$ de una trayectoria es un vector para el cual cada componente es un número real correspondiente a la cantidad de disparo de la correspondiente transición. Entonces un marcado M alcanzado desde M_0 por el disparo de una secuencia S puede ser deducido usando la ecuación fundamental (o ecuación de estado): $M=M_0 + W \cdot S$. La ecuación fundamental de una red de Petri continua es idéntica a la de una discreta, por lo que las propiedades de las RdP discretas deducidas a partir de dicha ecuación son también válidas para las RdP continuas. En particular los resultados de P-invariantes y T-invariantes son similares para una RdP continua y para una discreta.

Redes de Petri híbridas

Definición

Según [30], una red de Petri híbrida autónoma es una sextupla Q=<*P*,*T*,*Pre*,*Post*,M_0,*h*> tal que:

P={P1, P2, ..., Pn}es un conjunto finito y no vacío de lugares.

T={T1, T2, ..., Tm}es un conjunto finito y no vacío de transiciones.

$P \cap T = \varnothing$, es decir, P y T son conjuntos disjuntos.

h:$P \cup T \rightarrow$ {D,C}, llamada función híbrida, indica para cada nodo si es continuo o discreto.

Pre:P x T $\rightarrow R^+$ ó N es la función de incidencia previa.

Post:P x T $\rightarrow R^+$ ó N es la función de incidencia posterior.

M0: P$\rightarrow R^+$ es el marcado inicial.

(en las definiciones de Pre, Post y M_0, N corresponde al caso en que el nodo es discreto, y R^+ al caso en que es continuo)

Las funciones Pre y Post deben cumplir el siguiente criterio: si P_i y T_j son un lugar y una transición tal que Pi es discreta y Ti es continua, entonces deben cumplir que Pre(Pi,Tj)=Post(Tj,Pi).

En una RdP híbrida el vector característico $\underline{S}$ de una secuencia S es un vector para el cual cada componente es o bien un entero correspondiente al número de disparo de una transición discreta o bien un número real correspondiente a una cantidad de disparos de una transición continua. Un marcado M puede ser deducido a partir de un marcado M_0 debido a una secuencia S usando la ecuación fundamental: $M=M_0 + W \cdot S$. Por tanto de nuevo los resultados de T-invariantes y P-invariantes son equivalentes a las RdP discretas.

1.5.3. Herramientas

A continuación veremos una tabla con el resumen de las principales herramientas que se pueden encontrar para el tratamiento de las RdP así como un resumen de sus características principales. De nuevo no se pretende clasificar las herramientas existentes, ni siquiera enumerar todas las existentes. El objetivo vuelve a ser proporcionar una visión global de algunas de las herramientas disponibles y de sus características.

Herra-mienta	Descripción	Plataforma (entorno)	Tipo de red soportada	Funcionalidades	Notas/ Disponibilidad
ALPHA/Sim	ALPHA/Sim es una herramienta de simulación de eventos discretos de propósito general basada en el paradigma de las Redes de Petri	Estaciones de trabajo con XWindows, OpenWindows, o Motif	Estocástica, Temporizada, Red de Petri coloreada con colas, probabilidades y prioridades	EDITOR gráfico SIMULACIÓN y cálculo de simulaciones estocásticas Conexión con software externo	POLÍTICA: Precio especial para Universidades
ANARCO	ANARCO (ANAlisador de Redes de Petri COloreadas) es una herramienta para el análisis de Redes de Petri y Redes de Petri Coloreadas.	PC/DOS	Redes de Petri y Redes de Petri Coloreadas	Construcción del árbol de alcanzabilidad ANÁLISIS estructural: Viveza, Limitación, Lugares Invariantes, Transiciones Invariantes (sólo para Redes Ordinarias) Análisis Reducido (sólo para Redes Ordinarias) Algoritmo de Enumeración	La presente versión, v. 3.00, está disponible en dos "tipos". Modo Real y Modo DPMI. Modo Real está limitado a 640 K de DOS RAM. Modo DPMI puede usarse en todo RAM disponible. POLÍTICA: Gratis para propósitos no comerciales
CABERNET	CABERNET es un software del entorno de la ingeniería para la especificación y análisis en sistemas de tiempo real basados en Redes de Petri para datos, predicados, acciones, e información temporal.	Unix (requiere Compilador GNU C++)	Redes Temporizadas ER Jerárquicas	EDITOR gráfico SIMULACIÓN: ejecutable y animación ANÁLISIS: computación del árbol T-alcanzable para comprobar propiedades como la viveza y la seguridad para intervalos de tiempo dados (limitación invariante, limitación de respuesta) GENERACIÓN DE HERRAMIENTAS: Las funcionalidades básicas que proporciona esta herramienta pueden componerse de distintas maneras usando el lenguaje base de Redes de Petri para obtener nuevas herramientas.	La herramienta MERLOT está integrada en CABERNET y en soportes de Redes TB, una subclase de Redes ER Temporizadas. POLÍTICA: disponible a través de ftp anónimos
CAPSNET	REDES C.A.P.'s (Computer Aided Petri NET Simulator) es una herramienta de simulación para Redes de Petri.	MS-DOS	Redes Lugar/Transición	EDITOR gráfico SIMULACIÓN automática	POLÍTICA: herramienta disponible a través de ftp anónimos
COMBAG		Unix	Redes Predicado/Transición	EDITOR gráfico desde la herramienta PetriPote Descripción textual de redes Computación de invariantes lineales	COMBAG ha sido integrada en la estructura del software SERPE.
CPN/AMI	CPN/AMI es una versión de AMI, un conjunto de herramienta interactiva para creación, manipulación, simulación, transformación, análisis y comprobado de gráficas tipo con atributos.	Macintosh para el uso de interface (MACAO) conetado con una estación Sun Sparc para plataforma AMI	Redes AMI incluyendo CPN y Redes P/T	EDITOR gráfico (MACAO) ANIMACIÓN delante y detrás con condiciones de puntos rotos y extracción de datos Comprobador de sintaxis Compilador ANÁLISIS: Para Redes P/T: propiedades estructurales (limitaciones, limitaciones estructurales, conservación, repetitividad...), propiedades gráficas (cerrojos y trampas), Invariantes, Reducciones, prototipo ADA para CPN: Invariantes, Reducciones y Prototipos, la herramienta PROD para la construcción de la gráfica de alcanzabilidad ha sido integrada MACAO: interface de usuario para edición de gráficas y display de servicios	POLÍTICA: libre de cargo para universidades, disponible a través de ftp anónimos

				SANDRINE: Compilador de Redes de Petri para acceder a servicios AMI-CPN RIP (Research of Invariants and Properties for P/T Nets) MIAMI: sistema experto para el análisis y validación de propiedades combinatorias de gráficos MAPLE: análisis interactivo y verificación de propiedades de formalismos de Redes de Petri CPN/FARM (Flow Analysis and Reduction Method) CPN/DESIR (DEbugger and SImulatoR) CPN/TAGADA (Translation, Analysis and Generation of ADA code)	
DESIGN/CPN	DESIGN/CPN es un paquete de herramientas de soporte de modelado, simulación y verificación para la interpretación de Redes de Petri Coloreadas jerárquicas (con o sin tiempo).	SUN Sparc con Solaris, MAC	Redes de Petri Coloreadas Jerárquicas con o sin tiempo	La herramienta consiste en tres partes integradas: EDITOR gráfico soporte de construcción, modificación y chequeo de sintaxis de modelos DPN interactivos y simulador automático Herramienta de suceso grafica soporte de construcción y análisis de gráficas de sucesos	Las rutinas de DESIGN/OA para la manipulación de objetos gráficos están disponibles para el usuario vía un interface SML. POLÍTICA: Libre de cargo- El formulario de licencia puede ser descargado de Desing/CPN.www
DESIGN/OA	Las rutinas de DESIGN/OA para la manipulación de objetos gráficos está disponible para el usuario vía un interface SML.	MAC, MS-DOS, Unix	Redes de Petri (y todo otro tipo de gráficos directos/indirectos)	Las aplicaciones creadas con DESIGN/OA tienen tres pomponentes: DESIGN/OA Kernel: contiene rutinas y estructuras de datos DESIGN/OA Interface: librería de funciones para la manipulación de objetos gráficos. Módulo de desarrollo: código escrito en C que hace llamadas al Interface	Los modelos construidos con DESIGN/CPN pueden ser animados con DESIGN/OA. POLÍTICA: Descuento del 50% para universidades.
EASE	EASE es un entorno para transformaciones y reducciones de redes basado en estados basados en equivalencias.	DECMicroVAX	Redes 1-Safe cubiertas con monomarcadas S-invariante de las cuales la Redes SA son una subclase.	EASE es un sistema basado en tres componentes: TEBE (Tools for Exhibited Behaviour Equivalence) implementa un sistema reescrito para Redes de Petri. Toma como entrada una red y produce una red REDUCIDA de la misma clase preservando el comportamiento equivalente exhibido. TSTE (Tools for State Transformation Equivalence) implementa la construción de sistemas de estado de espacio como álgebra de transformación local de estado (LSTA) desde la que la representación canónica de la clase equivalente conteniendo el sistema original es sintetizada. SDB (System Data Base) es un conjunto de funciones que chequean la estructura de la red de entrada frente a la definición de redes superpuestas de autómata 1-safe (SA), y permite la COMPOSICIÓN de ellas y la DESCOMPOSICIÓN en componentes (elementales) partiendo de redes SA.	POLÍTICA: Libre de cargo
EXSPECT	La herramienta EXecutable SPECification es un marco de trabajo para especificaciones formales ejecutables de sistemas distribuido basada en un lenguaje funcional.	SunOS o SOLARIS en SPARC o Intel x86	Redes de Petri Coloreadas jerárquicas con tiempo, retrasos de tiempo asociados con marcas. Cada marca tiene un valor y un tiempo: tiempo de sistema en su producción y retraso de tiempo (determinado	EDITOR gráfico y COMPROBADOR DE TIPOS SIMULACIÓN interactiva: interpretador de ejecutables IAT (Interval Timed Petri Net Analysis Tool): calcula intervalos S y T y otras propiedades estructurales y de tiempo.	POLÍTICA: licencia para un solo usuario

			por la transición) denotando el menor tiempo que puede consumir.		
FORSEE	FORSEE (FORmal Systems Engineering Environment) contiene una colección de herramientas que provee de asistencia base computerizada en el comportamiento de sistemas concurrentes.	Estación de Trabajo Sun con Unix	Redes de Petri Coloreadas, Redes Lugar/Transición.	El entorno consiste en tres componentes: Paquete de Editado y Simulación Design/CPN Herramientas de verificación TORAS Herramienta de implementación automática PROMPT	POLÍTICA: Existe solo un prototipo de FORSEE
GRAF	GRAF es una etapa general e integrada para la edición y simulación de Redes de Petri jerárquicas.	MS-DOS	Redes Lugar/Transición con arcos inhibidores y prioridades	Editor gráfico para creación y manipulación de redes interactivas SIMULACIÓN Interactiva/automática ANÁLISIS estructural basado en la comprobación de la sintaxis Características jerárquicas	POLÍTICA: en la solicitud
HYPERNET CE	Hypernet es una aplicación, realizada con MetaDesign, que permite el modelado y el análisis de sistemas complejos.	Macintosh	Redes Condición/Evento	EDITOR gráfico SIMULACIÓN ANÁLISIS (Árbol e Invariantes alcanzabilidades) para la verificación de propiedades de red como pureza, simplicidad, viveza, seguridad La descripción gráfica de redes puede transformarse en un prólogo de clausulas y exportarse a otras aplicaciones	POLÍTICA: en la solicitud
INA	INA (Integrated Net Analyser) es una herramienta para el análisis de Redes de Petri.	MS-DOS, Sun-Unix	Redes Lugar/Transición (P/T) y Redes de Petri coloreadas (CPN) con tiempo y prioridades.	EDITOR textual Del mismo modo simulación ANÁLISIS: Análisis de propiedades estructurales como viveza, seguridad, cubrimiento, conservacionismo, cerrojos y trampas, chequeos de la descomposición de máquinas de estado si la red es una Red de Petri Ordinaria, Libre Elección, Máquina de Estado o un Grafo Marcado Computación de la gráfica de alcanzabilidad Reduciones simétricas Desarrollo de Redes de Petri Coloreadas	INA es un sucesor de las siguientes herramientas: Petri Net Machine (1986), PAN (1988), CPNA (1989), ATNA (1991). POLÍTICA: Disco DEMO para MS-DOS disponible a través de ftp anónimos
INCOME	INCOME se crea como un método versatil y una serie de herramientas de orientación al usuario, para modelado, simulación e implementación de procesos empresariales.	MS-DOS, MS Windows, Sun (Solaris), HP9000, IBM RS/6000; requiere Oracle	Redes Predicado/Transición	EDITOR gráfico SIMULACIÓN DE ANIMACIÓN GRÁFICA Modelos creados con INCOME pueden ser reusados en proyectos CASO (Oracle CASE and Oracle Designer/2000) El DICCIONARIO sirve como una reposición para toda información del proyecto. Está implementado como una base de datos Oracle7 y soportes distribuidos de etapas como Cliente/Servidor o Cliente/Agente/Servidor. El MODELADO DE PROCESOS es usado para la captura, documentación y evaluación de procesos empresariales.	POLÍTICA: Condiciones especiales para instituciones sin beneficios

				SIMULADOR: Las más importantes componentes del Simulador son: el KERNEL que realiza simulaciones de los modelos, el ANIMADOR que genera visualizaciones orientadas al usuario del comportamiento dinámico; y el ANALIZADOR que puede evaluar simulaciones de diferentes formas. INCOME MOBILE proporciona el uso con toda la funcionalidad del modelado del Proceso de Modelado pero no necesita la base de datos Oracle7.	
LOOPN	LOOPN (Language for Object-Oriented Petri Nets) es un lenguaje y un simulador para sistemas especificados en términos de Redes de Petri Temporizadas Coloreadas. Incluye rasgos de objetos orientados como la ineherencia, polimorfismo, que permiten la conveniente modularización de especificaciones complejas.	Unix	Redes de Petri Objeto jerárquicas	Simulación Generación de código C Características de Objetos orientados	POLÍTICA: Herramienta disponible a través de ftp anónimos
MERLOT	MERLOT es un conjunto de herramientas que implementa el algoritmo de análisis de la alcanzabilidad basado en el Arbol de Alcanzabilidad de Tiempo.	Unix	Redes TB	EDITOR gráfico (desde CABERNET) ANÁLISIS: computación del Arbol de alcanzabilidad de tiempo para el verificado de propiedades como viveza y seguridad, para intervalos de tiempo dados Tratamiento JERÁRQUICO (desde CABERNET)	MERLOT está integrado en CABERNET
METADESIGN	MetaDesign es un editor de Redes de Petri y un simulador aumentado con funciones para la solución de situaciones ambiguas.	Macintosh	Redes de Petri con extensiones	EDITOR gráfico SIMULACIÓN El análisis está hecho con herramientas diseñadas con MetaDesign, como HYPERNET CE o NETOBJ	MetaDesign tiene un lenguaje de programación, Desgin/OA POLÍTICA: 50% de descuento para Universidades
MOBY	MOBY es una heramienta orientada al objeto del análisis de Redes de Petri basado en Smalltalk 80.	MS-DOS, Unix, OS2, Macintosh (requiere Smalltalk 80)	Redes Condición/Evento, Lugar/Transición y Objeto (Redes temporizadas jerárquicas de altas prestaciones con	EDITOR gráfico SIMULADOR paso a paso y automático ANÁLISIS de las siguientes propiedades estructurales: viveza, seguridad, alcanzabilidad, cobertura e invariantes	POLÍTICA: herramienta disponible a través de ftp anónimos

			objetos como marcas, incl. tipos de arco).		
NETMAN	NETMAN sirve como una herramienta de edición que exporta a otros paquetes para el análisis y la simulación.	MS-DOS / MS-Windows 3.1	Redes de Petri Transición-Temporizada jerárquica (determinista y exponencial)	EDITOR gráfico Provisto de exportador funcional para el uso con otras herramientas	Las siguientes herramientas soportan actualmente la exportación a: XPN, SPNP, SimNet, POSES POLÍTICA: herramienta disponible a través de ftp anónimos
NETOBJ	NETOBJ (NET OBJect) es una herramienta para la edición y el análisis de las Redes de Petri Comunicadas, realizada en la base de HYPERNET.	Macintosh	Redes de Petri Comunicadas.	EDITOR gráfico con comprobadores de sintaxis SIMULACIÓN ANÁLISIS (durante la simulación) de propiedades, como la viveza, seguridad, alcanzabilidad	POLÍTICA: en la solicitud
ONE	ONE (OBJSA Nets Environment) consiste en diferentes módulos, cada uno soportando un paso diferente en el desarrollo incremental de la especificación de la red.	Sun4 (requiere DesignML y OBJ3 Interpreter)	Redes OBJSA	Módulos de la herramienta: ONESyst permite al usuario dar una especificación alta-baja del sistema. ONEGen sostiene la producción de las especificaciones ejecutables OBJSA proporcionando un editor gráfico de red y un soporte para el entorno de las especificaciones del marcado algebraico. ONERed soporta la Red transformación NET, que transfiere la información contenidad en la componente OBJSA(N,SPEC) desde la red N hasta la especificación algebraica asociada SPEC, preservando su semántica. ONEComp sostiene el mecanismo de composición OBJSA ONESim simula la especificación usando ecuaciones OBJ como reglas de escritura. ONEUnf realiza el desarrollo de una red OBJSA en una red 1-safe SA. ONEInv calcula las invariantes S/T del desarrollo. ONEVer genera la gráfica caso del desarrollo. ONEGSPN convierte Redes OBJSA en Redes de Petri Estocásticas Generalizadas.	POLÍTICA: gratis
PAPETRI	PAPETRI (Poste pour l'Analyse des réseaux de PETRI) es un desarrollo general e integrado para la edición y análisis de Redes de Petri.	Unix (requiere OpenLook)	Sistemas Lugar /Transición, Redes de Petri ordinarias, Redes de Petri Coloreadas, Redes Algebraicas	EDITOR interactivo gráfico (PETRIX) SIMULADOR gráfico Construcción de la gráfica de alcanzabilidad y cobertura que comprueba algunas propiedades de red (viveza, terminación...) y construcción de la gráfica de cobertura mínima Generación de Invariantes S y T (COMBAG) Análisis usando técnicas de reescritura (PETRIREVE): verifica la terminación de la Red de Petri para una clase inicial de marcados. Análisis y Simulación de Redes Algebraicas (VAERA) Composición/Descomposición de Redes Reducción de Redes	POLÍTICA: gratis para universidades a través de ftp anónimos
PENECA PNC	PENECA es una Herramienta de Simulación de Redes de Petri para el aprendizaje	MS-DOS	Redes de Petri P/T jerárquicas con condiciones, acciones, temporizadas y	EDITOR gráfico SIMULACIÓN: animación gráfica Generación de código C	PENACA tiene un interface para INA POLÍTICA: 60% de descuento para Universidades

	de estudiantes tanto como para el desarrollo de procesos tecnológicos, sistemas y dispositivos controlados, programación en tiempo real y diseño de hardware.		transiciones estocásticas.		
PEP	En PEP (Programming Environment based on Petri nets) se combinan el álgebra de procesos y Redes de Petri para modelar, simular, analizar y verificar sistemas paralelos.	Solaris 2.3 o 2.4, SUN OS 4.1.3, Linux	Redes P/T y Redes M (redes de altas prestaciones)	EDITOR gráfico SIMULACIÓN Verificación de propiedades en términos de fórmula lógica temporal El modelado puede realizarse con lenguaje de programación B(PN)^2 PEP consiste en cinco diferentes tipos de componentes: Editores para programas B(PN)^2 (lenguaje de programación de altas prestaciones), programas PBC (un álgebra de procesos llamada Petri Box Calculus), Redes de Petri, Simuladores de fórmulas y documentación de proyectos para Redes de Petri Algoritmos standard (como libre elección, sistema T, viveza, alcanzabilidad, reversibilidad) Algoritmos de comprobación de modelo para sistemas T y para Redes de Petri 1-safe, que pueden determinar si una Redes de Petri satisface la propiedad dada en términos de una fórmula lógica temporal	POLÍTICA: disponible sin cargo para usuarios no comerciales vía ftp para Solaris, para Sun OS y para Linux
PESIM		MS-DOS / MS-Windows	Redes Lugar/Transición con arcos inhibidores, Grafos Marcados, Redes de Petri Estocásticas	EDITOR gráfico y de texto SIMULACIÓN paso a paso ANÁLISIS: Construción del Árbol de Alcanzabilidad (para verificar viveza, seguridad, ...) Invariantes S y T Técnica de Reducción para Grafos Marcados Análisis de prestaciones de estado (probabilidades de estado, función de densidad de probabilidad de marcados para cada lugar, promedio de número de marcas en cada conjunto de lugares, ...)	POLÍTICA: herramienta disponible a través de ftp anónimos
PN^3-Editor	The PN^3-Editor es una Herramienta de Redes de Petri para la Especificación y Comunicación de Protocolos.	MS-DOS con MS Windows 3.x	Redes de Petri Jerárquicas	EDITOR BÁSICO DE REDES DE PETRI: editor multi-ventana con todos los rasgos necesarios para dibujar/editar y simular redes y sus estructuras jerárquicas EDITOR ALGEBRAICO: construcción de redes complejas usando operaciones de red (red atómica, composición secuencial, pre y postfix, elección, componentes paralelas, iteración, recursividad) NIVEL DE ARQUITECTURA: Este nivel es intentado para el diseño de sistemas complejos de alto nivel de composición y describe estructuras generales de sistemas, que pueden ser detalladas en niveles de red.	El Editor PN^3 es un sucesor del entorno de Redes de Petri composicionales. POLÍTICA: prototipo disponible con condiciones de respuesta especial para Universidades y Centros de Investigación
PNS - Petri Net Simulation	PNS es una Herramienta de dominio público para la Simulación de Redes de Petri.	Estación de trabajo Unix o PC con linux (requiere XWindows)	Redes Lugar /Transición Extendidas	EDITOR gráfico SIMULACIÓN/ANIMACIÓN con condiciones de puntos rotos	POLÍTICA: herramienta disponible a través de ftp anónimos
PNTBL-SIM	Petri Net TaBLe-based SIMulator	MS-DOS	Redes de Petri Coloreadas, Redes de Petri Temporizadas con	EDITOR de texto SIMULACIÓN interactiva o estadística ANÁLISIS de alcanzabilidad: árbol de cobertura	PNTBLSIM usa tablas de decisión como estructura de datos para la representación de

			arcos inhibidores		Redes de Petri POLÍTICA: gratis para usos académicos
POSES++	POSES++ es una simulación y desarrollo del modelado de modelos de redes extremadamente grandes (limitado sólo por el hardware) con reglas derivadas del lenguaje de programación C.	*Poses++ Server:* SUN Solaris 2.3 o superior, DEC OSF1 V3.0 o superior, viniendo pronto: Linux, Windows NT (quizás Windows95); más adelante: Compilador Gnu C++ (>=2.7.0), GNU make (>=3.74) *Poses++ IDE:* Windows3.1 o superior; más adelante: WINSOCK.DLL (e.j. Trumpet Winsock)	Redes de Petri Temporizadas C/E, P/T, CPN, Pr/T, Transición Temporizada/Arco y Redes de Petri Modulares, acceso por cola para predicados, expresiones booleanas y temporizadas para arcos, prioridades y paralelismo para transiciones	Modelado en lenguaje POSES++ (como C) Gráfica de ANIMACIÓN en línea o fuera de línea SIMULACIÓN remota de alta velocidad (sistema multicliente/multiservidor) Interface de módulos de software C La integración en otros sistemas de software (e.j. PPS) es posible Análisis estructural de estados de seguridad (contado de marcas, utilización,...)	POLÍTICA: herramienta disponible gratis bajo licencia a través de ftp anónimos
PROD TOOL	PROD es una herramienta del análisis de la alcanzabilidad para Redes Pr/T.	Unix, MS-DOS (requiere compilador y linkador C)	Redes Predicado /Transición	Las Redes son descritas en el lenguaje preprocesador C extendido con directivas de descripción de redes (prpp) ANÁLISIS: Generación de la gráfica de alcanzabilidad Depuración de Redes Pr/T a Redes P/T Un conjunto de Métodos para reducir la generación de estados de espacio (en el depurado) Programa de cuestiones gráficas (prueba) Computación de los componentes fuertes de conexión de la gráfica Lenguaje de cuestiones gráficas CTL (Computation Tree Logic) para la inspección de estados de espacio generados	Referencias disponibles en saturn.hut.fí POLÍTICA: libre de cargo a través de ftp anónimos para PC-para Unix
Product Net Machine	El Product Net Machine es una herramienta integrada para la especificación y análisis de Redes Producto, que puede modelar aplicaciones complejas de mundo real.	Symbolic -Lisp-Machine (Genera)	Redes Producto	EDITOR gráfico y de texto (código LISP) SIMULACIÓN controlada del usuario en modo simple o multi paso ANÁLISIS: Computación de la gráfica de alcanzabilidad REDUCCIÓN de la gráfica de alcanzabilidad Análisis de cerrojos, caminos de conclusión	POLÍTICA: herramienta disponible a través de ftp anónimos
PROMPT	PROMPT es una herramienta para la implementación de protocolos y otras aplicaciones desde un lenguaje de Redes de Petri de altas prestaciones hasta código C.	SUN Workstation (con Unix + SunView o X-Windows), VAX/VMS (con Curses), MS-DOS	Redes de Petri de Altas Prestaciones	PROMPT dispone de los siguientes componentes: Un compilador que traduce redes que han sido expresados en XNL (eXtended Net Language), en código C XDB, un depurador simbólico de pantalla completa, que permite la depuración de caracterizarse en la especificación de protocolo (expresado en XNL) SCI (System Control Interface), un interface para redes de ejecución de estudio y control (SIMULACIÓN) LFA (Log File Analyzer), una sola utilidad para la lectura y formato de archivos log creadas	PROMPT ha sido diseñada a partir de herramienta PROTEAN y es una componente de FORSEE POLÍTICA: precio negociable en la solicitud, licencia disponible para universidades

				durante la ejecución SCI Un conjunto de librerías que proporcionan funciones no-red como submodelos Facilidades para el usuario para incorporar su propio código C, para redes depuradas y para analizar la eficacia del funcionamiento Una colección de Redes de parámetros que ejerce el sistema para permitir comparaciones cuantitativas	
PSITool NET	Con PSITool NET el usuario puede construir, analizar, ejecutar y animar redes denominadas NET.	VAX-GPX y VAXstations (requiere X-Windows/Motif)	Redes NET: Redes Pr/T jerárquicas modificadas con tiempo y características estocásticas	La herramienta contiene cuatro componentes integradas: EDITOR sensible de sintaxis gráfica SIMULADOR: interactivo y fuera de línea ANALIZADOR. Análisis de rasgos de la Máquina de Redes de Petri que se han integrado los cuales consideran redes NET como redes P/T y de otra forma ignoran las inscripciones de red COMPONENTES de ANIMACIÓN: permite la visualización del comportamiento de los procesos ejecutados	POLÍTICA: las licencias pueden comprarse o alquilarse, precio en la solicitud
QPN Tool		Unix (requiere SunView)	Redes QPN y Libre Elección como una subclase estructural	EDITOR gráfico ANÁLISIS DE ESTRUCTURAS de estado ANÁLISIS: Análisis de Alcanzabilidad Invariantes S y T Algoritmo de eficiencia funcional para las Redes Libre Elección	POLÍTICA: en la solicitud
SANDS for COOPN	El entorno SANDS (Structured Algebraic Net Development System) ha sido desarrollado para construir especificaciones CO_OPN (Concurrent Object-Oriented Petri Nets).	SUN con OpenWindows 3.0	Redes Modulares Algebraicas	EDITORES gráficos con comprobador de sintaxix HERRAMIENTA DE TRANSFORMACIÓN con la librería de transformación provista de una transformación paso a paso basada en la equivalencia observacional El Simulador y el Monitorizado permiten un modo manual o automático de evolución, una exploración de la gráfica de alcanzabilidad (construida durante la simulación)... Verificador de propiedades temporales El compilador acepta un subconjunto de lenguaje CO-OPN y código de generación C para obtener una ejecución más eficaz	POLÍTICA: libre de cargo
SPNP	El Stochastic Petri Net Editor (SPNP) es una herramienta de Redes Petri basada en un formalismo como GSPN, llamado RN Estocástica.	Unix	RN Estocástica (SRN)	Las Redes son descritas por un lenguaje de entrada basado en C (CSPL- C basado en lenguaje de Redes de Petri Estocásticas) Construcción de la gráfica de alcanzabilidad Construcción del correspondiente eslabón Markov Temporizado Continuo/Determinado Características de transición y estado y análisis de caracterización	POLÍTICA: sin cargo a lugares académicos para uso no comercial
TORAS	TORAS es una herramienta para el análisis de Redes de Petri usando el análisis de la alcanzabilidad.	Estación de Trabajo Sun con Unix	Redes Lugar/Transición	Descripción textual de redes Análisis de alcanzabilidad usando: Método de conjunto pertinaz Técnica de análisis de la Alcanzabilidad de Holzmann	TORAS es un componente de FORSEE. POLÍTICA: todavía no disponible, pero existe prototipo
VISUAL SIMNET	Visual Simnet es un entorno para la simulación de sistemas, que son describos por Redes de Petri Estocásticas.	MS-DOS	Redes de Petri Temporizadas, Estocásticas (con diferentes distribuciones), Coloreadas con colas,	EDITOR de texto y gráfico SIMULACIÓN animada Análisis de la grafica de alcanzabilidad Análisis de características: tiempo de espera, longitud de cola, valor momentáneo, valor principal,... Filtro exportador para herramientas WINPETRI e INA	POLÍTICA: herramienta disponible a través de ftp anónimos

			prioridades, probabilidades, arcos inhibidores		
VOLTAIRE	Voltaire evoluciona desde un contrato de investigación de cuatro años entre Bell Canada's Quality Engineering Group y VLSI Design Laboratory en el Departamento de Ingeniería Eléctrica en la Universidad McGill usado para el entorno de un modelado de sistemas y un entorno de evaluación.		Redes de Petri Coloreadas jerárquicas con arcos inhibidores y extensiones estocásticas	SIMULACIÓN con verificación de coherencia ANÁLISIS cuantitativo Características objeto orientado	POLÍTICA: en solicitud
WIN PETRI		MS-DOS con MS-Windows (requiere compilador C++)	Redes Lugar/Transición y Redes Condición/Evento con arcos inhibidores	EDITOR gráfico Simulación paso a paso o automática usando la regla de transición ANÁLISIS de la gráfica de alcanzabilidad (lugares conflictivos, cerrojos, viveza, reversibilidad)	POLÍTICA: herramienta disponible a través de ftp anónimos
XPETRI		Sun, Unix, DEC Ultrix (requiere X11, Motif)	Redes de Petri	EDITOR gráfico SIMULACIÓN ANÁLISIS ESTRUCTURAL: Invariantes S y T Gráfica de ocurrencia Gráfica de cobertura Análisis de características continuas	POLÍTICA: herramienta disponible a través de ftp anónimos
XSIMNET		Unix (requiere XWindows)	Red de Simuluación Extendida SPN/CPN con jerarquías	Descripción de redes en SNL (Simulation Net Language) Interpretador para simulación enfatizando el resultado final Transformaciones de Redes para el análisis Estadísticas finales	POLÍTICA: herramienta disponible a través de ftp anónimos

1.5.4. Referencias

[1] L. Bernardinello, F. De Cindio, A survey of Basic Net Models and Modular Net Classes, LNCS vol. 609, Springer Verlag, 1992

[2] H. J. Genrich, K. Lautenbach, P. S. Thiagarajan, Elements of general Net Theory, LNCS vol. 84, Springer Verlag, 1980

[3] E. Best, C. Fernandez: Notations and Terminology on Petri Net Theory, Arbeitspapiere der GMD 195, 1986

[4] P. S. Thiagarajan, Elementary Net Systems, LNCS vol. 254, Springer Verlag, 1987

[5] G. Rozenberg, P. S. Thiagarajan, Petri Nets: Basic Notions, Structure, Behaviour, LNCS vol. 424, Springer Verlag, 1986

[6] M. Hack, Analysis of production schemata by Petri Nets, TR-94, MIT, Boston, 1972

[7] F. De Cindio, G. De Michelis, L. Pomello, C. Simone, A. Stragapede, Le Reti di Automi Sovrapposti: Una classe Modulare di Reti di Petri, ENEL-DSR-CRA, Milano, 1987

[8] M. Jantzen, R. Valk, Formal properties of Place/Transition Nets, LNCS vol. 84, Springer Verlag, 1981

[9] G. Winskel, Categories of Models for Concurrency, LNCS vol. 197, Springer Verlag, 1985

[10] R. Devillers, The semantics of capacities in P/T nets, LNCS vol. 424, Springer Verlag, 1990

[11] W. Reisig: On the semantics of Petri Nets, in: Neuhold, Chroust (eds.), Formal Models in Programming, North Holland Publ. Company, IFIP, 1985

[12] W. Reisig, Place/Transition Systems, LNCS vol. 254, Springer Verlag, 1987

[13] K. Lautenbach, Linear Algebraic Techniques for Place-Transition Nets, LNCS vol. 254, Springer Verlag, 1987

[14] E. Best, Structure Theory of Petri Nets: the Free Choice Hiatus, LNCS vol. 254, Springer Verlag, 1987

[15] F. Commoner, A. W. Holt, S. Even, A. Pneuli: Marked Directed Graphs. JCSS vol. 5, 1971

[16] C. Dimitrovici, U. Hummert, L. Petrucci, Semantics, Composition and Net Properties of Algebraic High-level Nets, LNCS vol. 524, 1991

[17] J. Vautherin: Un modèle algébrique, basé sur les rèseaux de Petri, pour l'ètude des systèmes parallèles, Thèse de doctorat d'ingènieur, Universitè de Paris-Sud, 1985

[18] J. Vautherin: Parallel Systems Specifications with Colored Petri Nets and Algebraic Specifications, LNCS vol. 266, 1987

[19] U. Hummert, Algebraische Theorie von High-Level-Netzen, Doktorarbeit, TU Berlin 1989

[20] K. Jensen, Coloured Petri Nets: A high-level Language for System Design and Analysis, LNCS vol. 483, Springer Verlag 1990

[21] K. Jensen, Coloured Petri Nets and the Invariant Method, Theoretical Computer Science vol. 14, 1981

[22] A. Chizzoni, CLOWN: CLass Orientation With Nets, Tesi di Laurea, DSI Milano, 1994

[23] E. Battiston, F. De Cindio, G. Mauri, OBJSA Nets: a class of High-Level Nets having Objects as Domains, In: G. Rozenberg, Advances in Petri Nets 1988, LNCS vol. 340, Springer Verlag, 1988

[24] C. Ghezzi, D. Mandrioli, S. Morasca, M. Pezzé, A unified High-Level Petri Net formalism for time-critical systems, IEEE Transactions on Software Engineering, February 1991

[25] H. J. Burkhardt, P. Ochsenschläger, R. Prinoth, Product Nets - A formal description technique for cooperating systems, GMD - Studien Nr. 165, Sept. 1989

[26] G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad: Stochastic Well-Formed Coloured Nets and Symmetric Modelling Applications. IEEE Transactions on Computers, Vol. 42, No. 11, 1993

[27] J.-M. Ilié, O. Rojas, On Well-Formed Nets and Optimizations in enabling tests, LNCS vol. 691, Springer Verlag, 1993

[28] G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad, On Well-Formed Coloured Nets and their Symbolic Reachability Graph, 11th Int. Conf. on Applications and Theory of Petri Nets, Paris, France, 1990

[29] K. Jensen, G. Rozenberg, High-level Petri Nets - Theory and Application, Springer Verlag, 1991

[30] H. Alla, R. David: Continuous and Hybrid Petri Nets. Journal of Circuits, Systems and Computers, vol 8 num. 1. World Scientific Publishing Company. 1998

[31] [4] Silva, M. (1985). Las Redes de Petri: en la automática y la informática. AC, D.L., Madrid.

Otras publicaciones de interés

[aal93] W. M. P. van der Aalst, Interval Timed Coloured Petri Nets and their Analysis, LNCS vol. 691, Springer Verlag, 1993
[abk95] A. Attieh, M.Brady, W. Knottenbelt, P.S. Kritzinger, Functional and Temporal Analysis of Concurrent Systems, Protocol Workshop, 16th International Conference on Theory and Application of Petri nets, Turin, June 1995
[ac87] M. Ajmone-Marsan, G. Chiola, On Petri Nets with Deterministic and Exponentially distributed firing times, LNCS vol. 266, Springer Verlag, 1987
[ajs89] K. Albert, K. Jensen. R. M. Shapiro: Design/CPN. A tool package supporting the use of Coloured Petri Nets, Petri Net Newsletter 32, April 1989
[akp94] N. A. Anisimov, A. Kovalenko, P. Postupalski, Compositional Petri Net Environment, Proc. of the 1994 IEEE Symposium on Emerging Technologies and Factory Atomation, Tokyo, Japan, November 1994
[ani89] N. A. Anisimov, A Notion of Petri Net Entity for Communication Protocol Design, Institute for Automation and Control Processes, Vladivostok,1989
[ani91] N. A. Anisimov, An Algebra of regular Macronets for formal specifications of communication protocols, Computers and Artificial Intellignce, No. 6, 1991
[bat94] E. Battiston, Guida all'ambiente ONE, CNR, Progetto finalizzato "Sistemi informatici e Calcolo parallelo", Rapporto, Luglio 1994

[bau93] F. Bause, Queueing Petri Nets - A formalism for the combined qualitative and quantitative analysis of Systems, 5th Int. Workshop of Petri Nets and Performance Models, Toulouse, France, Oct. 1993
[bb91] M. Baldassari, G. Bruno, PROTOB: An Object Oriented methodology for developing Discrete Event Dynamic Systems, in: K. Jensen, G. Rozenberg (ed.s), High-Level Petri Nets. Theory and Application, 1991
[bg92] D. Buchs, N. Guelfi: Distributed System Specification using CO-OPN. IEEE, 3rd Workshop on Future Trends of Distributed Computing Systems, Taipei, April 1992
[bg93] D. Buchs, N. Guelfi, Formal development of actor programs using structured algebraic Petri Nets, LNCS vol. 694, Springer Verlag, 1993
[bil91] J. Billington, FORSEEing Quality Telecommunications Software, Invited paper for the first Australian Conference on Telecommunications Software (ACTS), Melbourne, April 1991
[bjp91] G. Berthelot, C. Johnen, L. Petrucci, PAPETRI: Environment for the Analysis of Petri Nets, LNCS vol. 531, Springer Verlag, 1991
[bv95] G. Bucci, E. Vicario, Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets, IEEE Transactions on Software Engineering, Dec. 1995
[bww88] J. Billington, G. R. Wheeler, M. C. Wilbur-Ham, PROTEAN: A High-Level PetriNet Tool for the Specification and Verification of Communication Protocols, IEEE Trans. Software Eng. vol. 14 no. 3, March 1988
[cba93] G. Conte, G. Balbo, M. Ajmone Marsan, G. Chiola, Generalized Stochastic Petri Nets: A Definition at the Net Level and its Implications, IEEE Transactions on Software Engineering, Vol. 19, 1993
[cbc92] G. Ciardo, A. Blakemore, P. F. Chimento, J. K. Muppala, K. S. Trivedi: Automated Generation and Analysis of Markov Reward Models using Stochastic Reward Nets, IMA Volumes in Mathematics and its Applications, Vol. 48, Springer Verlag, 1992
[cj85] C. Choppy, C. Johnen, PETRIREVE: Proving Petri Net properties with rewriting systems, LNCS vol. 202, Springer Verlag, 1985
[cm91] J. M. Couvreur, J. Martinez, Linear Invariants in Commutative High-Level Nets, in: K. Jensen, G. Rozenberg (ed.s), High-Level Petri Nets. Theory and Application, 1991
[ctm89] G. Ciardo, K. S. Trivedi, J. Muppala, SPNP: stochastic Petri net package, Proc. 3rd Int. Workshop on Petri Nets and Performance Models (PNPM89), IEEE Computer Society Press, 1989
[dan91] G. Degli Antoni, Communicanting Petri Nets, Rapporto interno n.86/91, DSI, Univ degli Studi Milano, Novembre 1991
[dtg84] J. B. Dugan, K. S. Trivedi, R. M. Geist, V. M. Nicola, Extended Stochastic Petri Nets: Applications and Analysis, Proc. PERFORMANCE '84, Paris, France, Dec. 1984
[eph92] E. Best, R. Pinder Hopkins, B(PN)^2 - a Basic Petri Net Programming Notation, Hildesheimer Informatik-Bericht no. 27/92, Univ. Hildesheim, 1992
[es91] J. Esparza, M. Silva, Circuits, handles, bridges and nets, LNCS, vol. 483, Springer Verlag, 1991
[fel86] F. Feldbrugge, Petri Net Tools, LNCS vol. 222, Springer Verlag, 1986
[fj91] F. Feldbrugge, K. Jensen, Computer Tools for High-level Petri Nets, in: K. Jensen, G. Rozenberg (ed.s), High-Level Petri Nets. Theory and Application, 1991
[fl93] H. Fleischhack, U. Lichtblau, MOBY - A tool for high-level Petri Nets with objects, Proc. IEEE/SMC 93, Le Touquet, Vol. 4, 1993
[fls93] Hans Fleischhack, Ulrike Lichtblau, Michael Sonnenschein, Ralf Wieting, Generische Definition {hierarchischer} {zeitbeschrifteter} {höherer} Petrinetze, Bericht der Arbeitsgruppe Informatik-Systeme Nr. AIS-13, Fachbereich Informatik, Universität Oldenburg, December 1993
[for86] I. R. Forman, petri - A Unix Tool for the analysis of Petri Nets, Proc.1986 Fall Joint Computer Conf., 1986
[gl73] H. J. Genrich, K. Lautenbach: Synchronisationsgraphen. Acta Informatica vol. 2, 1973
[gl81] H. J. Genrich, K. Lautenbach, System Modelling with High-Level Petri Nets, Theoretical Computer Science, vol. 13, 1981
[gl83] H. J. Genrich, K. Lautenbach, S-Invariance in Predecate-Transition Nets, in "Application and Theory of Petri Nets", IFB66, Springer Verlag, 1983

[gm89] J. A. Goguen, J. Meseguer, Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions, and partial operations, Technical Report SRI-CSL-89-10, Computer Science Lab, SRI International, July 1989
[grt95] B. Grahlmann, S. Römer, T. Thielke, B. Graves, M. Damm, R. Riemann, L. Jenner, S. Melzer, A. Gronewold, PEP: Programming Environment Based on Petri Nets, in Hildesheimer Informatik-Berichte no. 14/95, Univ. Hildesheim, 1995
[hjs86] P. Huber, K. Jensen, R. M. Shapiro: Design/CPN extensions - Timed simulation, Colour set restrictions and reporting facilities, Meta Software, version II, February 1986
[hol88] G. J. Holzmann: An improved protocol reachability analysis technique. Software Practice and Experience, Vol. 18, No. 2, Febr. 1988
[hsv89] K. M. van Hee, L. J. Somers, M. Voorhoeve, Executable Specifications for Distributed Information Systems, in E. D. Falkenberg, P. Lindgreen (eds.), Information System Concepts: an in-depth analysis, North Holland, 1989
[hsv91] K. M. van Hee, L. J. Somers, M. Voorhoeve, The EXSpect Tool, LNCS vol. 551, 1991
[kb94] R. K. Keller, G. von Bochmann, Petri Net based business modelling and simulation with the Macrotec environment, June 1994, Handout at tool presentation at the 15th International Conference on Application and Theory of Petri Nets, Zaragoza, Spain
[kel95] C. Kelling, TimeNET-SIM - a parallel Simulator for Stochastic Petri Nets, Proc. 28th Annual Simulation Symp., Phoenix, Arizona, USA, 1995
[kfl87] J. Kaltwasser, G. R. Friedrich, P. Leipner, B. Müller, T. Vieweg, Dialog-orientiertes graphisches Petri-Netz-Entwicklungssystem (DIOGENES), Akademie der Wissenschaften der DDR, ZKI-Informationen 4/87, Berlin,1987
[klo93] R. K. Keller, R. Lajoie, M. Ozkan, F. Saba, X. Shen, Tao Tao, G. von Bochmann, The Macrotec toolset for CASE-based business modelling, Proc. of the 6th International Workshop on Computer-Aided Software engineering, Singapore, July 1993
[lin92] C. Lindemann, DSPNexpress: A Software package for the efficient solution of deterministic and stochastic Petri Nets, Proc. 6th International Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, Edinburgh, Great Britain, 1992
[lin93] C. Lindemann, Performance Modelling using DSPNExpress, LNCS vol. 729, Springer Verlag, 1993
[lin94] C. Lindemann, Stochastic Modeling using DSPNexpress, Oldenbourg Verlag, 1994
[lm91] C. Lin, D. C. Marinescu, Stochastic High-Level Petri Nets and Applications, in: K. Jensen, G. Rozenberg (ed.s), High-Level Petri Nets. Theory and Application, 1991
[mb83a] M. Menasche, B. Berthomieu, An Enumerative Approach for Analysing Time Petri Nets, Proceedings of the Information Processing 83, IFIP - Ninth World Congress, Paris, September 1983
[mb83b] M. Menasche, B. Berthomieu, Time Petri Nets for Analysing and Verifying Time Dependent Protocols, Proceedings of the Third International Workshop on Protocol Specification, Testing and Verification, Zurich, June 1983
[men85] M. Menasche, PAREDE: An Automated Tool for the Analysis of Time(d) Petri Nets, Proceedings of the International Workshop on Timed Petri Nets, Torino, July 1985
[met94] S. Metge et al., SURF-2: outil d'evaluation de la sret de fonctionnement par chaines de Markov et reseaux de Petri stochastiques. 9eme Colloque International de Fiabilite et de Maintenabilit et ESREL '94 (European Safety and Reliability Conference), Lau Baule, France, 1994
[mol86] M. K. Molloy, A CAD Tool for Stochastic Petri Nets, Proc. 1986 Fall Joint Computer Conf., 1986
[ms93] J. F. Meyer, W. H. Sanders: Specification and construction of performability models, in Second Int. Workshop on Performability Modelling of Computer and Communication Systems, Le Mont Saint-Michel, France, June 1993
[mur89] T. Murata, Petri Nets: Properties, Analysis and Applications, Proc. IEEE vol. 77, no. 4, April 1989
[nat80] S. Natkin: Les Reseaux de Petri Stochastique et leur Application a l'Evaluation des Systèmes Informatique, Thèse de Docteur Ingegneur, CNAM, Paris, France, 1980
[och91] P. Ochsenschläger, Die Produktnetzmaschine - Eine Übersicht, Arbeitspapiere der GMD 505, Jan. 1991

[och94a] P. Ochsenschläger, Verification of Cooperating Systems by simple homomorphisms using the Product Net Machine, Workshop Algorithmen und Werkzeuge für Petri Netze, Berlin, 1994
[och94b] P. Ochsenschläger, Verfikation von SmartCard-Anwendungen mit Produktnetzen, 1994
[oem90] H. Oswald, R. Esser, R. Mattmann, An environment for specifying and executing hierarchical Petri Nets, Proc. Int. Conf. on Software Engineering, IEEE, 1990
[op95] P. Ochsenschäger, R. Prinoth, Modellierung verteilter Systeme. Konzeption, Formale Spezifikation und Verifikation mit Produktnetzen, Programm für Angewandte Informatik, Vieweg Verlag, to appear April1995
[par90] K. R. Parker: The PROMPT automatic implementation tool - Initial impressions. Proc. 3rd Int. Conf. on Formal Description Techniques, Madrid, Spain, Nov. 1990
[ps93] L. Peters, R. Schultz: The application of Petri Nets in Object-Oriented Enterprise simulation. Hawaii International Cong. on system Sciences (HICSS-26), January 1993 [pt91] P. Parent, O. Taniv: Voltaire: a discrete event simulator. 4th Int. Workshop on Petri Nets and Performance Models, Melbourne, Australia, 1991
[ram74] C. Ramchandani, Analysis of asynchronous concurrent systems by Petri Nets, Cambridge, MA: MIT, Project MAC, TR-120, Feb. 1974
[rei85a] W. Reisig, Petri Nets. An Introduction, New York: Springer Verlag, 1985
[sa88] J. Snow, R. Albright, MetaDesign - The graphic tool for modelling complex systems, Version 2.3; Meta Software Corporation, Cambridge, 1988
[sd88] R. Shapiro, R. D. Druker, Design/OA, Meta Software Corporation, Cambridge, 1988
[sif77] J. Sifakis, Use of Petri Nets for performance evaluation, in: H. Beilner and E. Gelenbe (eds.), Measuring, Modelling and Evaluating Computer Systems, North Holland, 1977
[ska92] M. Skacel, Tools for Petri Nets drawing, simulation and analysis under MS Windows, dipl. thesis, Dep. of Computer Science and Engineering, Technical University of Brno, 1992
[skb93] X. Shen, R. K. Keller, G. von Bochmann, Macrotec at a glance, a new approach and toolset for business modelling and simulation, October 1993, Handout at demonstration session at the 1993 GAS conference, Toronto, Canada
[sm86] W. H. Sanders, J. F. Meyer, METASAN: A Performance Evaluation Tool based on Stochastic Activity Networks, Proc. ACM/IEEE-CS Fall Joint Computer Conference, Nov. 1986
[so93] W. H. Sanders, W. D. Obal II, Dependability Evaluation using UltraSAN, Software Demonstration in Proc. of the 23rd International Symposium on Fault-Tolerant Computing, Toulouse, France, IEEE Press, June 1993
[ssw94] S. Schöf, M. Sonnenschein, R. Wieting, Sequentielle und verteilte Simulation von THOR-Netzen, in: J. Desel, A. Oberweis, W. Reisig (ed.s), Workshop "Algorithmen und Werkzeuge für Petri Netze, Berlin, October 1994
[ssw95] S. Schöf, M. Sonnenschein, R. Wieting, High-level Modeling with THOR Nets, Proceedings of the 14th International Congress on Cybernetics, Namur, Belgium, August 21-25, 1995 (to appear)
[sta88] P. H. Starke, Remarks on Timed Petri Nets, Proc. 9th European Workshop on Application and Theory of Petri Nets, 1988
[sta90] P. H. Starke, Analyse von Petri-Netz-Modellen. B. J. Teubner Verlag, Stuttgart, 1990
[sta91] P. H. Starke, PAN/CPNA: Petri-Netz-Analyse-Werkzeuge, Effizientes Engineering komplexer Automatisierungssysteme, TU Braunschweig, 1991
[sur90] Z. Suraj, GRAPH: A graphical system for Petri Net design and simulation, Petri Net Newsletter no. 35, 1990
[ucs91] Unico Computer Systems: PROMPT System User Manual, Unico Computer Systems and Telecom Australia, Sept. 1991
[var93] K. Varpaaniemi, Efficent Detection of Deadlocks of Petri Nets, Helsinki University of Technology, Digital Systems Laboratory Report A26, Espoo, October 1993
[var94] K. Varpaaniemi, On Computing Symmetries and Stubborn Sets, Helsinki University of Technology, Digital Systems Laboratory Report B12, Espoo, April 1994
[var95] S. Varallo, Laboratorio su alcune teorie dei processi, Tesi di Laurea, DSI, Univ. degli Studi di Milano, 1995
[vr92] K. Varpaaniemi, M. Rauhamaa, The Stubborn Set Method in Practice, Lecture Notes in Computer Science vol. 616, Springer Verlag, 1992.

Páginas WEB

http://www.daimi.aau.dk/PetriNets
WELCOME TO THE WORLD OF PETRI NETS
http://diana.cps.unizar.es/banares/
PETRI NETS IN KNOWLEDGE BASED CONTROL. MODELS FOR MANUFACTURING SYSTEMS SIMULATION
http://dreyer.dsic.upv.es/users/oom/reports.html
REPORTAJES TÉCNICOS sobre modelo OASIS que es de RdP orientadas a objetos
http://www.ac.upc.es/homes/marcoa/Papers/abstract_tesina.html
SÍNTESIS DE CIRCUITOS ASÍNCRONOS MEDIANTE LA TRADUCCIÓN DE CIRCUITOS DE SINCRONIZACIÓN A REDES DE PETRI
http://www.informatik.hu-berlin.de/PNT/
CONCEPTION, THEORETICAL FOUNDATION AND VALIDATION OF AN APPLIED PETRI NET TECHNOLOGY
http://www.informatik.hu-berlin.de/PNT/pnt-overview.html
PROJECT OVERVIEW DETAILED RESULTS OF NSE. Conception, theoretical foundation and validation of an applied Petri Net Technology
http://www.informatik.hu-berlin.de/PNT/pnt-public.html
PROJECT PUBLICATIONS (Universidad de Berlín)
http://www.informatik.hu-berlin.de/top/pnk/index-engl.html
ANÁLISIS Y SIMULACIÓN DE LAS REDES DE PETRI (PETRI NET KERNEL) contiene software, publicaciones, aplicaciones, contactos y linca con otras direcciones..
http://www.informatik.hu-berlin.de/top/pnk/anwendungen-engl.html
APPLICATIONS. THE FOLLOWING APPLICATIONS BASED ON THE PETRI NET KERNEL EXIST:
http://www.informatik.hu-berlin.de/top/pnk/literatur-engl.html
PUBLICATIONS
http://www.laas.fr/~robert/workshop.d/index.html
CONFERENCE ON APPLICATIONS AND THEORY OF PETRI NETS TOULOUSE, JUNE 23, 1997
http://www.informatik.uni-hamburg.de/TGI/tools/tools_eng.html
RENEW IS A JAVA-BASED PETRI NET TOOL THAT WAS DEVELOPED HERE AT HAMBURG.
http://ls4-www.informatik.uni-dortmund.de/QPN/QPN_article/qpn_final/qpn_final.html
QUEUEING PETRI NETS A FORMALISM FOR THE COMBINED QUALITATIVE AND QUANTITATIVE ANALYSIS OF SYSTEMS
Enfoque desde un punto de vista matemático de las Redes de Petri
http://sun195.iit.unict.it/~webspn/webspn2/website/other/paper.html
A WEB-ACCESSIBLE PETRI NET TOOL. Redes estocásticas
http://www.ee.umanitoba.ca/tech.archive

TECHNICAL REPORTS sobre AUTOMATA, COMPUTATIONAL INTELLIGENCE, DATA AND SIGNAL COMPRESSION, FORMAL METHODS IN SYSTEM DESIGN, FUZZY SYSTEMS, GENETIC ALGORITHMS, MATHEMATICS, MICROELECTRONICS AND SOFTWARE SYSTEMS, NEURAL NETWORKS, PARALLEL PROCESSING, PETRI NETS, REAL-TIME SYSTEMS, SPECIFICATION AND DESCRIPTION LANGUAGE (SDL)
http://www.ee.uwa.edu.au/~braunl/pns/
PNS - AN S/T PETRI-NET SIMULATION SYSTEM
http://www.informatik.uni-bremen.de/uniform/vm97/methods/m-pnet.htm
PETRI NETWORKS
http://www.cis.um.edu.mt/~jskl/petri.html
PETRI NETWORKS
Ideas básicas sobre las redes de Petri, programa Psim para simularlas
http://www.cis.um.edu.mt/~jskl/simweb/index.html
SIMULATOR OF QUEUEING NETWORKS .
http://pdv.cs.tu-berlin.de/~azi/petri.html
PETRI NETS
http://www.daimi.au.dk/PetriNets/tools/complete_db.html
COMPLETE OVERVIEW OF PETRI NETS TOOLS DATABASE
Complete Overview of Petri Nets Tools Database
http://ls4-www.informatik.uni-dortmund.de/QPN/
QUEUEING PETRI NETS (QPNS)
http://www.cs.purdue.edu/homes/saw/cs406/lectures/11/petrinets-1.html
http://www.elet.polimi.it/Users/DEI/Sections/Compeng/Mauro.Pezze/Tutorials/PNPM99/slides/index.htm
TIME PETRI NETS A PRIMER INTRODUCTION (Software engineering and petri nets)
http://www.abo.fi/~atorn/Petri/PCont.html
Course material prepared by Aimo Törn in the Spring 1998 for a course on Petri Nets at the Computer Science Department of Åbo Akademi University and TUCS Turku, Finland
http://www.informatik.hu-berlin.de/PNT/Coll/index.html
COLLOQUIUM ON PETRI NET TECHNOLOGIES
http://www.cs.pitt.edu/~chang/365/2petri.html
PETRI-NET AND AUGMENTED-PETRI-NET
http://www.mech.kuleuven.ac.be/pma/project/complan/pnppm.html
PETRI NET PROCESS ; PLANNING MODULE
http://home.arcor-online.de/wolf.garbe/petrinets.html
TERMS FOR PETRI-NETS AS DEFINED IN THE LITERATURE: Dirección que trata sobre los términos básicos conformados en las redes de Petri
http://www-src.lip6.fr/logiciels/framekit/cpn-ami2.rdp-arrange.html
PETRI NET BEAUTIFIER. Como se trabaja con Macintosh las redes de Petri
http://www.cse.unsw.edu.au/dblp/db/conf/ac/petri2.html
ADVANCED COURSE: PETRI NETS
http://everest.ento.vt.edu/~sharov/PopEcol/lec1/petrinet.html

Diferentes aplicaciones de Petri
http://www.google.com/search?q=cache:worldserver.oleane.com/adv/elstech/petrinet.htm+petri+nets&hl=es
INTRODUCTION TO PETRI NETS
http://www.comp.lancs.ac.uk/computing/users/angie/petri.htm
Introducción a Redes de Petri
http://www.autom.dist.unige.it/autom_ita/software/petri_net.html
http://citeseer.nj.nec.com/did/107580
ELECTRONIC SYSTEM DESIGN AUTOMATION USING HIGH LEVEL PETRI NETS (CORRECT)
http://www.laas.fr/~robert/logic.d/austra.html
Petri nets and Artificial Intelligence
http://www-dse.doc.ic.ac.uk/~nd/surprise_97/journal/vol4/wll1/petriex.htm
http://www.ee.umanitoba.ca/tech.archive/pns-ex.html
PETRI NETS EXAMPLE
http://www.astem.or.jp/~tanabe/work/icpn97.html
TIMED PETRI NETS AND TEMPORAL LINEAR LOGIC
http://wwwis.win.tue.nl/~wsinwa/jcsc/node5.html
HIGH-LEVEL PETRI NETS
http://viking.gmu.edu/http/syst511/vg511/AppC.html
AN INTRODUCTION TO PETRI NETS
http://home.datacomm.ch/wkeller/petri/dokSem/sld001.htm
PETRI NETS FOR REVERSE ENGINEERING
http://www.ele.puc-rio.br/~menasche/petri/petridef.html
A SHORT INTRODUCTION TO PETRI NETS
http://www.informatik.uni-stuttgart.de/ifi/ti/fap98/fap98.html
WORKSHOP ON FORMAL LANGUAGES, AUTOMATA AND PETRI-NETS
http://www.inria.fr/RRRT/RR-1520.html
PARALLEL SIMULATION OF STOCHASTIC PETRI NETS USING RECURSIVE EQUATIONS
http://markun.cs.shinshu-u.ac.jp/kiso/projects/petrinet/pds/design/move-e.html
DESIGNING PETRI
http://dcr.rpi.edu/Research/VOPN.web/VOPN.rtf.20no98.html
EVOLUTION OF VIRTUAL OBJECTS IN PETRI NET MODELED COLLABORATIVE BEHAVIOR
http://www.univ-tlse1.fr/ceriss/COOpubli.html
THE CLIENT-SERVER PROTOCOL FOR COMMUNICATION OF PETRI NETS
http://yoneda-www.cs.titech.ac.jp/Papers/timed-e.htm
VERIFICATION OF TIMED SYSTEM BASED ON TIME PETRI NETS
http://etourtelot.free.fr/download/petri/
PETRI NET SIMULATION
http://www.research-projects.unizh.ch/oek/unit35100/area479/p1754.htm PENREE - PETRI NETS FOR REVERSE ENGINEERING PENREE - Petri Nets for Reverse Engineering
http://dalton.abo.fi/~atorn/Petri/P31.html

http://www.abo.fi/fak/mnf/infbeh/software/simnet/
SIMNET, FOR EXECUTING SIMULATION NETS
http://www.amazon.de/exec/obidos/ASIN/3540582762/artvisitwww/028-2889798-4866159
COLOURED PETRI NETS. BASIC CONCEPTS, ANALYSIS METHODS AND PRACTICAL USE.
http://www.brics.aau.dk/BRICS/RS/94/3/index.html
Lógica lineal en las RdP
http://206.67.72.201/catalog/np/dec98np/3-540-65306-6.html
LECTURES ON PETRI NETS I: BASIC MODELS
http://206.67.72.201/whatsnew.html
LECTURES ON PETRI NETS I: BASIC MODELS ADVANCES IN PETRI NETS
http://www.brics.dk/RS/95/39/index.html
PETRI NETS, TRACES, AND LOCAL MODEL CHECKING
http://cpe.gmu.edu/courses/ece545/lectures/545_13/sld024.htm
PETRI NETS EXAMPLES
http://www.gmd.de/SCAI/Projectpages/petrinet/petrinet.html
PETRI NETS AND COMPUTER ALGEBRA
http://www.cas.mcmaster.ca/~cs3sd3/cwnotes/mdlsl/sld058.htm
MODELING: CONCURRENCY AND CONTENTION (SIMULTANEOUS RESOURCE USAGE) ANALYSIS: REACHABILITY TREE, CTMC ANALYSIS
http://vega.icu.ac.kr/~cyu/courses/perf98/note_petri/index.htm
PETRI NET (FROM "PETRI NETS" BY J.L.PETERSON, ACM COMPUTING SURVEYS,
http://www.csr.unibo.it/~lumini/pindar/petrinet.htm
PETRINET
http://taylor.ces.clemson.edu/ie340/files/340-16.htm
ARQUITECTURA DEL NIVEL DE RED
http://www.msel.naoe.t.u-tokyo.ac.jp/staffs/aoyama/Class/Info_system/No3/sld036.htm
http://rutcor.rutgers.edu/~pinzon/papers/rrr1/node38.html
USING PETRI NETS
http://user.cs.tu-berlin.de/~gebert/Simulation/petrinete.html
INTRODUCTION TO PETRI NETS (Stochastic Petri-Nets)
http://www.ele.puc-rio.br/~menasche/petri/petridef.htm
A Short Introduction to Petri Nets
http://www.ship.edu/~jlsieb/theory/App-h40.htm#Petri%20Nets
PETRI NETS
ftp://ftp.informatik.uni-stuttgart.de/pub/petri-nets/
http://www.icsi.berkeley.edu/techreports/1992.abstracts/tr-92-008.html
LINEAR TIME ALGORITHMS FOR LIVENESS AND BOUNDEDNESS IN CONFLICT-FREE PETRI NETS
http://www.aisa.uvigo.es/fvazquez/cursos/petri/index.htm
INTRODUCCIÓN A LAS REDES DE PETRI. (40 páginas). Transparencias
http://www.aisa.uvigo.es/software.html

SOFTWARE DE LIBRE DISPOSICIÓN DESARROLLADO EN EL DEPARTAMENTO
http://www.diee.unica.it/~aldo/bibliohpn.html
BIBLIOGRAPHY ON HYBRID PETRI NETS
Dip. di Ingegneria Elettrica ed Elettronica, Università di Cagliari, Italy.
http://www.laas.fr/~robert/workshop.d/index.html
Manufacturing and Petri Nets
http://www.di.unito.it/~portinal/ispn.html
APPLICATIONS AND THEORY OF PETRI NETS
Petri Nets'99 and 98 Conference Program
http://homes.dsi.unimi.it/~alberti/Pubbl/OO-Petri.html
MODELLING GEOMETRIC OBJECTS WITH OBJSA NETS
http://www.ee.umanitoba.ca/tech.archive/mat1.html
MATRIX REPRESENTATION OF PETRI NETS
http://sadko.ncl.ac.uk/~nay/hwpn/hwpn.html
WORKSHOP ON HARDWARE DESIGN AND PETRI NETS (HWPN)
http://www.dbis.informatik.uni-frankfurt.de/~door/docs/spadecourse/sld032.htm
Transparencias (32 de 145) sobre RdP
http://www.math.upatras.gr/~vasilis/petri.html
Petri Nets: a Formal Method for Systems Modeling and Analysis
Bibliograpy on Petri Nets
http://dalton.abo.fi/~atorn/Petri/P42.html
COLOURED PETRI NETS,
http://faculty.winthrop.edu/snyderr/CS475.WEB/petri-01.asp#1v
PETRI NETS: INTRODUCTION
http://www.laas.fr/~robert/logic.d/austra.html
PETRI NETS AND ARTIFICIAL INTELLIGENCE
http://www.aut.utt.ro/~mappy/petri/nets/CPN.html
COLOURED PETRI NETS (CPN)
http://faculty.winthrop.edu/snyderr/9C.OLD/CS475.WEB/petri-01.htm
PETRI NETS: INTRODUCTION
http://taylor.ces.clemson.edu/ie340/files/340-19.htm
COLORED PETRI NETS
http://www.laas.fr/~robert/hybrid.d/adedops95.html
PETRI NETS FOR CONTROL AND MONITORING: SPECIFICATION, VERIFICATION AND IMPLEMENTATION,
http://www.aut.utt.ro/~mappy/petri/nets/pn-intro.html
ORDINARY PETRI NETS – INTRODUCTION ; 1-SAFE NET SYSTEMS; CONDITION/EVENT (C/E) - SYSTEMS ; Elementary Net (EN) System ; Extensions to Free Choice Nets ; FREE CHOICE (FC) SYSTEMS
http://www.laas.fr/~robert/logic.d/ieeesmc93_dia.html
PETRI NETS AND LINEAR LOGIC FOR PROCESS ORIENTED DIAGNOSIS
http://www.aut.utt.ro/~mappy/petri/nets/PRT.html
PREDICATE TRANSITION NETS
http://www.aut.utt.ro/~mappy/petri/nets/OPN.html

OBJECT PETRI NETS
http://www.dcs.ed.ac.uk/home/lfcsreps/91/ECS-LFCS-91-172/
Verifying Temporal Properties of Systems with Applications to Petri Nets
http://www.aut.utt.ro/~mappy/petri/nets/mod-intro.html
MODULAR PETRI NETS – INTRODUCTION
http://www.fee.vutbr.cz/~vojnar/Articles/springer97/springer97.abstract.html.cs.iso-8859-10
PNTALK - A COMPUTERIZED TOOL FOR OBJECT ORIENTED PETRI NETS MODELLING
http://www.e-technik.uni-kl.de/litz/ENGLISH/members/frey/Abstracts/SIM98-2.htm
*Simulation of Hybrid Systems based on Interpreted Petri Nets.*Proceedings of the IEE International Conference on Simulation
http://karna.cs.umd.edu:3264/papers/SG92:debit/SG92:debit.html
PLACE/TRANSITION NETS WITH DEBIT ARCS
http://www.pdv.cs.tu-berlin.de/~azi/texte/osaka_info.html
PERFORMANCE AND DEPENDABILITY EVALUATION OF MANUFACTURING SYSTEMS USING PETRI NETS
http://sirio.dit.upm.es/~str/silva.html
TRANSPARENCIAS SOBRE SISTEMAS DISCRETOS Y REDES DE PETRI BASADAS EN EL LIBRO DE MANUEL SILVA
http://lucas.cdf.udc.es/main/planes/indus/3/aut-elec/auto-ind.htm
Special Issue on Flexible Workflow Technology Driving the Networked Economy.
http://wwwis.win.tue.nl/~wsinwa/csse_special_issue.html
Special Issue on Flexible Workflow Technology Driving the Networked
http://wwwis.win.tue.nl/~wsinwa/jcsc/node1.html
THE APPLICATION OF PETRI NETS WORKFLOW.
THEORETICAL FOUNDATIONS OF COMPUTER SCIENCE
http://www.informatik.uni-hamburg.de/TGI/publikationen/publikationen_home_eng.html
http://www4.cs.uni-dortmund.de/QPN/
QUEUEING PETRI NETS (QPNS)
http://polaris.dit.upm.es/~str/silva.html
Transparencias sobre sistemas discretos y redes de Petri basadas en el libro de Manuel Silva Las Redes de Petri en la Automática y en la Informática. Transparencias de redes de Petri
http://claymore.engineer.gvsu.edu/eod/mathmod/petri/petri.html
MATHEMATICAL MODELLING. Introducción general a las redes de Petri.
http://www1.gratisweb.com/ivanherrera/
MODELOS DE REDES Y REDES DE PETRI
http://gehtnix.fernuni-hagen.de:8000/se2-eng/ke5/node11.html
Explicación teórica sobre las redes de Petri, se establece una introducción sobre los lugares, transiciones (elementos de que se compone las redes de Petri), clasificando los objetivos y lugares de diseño:

Tema 2

METODOLOGÍA Y ARQUITECTURAS PARA LA AUTOMATIZACIÓN DE PROCESOS COMPLEJOS.

2.1. Introducción

En el presente capítulo, en el que comienza la aportación innovadora de esta tesis, se analiza la forma más eficiente de empleo de las herramientas de automatización para el diseño de aplicaciones industriales, especialmente en procesos complejos o muy extensos.

La aportación se divide principalmente en tres puntos. En el primero se comienza a analizar cómo se pueden implementar sobre los dispositivos industriales los diseños generados con herramientas gráficas, aunque a tal respecto en el siguiente capítulo se desarrolla toda la metodología necesaria para su realización. En el punto segundo se analiza la influencia de desarrollar un modelo apropiado del sistema. En el tercero de los puntos importantes se presentan alternativas al control clásico, que consiste en control mediante PLC y supervisión con SCADA, proponiendo un control redundante adaptativo capaz de determinar malos funcionamientos de la automatización (ante lo que seguiría funcionando con la automatización redundante) e incluso capaz de detectar desviaciones en el funcionamiento del sistema, causados por factores como el envejecimiento de las instalaciones o el mal funcionamiento de alguno de los sistemas.

Se ha decidido agrupar esos tres puntos, correspondientes a los apartados 2, 3 y 4 de este tema, por la estrecha relación que mantienen entre sí, pero se ha realizado una introducción dentro de cada uno para indicar sus peculiaridades.

2.2. RdP en Automatizaciones Industriales

2.2.1. Introducción

Los procesos industriales automáticos vienen evolucionando desde hace décadas, y en muchas ocasiones más rápido de lo que han podido hacer muchas plantas industriales. Esto es debido a que las grandes inversiones necesarias para la creación de una factoría de última generación requieren un tiempo de amortización en el cual las técnicas y tecnologías empleadas se quedan desfasadas. En estos casos se requiere una adaptación de las instalaciones y del proceso productivo para conseguir un sistema de producción más moderno y eficaz, y si fuese posible sin excesivos recursos adicionales.

Existen infinidad de ejemplos que hacen patente lo que se acaba de comentar, como pueden ser las bodegas de crianza, las plantas de producción de cerámica, las naves de

prefabricados de construcción, las de componentes para automoción, etc. Tomando el caso de las bodegas de crianza, las grandes bodegas que se construyeron necesitaron grandes inversiones para conseguir los recintos subterráneos con las condiciones físicas ideales. Posteriormente con la aparición de los sistemas automáticos necesitaron adaptar dichas instalaciones a sistemas de almacenaje automáticos, para poder competir con las nuevas instalaciones que sí que lo incorporaban. E igualmente ha ido ocurriendo con las mejoras que han ido apareciendo como sistemas de monitorización y de supervisión, herramientas de gestión global, etc.

En este contexto de evolución y desarrollo de las automatizaciones, muchas veces condicionado por la herencia anterior, se debe tener especial cuidado en realizar la adaptación de una manera coherente y previsora, intentando buscar no solamente la consecución del objetivo sino también una realización clara, estructurada, y sistemática, que condicione lo menos posible la adaptación de las futuras innovaciones que haya que aplicar. Pero por desgracia este tipo de previsión se da con poca frecuencia, especialmente en la pequeña y mediana industria, en la que suele primar la resolución rápida del problema antes que una solución de futuro.

Por otro lado, hemos visto en el tema anterior que las RdP constituyen una herramienta, teórica y práctica, de inestimable ayuda para el modelado de sistemas y el desarrollo de automatizaciones en estas plantas, especialmente en sistemas secuenciales y concurrentes. Hay que decir que es más habitual la utilización de GRAFCET que RdP en las automatizaciones industriales, principalmente debido a que las RdP son más desconocidas, aunque más potentes. Sin embargo, puesto que los GRAFCET se pueden interpretar como un subconjunto de las RdP (todo esquema en GRAFCET puede ser traducido a una RdP, y de una manera muy sencilla) a partir de ahora y durante el resto de esta tesis nos referiremos a ambas herramientas de la automatización nombrando de forma genérica RdP.

En muchas ocasiones, una vez que se ha diseñado la RdP que controla el proceso, se traduce al lenguaje del autómata (generalmente en forma de diagrama de contactos), y se pierde ya el concepto global del funcionamiento que nos proporcionaba esa red. En otras ocasiones ni siquiera se diseña la RdP que modela el sistema, sino que se programa éste directamente en lenguaje de contactos, de una manera poco sistemática y clara, o incluso en ocasiones se genera el programa secuencial del autómata traduciendo literalmente los contactos físicos existentes a base de relés (lógica cableada) antes de introducir los autómatas programables.

Precisamente para aprovechar las ventajas de las RdP, su potencia y su simplicidad, se realiza junto a la simulación del proceso productivo, la simulación de la Red de Petri que define su funcionamiento, ya sea la que se ha empleado para el desarrollo del programa del autómata o una que modeliza tal desarrollo. Con esto, tenemos en la monitorización toda la información del estado del sistema, ya que con la visualización de la planta tan sólo tenemos información de las entradas y salidas, lo cual no es suficiente en los sistemas secuenciales. Además así podemos emplear todas las posibilidades de análisis, optimización y detección de errores, ampliamente estudiadas en las RdP, para mejorar el proceso, y de una manera clara, estructurada, y sistemática, para facilitar posteriores modificaciones o ampliaciones.

Esta simulación de redes de Petri supervisoras del sistema automatizado ha sido aplicada en industrias colaboradoras con esta línea de investigación, con resultados muy satisfactorios, de dos formas principalmente: mediante programas informáticos (con metodología orientada a objetos) capaces de comunicar con los autómatas programables para compartir su información, y también aprovechando los sistemas gráficos de supervisión (SCADA) que se emplean en muchas plantas industriales.

2.2.2. Implementación de las Automatizaciones

Como acabamos de comentar, cuando se aborda el problema de automatizar un proceso complejo de eventos discretos, en sistemas secuenciales y concurrentes, si no se emplea una metodología adecuada para su desarrollo el resultado puede ser una automatización con algún error encubierto o en todo caso sumamente complicada de entender, y por tanto de modificar o mejorar, para cualquiera que no la haya diseñado (e incluso para quien lo haya hecho, al cabo de no mucho tiempo). Con el empleo de las RdP para el desarrollo de tales sistemas contamos con una herramienta sencilla, potente, fácil de entender, y con numerosos estudios teóricos que nos avalan para detectar ciertos errores intrínsecos [20]. Así, la modelización de un sistema que presente recursos compartidos, concurrencias, autorizaciones memorizadas o sin memorizar, sincronizaciones mutuas con evoluciones simultáneas o con desactivación de subsistemas, etc., es una labor muy simple de realizar mediante RdP y sin embargo puede resultar sumamente compleja de programar directamente en lenguaje de contactos de un autómata programable [20].

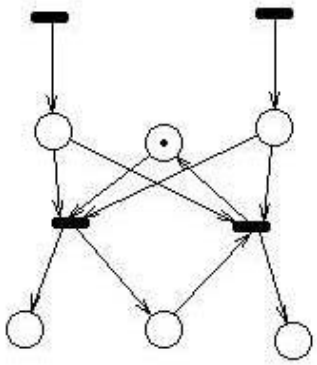


Figura 1: Ejemplo de sincronización con
desactivación mediante RdP

La mayoría de las automatizaciones están controladas con autómatas programables a nivel de planta, muchas veces conectados en red entre sí y con otros sistemas, a través de buses de campo, y en ocasiones estos buses conectados a otras redes de mayor ámbito. Estos autómatas programables guardan en sus memorias el programa de funcionamiento con el cual controlan sus salidas en función de las entradas y de los estados internos. Estos programas de los autómatas están realizados en el lenguaje propio de su marca y modelo, y habitualmente pueden ser realizados tanto en listado de instrucciones como en contactos (que básicamente es una forma semigráfica de

representar el listado de instrucciones). Puesto que el término en inglés para este último tipo de contactos de autómata (ladder) está sumamente extendido en la bibliografía (incluso la de nuestra lengua) en adelante también será utilizado.

Programando así los autómatas podemos implementar las RdP empleadas para la resolución de la automatización, y el autómata evoluciona tal cual lo haría la RdP correspondiente. Pero puesto que el autómata evoluciona según su diagrama de contactos, se pierde en el autómata el concepto de la RdP que rige la automatización, pasando a un concepto de listado de contactos, que es mucho más difícil de seguir y de comprender, y que sin embargo no aporta más información que la que aporta la RdP de una manera gráfica e intuitiva [17]. Por eso resulta muy interesante disponer de una representación gráfica de la evolución del sistema en forma de la RdP que lo maneja.

Cuando tenemos un programa secuencial de autómata que no viene de una traducción de una RdP, siempre es posible determinar una RdP que realice lo mismo. En estos casos puede ocurrir que el programa en contactos sea más simple que la RdP, pero aun así siempre se gana en claridad y comprensión del proceso.

Esto último podemos verlo en las figuras 2 y 3, en las que tenemos el programa en contactos de un típico mando bimanual de seguridad y una RdP que modela su funcionamiento. Este mando es muy empleado en máquinas peligrosas, tipo prensas o sierras, para proteger las manos de los operarios. Consiste en que la salida sólo se activa mientras se pulsan los dos pulsadores, que además deben pulsarse en menos de un cierto tiempo (normalmente medio segundo). Transcurrido ese tiempo desde que se pulsa el primero, si la salida está activa y se suelta alguno de los dos, se desactiva la salida y queda bloqueada hasta que se suelten ambos pulsadores.

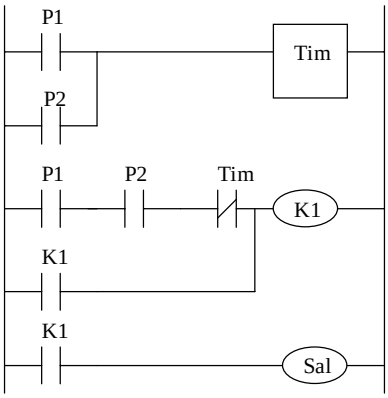


Figura 2: Mando bimanual de seguridad programado en lenguaje de contactos

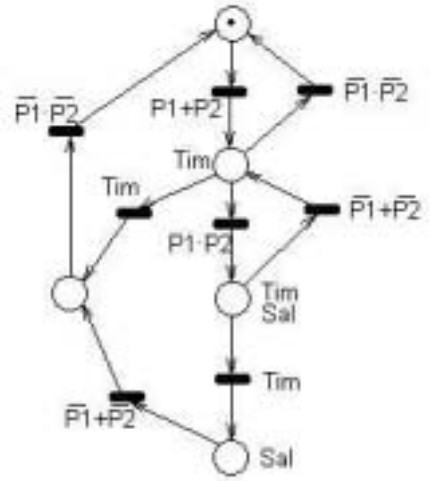


Figura 3: Red de Petri de un mando bimanual de seguridad

Vemos que es más simple el programa en contactos, pero en la RdP vemos en todo momento dónde tenemos el sistema y sus posibilidades de evolución. De todas formas puede considerarse este ejemplo como una excepción, ya que no es frecuente encontrar este tipo de sistemas secuenciales que sean más sencillos modelados en contactos, y lo más normal es que el programa del autómata consista en unas series de biestables a modo de secuenciadores, conectados entre sí mediante funciones algo más complejas.

A todo lo comentado anteriormente hay que añadir que ciertos modelos y marcas de autómatas programables permiten la programación en GRAFCET, así como la visualización en el propio GRAFCET de control de la evolución del proceso. Esto confiere a tales modelos una gran ventaja sobre los que no lo permiten, si bien aun es más ventajoso el empleo de RdP de supervisión y simulación del proceso por tres principales razones: no se necesita que los autómatas sean de ningún tipo especial, se pueden representar RdP correspondientes a más de un autómata simultáneamente (o a todo el proceso global de la planta), y además, como ya hemos comentado, la potencia como herramienta de las RdP es muy superior a la del GRAFCET.

2.2.3. Supervisión con SCADA

Cada vez más las plantas industriales automatizadas cuentan con un sistema SCADA para monitorización y supervisión de la planta y del sistema productivo, como el mostrado en la Figura 4.

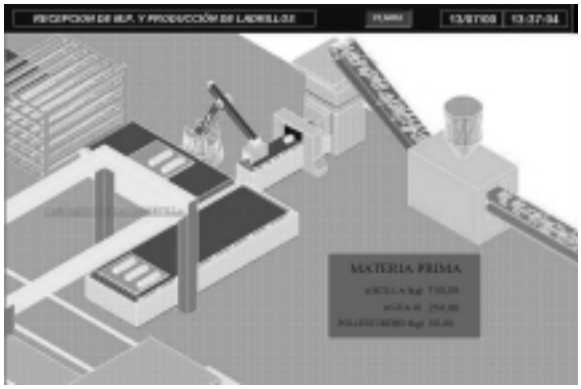


Figura 4: Monitorización con SCADA de zona de planta industrial

Los modernos paquetes de software SCADA incluyen herramientas para realizar tales funciones de una manera muy eficiente y no muy complicada [12], y además cuentan con otras posibilidades incorporadas tales como gestión de recetas, control remoto vía módem, gestión de alarmas, gráficos de tendencias, históricos de datos, análisis y tratamiento de información, etc. Sus interconexiones con la planta a través del computador se muestran en la figura 5. Además veremos más adelante que es posible incluir otra conexión directa adicional entre el computador que controla el SCADA y la planta para, programando el SCADA convenientemente, realizar un control redundante paralelo al del autómata, que sea capaz de detectar errores de funcionamiento, e incluso realizar un control de emergencia si ocurre esto.

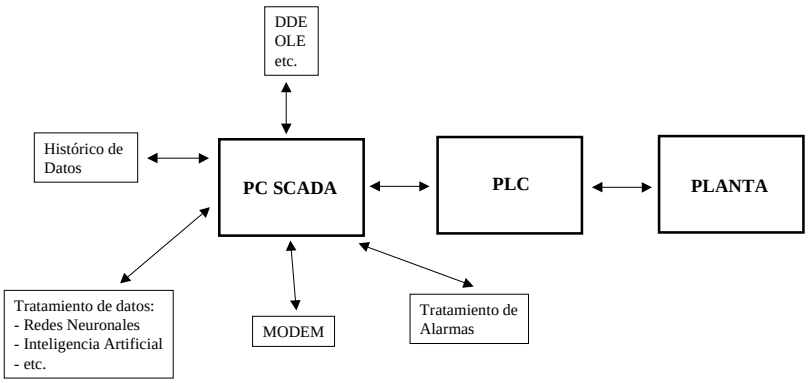


Figura 5: Diagrama de automatización supervisada con SCADA

En la figura 6 se muestra un ejemplo de planta monitorizada mediante SCADA. Con él las posibilidades de control y gestión de la planta mejoran de manera muy considerable [4], y además permiten realizar la implementación de la Red de Petri de supervisión mediante sus propias aplicaciones gráficas y de comunicaciones, para visualizar, a la vez que la planta, la evolución del proceso y de sus estados internos.

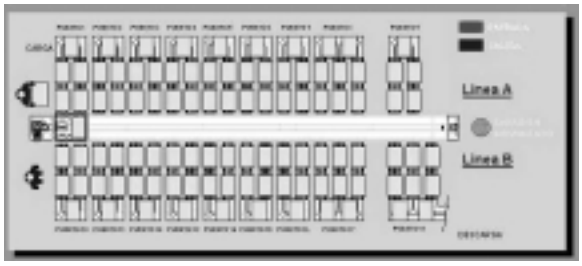


Figura 6: Planta industrial de montaje en estaciones de trabajo supervisada con SCADA

Así la representación de la evolución de la RdP es muy sencilla puesto que tan sólo hay que correlacionar los "lugares" y las "transiciones" con las posiciones de memoria correspondientes del autómata, para así visualizar el estado interno del sistema, que en ocasiones no puede ser visualizado con la representación de la planta.

Además, como veremos en el apartado 2.4.7, tenemos la posibilidad de diseñar sin coste adicional un sistema redundante supervisor para alguna zona especialmente importante o peligrosa. Para ello tan sólo tenemos que programar la RdP que hemos implementado en el SCADA para que "evolucione" según los eventos, en vez de tan sólo monitorizar la evolución del autómata. Así podemos comparar ambas evoluciones, la del autómata y la del computador sobre el que se ejecuta el SCADA, para detectar posibles conflictos o errores en el funcionamiento.

Todo esto se puede hacer igualmente sin la necesidad de un paquete informático SCADA, simplemente construyendo la aplicación informática a medida. Con ello evitamos el tener que comprar una licencia del SCADA comercial, y además tenemos la ventaja de poder realizar la aplicación más a medida de nuestras necesidades. Las desventajas son que hay que realizar la aplicación informática, incluyendo el sistema de comunicación con los autómatas programables, lo cual lleva mucho tiempo, y además no podemos beneficiarnos de las aplicaciones que ya lleva implementado el SCADA (drivers y controladores para PLCs y buses de campo comerciales, gestión de recursos de la CPU, etc.).

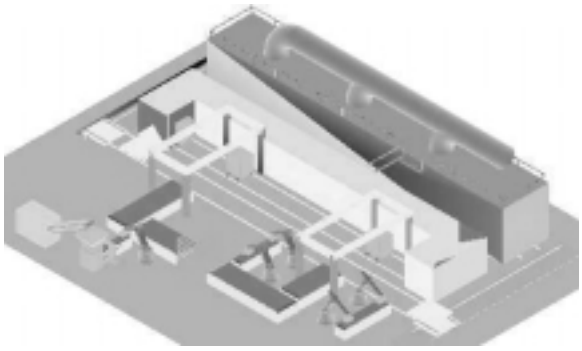


Figura 7: Supervisión de zona de planta industrial mediante programa a medida.

Con las actuales técnicas de programación, especialmente con metodología orientada a objetos, se consiguen buenas aplicaciones de RdP de supervisión en plantas que no disponen de un sistema SCADA incorporado, y sin excesivo esfuerzo. En concreto se

han obtenido aplicaciones satisfactorias programadas en Visual Basic, C++ y Java (ver figura 7).

2.2.4. Resultados

Como parte de los resultados se pueden mostrar dos de los ejemplos en los que se han simulado estas redes. El primero lo podemos ver en las figuras 8 y 9, donde tenemos una planta automática de producción de componentes para automóvil en la que las piezas deben pasar sucesivamente por los nueve puestos de producción superiores o por los nueve inferiores, empleando un único transbordador común a todos los puestos. Con la automatización que existía se daba prioridad tan sólo a los puestos más avanzados, con lo cual el sistema funcionaba pero de forma poco eficiente. Al implementarse la red de supervisión que modelaba el funcionamiento se pudo comprobar dónde y por qué aparecían los cuellos de botella que acaparaban el recurso compartido del carro transbordador.

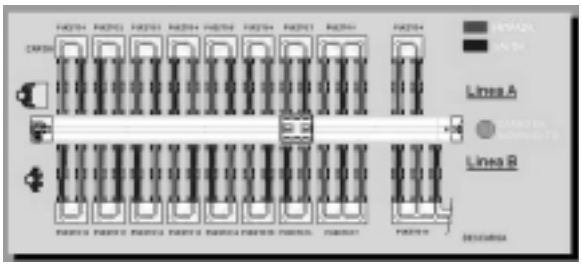


Figura 8: Automatización con recurso común de prioridades adaptativas

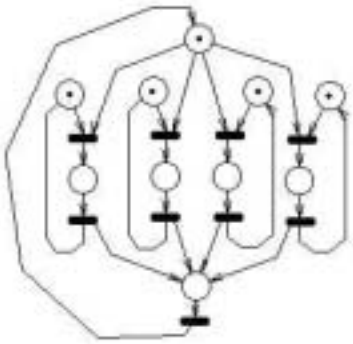


Figura 9: Red de Petri para recurso común

En las figuras 10 y 11 tenemos otro ejemplo de planta industrial, en este caso de producción de diversos tipos de ladrillos (este ejemplo real será ampliamente analizado en capítulos posteriores). Existen dos cruces de vías con puentes grúa, así como cuatro robots empleados todos ellos en dos tipos de procesos distintos y dos transbordadores que funcionan como sistemas de concurrencia a otros eventos. Todos estos elementos constituyen recursos compartidos. Con ello tenemos un caso claro de la complejidad del análisis global de ciertos sistemas. Pero además, en este caso, cuando se hizo el análisis de los datos recogidos en planta para la detección de errores mediante redes neuronales, se comprobó que las variables con más influencia en los errores eran estados internos

(más influencia que las entradas y salidas) con lo cual no era suficiente recolectar los valores de planta.

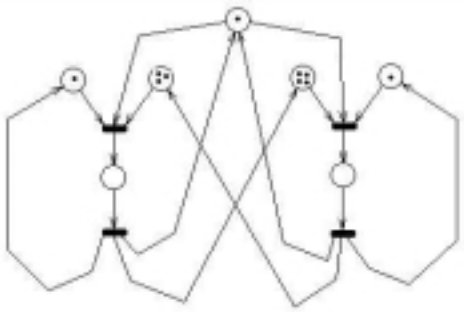


Figura 10: Red de Petri de sistema generador-consumidor con recurso compartido

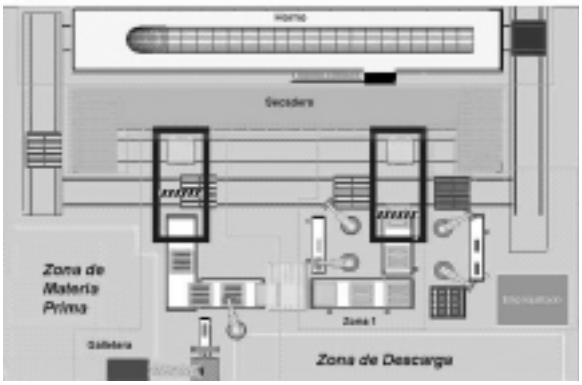


Figura 11: Automatización de planta con múltiples recursos compartidos

En algunas de las aplicaciones realizadas también ha dado muy buen resultado emplear estas redes no sólo para supervisión sino también para el control de los eventos discretos del proceso (como se explica en la sección 4 de este capítulo), al igual que se suele realizar con los paquetes SCADA comerciales, si bien resulta recomendable tener la posibilidad de seleccionar el control desde la propia RdP o desde alguna pantalla específica de control.

2.3. Sistema, automatización y sistema automatizado

Cuando se realizan las automatizaciones es frecuente comenzar a modelar el conjunto sistema-automatización directamente, en el cual el dispositivo encargado de la automatización, PLC, en función de las entradas y del programa que tiene produce las salidas de control adecuadas. En estos casos no se presta demasiada atención al funcionamiento del sistema.

Sin embargo en procesos complicados merece la pena desarrollar la automatización, pero basándose en un modelo que se realice sobre el funcionamiento del sistema. Empleando las habituales herramientas de modelado se pueden realizar por separado ambos modelos: el de la automatización (el programa que irá al autómata) y el de funcionamiento del sistema, tal cual se muestra en la figura 12. A continuación veremos las características y ventajas de este tipo de proceder.

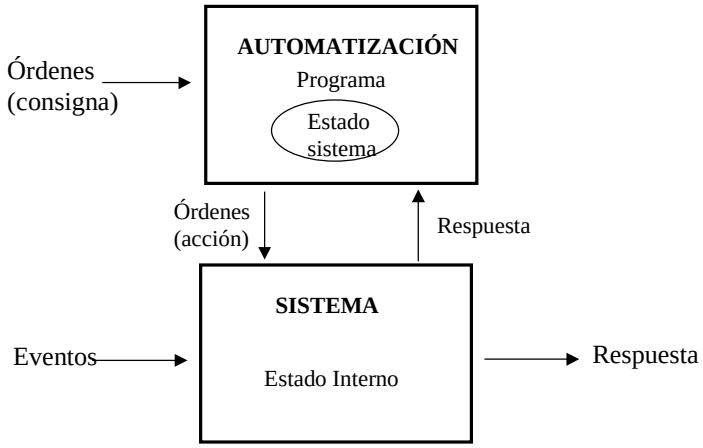


Figura 12: Modelado de la automatización y del sistema

2.3.1. Modelado del Sistema

Modelar el sistema consiste en realizar una representación de cómo van a evolucionar las salidas (que nos interesen) en función de la secuencia de entradas. Como ya hemos visto anteriormente para los procesos secuenciales, se debe conocer el estado del sistema para determinar cómo serán las salidas en función de las entradas en ese instante, teniendo en cuenta que también el estado interno evoluciona en función de las entradas. No nos entretendremos en las teorías de las máquinas de estados ni en las representaciones de Moore y de Mealy [22].

Pero sí es interesante analizar que el sistema recibe como entradas tanto los eventos que le sucedan (p.e., llegada de una pieza nueva) como las órdenes de actuación (o de acción) que le envía la automatización. En virtud de dichas entradas y de su estado interno el sistema produce unas respuestas y modifica su estado interno. De dichas respuestas, las que interesen son transmitidas a la automatización. Un evento puede ser prácticamente equivalente a una respuesta (ante el evento de llegada de una pieza se produce la respuesta de tener activo el sensor de llegada de piezas), con lo cual la automatización se puede enterar (indirectamente) de los eventos.

Otra cuestión interesante a la hora de realizar el modelado del sistema es que se puede realizar un modelo más o menos exacto en función de las necesidades. A veces es suficiente una aproximación simple al sistema, y otras veces puede ser crítico tener un modelo prácticamente exacto al sistema.

2.3.2. Automatización del proceso

La automatización del proceso consiste en diseñar un modelo, que luego pueda implementarse en algún dispositivo, que ante las respuestas del sistema y las consignas que le proporcione el usuario produzca las correspondientes órdenes al sistema para que éste evolucione como interesa.

Como entradas la automatización tiene las órdenes del usuario (que se puede entender como una consigna, continua o discreta) y la información que recibe del sistema. Como salidas tiene las órdenes de actuación al sistema.

La automatización es (habitualmente) secuencial, por lo que tendrá su estado interno que identifica su comportamiento en cada momento frente a las entradas. La información del estado del sistema puede estar contenida dentro de dicho estado interno de la automatización, o también puede recibirse del propio sistema como parte de la respuesta. Por ejemplo, si estamos automatizando un garaje con capacidad para 10 coches, la automatización puede levar la cuenta de los que van entrando y saliendo, o puede solicitar al sistema la información en cada momento, si es que el sistema dispone de un sensor del número de coches.

2.3.3. Confrontación de simulación y automatización.

El desarrollo del modelo del sistema previamente a la realización de la automatización que se incluirá en los dispositivos reales presenta varias ventajas:

- Al modelizar se tienen en cuenta los aspectos más interesantes del sistema y se obvian los que no afectan al funcionamiento global, incluso siendo más complejos muchas veces que los que realmente interesan. Así estaremos trabajando con un sistema simplificado (con el modelo útil) y nos podemos olvidar del resto.

- Para comprobar si la automatización es correcta se pueden "enfrentar" la automatización con el modelo del sistema en un programa de simulación de la herramienta empleada. Con ello, además, podemos hacer un análisis del comportamiento del sistema automatizado, es decir, podemos ver cómo será (idealmente) la producción.

- Pero además, lo mismo que se traduce al dispositivo de control automático la automatización realizada mediante la herramienta de modelado (en nuestro caso del Grafcet o de la RdP al autómata programable), también podemos traducir el modelo del sistema y de nuevo enfrentarlos. La característica adicional respecto al caso anterior es que ahora se tiene una simulación en tiempo-real dentro del propio dispositivo de control (ahora en vez de controlar la planta controla el modelo que tiene introducido de ella). De esta manera no sólo se comprueba que está bien realizada la automatización, sino también que está bien implementada en el dispositivo de control. Este sistema ha supuesto la reducción a casi la mitad (según estimaciones) del tiempo empleado en corrección de errores de implementación de la automatización de los dispositivos, en las industrias colaboradoras con la línea de investigación.

2.3.4. Ejemplo

Todo esto lo podemos entender gráficamente mediante un simple ejemplo. El ejemplo consiste en la automatización de una cinta transportadora en una planta dedicada a la fabricación de ladrillos. La descripción del sistema se puede ver en la figura 13.

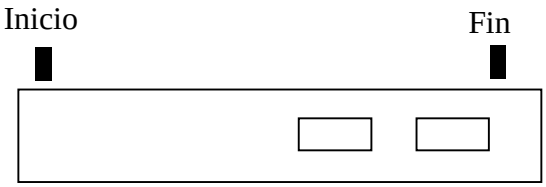


Figura13: Cinta transportadora para modelar y automatizar.

Existen dos sensores colocados en los extremo de la cinta, uno al principio de la misma (INICIO) y otro al final (FIN). La cinta tiene una capacidad máxima para tres piezas, y siempre que sea posible debe avanzar hasta que haya un ladrillo en el extremo final, de tal forma que pueda ser cogido por la automatización siguiente (por ejemplo un robot).

Automatización del proceso
Para llevar a cabo la automatización del funcionamiento de la cinta debemos tener claro qué entradas y qué salidas va a utilizar la automatización. Éstas se detallan en la siguiente figura:

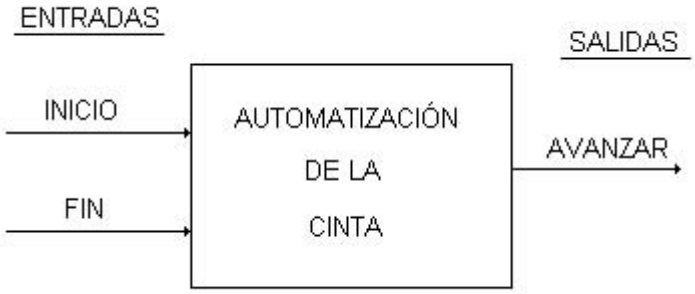


Figura14: Entradas y salidas en la automatización del ejemplo

Por lo tanto nuestra automatización no dispondrá de consigna (siempre va a funcionar igual), y las entradas van a ser sólo las señales de los sensores. Fijarse que como debemos saber el número de piezas, y dicha información no se recibe como entrada del sistema, entonces tendremos que generar y almacenar dicha información en el estado interno de la automatización. Con todo ello una solución puede ser la que se muestra a continuación en la Figura 15.

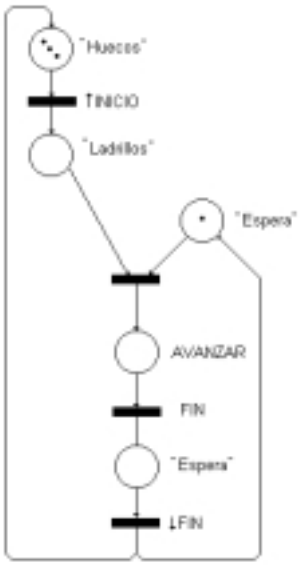


Figura 15: Automatización de la cinta transportadora.

Modelado del sistema

Una vez realizada la automatización de la cinta transportadora, el siguiente paso consiste en realizar la simulación del sistema (en este caso sólo nos interesan los sensores). Esas salidas del sistema se activarán en función de las órdenes procedentes de la automatización pero también en función de los eventos externos. Téngase en cuenta que el sistema cinta interactúa con el exterior (un proceso de fabricación mucho mayor), de manera que sobre la cinta se producen eventos procedentes de otras máquinas, robots, etc.

Los eventos, las entradas y salidas de la simulación son:

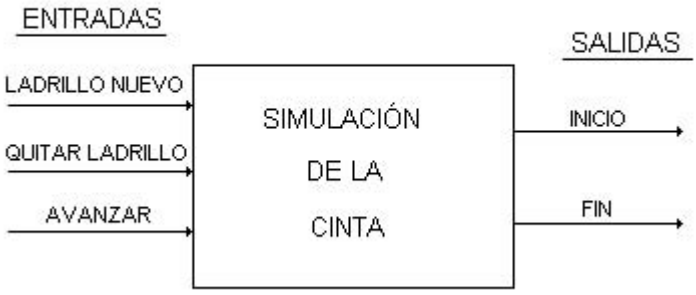


Figura16: Entradas y salidas en el modelo del ejemplo

La simulación de la cinta se ha realizado mediante la siguiente RdP:

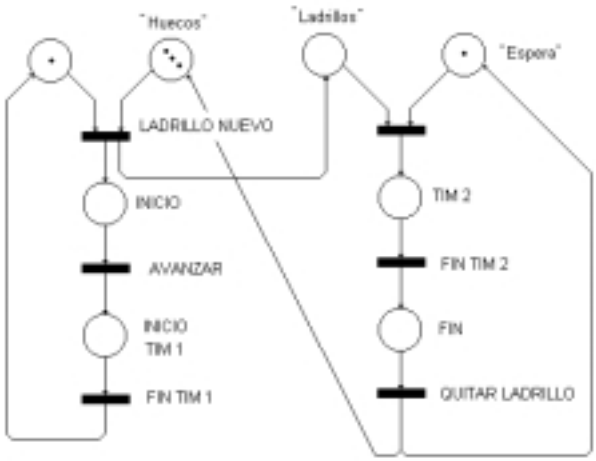


Figura 17: Modelo del sistema del ejemplo

El sistema parte de un estado inicial en el que sobre la cinta no existe ningún ladrillo. Cuando mediante el evento externo "Ladrillo Nuevo", se coloca un ladrillo en la cinta se simula el comportamiento del sensor "Inicio". Para ello se activa el sensor, y mediante un temporizador se simula el tiempo que permanece el sensor activado por el paso del ladrillo mientras la cinta se encuentra en marcha. De forma similar, cuando sobre la cinta se encuentra algún ladrillo, se temporiza el tiempo que tardaría el ladrillo en recorrer la cinta hasta la activación del sensor "Fin". Cuando el ladrillo llega al final de la cinta está en disposición de ser retirado, cosa que ocurrirá mediante el evento externo "Quitar Ladrillo".

Se debe observar en este modelo de la cinta que independientemente del número de ladrillos que se encuentren sobre la cinta, o del tiempo que hayan estado avanzando los ladrillos sin llegar al final, siempre tardan el mismo tiempo en recorrer la cinta desde que se da la orden de avanzar hasta que llega el primero. Puede verse por tanto que este modelo no es exacto, pero si es suficiente para el proceso que estamos automatizando, a partir de ahora trabajaremos tan sólo con este modelo. Evidentemente si se necesitase el modelo exacto habría que tener en cuenta el tiempo que cada ladrillo lleva avanzando para determinar con exactitud la distancia a la que se encuentran del final, y cuándo lo alcanzan.

Funcionamiento simultáneo de simulación y automatización

El siguiente paso de diseño consiste en realizar la fusión de la automatización y simulación de la planta en el autómata, con el fin de detectar anomalías en el funcionamiento y depurar el mismo. Esta fusión puede realizarse en la fase de las propias herramientas de modelado (Grafcet ó RdP) para llevarlas a un programa de simulación de dicha herramienta, o puede realizarse una vez traducidas esas herramientas al lenguaje del dispositivo. En este caso se ha realizado de ambas formas:

- se ha dibujado la RdP conjunta, comunicada mediante eventos (Figura 18). Son pocos los simuladores que permiten la utilización de eventos de entrada y salida

para coordinar subredes, pero con un poco de experiencia es muy sencillo sustituirlos por elementos estructurales (lugares, arcos y transiciones).

- directamente sobre el PLC, más concretamente un autómata CQM1 de OMRON, si bien sería análogo en cualquier otro tipo.

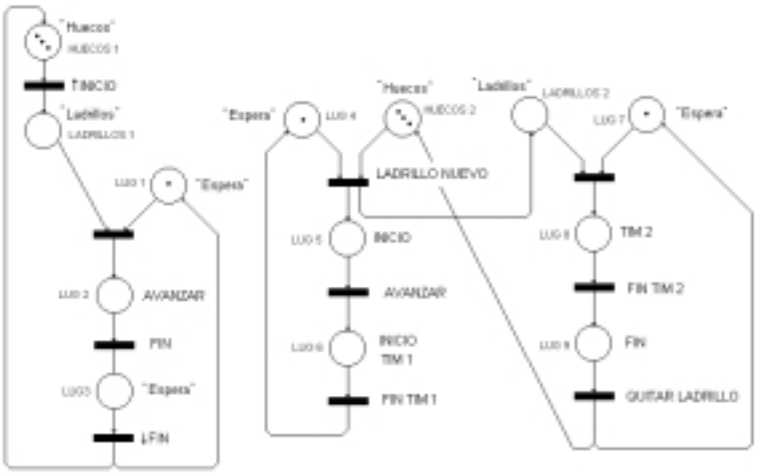


Figura 18: Simulación y automatización comunicadas mediante eventos

Para la realización sobre el PLC, primeramente se ha realizado la comprobación de la automatización de la planta, mediante la traducción de la RdP de automatización a lenguaje de autómata y realizando la comprobación con el programa introducido en el propio PLC. Una vez depurada la automatización se ha realizado lo mismo con el modelo del sistema. Posteriormente se han unido ambos, que en realidad se puede considerar como una modelización del sistema automatizado. Sus entradas y salidas son por tanto:

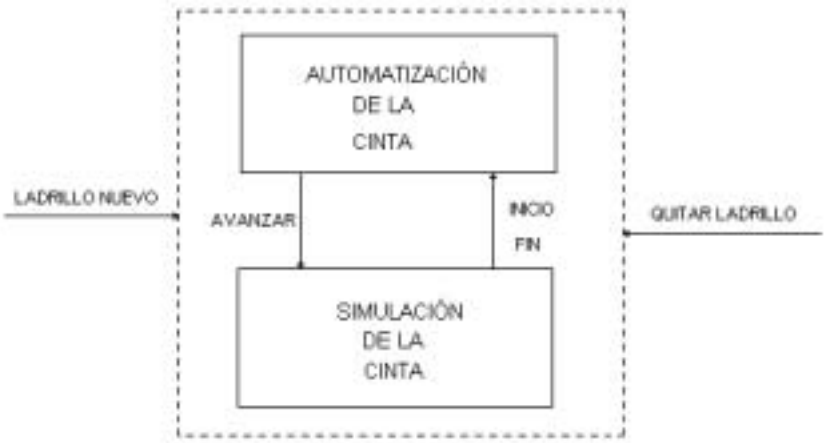


Figura 19: Entradas y Salidas del modelo del sistema automatizado

La automatización y la simulación unidas forman un todo que sólo posee las entradas de los eventos externos de "Ladrillo Nuevo" y "Quitar Ladrillo".

Implementación en PLC

El mapeado de memoria del ejemplo es el siguiente:

Entradas:	Salidas:	Variables:
Quitar ladrillo: IR 000.00 Ladrillo muevo: IR 000.01 Inicio: IR 000.02 Fin: IR 000.03	Avanzar: IR 100.00	Huecos1: DM 0100 Ladrillos1: DM 0101 Huecos2: DM 0102 Ladrillos2: DM 0103

Estados intermedios:	Bits auxiliares de detección de flancos:	Bits de propósito general:
Lug1: IR 016 Lug2: IR 017 Lug3: IR 018 Lug4: IR 019 Lug5: IR 020 Lug6: IR 021 Lug7: IR 022 Lug8: IR 023 Lug9: IR 024	Aux1: IR 025.00 Aux2: IR 025.01 Temporizadores: Timer1: TIM 000 Timer2: TIM 001	Activo primer ciclo de scan: SR 253.15 Bit de siempre a ON: SR 253.13 Bit de "mayor que": SR 255.05

Figura 20: Mapeado de entradas y salidas del ejemplo.

y el listado en mnemónico de la aplicación es el que se muestra a continuación:

```
00000  LD        253.15                  SCAN
00001  OR        000.15                  RESET
00002  MOV       #0001 016               #0001 LUG1
00003  MOV       #0000 017               #0000 LUG2
00004  MOV       #0000 018               #0000 LUG3
00005  MOV       #0001 019               #0001 LUG4
00006  MOV       #0000 020               #0000 LUG5
00007  MOV       #0000 021               #0000 LUG6
00008  MOV       #0001 022               #0001 LUG7
00009  MOV       #0000 023               #0000 LUG8
00010  MOV       #0000 024               #0000 LUG9
00011  MOV       #0000 DM0101            #0000 LADRILLOS
00012  MOV       #0003 DM0100            #0003 HUECOS
00013  MOV       #0000 DM0103            #0000 LADRILLOS2
00014  MOV       #0003 DM0102            #0003 HUECOS2

00015  LD        000.02                  INICIO
00016  OR        021.00                  LUG6.00
00017  OR        022.00                  LUG7.00
00018  DIFU      025.00                  AUX1

00019  LD        253.13                  ON
00020  CMP       DM0100 #0000            HUECOS #0000

00021  LD        255.05                  MAYOR
```

```
00022  AND       025.00                  AUX1
00023  INC       DM0101                  LADRILLOS
00024  DEC       DM0100                  HUECOS

00025  LD        253.13                  ON
00026  CMP       DM0101 #0000            LADRILLOS #0000

00027  LD        255.05                  MAYOR
00028  AND       016.00                  LUG1.00
00029  INC       017                     LUG2
00030  DEC       DM0101                  LADRILLOS
00031  DEC       016                     LUG1

00032  LD        017.00                  LUG2.00
00033  OUT       100.00                  AVANZAR

00034  LD        017.00                  LUG2.00
00035  LD        000.03                  FIN
00036  OR        024.00                  LUG9.00
00037  AND LD
00038  INC       018                     LUG3
00039  DEC       017                     LUG2

00040  LD        000.03                  FIN
00041  OR        024.00                  LUG9.00
00042  DIFD      025.01                  AUX2

00043  LD        025.01                  AUX2
00044  AND       018.00                  LUG3.00
00045  INC       016                     LUG1
00046  INC       DM0100                  HUECOS
00047  DEC       018                     LUG3

00048  LD        253.13                  ON
00049  CMP       DM0102 #0000            HUECOS2 #0000

00050  LD        255.05                  MAYOR
00051  AND       019.00                  LUG4.00
00052  AND       000.00                  LADRILLO_NUEVO
00053  INC       020                     LUG5
00054  DEC       019                     LUG4
00055  INC       DM0103                  LADRILLOS2
00056  DEC       DM0102                  HUECOS2

00057  LD        020.00                  LUG5.00
00058  AND       100.00                  AVANZAR
00059  INC       021                     LUG6
00060  DEC       020                     LUG5

00061  LD        021.00                  LUG6.00
00062  TIM       000 #0050               TIMER1 #0050

00063  LD        021.00                  LUG6.00
00064  AND       TIM000                  TIMER1
00065  INC       019                     LUG4
00066  DEC       021                     LUG6

00067  LD        253.13                  ON
00068  CMP       DM0103 #0000            LADRILLOS2 #0000
```

```
00069  LD        255.05                  MAYOR
00070  AND       022.00                  LUG7.00
00071  INC       023                     LUG8
00072  DEC       022                     LUG7
00073  DEC       DM0103                  LADRILLOS2

00074  LD        023.00                  LUG8.00
00075  TIM       001 #0070               TIMER2 #0070

00076  LD        023.00                  LUG8.00
00077  AND       TIM001                  TIMER2
00078  INC       024                     LUG9
00079  DEC       023                     LUG8

00080  LD        024.00                  LUG9.00
00081  AND       000.01                  QUITAR_LADRILLO
00082  INC       022                     LUG7
00083  DEC       024                     LUG9
00084  INC       DM0102                  HUECOS2

00085  END
```

Figura 21: Listado del programa de autómata en mnemónico del ejemplo

Más adelante, como parte de esta misma tesis, se analizará la forma de traducir automáticamente desde la RdP hasta el lenguaje del autómata programable o el de algún simulador (en este caso Matlab).

Monitorización

Puesto que éste es un ejemplo sencillo para indicar la forma de implementación de la automatización total, también se ha realizado la monitorización de la planta mediante el paquete SCADA FIX 32, conectado al autómata CQM1 de OMRON.

Para ello se han empleado los siguientes elementos:

- Driver OMR de comunicación con autómatas de OMRON.
- Creación de variables o bloques en la base de datos.
- Diseño de una pantalla de monitorización.

El primer elemento utilizado es el driver de comunicación para autómatas de OMRON. Este driver permite la creación de los "Pool Records" necesarios para poder acceder a las zonas de memoria del autómata y poder realizar la monitorización de la cinta en función de los valores de los bits y canales que se han utilizado en el programa del autómata.

La configuración del driver OMR se muestra en la siguiente figura:

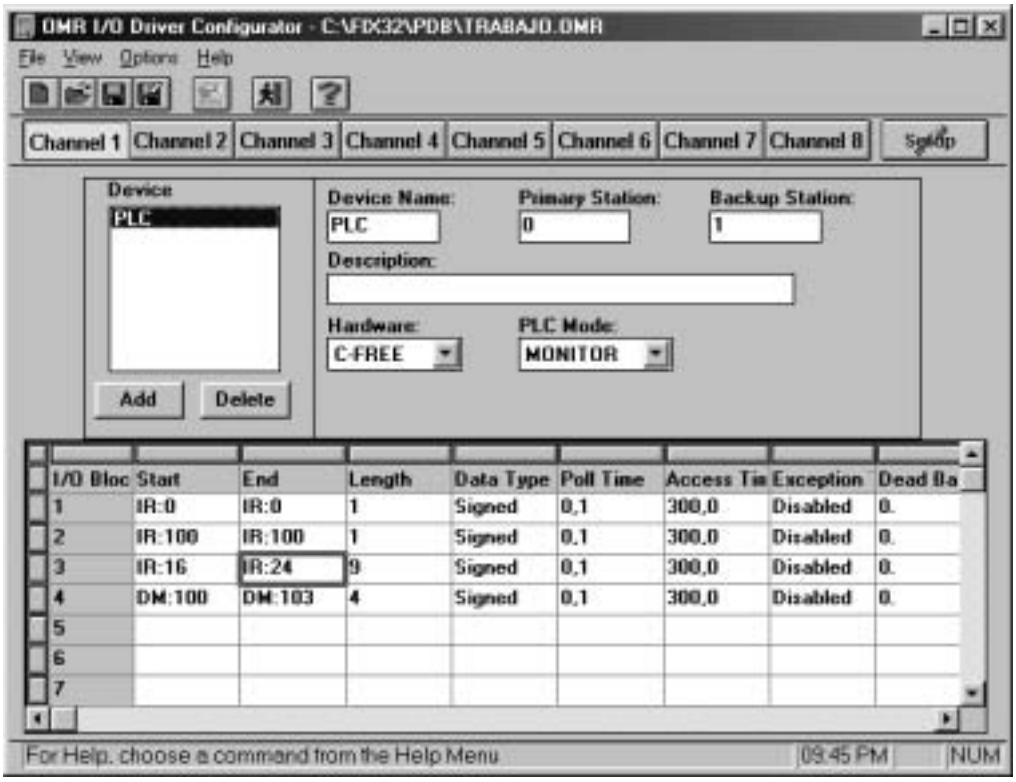


Figura 22: Configuración del driver de OMRON.

Fue necesaria la creación de los siguientes "Pool Records", que luego fueron asociados con bloques de la base de datos:

- IR 0: asociado al primer canal de entradas, para conocer el estado de las entradas, "Ladrillo nuevo", "Quitar ladrillo", "Inicio" y "Fin".
- IR 100: asociado al primer canal de salidas, para conocer el estado de la salida "Avanzar".
- IR 016 → IR 024: asociados a los lugares de la Red de Petri.
- DM 100 → DM 103: asociados a los DM's que almacenan el número de huecos y de ladrillos para la automatización y para la simulación de la cinta.

El siguiente paso es la creación de los bloques o variables de la base de datos. Se han empleado los siguientes bloques, asociados a los "Pool Records" creados anteriormente:

Tag Name	Type	I/O Device	I/O Address
MHUECOS	AI	OMR	PLC:DM:100
MHUECOS2	AI	OMR	PLC:DM:102
MLADRILLOS	AI	OMR	PLC:DM:101
MLADRILLOS2	AI	OMR	PLC:DM:103
MAVANZAR	DI	OMR	PLC:IR:100:0
MLADRILLO_NUEVO	DI	OMR	PLC:IR:000:0
MQUITAR_LADRILLO	DI	OMR	PLC:IR:000:1
MINICIO	DI	OMR	PLC:IR:000:2
MFIN	DI	OMR	PLC:IR:000:3
MLUG1	DI	OMR	PLC:IR:016:0
MLUG2	DI	OMR	PLC:IR:017:0
MLUG3	DI	OMR	PLC:IR:018:0
MLUG4	DI	OMR	PLC:IR:019:0

MLUG5	DI	OMR	PLC:IR:020:0
MLUG6	DI	OMR	PLC:IR:021:0
MLUG7	DI	OMR	PLC:IR:022:0
MLUG8	DI	OMR	PLC:IR:023:0
MLUG9	DI	OMR	PLC:IR:024:0

Figura 23: Bloques empleados en la aplicación SCADA.

El último paso es la creación de la pantalla de monitorización. Esta consiste en la monitorización de la evolución de la RdP que representa el estado interno del sistema. El aspecto de esta pantalla de monitorización, en un determinado instante es el siguiente:

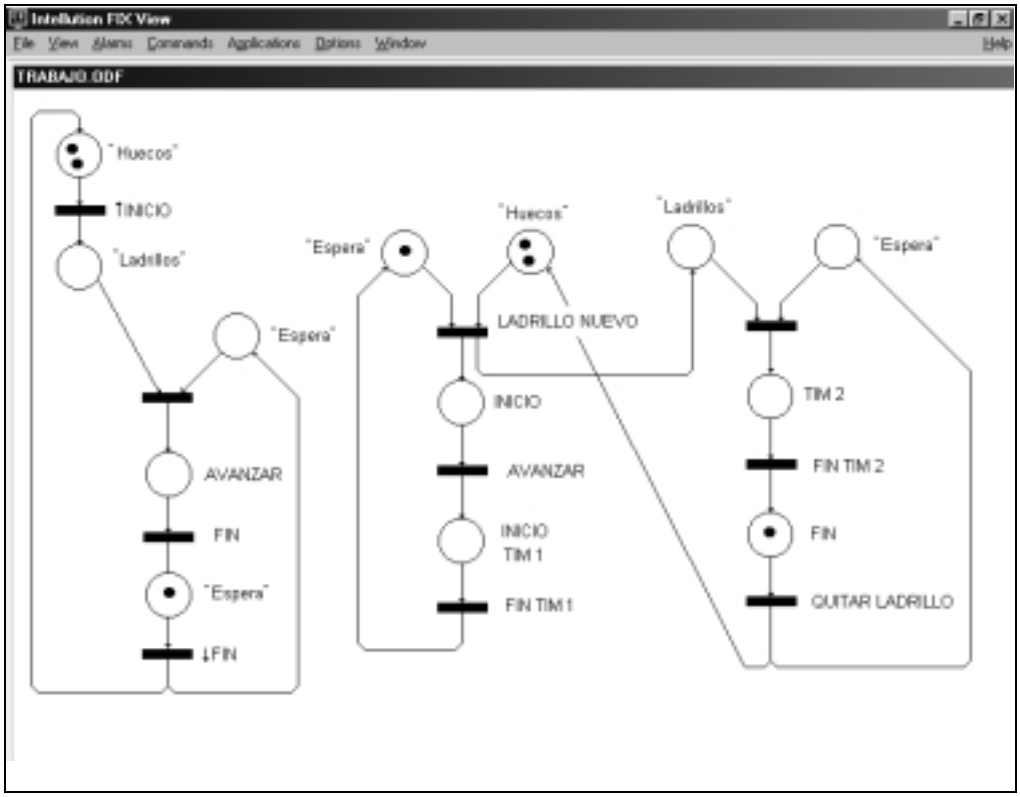


Figura 24: Pantalla de monitorización de la RdP del sistema en un paquete SCADA.

2.4. Automatización con PLC y SCADA

2.4.1. Introducción

Cuando una automatización es implementada por medio de una herramienta de descripción, todo el poder y propiedades de la herramienta son usados en el sistema. Hemos visto que las RdP ofrecen muchos estudios y trabajos que permiten deducir un alto número de propiedades de su comportamiento, y hay muchas herramientas y aplicaciones basadas en computador capaces de llevar a cabo rápida y fácilmente su análisis [23, 24] (ver el Tema 1). Al igual que en la programación orientada a objetos, el

modelado de sistemas por medio de RdP hereda algunas propiedades y características del comportamiento del sistema (red interpretada) que pueden ser deducidas desde las características de la red autónoma, independientemente del significado físico de la red empleada.

Sin embargo, los resultados de RdP autónomas sobre RdP interpretadas o temporizadas no pueden generalizarse, puesto que algunas otras propiedades dependen de la interpretación de la red. Esto es debido al hecho de que dentro del conjunto de marcados alcanzables en la red autónoma, algunos valores pueden estar fuera del conjunto de los válidos en la red no autónoma. Aquí, aunque debería desarrollarse un análisis de las propiedades de la red interpretada, también es muy interesante incluir aplicaciones de la simulación para ser capaces de analizar el comportamiento del sistema. De este modo el poder de cálculo de los sistemas informáticos puede ser usado para deducir el comportamiento en maniobras complicadas o en partes donde un análisis formal y metodológico es extremadamente complicado. Esta simulación también ayuda a desarrollar comportamientos avanzados del sistema, tales como controles redundantes, supervisión, control adaptativo, etc.

En esta sección se analiza la implementación de la automatización, supervisión y simulación de sistemas automáticos por medio de RdP en automatizaciones industriales, en las cuales se emplean en equipo los PLCs de control con los PCs de supervisión, para conseguir mejorar las prestaciones.

2.4.2. Automatización

Como hemos visto en la sección anterior, cuando queremos desarrollar un sistema complejo, el uso de una metodología muy estructurada que permita aplicar técnicas bien conocidas y resultados a nuestro sistema puede ser indispensable. La primera tarea a llevar a cabo es la descripción y modelado del sistema, que consiste en una representación simple pero válida y fiable de su comportamiento. Dependiendo del tipo de sistema, habrá algunas herramientas de modelado más apropiado que otras. Para los sistemas de eventos discretos, concurrentes, con paralelismos y sincronizaciones, las RdP constituyen una herramienta muy clara y poderosa. El paso siguiente consistía en llevar a cabo la automatización. El dispositivo donde la automatización es implementada recibe las señales del sistema como entradas, y envía las órdenes como salidas. También hemos visto que hay muchos dispositivos donde, en la práctica, las automatizaciones son implementadas, pero normalmente esta implementación no se realiza de un modo gráfico, sino por medio de lenguajes de programación. Y cuando es posible implementarlos de un modo gráfico, no es usual hacerlo con RdP. Así, es necesario traducir la RdP usada para el modelado al lenguaje del programa del dispositivo. Ésta es una tarea simple y muy mecánica, pero después se mostrará que es necesario tener cuidado con ciertos resultados.

La automatización está normalmente incorporada con un sistema de monitorización con el que la evolución de las entradas y las salidas del sistema y sus estados internos pueden ser expuestos de una manera gráfica. También se ha visto que no sólo puede ser muy interesante visualizar el sistema sino también la herramienta gráfica (la RdP) que

lo modela y lo describe [11]. Ello permite identificar errores en el funcionamiento, desarrollar el arranque del sistema, visualizar el proceso global, etc. Cuando el sistema de monitorización, además de leer los valores de las entradas, salidas y estados internos del sistema, puede modificarlos en tiempo real [6], entonces se trata de un sistema de supervisión.

2.4.3. Monitorización

La monitorización del sistema consiste en la capacidad de mostrar sus datos en tiempo real, incorporando formatos alfanuméricos y gráficos, y se realiza mediante un SCADA que consiste en un software de control de producción con acceso mediante comunicación digital a los elementos de control de la planta, y con interface gráfica de alto nivel con usuario [25]. Se pueden realizar a medida (programados en lenguajes de programación de propósito general), o bien se puede emplear alguno comercial de los muchos que existen en el mercado (Figuras 25 y 26).

Si bien la monitorización del sistema es imprescindible por la facilidad y fiabilidad que ofrece de manejo [11], es poco lo que aporta innovador a la automatización. Sin embargo los SCADAs no sólo realizan monitorización, sino también supervisión, adquisición y almacenamiento de datos, y otras tareas adicionales importantísimas (tratamiento de datos on-line, comunicación con base de datos, operaciones sobre hojas de cálculo, control remoto, etc.) que permiten un control mucho más potente.

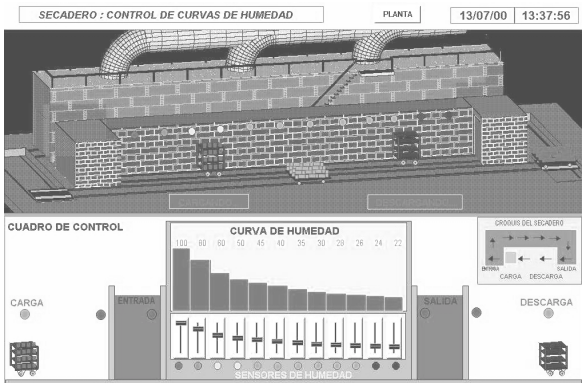


Figura 25: Aplicación SCADA de un proceso de producción industrial.

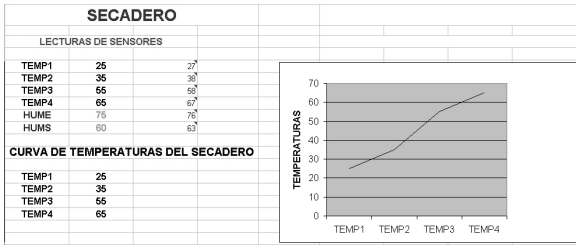


Figura 26: Utilización de hoja de cálculo y base de datos mediante SCADA.

2.4.4. Simulación

La simulación se emplea muchas veces como método de análisis del sistema de producción. Sin embargo el análisis por simulación sólo es preciso cuando se realiza de forma exhaustiva, por lo que es conveniente realizar, siempre que sea posible, análisis de tipo cualitativo.

La simulación del proceso puede ser también muy interesante para conseguir diseños de altas prestaciones. Al simular el proceso podemos detectar errores de funcionamiento antes de que se produzcan. Por otro lado, y esto es muy importante, podemos comparar el funcionamiento real con la simulación, para detectar desviaciones del funcionamiento de la planta.

Los propios SCADAs disponen a menudo de herramientas de simulación, por lo que no hace falta recurrir a otras aplicaciones para ello, como se muestra en el ejemplo de la Figura 27.

Figura 27: Simulación de proceso industrial con SCADA

También se puede recurrir a otras aplicaciones de simulación, desarrolladas a medida o comerciales. Entre estas últimas destacan las aplicaciones de simulación para realidad virtual [26], cada vez más potentes y avanzadas. Aplicaciones de simulación se han desarrollado en el marco de este trabajo en varios lenguajes de programación (incluyendo realidad virtual), como se verá posteriormente.

Otra forma de simulación consiste en implementar en el propio PLC un programa de simulación de la planta que funcione conjuntamente a la automatización, pero sin estar conectada la planta. Puede parecer un inconveniente de este método el que la simulación no sea gráfica, sino en el entorno del PLC, pero podemos hacerla gráfica simplemente empleando la monitorización sobre el programa de simulación, igual que si fuese sobre la planta real.

Además, como ya hemos visto anteriormente, otra gran ventaja que presenta este método es que las RdP que empleemos para desarrollar esta simulación se pueden juntar a las de la automatización para conseguir un modelo conjunto del sistema planta-automatización, que podemos correr sobre cualquier programa de simulación de redes

de Petri. Este método es además muy interesante para comprobar el correcto funcionamiento del programa del PLC e incluso de la monitorización y la simulación.

2.4.5. Supervisión

La supervisión consiste en la capacidad de monitorización junto a la de cambiar los datos y programas de los elementos del sistema desde el terminal [4]. Con el SCADA de las Figuras 25 y 27 podemos modificar desde la pantalla gráfica las consignas de la temperatura del horno, para que el sistema real evolucione hasta obtener la curva de temperatura que necesita [12]. Igualmente se puede hacer desde la hoja de cálculo (Figura 26) o desde una base de datos, puesto que se emplean tanto para monitorizar como para supervisar.

2.4.5.1. Implementación de Supervisión y Simulación

La supervisión (y por consiguiente también la monitorización) normalmente se lleva a cabo por medio de una computadora, bien usando un lenguaje de programación (generalmente visual y orientado a objetos), o por medio de una aplicación de computador SCADA. La ejecución con un lenguaje de computadora es más abierta, y permite más opciones, pero su desarrollo y modificación también son más caros. El uso de SCADAs proporciona la ventaja adicional de incluir los drivers y las herramientas de la programación.

La simulación de la RdP simplemente consiste en programar la evolución de la propia RdP de una forma controlada por el usuario en lugar de por los eventos del sistema. Generalmente es implementada por medio de una computadora, y con el mismo sistema usado para llevar a cabo la monitorización (de hecho, la simulación consiste en una monitorización de la evolución del modelo programado en la computadora, controlado por el usuario).

Por consiguiente, la simulación por medio de un lenguaje de programación es similar a la programación llevada a cabo para la automatización, excepto que las entradas y salidas son ahora simuladas por medio de las variables internas.

La supervisión y la simulación por medio de una aplicación SCADA dependen de la aplicación empleada, pero es muy similar a la utilización de un lenguaje de programación de alto nivel. La implementación de un pequeño sistema a modo de ejemplo, como el mostrado en la Figura 28, realizada con un SCADA típico (en este caso el empleado a lo largo de esta tesis) podría ser realizada como se muestra en la propia Figura 28 y en el programa de la Figura 29.

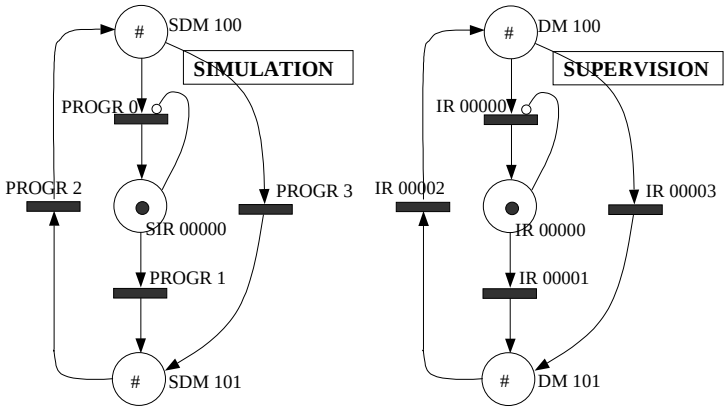


Figura 28: Supervisión y simulación con aplicación SCADA.

El gráfico anterior corresponde a una pantalla de SCADA con las dos RdP, una para la supervisión del sistema y otra para la simulación, según el programa siguiente, que es similar a un programa de automatización pero en el lenguaje de programación del SCADA.

```
Tag Name   Type              Device  Address
DM100      Analog Input      PLC     DM100
DM101      Analog Input      PLC     DM101
CONDIC     Boolean
IR10000    Digital Input     PLC     IR10000
IR00000    Digital Output    PLC     IR00000
IR00001    Digital Output    PLC     IR00001
IR00002    Digital Output    PLC     IR00002
IR00003    Digital Output    PLC     IR00003

S_DM100    Analog Register   SIM 1
S_DM101    Analog Register   SIM 2
S_CONDIC           Boolean
S_IR00000  Digital Input     SIM 5:0
S_CONDIC2          Digital Output      SIM 11:0
S_IR00001  Digital Register  SIM 5:1
S_IR00002  Digital Register  SIM 5:2
S_IR10000  Digital Register  SIM 4:0

PROGR0     Program
          0 IF   S_DM100  <       0,50 GOTO 4
     1 IF   S_IR10000  >       0,50 GOTO 4
     2 CLOSE     S_IR10000
     3 ADDOUT        -1,00 TO S_DM100
PROGR1             Program
          0 IF   S_IR10000  <       0,50 GOTO 3
     1 ADDOUT        1,00 TO S_DM101
     2 OPEN      SIR10000
PROGR2             Program
          0 IF   S_DM101  <       0,50 GOTO 3
     1 ADDOUT        1,00 TO S_DM100
     2 ADDOUT        -1,00 TO S_DM101
PROGR3             Program
          0 IF   S_DM100  <       0,50 GOTO 3
     1 ADDOUT        1,00 TO S_DM101
```

```
2 ADDOUT        -1,00 TO S_DM100
```

Figura 29: Programa para supervisar y simular las RdP anteriores mediante aplicación SCADA

2.4.6. Control con SCADA

Pero los SCADAs disponen además de métodos de control de proceso y automatización, tal cual se haría con un sistema informático [27] (figura 30).

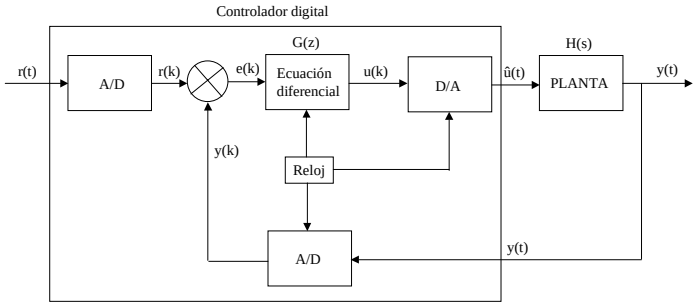


Figura 30: Esquema general de control en lazo cerrado con sistema informático.

Muchos de estos sistemas disponen de bloques de control PID (Figura 31), y de la posibilidad de implementar otros sistemas de control más complicados. Además esos bloques de control pueden combinarse en estructuras realimentadas simples (Figura 32) o más complejas (maestro esclavo de la Figura 33) tal como se haría con cualquier otro sistema de control [19].

$$\Delta y_n = K_P \cdot \beta \cdot \left(E_n - E_{n-1}\right) + \frac{T}{T_I} \cdot \left(F_n - y_{n-1}\right) +$$

$$+ \frac{T \cdot K_P \cdot E_n}{T_I} + \frac{K_P \cdot T_D \cdot \gamma}{6 \cdot T \cdot \left(T_D \cdot \alpha + 1\right)} \cdot \left(E_n + 3E_{n-1} - 3E_{n-2} - E_{n-3}\right)$$

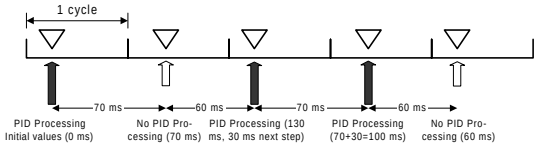


Figura 31: Funcionamiento de los bloques PID del SCADA.

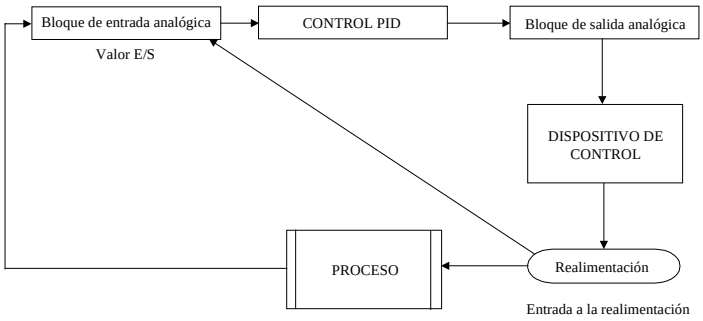


Figura 32: Control simple en lazo cerrado con bloques PID de un SCADA.

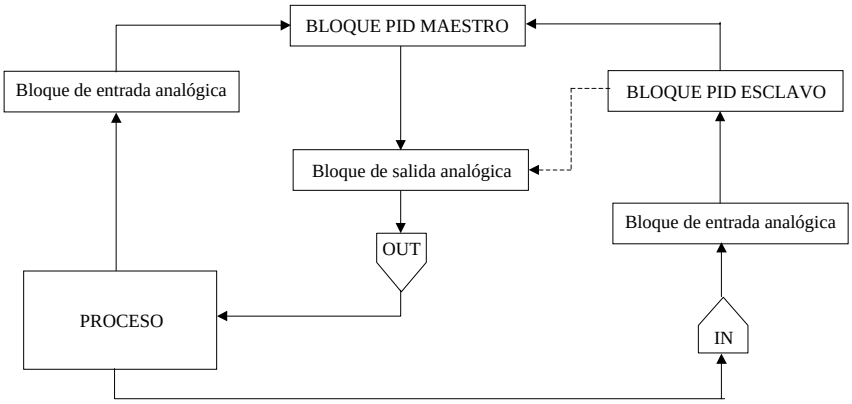


Figura 33: Control maestro-esclavo con bloques PID de un SCADA.

2.4.7. Control Supervisor Redundante Adaptativo

Acabamos de ver que puesto que un SCADA es usado para monitorizar y supervisar el sistema, podría ser interesante usarlo para simular el control (Figura 36). Esto no es difícil de desarrollar porque el uso de SCADAs incluyen elementos de control como por ejemplo bloques PID (aunque también este control puede desarrollarse numéricamente sin necesidad de estos bloques).

De esta manera, una comparación entre el control real y el control simulado nos permite detectar cualquier fallo en el control. Este control redundante podría implementarse en el PLC, pero esto podría incrementar el tiempo del ciclo, y por consiguiente empeoraría las propiedades dinámicas del control.

El SCADA también puede usarse para simular el proceso. Esto nos permite comparar los sistemas real y virtual o simulado (comparación 2 en Figuras 36 y 37), y entonces detectar fallos en el modelo del sistema, o fallos en el funcionamiento del sistema. Al igual que en la primera comparación, esta simulación puede hacerse también en el control PLC, pero suele ser mejor hacerlo en el SCADA (para evitar incrementar el tiempo de respuesta del control). Aún así, cosa importante, si se decide realizar desde el PLC (porque le sobran recursos) se tiene la ventaja de que puede estar ya disponible, si es que se ha realizado la automatización tal como se indicaba en el apartado anterior, donde se proponía simular (off-line) el funcionamiento de la automatización mediante su ejecución simuntánea (enfrentada) al modelo del sistema implementado también en el PLC. De esta manera el PLC controla la simulación en tiempo real de la planta, y puede ser verificado el funcionamiento correcto del control del PLC. De cualquier modo, esta verificación del control del PLC también puede llevarse a cabo usando la simulación de la planta implementada en el SCADA. En ambos casos, el sistema virtual puede ser monitorizado en el SCADA durante la verificación del control PLC.

Este sistema redundante constituye un control avanzado que mejora substancialmente la ejecución obtenida. Puede detectar fallos en la automatización, e incluso funcionar

correctamente cuando ocurran esos fallos, puesto que el control redundante puede manejar el sistema, y es también capaz de detectar errores en el modelado del sistema o su comportamiento (Fig. 35 y 36)[12]. Esta última característica también permite la modificación en tiempo real de los parámetros de la automatización y de la simulación (control adaptativo, Fig 37) para optimizar el sistema [19].

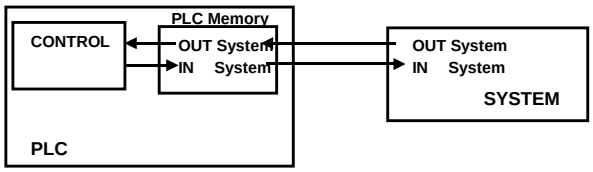


Figura 34: Control habitual con PLC

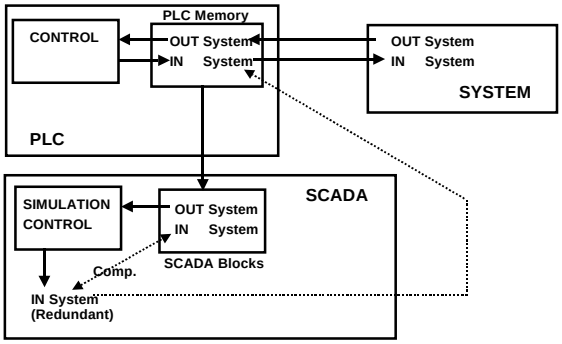


Figura 35: Detección de errores mediante control con PLC y simulación con SCADA

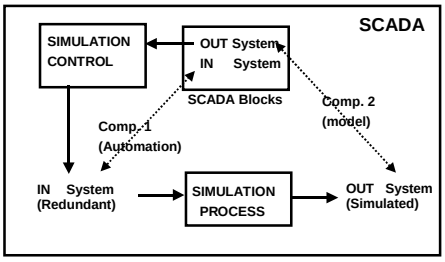


Figura 36: Simulación del control y del proceso con el SCADA

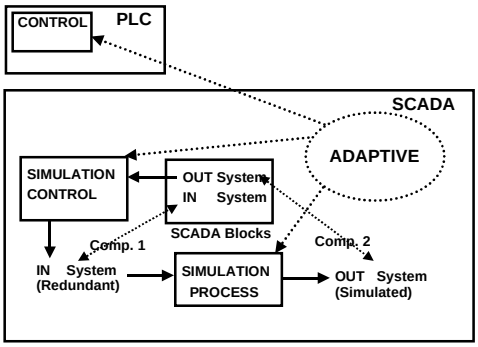


Fig. 37: Control adaptativo y redundante

Con todo esto vemos que se puede pasar de típico esquema en el que el sistema es controlado por un PLC y monitorizado y supervisado por un SCADA, a otro tipo de arquitectura, que enriquece sustancialmente a la anterior y la hace mucho más robusta sin a penas incrementar el precio, en la que también se simula con el PLC y/o con el SCADA, y se realiza la automatización sobre el SCADA.

2.5. Conclusiones

Como conclusión cabe destacar las ventajas de que las automatizaciones se basen en algún sistema formal y metodológico (se proponen redes de Petri pero hay otros, que incluso en algunas aplicaciones pueden ser más apropiados). No siempre se usan, pero en todo caso siempre existe una equivalencia de la automatización usada con otra basada en tales métodos, y al emplearlas disponemos de todas las prestaciones que ofrecen. Además la monitorización de la red de Petri que dirige el automatismo, se puede realizar a muy bajo coste (si se emplea un SCADA, sin coste adicional), y permite una supervisión global más completa y útil que la que se obtiene monitorizando sólo la planta.

Con esto se consiguen unas prestaciones muy interesantes: se dispone de una representación gráfica muy clara e intuitiva con la que poder conocer el estado interno del sistema (no sólo sus entradas y salidas) y su evolución, se pueden emplear los amplios estudios existentes sobre RdP para detectar fallos y mejorar prestaciones en el programa del autómata, se puede ampliar la automatización con las aplicaciones que ofrece el SCADA (especialmente en partes críticas), etc. Esta aplicación se puede realizar sobre el paquete SCADA de monitorización del proceso, si es que se emplea uno en la planta, y si no es así se puede implementar como una aplicación específica a medida programada en lenguajes orientados a objetos.

Otra conclusión importante es que el empleo de un modelo del sistema a automatizar, empleando las mismas herramientas que se emplearán posteriormente en la automatización, permite, aparte de la evidente posibilidad de trabajar con las características que nos interesen del sistema sin entretenernos en las superfluas, detectar errores en el diseño de la automatización, comprobar el sistema automatizado (sistema más automatización) empleando los simuladores correspondientes a las herramientas, y sobre todo, garantizar el correcto funcionamiento de la automatización, al comprobarlo

off-line pero en tiempo-real sobre el modelo del sistema implementado sobre el mismo dispositivo de control.

Además esa implementación del modelo del sistema (sin automatizar) sobre el dispositivo empleado para la automatización, puede servir también para detección de errores en el propio sistema (por envejecimiento o por averías). Para ello se emplea una forma de control supervisor redundante empleando PLC y SCADA, así como elementos adicionales para realizar las comparaciones. Dicha propuesta ha sido implementada en una de las empresas colaboradoras con la línea de investigación, con un resultado muy satisfactorio. La implementación en empresa real era necesaria para determinar el método más apropiado para detectar a partir de los datos obtenidos en el funcionamiento cuándo existía una desviación considerable de los datos esperados y los reales, que indicase que podía ocurrir un error productivo. De ahí se dedujo que variaba mucho de una automatización a otra, pero el empleo de redes neuronales con el suficiente entrenamiento ha proporcionado buenos resultados en todos los sistemas analizados.

2.6. Referencias

[1] Xing KY., Hu BS., Chen HX., "Deadlock avoidance policy for Petri-net modelling of flexible manufacturing systems with shared resources", IEEE Trans on Automatic Control 1996, 41 (2) ,289-294 (1996)
[2] Aström, K.J., and B. Wittenmark (1997). Computer controlled systems. Prentice-Hall, Englewood Cliffs, New Jersey.
[3] Aström, K.J., C.C. Hang, P. Persson and W.K. HO (1992). Towards intelligent PID control. Automatica, 28, 1-9.
[4] Balcells, J. and J.L. Romeral (1997). Automatas programables. Marcombo Boixareu Editores, Barcelona.
[5] Belli F, Grosspietch KE., "Specification of fault-tolerant systems issues by predicate/transition nets and regular expressions: Approach and case study", IEEE Trans on Software Engineering, 17 (6), 513-526 (1991)
[6] Burns, A. and A. Wellings (1996). Real-Time Systems and Programming Languages. Addison-Wesley, California.
[7] David R., "Grafcet: A powerful tool for specification of logic controllers", IEEE Trans on Control Systems Technology, 3 (3) , 253-268 (1995)
[8] Ezio, F. (1993). Ceramic technology. Iberian Faenza Editrice, Milan.
[9] Giua A, DiCesare F, Silva M., "Petri net supervisors for generalized mutual exclusion constraints", 12th IFAC World Congress Sydney, 1 , 267-270 (1993)
[10] Jensen K., "Coloured Petri nets: basic concepts, analysis methods and practical use", EATCS Monographs on Theoretical Computer Science, Springer, Berlin (1994)
[11] Jiménez E., "Redes de Petri de Supervisión y Simulación de Procesos Industriales Automatizados" XXI Jornadas de Automática CEA-IFAC, (2000)
[12] Jiménez E., Miruri JM., Martínez de Pisón JF., Gil M., "Supervised Real-Time Control with PLCs and SCADA in Ceramic Plant", 6th IFAC Workshop on Algorithms and Architectures for Real-Time Control, 1 , 221-226 (2000)

[13] Koivo, H.N. and J.T. Tanttu (1991). Tuning of PID controllers: survey of SISO and MIMO techniques. IFAC Intelligent tuning and adaptive control Proceedings, Singapore, pp. 75-80.
[14] Laplante, P.A. (1993). Real-Time Systems Design and Analysis: An Engineer Handbook. IEEE Press.
[15] Lee DY, DiCesare F.., "Scheduling flexible manufacturing systems using Petri nets and heuristic search", IEEE Trans on Robotics and Automation, 10 (2) , 123-132 (1994)
[16] Levi, A. (1990). Real-Time System Design. McGraw-Hill, New York.
[17] Malpica, J. A. (1998). Introducción a la teoría de autómatas. Ed. Universidad de Alcalá, Madrid.
[18] Morriss, B. (1999). Programmable Logic Controllers. Prentice Hall, New Jersey.
[19] Ogata, K.(1998). Engineering of Modern Control. Spanish American Prentice Hall, Mexico.
[20] Silva, M. (1985). Las Redes de Petri: en la automática y la informática. AC, D.L., Madrid.
[21] Villarroel JL., Muro P., "Using Petri net models at the coordination level for manufacturing systems control", Robotics and Computer-Integrated Manufacturing, 11 (1), 41-50 (1994)
[22] Herbert Taub, "Circuitos digitales y microprocesadores", Mc Graw Hill, 1984
[23] N. Konstas, S. Lloyd, H. Yu, C. Chatwin. Generic Net Modelling Framework for Petri Nets. IASTED Intelligent Systems and Control ISC'99 (1999)
[24] W.T. Goh, Z. Zhang. Autonomous Petri-Net for Manufacturing System Modelling in an Agile Manufacturing Environment. IASTED International Confer. Robotics and Applications 1999.
[25] Mitormat S.L. Terrassa. Manual HTERM, 1993.
[26] Pascual Gonzalez López, Jesús García-Consuegra Bleda (1998). Informática Gráfica. Ediciones de la Universidad de Castilla-La Mancha.
[27] Paredes, P., El PC en aplicaciones industriales. Automática e instrumentación, nº 271, pag. 73, febrero 1993.

Tema 3

IMPLEMENTACIÓN DE LAS AUTOMATIZACIONES EN LOS DISPOSITIVOS INDUSTRIALES DE CONTROL AUTOMÁTICO.

3.1. Implementación en los dispositivos

3.1.1. Introducción

Cuando un sistema físico es modelado, las restricciones debidas a las limitaciones tecnológicas deberían ser incluidas como restricciones en la interpretación de la RdP, puesto que el comportamiento del sistema implementado puede ser ligeramente diferente del comportamiento en el sistema ideal usado en el modelo. Y estas pequeñas diferencias, que normalmente no tienen importancia, a veces pueden llevar al control automático a una operación errónea. Por ejemplo dos eventos simultáneos pueden ser detectados por nuestro autómata en momentos diferentes, pueden ocurrir retrasos en las entradas y salidas, la evolución de las marcas podría no suceder en un punto particular, etc. Por consiguiente, es muy importante saber el comportamiento real del sistema físico donde el control automático será implementado.

También, si queremos usar la red eficazmente en el modelado, automatización, o simulación, debería ser interpretado correctamente el significado de nuestro modelo en la fase de diseño, para que corresponda perfectamente al comportamiento de nuestro sistema. Si es posible realmente implementar la RdP interpretada, es decir, desarrollar la red que intrínsecamente incorpora las restricciones debidas a la interpretación, esta red se usará como el propio sistema, bien para determinar las propiedades o bien llevar a cabo la implementación. Otras veces, será necesario usar las RdP como la base del modelado, pero introduciendo algunas características adicionales que el sistema real presenta para conseguir el modelo del sistema real. En este caso, será necesario implementar esas funciones adicionales a la RdP, y se deberá prestar atención al trasladar el comportamiento de la RdP usada al modelo (o el sistema real).

Por tanto la aportación de las siguientes secciones de este capítulo consisten en el análisis de los efectos en los sistemas reales en los que es implementado el control, así como la metodología para su correcta implementación, y algunas de las características especiales implementadas en RdP o con ellas para conseguir un modelo más similar al real.

3.1.2. Implementación de las RdP en los Dispositivos de Automatización

Como se acaba de comentar, la RdP es usada principalmente como una herramienta en el modelado y automatización del sistema, y después esa automatización es implementado bien sobre un PLC (controlador lógico programable) [6] o en una computadora o IPC (Industrial-PC). Cuando se traduce la RdP al dispositivo de control es esencial conocer perfectamente el lenguaje de programación empleado (o la aplicación del computador), así como la operación interna del sistema de control (el PLC o el IPC [4]). La falta de conocimiento de cualquiera de ellos podría llevar a un funcionamiento diferente del esperado. Entonces el dispositivo de control podría realizar una acción errónea (mejor dicho, no deseada), pese a cumplir las instrucciones.

Normalmente es aconsejable distinguir entre la fase de modelado y la de implementación. Es decir, desarrollar primero la RdP que modela el comportamiento, incluso cuando es una RdP con funciones adicionales debido a alguna característica especial del sistema, y después traducir ese modelo al lenguaje del dispositivo de control (sin tener ya en cuenta la planta, sino sólo su modelo). Si esto no se hace, a menudo pueden modificarse algunas formas de comportamiento en la fase de traducción, mejorando algunas partes, pero dañando por otro lado otras.

A modo de ejemplo se traduce la pequeña RdP de la Figura1. Para implementar esta RdP, es decir, para traducirla al sistema de programación, el primer paso consiste en elaborar un mapa de las entradas y salidas del sistema de control, y asociar una posición de memoria o una variable interna a cada valor, de acuerdo con su funcionamiento. Por consiguiente, los valores binarios pueden representarse por una variable booleana o por un bit de memoria, mientras las otras requieren una variable entera (o un contador) o una palabra de memoria (o canal).

La implementación de esa RdP en el lenguaje del PLC depende del tipo y modelo del PLC, así como del sistema de programación (casi todos los PLCs permiten código mnemónico y ladder, y algunos otros, también GRAFCET o incluso herramientas gráficas más avanzadas). Esto podría realizarse de la forma que se indica en la tabla de la figura 2, el programa de la figura 3 y el diagrama de contactos de la figura 4.

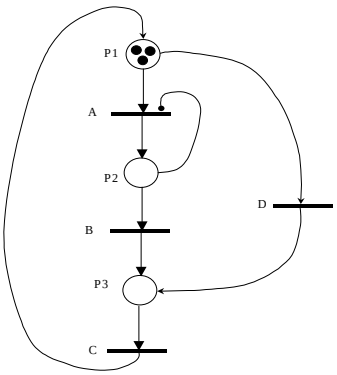


Figura 1. RdP elemental de ejemplo para traducción.

Variable	Type	Memory address
A	bit	IR 000.00
B	bit	IR 000.01
C	bit	IR 000.02
D	bit	IR 000.03
P1	integer	DM 0100
P2	bit	IR 100.00
P3	integer	DM 0101

Figura 2: Tabla con el mapeado de memoria

```
LD  253.13
CMP         DM0100 #0000
LD  255.05
OUT         TR0
AND NOT     100.00
LD  000.00
OR  016.00
AND LD
SET         100.00
@DEC        DM0100
LD  TR0
AND         000.03
INCDM0101
DEC         DM0100
```

```
LD          100.00
AND         000.01
@INC        DM0101
RSET        100.00
LD          253.13
CMP         DM0101 #0000
LD          255.05
AND         000.02
@INC        DM0100
@DEC        DM0101
LD          253.15
MOV         #0003 DM0100
END
```

Figura 3: Implementación en mnemónico del programa de la Figura 1.

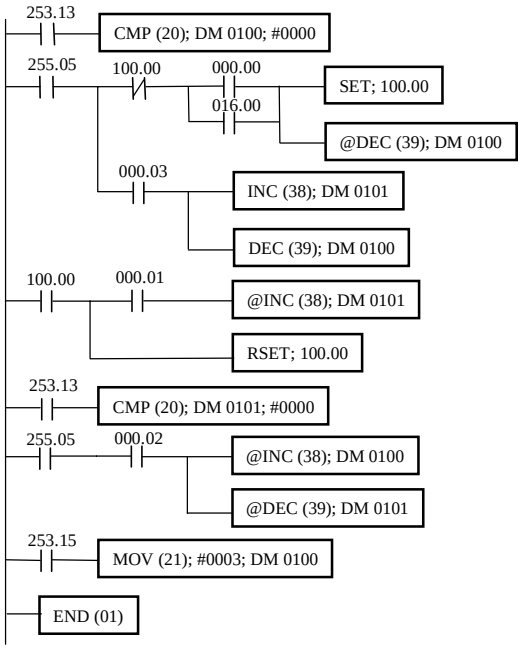


Fig. 4. Implementación en Ladder correspondiente a la Figura 1.

La automatización empleando una computadora no es tan frecuente. Puede llevarse a cabo por medio de una aplicación de control donde la programación dependa de la

aplicación correspondiente (aquí incluso puede programarse directamente en GRAFCET o en otros idiomas gráficos [3]), o también puede llevarse a cabo por medio de un programa personalizado en algún lenguaje de programación (generalmente un lenguaje orientado a objetos). En ambos, el PC debe tener una tarjeta de adquisición de datos (dispositivos de entrada-salida) que habilita la entrada de datos desde el sistema y la salida al mismo. En un lenguaje de programación genérico, su traducción podría ser como sigue en el programa de la figura 5.

Éstos son solamente ejemplos de la implementación en algunos de los posibles sistemas de programación. Aunque el PLC es uno de los más frecuentemente usados, la implementación puede llevarse a cabo con cualquier otro sistema del control, con tal de que el programa ejecute las instrucciones. Hay también otras maneras de implementar RdP en el mismo dispositivo que son igualmente válidas, como los métodos matriciales, entre otros, pero en todos ellos es necesario tener cuidado con ciertos aspectos. Hay también algunos sistemas de implementación que permiten la programación en GRAFCET, y entonces a veces puede asociarse un contador a los valores no binarios para acercarse el modelado mediante RdP (de muy bajo nivel). Además, otros permiten la implementación directa de una manera gráfica por medio del dibujo de la RdP, aunque desgraciadamente, esto no es muy frecuente.

```
variables
p1,p2:integer; p2:boolean;
A,B,C,D: boolean input;
program
    p1:=3; p2:=false; p3:=0
        repeat
            if A and p1>0 and not p2
                then p1:=p1-1; p2:=true
            if B and p2
                then p2:=false; p3:=p3+1
            if C and p3>0
                then p3:=p3-1; p1:=p1+1
            if D and p1>0
                then p1:=p1-1; p3:=p3+1
until end_of_program
end program
```

Figura 5. Implementación correspondiente a la RdP de la Figura 1 en lenguaje de programación genérico.

3.1.3. Análisis de la Implementación. Influencias del Dispositivo Usado

Cuando se ha determinado la RdP y el objetivo es implementarla en el dispositivo de control (PLC o IPC), es necesario tener presente el funcionamiento intrínseco del dispositivo para asegurar su comportamiento apropiado.

En la implementación llevada a cabo en el ejemplo anterior, se ha usado un razonamiento lógico para modelar la evolución de una RdP en un leguaje PLC o en una computadora,

si *Transición_Sensibilizada* y *Evento* entonces (*disparo_transición*) **(1)**

donde *disparo_transición* implica modificar las marcas de todos los valores afectados por el encendido de la transición (*t y t*).

Los valores binarios de la red pueden ser almacenados en el dispositivo como variables booleanas o como bits de memoria. Sin embargo, en el modelado de valores binarios con *p>1 o p*>1, sería necesario poner a 1 ó 0 (set o reset) el marcado de p desde varios puntos del programa (varias transiciones). Esto no puede hacerse en la mayoría de los PLCs, puesto que cada bit puede solamente ser modificado desde un único punto en el programa. Las instrucciones de lenguaje "incrementar" y "decrementar" pueden ser usadas desde varios puntos del programa, pero solamente con valores enteros (o contadores, o palabras = 16 bits). Entonces es necesario usar un entero en lugar de una variable booleana para este tipo de valores (a pesar de corresponder a valores binarios) debido al funcionamiento del dispositivo de automatización.

Debe también tenerse cuidado si la solución propuesta consiste en modificar las marcas de los bits correspondiendo al valor con un "or" lógico de los valores $t \in *p$ (o los valores $t \in p*$) y sus eventos asociados, es decir, si el razonamiento lógico es:

si *alguna_ transición_de_entrada* y *evento* entonces *incrementar_marcado*
si *incrementado_siguiente* entonces *decrementar_marcado* **(2)**

En este razonamiento, donde simplemente parece haber un cambio en el orden (y el punto de vista desde los valores, en lugar de desde las transiciones), podría cometerse un error, puesto que los dispositivos (PLCs o computadoras) son secuenciales, y de esta manera, están ejecutando las órdenes que corresponden independientemente a un solo disparo de transición. Este tipo de error es desgraciadamente frecuente, debido a la herencia de los sistemas implementados por medio de lógica cableada.

Esto puede verse mejor con el ejemplo de la RdP de la Figura 6, que simplemente es una red secuencial. Con este marcado la red es binaria, y puede ser implementada por medio de bits ya que $|*p|=1$ y $|p*|=1 \ \forall$ p. Una implementación por medio del razonamiento lógico (2) produce el programa correspondiente a la Figura 8. Una implementación por medio de la lógica (1), que es la usada en todos los ejemplos anteriores, produce el programa mostrado en la Figura 7. Aunque los dos pueden parecer similares, son diferentes. Siguiendo el programa mostrado en la Figura 7, si los eventos asociados a t2 y t3 son siempre ciertos, y en un cierto instante, t1 pasa a cierto, desde este momento en adelante, la marca que inicialmente estaba en p1 irá desde p1 hasta p4 saltando los valores intermedios, como es necesario para completar la evolución correcta de la red. Sin embargo, siguiendo la evolución según el programa en Figura 8, la evolución de las marcas es la mostrada en la Figura 10.

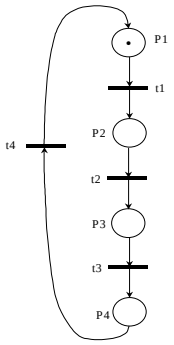


Figura 6: RdP secuencial

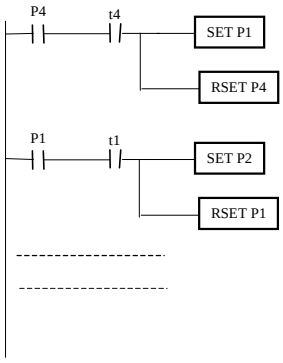


Figura 7: Programa ladder de la RdP de la Figura 6

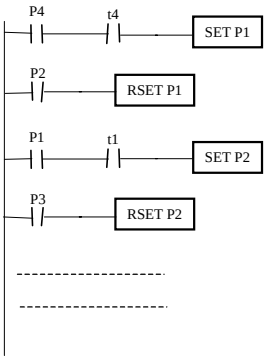


Figura 8: Programa ladder de la RdP de la Figura 6.

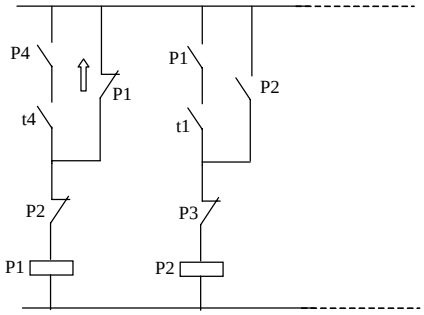


Figura 9: Lógica cableada para la RdP de la Figura 6

La razón por la cual es usual encontrar algunas implementaciones de esta manera incorrecta (Figura 8) es la herencia de la lógica cableada. Si el propio esquema desarrollado en GRAFCET se traduce a contactos eléctricos, queda como se muestra en la Figura 9, y traducida esta última a ladder da el resultado de la Figura 8.

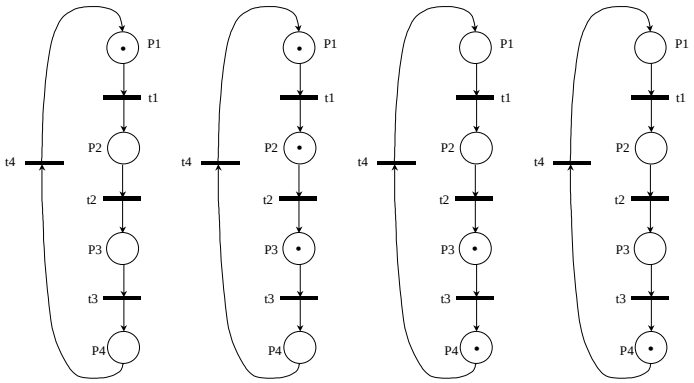


Fig. 10. Evolución de acuerdo al programa ladder de la Figura 6.

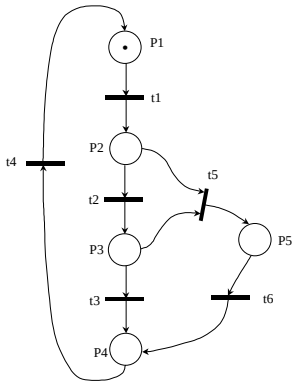


Fig. 11. RdP con diferentes comportamientos, dependiendo de la implementación en el dispositivo lógico.

De hecho, los resultados son iguales cuando los programas llevan a cabo secuencias simples, y éste es frecuentemente el caso (como en el ejemplo mostrado), puesto que la última marca es la misma y la evolución es muy rápida (idealmente instantánea). Pero en ciertas RdP los resultados no son los mismos. Por ejemplo, en la RdP de la Figura 11, dónde p5 no está nunca activo según el marcado inicial, pero debido a la implementación en el dispositivo lógico, podría llegar a estar marcado, lo cual es erróneo.

Todos esto sucede porque el PLC ejecuta sus instrucciones en orden secuencial (una a una). También es interesante señalar que en una ramificación el programa guarda el valor evaluado, que será igual para todas las ramas, sin evaluarlo de nuevo aunque en algunas ramas varios elementos anteriores hayan cambiado. Todo esto se ha analizado para la implementación en un PLC, pero es igual si se implementa en un PC por medio de un programa de ordenador (Figuras 12 y 13) en cualquier lenguaje secuencial (para los lenguajes no secuenciales, tales como ADA, el resultado es diferente).

```
p1:=1
repeat
  if p1 and t1 then (p2:=1; p1:=0)
  if p2 and t2 then (p3:=1; p2:=0)
  if p3 and t3 then (p4:=1; p3:=0)
  if p4 and t4 then (p1:=1; p4:=0)
until end_ program
```

Figura12: Programa correspondiente a una correcta implementación de la RdP.

```
p1:=1
repeat
  if p4 and t4 then p1:=1
  if p2 then p1:=0
  if p1 and t1 then p2:=1
  if p3 then p2:=0
  if p2 and t2 then p3:=1
  if p4 then p3:=0
  if p3 and t3 then p4:=1
  if p1 then p4:=0
until end_program
```

Figura 13: Programa correspondiente a una incorrecta interpretación de la RdP, debido al orden en las instrucciones

3.2. Algoritmos de traducción a PLC

3.2.1. Metodología para la Implementación en PLC

Una vez se tiene modelado el sistema mediante una RdP se debe implementar ésta en el autómata programable. A continuación se verá una metodología para realizar dicha implementación, que también puede ser vista como una traducción de un lenguaje RdP (viendo las RdP como un lenguaje de descripción de sistemas de eventos discretos (SED)) a lenguaje de contactos de autómata (ladder) o mnemónico.

Dado que los PLCs pueden trabajar con matrices (aunque de una manera poco sencilla y no muy eficiente) se podría pensar en realizar la implementación calculando las salidas y el nuevo estado interno a partir de las entradas y el estado interno previo, de acuerdo a la ecuación de estado de la red: $M_K = M_0 + \boldsymbol{C} \cdot \overline{\sigma}$. Sin embargo no suele ser la mejor manera de realizarlo, por diversas cuestiones:

- Los problemas que ocasionan características especiales de la red. Por ejemplo, una red que no sea pura cuando tiene asociados eventos a los flancos de los lugares

- La dificultad de ampliar o modificar el sistema, cosa habitual, y una de las grandes ventajas del empleo de RdP

- La mayor dificultad a la hora de depurar errores o de buscar nuevas alternativas en la programación del PLC

- La dificultad de los cálculos matriciales en los autómatas programables, y la ineficiencia al manejar posiciones de memoria de valores enteros para todos los lugares, incluidos los binarios.

Por todo ello la forma más empleada y habitualmente la más eficiente consiste en la traducción una a una de las transiciones de la red.

Una transición *t* presenta en general un cierto número de arcos provenientes de los lugares de entrada ${}^\bullet t$ (${}^\bullet t = \{p \in P \mid \alpha(p,t) > 0\}$), incluidos los arcos inhibidores, y otro número de arcos que salen hacia los lugares de salida de t, $t^\bullet$ ($t^\bullet = \{p \in P \mid \beta(t,p) > 0\}$). Sin embargo, dado que muchos de los lugares son binarios en la práctica, y por tanto se pueden representar y manejar con posiciones de memoria tipo bit de una manera más cómoda y eficiente, conviene distinguir entre los lugares binarios y los que no lo son (si bien se pueden considerar en general todos como no binarios y el funcionamiento es correcto). Por lo tanto, una transición tendrá en general la forma que muestra la Figura 14. De momento consideramos transiciones sin retardo o temporización, y estos se introducirán más adelante.

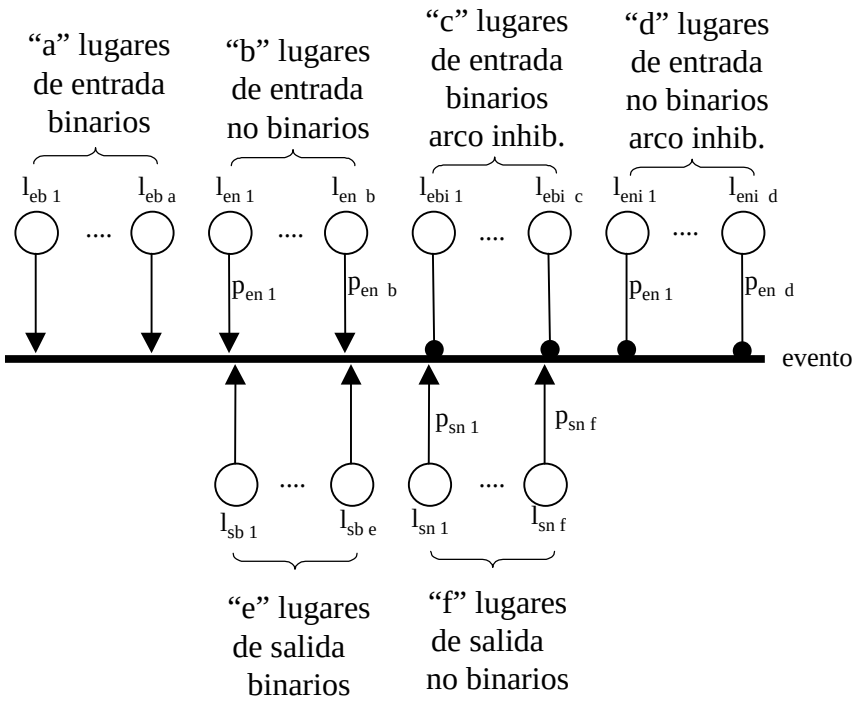


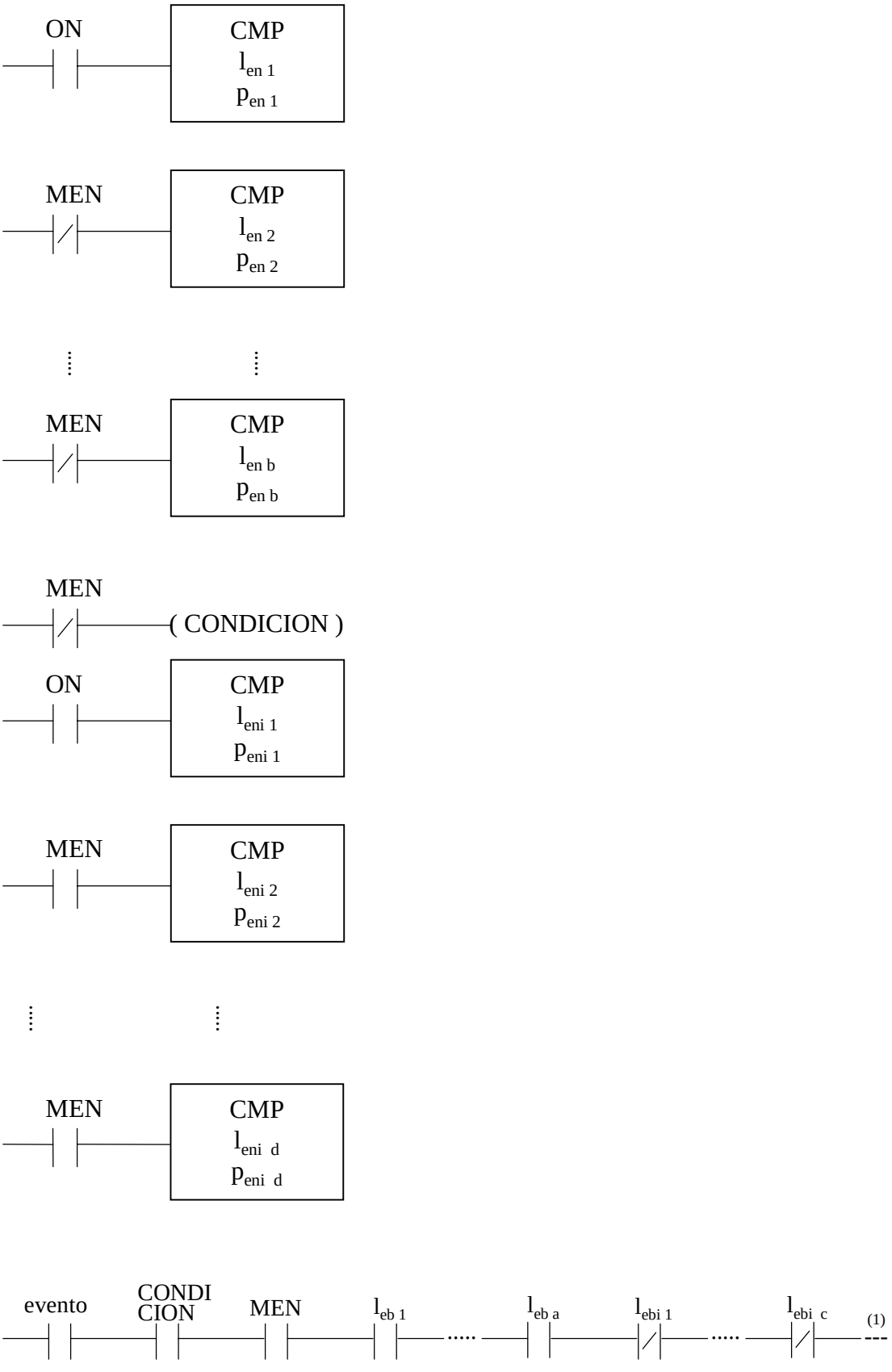
Figura 14: Transición genérica desde el punto de vista de un PLC

donde:

$l_{eb\ i}$	es el lugar de entrada binario *i*
a	es el número de lugares de entrada binarios
$l_{en\ i}$	es el lugar de entrada no binario *i*
$p_{en\ i}$	es el peso del arco de entrada no binario *i*
b	es el número de lugares de entrada no binarios
$l_{ebi\ i}$	es el lugar de entrada binario con arco inhibidor *i*
c	es el número de lugares de entrada binarios con arco inhibidor
$l_{eni\ i}$	es el lugar de entrada no binario con arco inhibidor *i*
$p_{eni\ i}$	es el peso del arco de entrada no binario con arco inhibidor *i*
d	es el número de lugares de entrada no binarios con arco inhibidor
$l_{sb\ i}$	es el lugar de salida binario *i*
e	es el número de lugares de salida binarios
$l_{sn\ i}$	es el lugar de salida no binario *i*
$p_{sn\ i}$	es el peso del arco de salida no binario *i*
f	es el número de lugares de salida no binarios
evento	es el evento asociado a la transición

Figura 15: Interpretación de la nomenclatura en la Figura 14

La implementación de una transición global como la anterior es como se muestra a continuación:


ON
CMP
len 1
pen 1
MEN
CMP
len 2
pen 2
MEN
CMP
len b
pen b
MEN
(CONDICION)
ON
CMP
leni 1
peni 1
MEN
CMP
leni 2
peni 2
MEN
CMP
leni d
peni d
evento
CONDICION
MEN
leb 1
leb a
lebi 1
lebi c
(1)

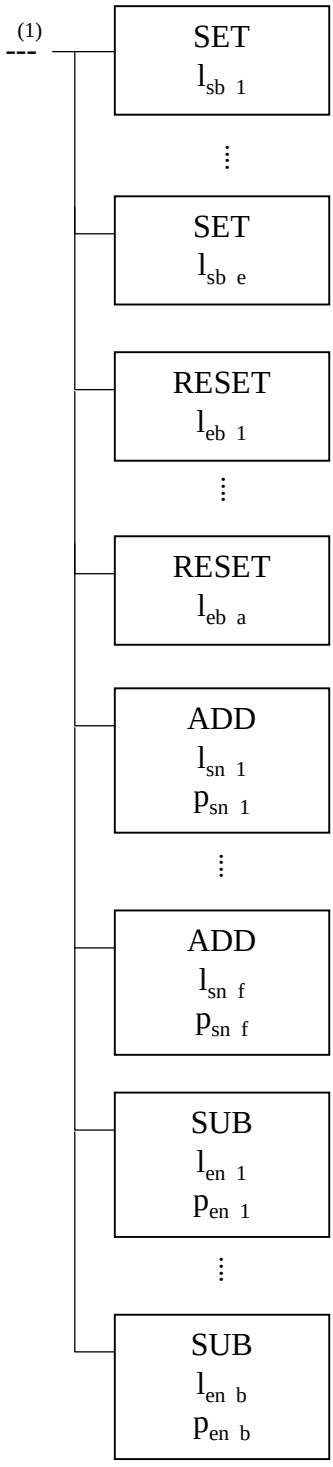


Figura 16: Implementación en Ladder
de una transición global en un PLC.

donde:

ON	es un bit siempre activo
MENOR	es el bit de comparación que indica menor
MAYOR	es el bit de comparación que indica mayor

CONDICION es un bit auxiliar empleado para todas las transiciones de la red

El diagrama de contactos se ha roto, indicándose con el símbolo (1), simplemente para poder meterlo en la hoja, pero hay que suponer que va todo seguido. De todas formas cuando se indique más adelante cómo se realiza este tipo de implementaciones con transiciones temporizadas, va a venir muy bien ese *corte* que se ha realizado, puesto que es precisamente ahí donde se insertarán los elementos de la temporización.

Ese mismo programa, desarrollado en lenguaje mnemónico queda de la siguiente manera:

LOAD ON
CMP $l_{en\ 1}$, $p_{en\ 1}$

LOADNOT MEN
CMP $l_{en\ 2}$, $p_{en\ 2}$

⋮

LOADNOT MEN
CMP $l_{en\ b}$, $p_{en\ b}$

LOADNOT MEN
OUT Condicion

LOAD ON
CMP $l_{eni\ 1}$, $p_{eni\ 1}$

LOADNOT MEN
CMP $l_{eni\ 2}$, $p_{eni\ 2}$

⋮

LOADNOT MEN
CMP $l_{eni\ d}$, $p_{eni\ d}$

LOAD evento
AND Condicion
AND MEN
AND $l_{eb\ 1}$

⋮

AND $l_{eb\ a}$
AND $l_{ebi\ 1}$

⋮

AND $l_{ebi\ c}$

SET $l_{sb\ 1}$

⋮

SET $l_{sb\ e}$

SET $l_{eb\ 1}$

⋮

SET $l_{eb\ a}$

ADD $l_{sn\ 1}$, $p_{sn\ 1}$, $l_{sn\ 1}$

⋮

ADD $l_{sn\ f}$, $p_{sn\ f}$, $l_{sn\ f}$,

ADD $l_{en\ 1}$, $p_{en\ 1}$, $l_{en\ 1}$,

⋮

ADD $l_{en\ b}$, $p_{en\ b}$, $l_{en\ b}$,

Figura 17: Implementación en mnemónico
de una transición global en un PLC.

En el caso de que b=0 (no hay lugares de entrada no binarios), además de eliminar sus líneas correspondientes, se debe eliminar la línea "AND Condicion"

En el caso de que d=0 (no hay arcos inhibidores no binarios), además de eliminar sus líneas correspondientes, se deben eliminar "AND MEN" así como "ANDNOT MEM" y "OUT Condicion", y se debe sustituir la línea "AND Condicion" por "ANDNOT MEN".

Una vez realizado lo anterior para cada transición de la RdP, no se debe olvidar inicializar la red con el marcado inicial, es decir, cargar con su marcado inicial las posiciones no binarias (incluyendo las que están a cero), así como poner a SET las posiciones binarias inicialmente activadas y a RESET las inicialmente desactivadas. Evidentemente esto último sólo se debe realizar en el primer ciclo de SCAN del autómata. Además, cuando se trabaje con posiciones de memoria que se resetean automáticamente cada vez que se pone en marcha el autómata no es necesario inicializarlas a cero ni a reset. Aun así se recomienda reininicializar también esas posiciones, puesto que el tiempo de ciclo del autómata es exactamente igual (cuando se evalúa que no es primer ciclo de SCAN se salta directamente al siguiente bloque), y ante posibles traducciones del programa a otro tipo de máquina u otra aplicación puede ser bueno que aparezca.

3.2.2. Retardos o temporizaciones

Existe una extensa teoría sobre la interpretación temporal de las RdP, así como numerosos estudios y artículos de gran calidad sobre el tema (ver referencias sobre RdP

del tema 1). Estos estudios son ciertamente importantes en aspectos como el análisis y evaluación de prestaciones. En la fase de automatización de los procesos industriales, normalmente la temporización está definida, es decir, se conocen los tiempos de retardo de los procesos. Pero además del tiempo de retardo se debe definir el tipo de temporización.

Vamos a tratar de encontrar los tipos de temporización para poder definir mediante ellos *cualquier proceso industrial*, y también vamos a clasificarlos y agruparlos. Estos tipos de retardo, como ya se ha dicho, están ampliamente analizados desde un punto de vista formal y matemático, pero ahora se va a ver desde el punto de vista de los procesos industriales, modelados mediante RdP.

Es muy frecuente al modelar o automatizar un proceso industrial asignar sin más un tiempo de retardo, sin pensar en qué tipo de retardo se trata, y ello lleva muchas veces a un error que además es posible que cueste esfuerzo detectar posteriormente.

A. Retardos en redes no coloreadas

A.1.Retardos en el disparo de la transición

Cuando tenemos una RdP en la que el disparo de cada transición t_i dura un tiempo $\tau(t_i)$ tenemos una RdP temporizada (RdPT). Formalmente se puede decir que una RdPT es un par $\langle \boldsymbol{R}, Z \rangle$ tal que $\boldsymbol{R}$ es una RdP y Z es una función que asigna un número real no negativo z_i a cada transición de la RdP. $z_i = Z(t_i)$ recibe el nombre de tiempo de disparo de la transición t_i.

Otra forma de ver el mismo caso consiste en decir que en una RdPT una marca puede estar dispuesta o indispuesta. Cuando la marca está indispuesta es como si no existiese a efectos de disparo de la transición. Cuando aparece una marca en un lugar entonces está indispuesta, y permanece así durante $\tau(t_i)$ unidades de tiempo, desde que llega y desde que se produce el disparo de la transición.

Como caso real podemos encontrar esta temporización en un proceso en el que la transición representa el tiempo que tarda un robot en coger una pieza de una zona (un lugar l_1) y llevarlo a otra (otro lugar l_2).

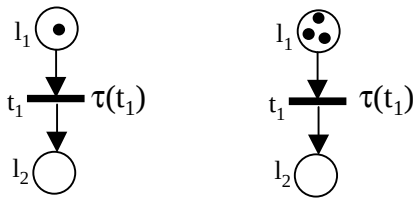


Figura 18: Retardos en el disparo de la transición

De esta manera aunque haya varias marcas en l_1, todas tienen que esperar el tiempo de la transición desde que llegan y también desde que se ha disparado la transición la

última vez (debido a otra transición distinta). Si llegan todas a la vez, esperan a que se dispare la transición con la primera marca, y después cada una de las siguientes esperará ese mismo tiempo desde que se dispare la anterior.
Este es el tipo de retardo que presenta una RdPT, es decir, si estamos modelando un proceso e incluimos una temporización en una transición, el comportamiento que presenta el modelo es el que acabamos de describir.

A.2. Retardos en la sensibilización de la transición

Este tipo de temporización consiste simplemente en un tiempo de espera desde que se dan las condiciones de sensiblización de la red hasta que se pueden producir los disparos de ésta. Es decir, es como si se incluyese como condición al disparo de la transición: "llevar t segundos cumpliendo que $\forall p \in {}^\bullet t, M(p) \geq \alpha(p,t)$ y cumpliendo el evento asociado a la transición".

Podemos verlo desde el mismo punto de vista de marcas dispuestas e indispuestas, considerando también que cuando la marca está indispuesta es como si no existiese a efectos de disparo de la transición, y que cuando aparece una marca en un lugar entonces está indispuesta, y permanece así durante $\tau(t_i)$ unidades de tiempo. Pero la diferencia es que ahora cuando se produce el disparo de la transición (debido a otra marca distinta) no se vuelve a empezar a esperar.

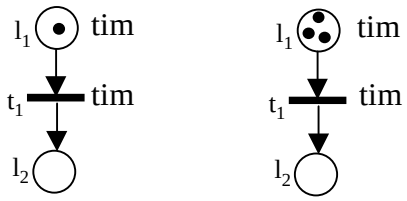


Figura 19: Retardos en la sensibilización de la transición

Ahora cuando hay varias marcas en l_1, si han llegado todas a la vez, esperan a que transcurra el tiempo de la temporización y en ese momento se consideran todas dispuestas y todas ellas pueden evolucionar a l_2 sin tener que volver a esperar. Resulta evidente que este tipo de retardo es totalmente equivalente al anterior cuando se trata de redes binarias.

Podemos encontrarla por ejemplo en una compuerta que tarda t segundos en abrirse, se abre cada vez que llega algún vagón (marca), y permanece abierta hasta que han pasado todos los vagones.

Este tipo de retardos deriva de los que se emplean en GRAFCET, donde los Estados son una especie de lugares binarios, y por eso su representación es análoga a la de los retardos en GRAFCET, indicando la temporización en el lugar y el fin de la temporización en la transición. Los retardos corresponden simplemente a retardos a la conexión. Este tipo de retardo es el que más se encuentra en los procesos industriales.

Hasta tal punto es así que los autómatas programables industriales más frecuentes en el mercado es el único que incluyen directamente, y los demás tipos de temporizaciones se realizan a partir de este tipo de elementos.

Los tipos de temporizadores que normalmente derivan de este tipo de retardo, y que se pueden construir a partir de él, son los que comentaremos a continuación, así como las RdP para conseguirlos. En estos temporizadores se habla de *entrada* y *salida* porque un temporizador es un dispositivo capaz de retardar una orden de salida un cierto tiempo en respuesta a la señal de mando de entrada.

A.2.1 Temporizador a la conexión

Activa la salida cuando lleva cierto tiempo recibiendo entrada, y se desactiva la salida cuando cesa la entrada. Consiste simplemente en aplicar un retardo a la conexión para activar la salida (Figura 20).

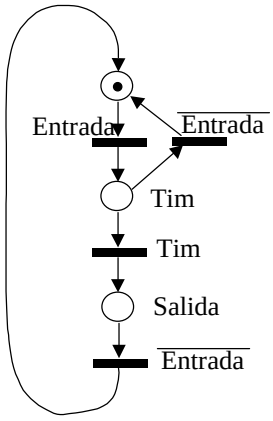


Figura 20: Esquema del temporizador a la conexión

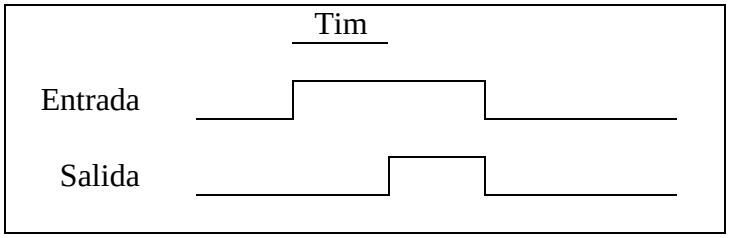


Figura 21: Cronograma del temporizador a la conexión

A.2.2 Temporizador a la desconexión

La salida se activa inmediatamente con la entrada, pero permanece un cierto tiempo después de desaparecida ésta (Figura 22).

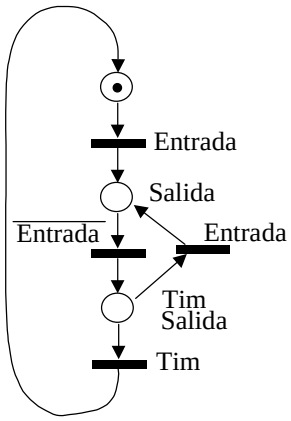


Figura 22: Esquema del temporizador a la desconexión

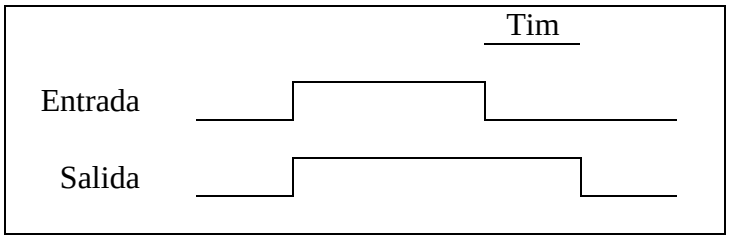


Figura 23: Cronograma del temporizador a la desconexión

A.2.3 Temporizador a la conexión y a la desconexión

Combina los dos tipos anteriores, es decir, la salida tarda en activarse un cierto tiempo mientras recibe entrada, y una vez activada tarda un cierto tiempo en desactivarse cuando deja de recibir señal. Los tiempos de activación y desactivación no tienen por que ser iguales (Figura 24).

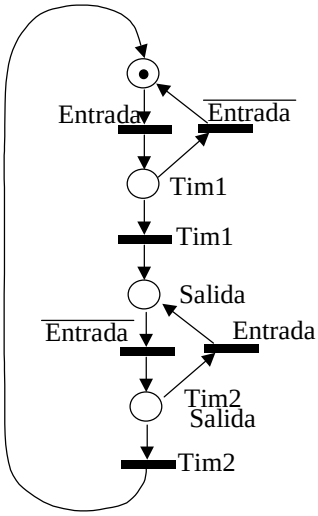


Figura 24: Esquema del temporizador a la conexión-desconexión

No se debe caer en la tentación de pensar que puede construirse uno exactamente igual simplemente enlazando los dos anteriores, es decir, empleando la salida de un temporizador a la conexión como la entrada de uno a la desconexión. Para muchos tipos de entradas el comportamiento sí es igual, pero para otras no lo es (Figura 25)

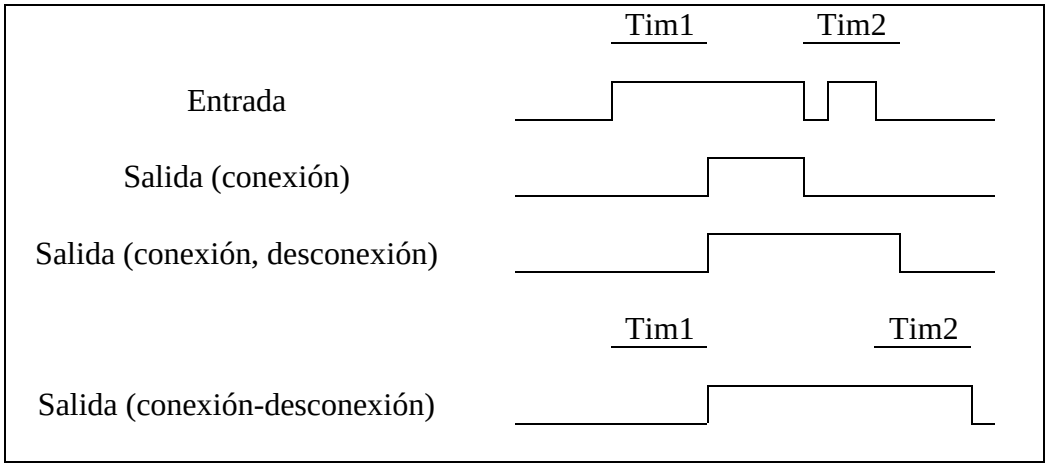


Figura 25: Cronograma del temporizador a la conexión-desconexión

A.2.4 Temporizador Impulso

La salida se mantiene activa mientras dure activa la condición de marcha pero con un tiempo máximo (Figura 26).

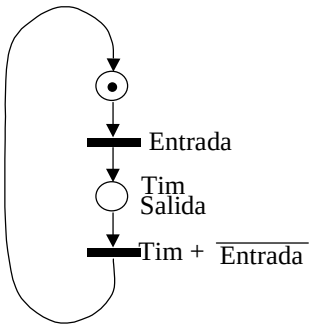


Figura 26: Esquema del temporizador impulso

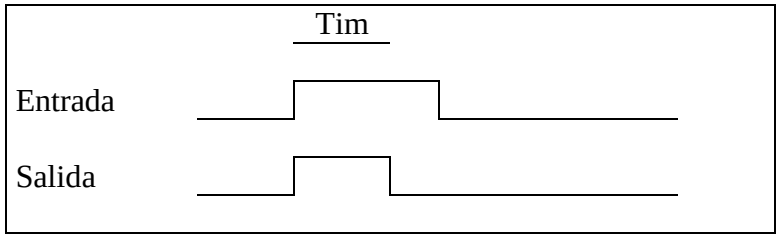


Figura 27: Cronograma del temporizador impulso

A.2.5 Temporizador Monoestable o Conformador de Impulso

Cuando se activa la entrada se activa también la salida durante un tiempo constante e independiente de que se mantenga o no la entrada (Figura 28).

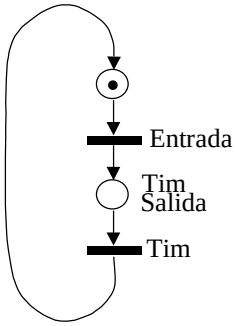


Figura 28: Esquema del temporizador monoestable

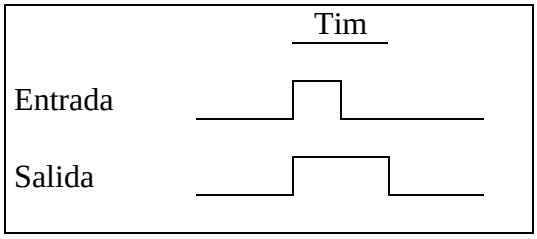


Figura 29: Cronograma del temporizador monoestable

A.3 Retardos asociados a las marcas

Otro tipo de retardo es el que se puede asociar a todas las marcas que accedan a un lugar. Entonces, cuando una marca accede a un lugar, debe esperar cierta cantidad de tiempo antes de estar dispuesta, y pasado dicho tiempo ya está dispuesta en todo momento.

Este tipo de transición puede darse en la industria cuando tengamos un proceso, como por ejemplo un horno (representado por un lugar), en el que los elementos (las marcas) deben estar al menos una cantidad de tiempo antes de poder salir.

En realidad este tipo de retardo es equivalente a los dos anteriores si se consideran redes coloreadas (que enseguida analizaremos) y se considera que cada elemento tiene un color diferente. Hay que notar que al tener cada marca un color diferente, para cada color la red es binaria, por lo cual los dos tipos anteriores de retardo son el mismo (cosa que debía ocurrir para ser coherente la propiedad anterior).

B. Retardos en redes coloreadas

Se ha comentado anteriormente que una RdP coloreada es aquella que presenta marcas de diversos colores (que las distingue de las de colores distintos), en la que la evolución es independiente para cada tipo de color.

Para implementar en los dispositivos modelos llevados a cabo con redes coloreadas hay que tener en cuenta en cada lugar el número de marcas que tenemos de cada color. Siendo así no existe ninguna diferencia con lo visto en el punto anterior. Por lo tanto tenemos los mismos tres tipos de retardos.

No hay que caer en la trampa de pensar que el tercer tipo de retardo de los vistos anteriormente, el asociado a cada marca en un lugar, está incluido en los otros dos en redes coloreadas. Ello sólo ocurre cuando los lugares no pueden tener más de un elemento distinto (más de una marca) de cada color, es decir, cuando para cada color son binarias.

3.2.3. Implementación de Retardos

Una vez analizados los tipos de retardos, veamos cómo se implementan en los dispositivos industriales en los que se ha implementado la RdP del modelo.

A. Retardos en redes no coloreadas

A.1 Retardos en el disparo de la transición

Cuando se den las condiciones de sensibilización de la transición se pone en marcha un temporizador, cuya condición de activación se debe incluir entre las condiciones para que se produzca el disparo. Y además se debe reinicializar el temporizador con cada disparo de la transición, lo cual se logra con una condición negada del temporizador.

Por lo tanto, para el caso más sencillo posible, formado por un solo lugar de entrada y uno de salida (ambos no binarios en general), como se muestra en la figura 30, la traducción es la siguiente (Figura 31):

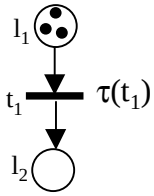
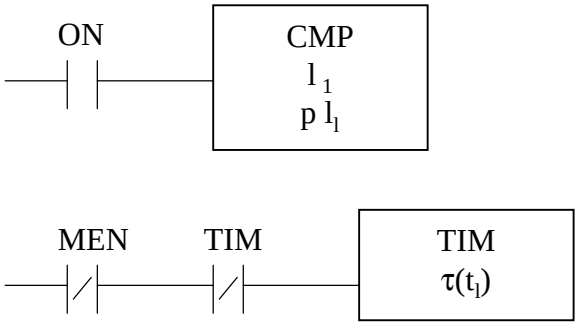


Figura 30: Retardo en el disparo de la transición



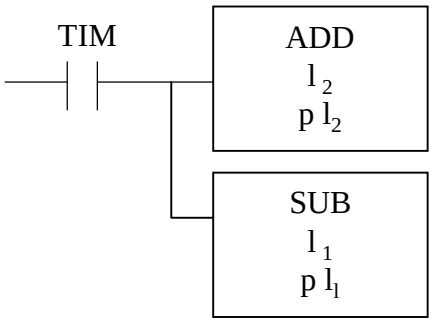


Figura 31: Implementación en ladder

o en mnemónico:

```
LOAD ON
CMP l1,p l1
LOADNOT MEM
ANDNOT TIM
TIM τ( ti)
LOAD TIM
ADD l2, p l2
SUB l1, p l1
```

Figura 32: Implementación en mnemónico

Para un caso general en el que se tengan más lugares de entrada y salida, incluyendo binarios y arcos inhibidores, la traducción consiste en insertar los elementos de temporización entre las condiciones de disparo y la actuación que éste produce (tal como acabamos de hacer). Es decir, en el gráfico de la Figura 16 de la sección anterior, que habíamos seccionado para que entrase en la hoja, ahora hay que dividirlo de verdad y añadir lo siguiente sobre las anotaciones (1) que tenía:

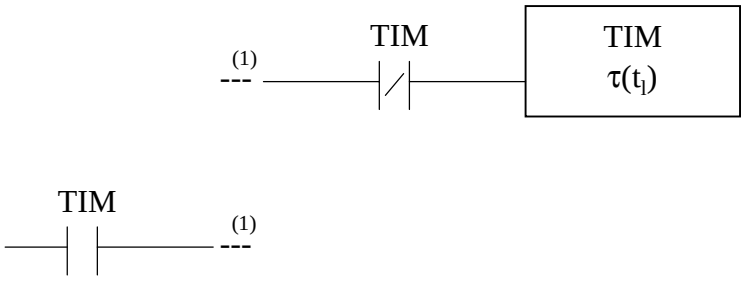


Figura 33: Implementación (en ladder) para una transición global

Con lo que para un caso general, como el que se indicó en la Figura 16, la traducción en mnemónico sería:

LOAD ON
CMP $l_{en\ 1}$, $p_{en\ 1}$

LOADNOT MEN
CMP $l_{en\ 2}$, $p_{en\ 2}$

⋮

LOADNOT MEN
CMP $l_{en\ b}$, $p_{en\ b}$

LOADNOT MEN
OUT Condicion

LOAD ON
CMP $l_{eni\ 1}$, $p_{eni\ 1}$

LOADNOT MEN
CMP $l_{eni\ 2}$, $p_{eni\ 2}$

⋮

LOADNOT MEN
CMP $l_{eni\ d}$, $p_{eni\ d}$

LOAD evento
AND Condicion
AND MEN
AND $l_{eb\ 1}$

⋮

AND $l_{eb\ a}$
AND $l_{ebi\ 1}$

⋮

AND $l_{ebi\ c}$
ANDNOT TIM
SET TIM

LOAD TIM
SET $l_{sb\ 1}$

⋮

SET $l_{sb\ e}$

SET $l_{eb\ 1}$

⋮

SET $l_{eb\ a}$

ADD $l_{sn\ 1}$, $p_{sn\ 1}$, $l_{sn\ 1}$

⋮

ADD $l_{sn\ f}$, $p_{sn\ f}$, $l_{sn\ f}$,

ADD $l_{en\ 1}$, $p_{en\ 1}$, $l_{en\ 1}$,

⋮

ADD $l_{en\ b}$, $p_{en\ b}$, $l_{en\ b}$,

Figura 34: Implementación (en mnemónico) para una transición global

A.2 Retardos en la sensibilización de la transición

Hay que conseguir lo mismo que en el apartado anterior, con la única diferencia de que ahora no se debe resetear el temporizador con el disparo de temporizaciones precedentes. Por lo tanto, la traducción del caso más simple (Figura 30), queda de la siguiente manera:

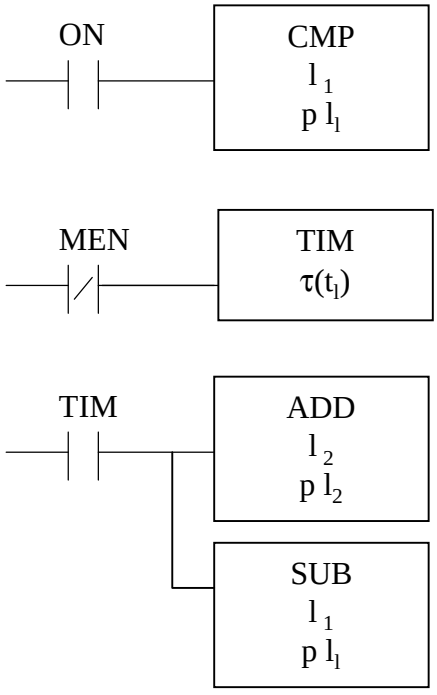


Figura 35: Traducción del sistema de la Figura 30

Y la traducción del caso general (respecto a la traducción sin retardo) queda:

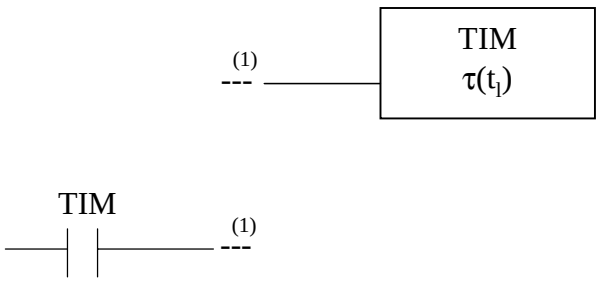


Figura 36: Modificación sobre la traducción
de una transición genérica(Fig 16)

Y en la representación en mnemónico tan sólo hay que eliminar la instrucción "LOADNOT TIM".

Conocida la forma de implementar estas transiciones, y las construcciones de los temporizadores con RdP (Figuras 20 a 29), se puede implementar cada uno de los temporizadores vistos previamente. Sin embargo, ya que son bloques con una entrada y una salida sobre los que no se suelen realizar modificaciones, es habitual construirlos de forma directa, que además en estos casos suele ser bastante más corta y compacta. A continuación vemos como pueden implementarse de forma directa:

A.2.1 Temporizador a la conexión

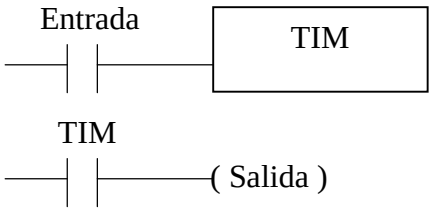


Figura 37: Temporizador a la conexión

A.2.2 Temporizador a la desconexión

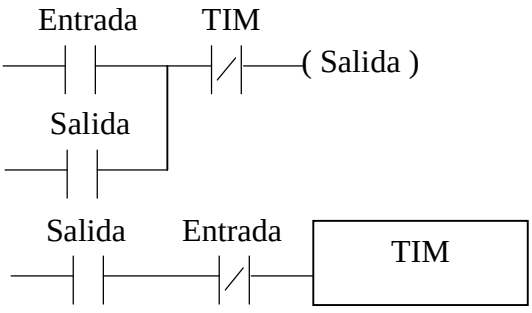


Figura 38: Temporizador a la desconexión

A.2.3 Temporizador a la conexión y a la desconexión

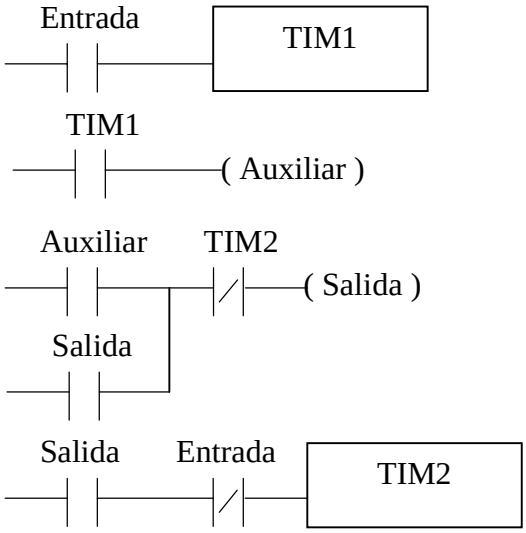


Figura 39: Temporizador a la conexión y a la desconexión (conexión-desconexión)

Se puede observar de nuevo que este temporizador no consiste en meter como entrada del temporizador a la desconexión la salida de uno a la conexión (ver figura 25). Esta diferencia en el comportamiento esperado se consigue poniendo como condición de corte (LOADNOT) del temporizador de desconexión (TIM2) la entrada global (E) en vez de la entrada del bloque de desconexión (AUX, que también es la entrada del bloque de desconexión).

A.2.4 Temporizador Impulso

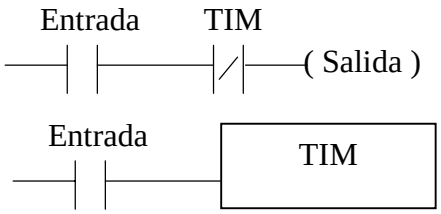


Figura 40: Temporizador impulso

A.2.5 Temporizador Monoestable o Conformador de Impulso

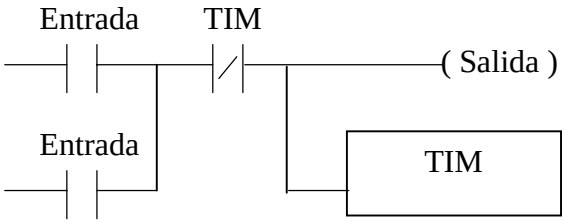


Figura 41: Temporizador monoestable

B. Retardos asociados a las marcas y Retardos en redes coloreadas

En estos tipos de retardo obligatoriamente se debe emplear un temporizador para cada marca posible en los lugares, o bien guardar en una lista el momento en que cada marca ha llegado al lugar. La segunda opción es mucho más económica en cuanto a recursos del PLC, y es por tanto la que se emplea habitualmente. La forma de realizar dicha implementación es muy similar a la forma de simular las temporizaciones en Matlab (tanto este tipo de retardos como los de los tipos anteriores) por lo que se remite a dicha sección.

3.3. Conclusiones

En este apartado se ha analizado la influencia de los dispositivos usados según la forma de traducción desde las herramientas gráficas a los PLCs, desde un doble punto de vista, teórico y práctico, basado en la investigación anterior y en la implementación real subsecuente. Se usan las redes de Petri como una herramienta importante para control de sistemas de eventos discretos con procesos concurrentes, en todas sus fases: modelado, automatización, supervisión y simulación.

En resumen, la contribución de este tema es el análisis de la implementación sobre los PLCs de los sistemas diseñados sobre RdP, así como una detallada metodología para su correcta implementación que permita realizarlo de forma sistemática y sin errores. Dicha metodología contempla diferentes tipos de temporización en las transiciones.

3.4 Referencias

1.Belli, F., Grosspietch, KE.: Specification of fault-tolerant systems issues by predicate/transition nets and regular expressions: Approach and case study. IEEE Trans on Software Engineering (1991) 17 (6), 513-526

2. Burns, A., Wellings, A.: Real-Time Systems and Programming Languages. Addison-Wesley, California (1996)

3. David, R.: Grafcet: A powerful tool for specification of logic controllers. IEEE Trans on Control Systems Technology (1995) 3 (3) , 253-268

4. Jiménez, E., Miruri, JM., Martínez de Pisón, J.F., Gil, M.: Supervised Real-Time Control with PLCs and SCADA in Ceramic Plant. 6th IFAC Workshop on Algorithms and Architectures for Real-Time Control (2000) 1 , 221-226

5. Jiménez, E.: Redes de Petri de Supervisión y Simulación de Procesos Industriales Automatizados. XXI Jornadas de Automática CEA-IFAC (2000)

6. Morriss, B.: Programmable Logic Controllers. Prentice Hall, New Jersey (1999)

7. N. Bhandari, D.K. Rollins. Superior Semi-Empirical Dynamic Predictive Modeling that Addresses Interactions. IASTED Intelligent Systems and Control ISC'99 (1999)

8. N. Konstas, S. Lloyd, H. Yu, C. Chatwin. Generic Net Modelling Framework for Petri Nets. IASTED Intelligent Systems and Control ISC'99 (1999)

9. Z. Bingul, A.S. Sekmen, S. Palaniappan, S. Sabatto. An Application of Multi Dimensional Optimization Problems using Genetic Algorithms. IASTED Intelligent Systems and Control ISC'99 (1999)

10. W.T. Goh, Z. Zhang. Autonomous Petri-Net for Manufacturing System Modelling in an Agile Manufacturing Environment. IASTED International Confer. Robotics and Applications 1999.

Tema 4

METODOLOGÍA DE SIMULACIÓN MEDIANTE LENGUAJES DE PROGRAMACIÓN

4.1. Introducción

Además de la simulación que hemos visto en los capítulos anteriores, en la que se simulaba el programa implementado sobre el PLC, es muy conveniente simular en un computador cómo va a funcionar la producción con ese mismo programa y con los parámetros empleados.

La realización de esta nueva simulación tiene varios cometidos:

- Sirve de comprobación para corroborar que la simulación realizada sobre el PLC da los mismos resultados (verificación de la simulación en el PLC)

- Permite realizar pruebas de modificación de parámetros antes de simularlas sobre el PLC, y por lo tanto sin tener que desconectar el funcionamiento de éste en la planta

- Permite realizar dichas modificaciones de una manera más ágil, puesto que los lenguajes de programación están más indicados para ello que el autómata programable

- Incluso se puede programar sobre esta simulación un algoritmo de búsqueda de la solución óptima mediante técnicas avanzadas (inteligencia artificial, algoritmos genéticos u otros métodos), para una vez encontrada poderla simular tanto sobre el PLC como sobre el propio PC de una manera detallada

Para realizar esta simulación se puede emplear cualquier lenguaje de programación de propósito general, por ejemplo los empleados para las diversas aplicaciones informáticas que componen esta tesis (C, Visual Basic, Java, etc.). Sin embargo puede resultar muy interesante realizar dicha aplicación sobre Matlab, por su potencia en el manejo de información en forma matricial, así como por la facilidad a la hora de ser programado y de representar los resultados en forma de gráficas.

La realización de la simulación en cualquier otro programa simplemente consiste en aplicar los mismos principios sobre el lenguaje de programación correspondiente. Y esos principios no son otra cosa que los empleados para realizar la traducción de las RdP al lenguaje de contactos del autómata programable, pero con unas reglas de juego algo distintas (las del computador en vez de las del autómata programable).

4.2.Metodología de Traducción a Matlab

Veamos por lo tanto cómo se realiza la traducción a Matlab de la RdP correspondiente a la automatización. La simulación se hará de nuevo recorriendo todas las transiciones de la red, y sobre cada una de ellas se va a comprobar si se debe disparar, y si es así, se realiza el disparo correspondiente. En primer lugar hay que tener en cuenta que ahora no hay que distinguir entre lugares binarios y no binarios, puesto que en las aplicaciones informáticas se puede trabajar con ambos tipos de datos de una manera prácticamente igual (al contrario de lo que ocurría con los PLCs). Por lo tanto, una transición genérica será de la forma que indica la Figura 1.

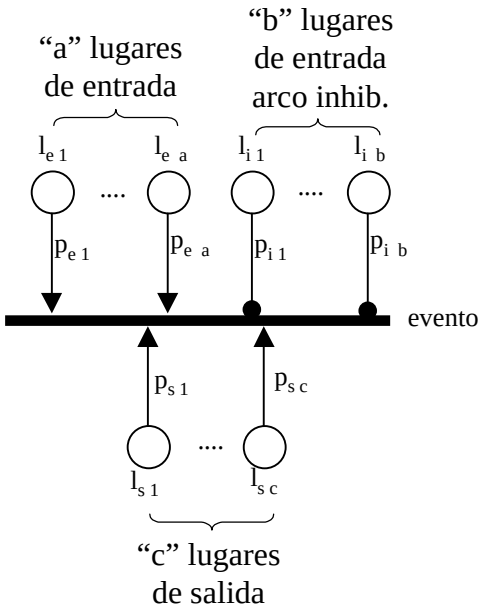


Figura 1: Transición genérica en una RdP para traducirla a Matlab

donde:

$l_{e\,i}$	es el lugar de entrada *i*
a	es el número de lugares de entrada
$p_{e\,i}$	es el peso del arco de entrada *i*
$l_{i\,i}$	es el lugar de arco inhibidor *i*
b	es el número de lugares de entrada con arco inhibidor
$p_{i\,i}$	es el peso del arco de entrada inhibidor *i*
$l_{s\,i}$	es el lugar de salida *i*
c	es el número de lugares de salida binarios
$p_{s\,i}$	es el peso del arco de salida *i*
evento	es el evento asociado a la transición

Aunque en Matlab se pueden emplear variables sin más (en realidad son matrices 1x1), dado que es un entorno adaptado para trabajar más eficientemente con matrices, conviene definir una matriz Y que tenga una columna por cada lugar de la red, y que

tendrá tantas filas como pasos se realicen en la simulación. Cada fila va a corresponder, por tanto, al marcado de la red en un instante dado.

Con todo lo anterior el algoritmo de traducción de la RdP queda como se indica en la Figura 2.

```
Y(1,:)=[ l1o, l1o,...,lno];
for i=2: num_pasos
      Y(i,:)= Y(i-1,:); % copio la fila igual a la anterior
  ⋮

      % inicio de la traducción de una transición
      if Y(i, le1)>= pe1 & ... & Y(i, lea)>= pea & Y(i, li1)< pi1 & ... &
      Y(i, lib)< pib & evento
            Y(i, le1)= Y(i, le1)- pe1; ... ;Y(i, lea)= Y(i, lea)- pea;
            Y(i, ls1)= Y(i, ls1)+ ps1; ... ;Y(i, lsa)= Y(i, lsa)+ psa;
      end
      % fin de la traducción de una transición
  ⋮

end
```

Figura 2: Algoritmo de traducción de RdP a Matlab

donde:

l_{io}	es el marcado inicial del lugar i
n	es el número de lugares en la red
num_pasos	es el número de pasos en la simulación
evento	es el evento (condición lógica) asociado a la transición,

donde se han representado los lugares por su nombre (por ejemplo $l_{e\ a}$) pero se debe sustituir por el número de codificación que tienen en la matriz de lugares Y (el número de columna en el que se representa su evolución),

y donde lo que se ha realizado es la traducción para una única transición, teniendo en cuenta que se debe repetir para cada una de las transiciones de la red.

La razón de iterar desde i=2 en vez de empezar en i=1 es que en Matlab se numeran las matrices desde 1 en adelante, y el valor 1 se emplea por tanto para el estado inicial. Más correcto sería que fuese el índice 0 el del marcado inicial, y dicho tratamiento es posible, pero se complica ligeramente el programa, por lo que se aconseja actuar de esta manera, y si hiciese falta (que no suele ser el caso) recordar que el contador marca una iteración más de las que hay. Además esa diferencia en los marcadores es fácilmente ajustable en la representación gráfica posterior, como veremos más adelante. Ello puede ser interesante cuando las iteraciones correspondan a parámetros temporales en las redes temporizadas.

4.3.Redes temporizadas

El algoritmo anterior de implementación de la RdP en Matlab trataba transiciones sin retardo o sin temporización, y por lo tanto la simulación correspondía a diversos pasos en la evolución de la red, pero sin ninguna interpretación temporal.

Si las transiciones tienen una interpretación temporal, que es el caso que vamos a analizar seguidamente, cada fila de la matriz que representa los marcados corresponderá al marcado en un instante de tiempo determinado. Por lo tanto, los parámetros que rigen el bucle de funcionamiento son temporales, como se muestra en la Figura 3.

```
t0=0; %el tiempo inicial de la simulación
tf=10; %el tiempo final de la simulación
paso=0.1; %el incremento de tiempo
tspan=t0:(tf/paso)-1; %los instantes de tiempo simulados
for i=2:(tf/paso)  %i es el contador del tiempo
   ....
end
```

Figura 3: Parámetros temporales en la simulación de las redes temporizadas

Además, el algoritmo de implementación debe tener en cuenta el tipo de retardo de las transiciones, tal como vimos anteriormente. Veamos cómo queda el algoritmo para los dos tipos de retardos.

4.3.1. Retardos en la sensibilización de la transición

Para implementar este tipo de retardo hay que asegurar no sólo que los lugares cuentan con el número de marcas suficientes, sino además que lo hacen desde hace cierto tiempo, es decir, que las últimas filas de la matriz Y (en las columnas que correspondan a la transición) lo cumplen.

La forma de implementarlo consiste simplemente en sustituir en la condición de disparo de cada transición de las que teníamos en la Figura 2 (if Y(i, l_{e1})>= p_{e1} & ... & Y(i, l_{ea})>= p_{ea} & Y(i, li_1)< p_{i1} & ... & Y(i, l_{ib})< p_{ib} & evento) las comparaciones escalares de los arcos no inhibidores por comparaciones vectoriales, es decir:

la comparación:	Y(i, l_{ex})>= p_{ex}	
pasa a:	Y(i-tim/paso:i, l_{ex}) >= p_{ex} * ones(tim/paso+1,1)	(1)

donde l_x es un lugar incidente a la transición (con arco no inhibidor), p_x es el peso de su arco, y *tim* es el valor de la temporización.

Para el caso frecuente de arcos con peso unitario, esa expresión se reduce considerablemente:

la comparación:	Y(i, l_{ex})>= 1	
pasa a:	all(Y(i-tim/paso:i, l_{ex}))	(2)

donde *all* es una función booleana que se cumple cuando todos los elementos de una matriz son no nulos.

Hay que tener en cuenta que las matrices no pueden tener índices negativos o cero (para que no dé error el programa en su funcionamiento). Por lo tanto, se debe impedir que en las primeras iteraciones el valor *i-tim/paso* dé errores por ese concepto. Eso se puede impedir de dos formas:

1) Indicando en vez del valor *i-tim/paso* el valor *max([1 i-tim/paso])*. De esta manera siempre será un valor positivo no nulo, como pretendíamos. En este caso, si el peso del arco es distinto de cero (el caso primero estudiado, según las ecuaciones (1)) hay que tener en cuenta que la matriz debe tener las mismas dimensiones que la matriz con la que se compara, por lo cual también hay que sustituir *ones(tim/paso,1)* por *ones(min([1 i-tim/paso]) , 1)*. Con todo esto una comparación queda de la siguiente forma:

```
if Y(max([1 i-tim/paso]):i,l_e1)>=
p_e1*ones(min([i tim/paso+1]),1)&...                    (3)
```

y para arcos con peso unitario queda:

```
if all(Y(max([1 i-tim/paso]):i, l_e1))&...              (4)
```

2) Imponiendo como condición al disparo de la transición, que haya pasado el tiempo mínimo. Para ello se incluye inmediatamente después del `if` (en las transiciones con temporización) como primera condición booleana:

```
i>tim/paso.                                             (5)
```

De esta manera, antes de que pase el tiempo de la temporización, sigue siendo negativo o nulo el valor *i-tim/paso* pero deja de ser problema porque en esos casos la condición anterior en el `if` (la que acabamos de introducir) ha resultado falsa y ya no se sigue evaluando. Evidentemente la condición indicada anteriormente debe ser la primera dentro de la instrucción `if` que testea si debe efectuarse el disparo de la transición con retardo.

4.3.1.1. Ejemplo

Como ejemplo veamos la realización de una RdP sencilla, como la que se muestra en la Figura 4.

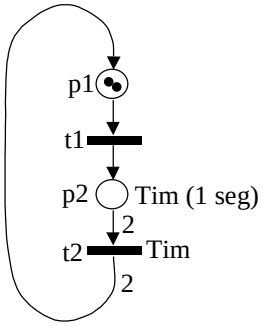


Figura 4: RdP con retardo en la sensibilización de la transición.

La traducción, según el primer método, se debe realizar según el algoritmo de la Figura 2, iterando con los parámetros temporales de la Figura 3 y de (1) (ó (2) si los arcos son de peso unitario), y sustituyendo las condiciones correspondientes a los arcos no inhibidores de incidencia previa a las transiciones con temporización de acuerdo a (3) en los casos generales y de acuerdo a (4) cuando los arcos tienen peso unitario.

Con todo ello la simulación es la que se presenta en la Figura 5.

```
close;
hold off;
t0=0;
tf=10;
paso=0.1;
tspan=t0:tf/paso-1;
tim=1;
Y=[2 0];
for i=2:tf/paso
   Y(i,:)=Y(i-1,:);
   if Y(i,1)>=1
      Y(i,1)=Y(i,1)-1; Y(i,2)=Y(i,2)+1;
   end
   if Y(max([1 i-tim/paso]):i, 2) >=2*ones(min([i tim/paso+1]),1)
      Y(i,2)=Y(i,2)-2; Y(i,1)=Y(i,1)+2;
   end
end
subplot(2,1,1), plot(tspan/10,Y(:,1),'r');
subplot(2,1,2), plot(tspan/10,Y(:,2),'b');
```

Figura 5: Programa correspondiente a la red de la Figura 4 según la forma primera. .

La traducción según el segundo método se debe realizar también según el algoritmo de la Figura 2, iterando con los parámetros temporales de la Figura 3 y de (1) (ó (2) si los arcos son de peso unitario), y poniendo como primera condición de la evaluación de las transiciones temporizadas la indicada en (5).

La simulación está representada en la Figura 6.

```
close;
hold on;
t0=0;
tf=10;
paso=0.1;
```

```
tspan=t0:tf/paso-1;
tim=1;
Y=[2 0];
for i=2:tf/paso
   Y(i,:)=Y(i-1,:);
   if Y(i,1)>=1
      Y(i,1)=Y(i,1)-1; Y(i,2)=Y(i,2)+1;
   end
   if i>tim/paso & Y(i-tim/paso:i, 2) >=2*ones(tim/paso+1,1)
      Y(i,2)=Y(i,2)-2; Y(i,1)=Y(i,1)+2;
   end
end
subplot(2,1,1), plot(tspan/10,Y(:,1),'r');
subplot(2,1,2), plot(tspan/10,Y(:,2),'b');
```

Figura 6: Programa correspondiente a la red de la Figura 4 según la forma segunda.

4.3.1.2. Coherencia

Ambos métodos analizados y empleados en el ejemplo anterior han de ser coherentes con el caso de que una transición sin temporizar es lo mismo que otra con tim=0. Efectivamente, en esos casos la aplicación de las ecuaciones (3), (4) y (5) no tiene ningún efecto:

- Con *tim=0* la expresión (3) queda así:

$$\texttt{if Y(max([1 i]):i,}l_{e1}\texttt{)>=}\ p_{e1}\texttt{*ones(min([i 1]),1)\&...}$$

y como i≥1, `max([1 i]=i y min([i 1]=1, con lo que ones(min([i 1]),1)=1, y por tanto la expresión queda:`

$$\texttt{if Y(i,}l_{e1}\texttt{)>=}\ p_{e1}\ \texttt{\&...}$$

que corresponde a la condición para una transición no temporizada

- Con *tim=0* la expresión (4) queda así:

$$\texttt{if all(Y(max([1 i]):i,}\ l_{e1}\texttt{))\&...}$$

y como i≥1, `max([1 i]=i, con lo que all(Y(max([1 i]):i,` l_{e1}`)) es equivalente a`

$$\texttt{if Y(i,}l_{e1}\texttt{)~=}\ 0\ \texttt{\&...}$$

y como Y es una matriz de términos no negativos, el que sea distinto de cero es porque es mayor que cero, y por tanto de nuevo corresponde a la condición para una transición no temporizada.

- Con *tim=0* la expresión (5) queda

i>0

que es algo que siempre se da por definición de i, con lo cual la condición que se ha añadido respecto a la transición sin temporizar no afecta, y por tanto son equivalentes.

Evidentemente esta coherencia en las ecuaciones (3), (4) y (5) es necesaria (aunque no suficiente) para comprobar que son correctas. Este tipo de comprobaciones resulta muy interesante para detectar equivocaciones en los índices o en las desigualdades, cosa relativamente frecuente si no se comprueba.

Esta coherencia permite además considerar el método con transiciones temporizadas (y peso de los arcos en general distinto de 1) como un caso general, en el que las transiciones sin temporización simplemente tendrán valor nulo de la constante de temporización. Esto hace que sea más sencillo para implementar de forma automática, puesto que no hay que distinguir entre tipos de transiciones, aunque el programa será algo menos eficiente (cosa poco importante en general, puesto que se emplea off-line).

4.3.2. Retardos en el disparo de la transición

En la implementación de este tipo de retardos hay que tener en cuenta todo lo que se ha visto en el retardo anterior y además hay que implementar que en cada disparo de la transición se comienza a contar, es decir, hay que asegurar que han pasado τ_{act} unidades de tiempo desde el último disparo de la transición.

Una forma de implementar todo esto consiste en memorizar los instantes de disparo de las transiciones, e imponer para los nuevos disparos que haya transcurrido el tiempo necesario. Entonces al programa que teníamos hay que añadir lo siguiente:

- antes de las iteraciones: *disparo=ones(1,num_transic);* %para definir la matriz de instantes de disparo

- en las condiciones de las transiciones: *if i-disparo(trans_actual)>=(τ_{act} /paso) & ...*

- en las acciones dentro de los if: *disparo(trans_actual)=i*

donde:

num_transic	es el número de transiciones con retardos de este tipo
trans_actual	indica el número de la transición que se está tratando dentro de la matriz disparo
τact	indica la temporización de la transición que se está tratando

Conviene fijarse que en la simulación de este tipo de retardos no es necesario tener en cuenta la posibilidad de que los índices se conviertan en negativos o ceros, como ocurría en el caso anterior, puesto que ahora la primera condición dentro de los `if` que testean el disparo de estas transiciones, es *i-disparo(trans_actual)>=τ_{act}/paso*. Como disparo(trans_actual) $\geq 1 \Rightarrow$ i $\geq$ (τ_{act} /paso) +1 $\Rightarrow$ i - τ_{act} / paso $\geq$ 1, y como $\tau_{act} \geq 0 \Rightarrow$ i $\geq$ 1, con lo cual está demostrado que siempre será índice positivo no nulo el que se

evalúe después de dicha condición. De nuevo hay que asegurarse de que sea esa la primera condición en el `if` (es decir, sí influye el orden de las condiciones lógicas).

4.3.2.1. Ejemplo

Como ejemplo veremos uno similar a los anteriores, que es el que se muestra en la Figura 7. El programa que implementa este ejemplo está impreso en la Figura 8.

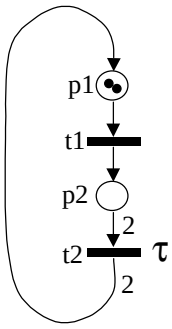


Figura 7: RdP con retardo en el disparo de la transición.

```
close;
hold on;
t0=0;
tf=10;
paso=0.1;
tspan=t0:tf/paso-1;
tim=1;
Y=[2 0];
disparo=[1];
for i=2:tf/paso
   Y(i,:)=Y(i-1,:);
   if Y(i,1)>=1
      Y(i,1)=Y(i,1)-1; Y(i,2)=Y(i,2)+1;
   end
   if i-disparo(1)>=tim/paso & Y(i-tim/paso:i,2)>=2*ones(tim/paso+1,1)
      Y(i,2)=Y(i,2)-2; Y(i,1)=Y(i,1)+2;disparo(1)=i;
   end
end
subplot(2,1,1), plot(tspan/10,Y(:,1),'r');
subplot(2,1,2), plot(tspan/10,Y(:,2),'b');
```

Figura 8: Programa correspondiente a la red de la Figura 7.

4.3.2.2. Coherencia

E igualmente podemos comprobar que esta implementación de la transición temporizada es coherente con el hecho de que una temporización igual a cero equivale a no tener temporización. Así, la condición que se ha impuesto nueva,

$$i\text{-}disparo(trans\_actual) >= \tau_{act}/paso$$

cuando $\tau_{act} = 0$ queda así:

i-disparo(trans_actual)>=0

lo cual se cumple siempre y no impone nada nuevo, ya que *disparo* almacena un valor de *i*, de una iteración anterior o de la actual, por lo que nunca puede ser mayor que el i de la iteración en curso.

4.3.3. Red totalmente temporizada

En los dos ejemplos precedentes, en los que se quería mostrar la forma de implementar las transiciones con retardos, se han empleados redes mixtas, desde el punto de vista de contener transiciones con y sin retardo, puesto que tan sólo incluían una transición con retardo cada una.

Se ha realizado así para centrar la atención en ese tipo de transiciones, pero más adelante, en el apartado siguiente, veremos que los métodos de implementación que se han visto anteriormente no son correctos con ese tipo de redes, y deben ser implementados en redes con retardos (de cualquiera de los tipos) en todas sus transiciones.

Sin embargo ese tipo de redes es muy frecuente en la práctica, puesto que a menudo los lugares unidos por transiciones sin retardo se unen (cuando es posible) en un macrolugar, y entonces se tiene una red totalmente temporizada.

4.3.3.1. Ejemplo

Veamos un ejemplo de un sistema industrial que emplea una red totalmente temporizada. Se trata de un ejemplo simple pero real, por lo cual será empleado en varias ocasiones a lo largo de este capítulo, y va a servir de ensayo a escala del sistema real completo y extremadamente complejo que se analiza más adelante en otro capítulo y que constituye uno de los pilares de este trabajo.

El sistema, representado en la Figura 8b, consiste simplemente en dos líneas de entrada, A y B, y una línea de salida. La línea A consiste en una máquina (la máquina 1-a) que carga piezas de tipo A en un buffer (el buffer 1-a). La transición t1a representa el tiempo que tarda la máquina en cargar la pieza, y la transición t2a representa el tiempo que tarda en descargarla. Tanto la máquina 1-a como el buffer 1-a están representados por dos lugares (el lugar que representa los elementos y su complementario que representa los huecos). La línea de entrada de piezas B es exactamente igual a A. Después, la máquina 2 ensambla una pieza de A del buffer 1-a con una pieza de B del buffer 1-b, y la pieza resultante la deposita en el buffer 2. De nuevo t3 y t4 representan los tiempos de cogida y de dejada de las piezas respectivamente. Por último la máquina 3 coge las piezas resultantes (igualmente con tiempo de acceso a coger y dejar) y las saca como producto. Además el número total de piezas en el proceso está limitado (por

ejemplo debido al número de recipientes o a la energía necesaria) lo cual se representa devolviendo desde el final del proceso 'los recipientes' al principio.

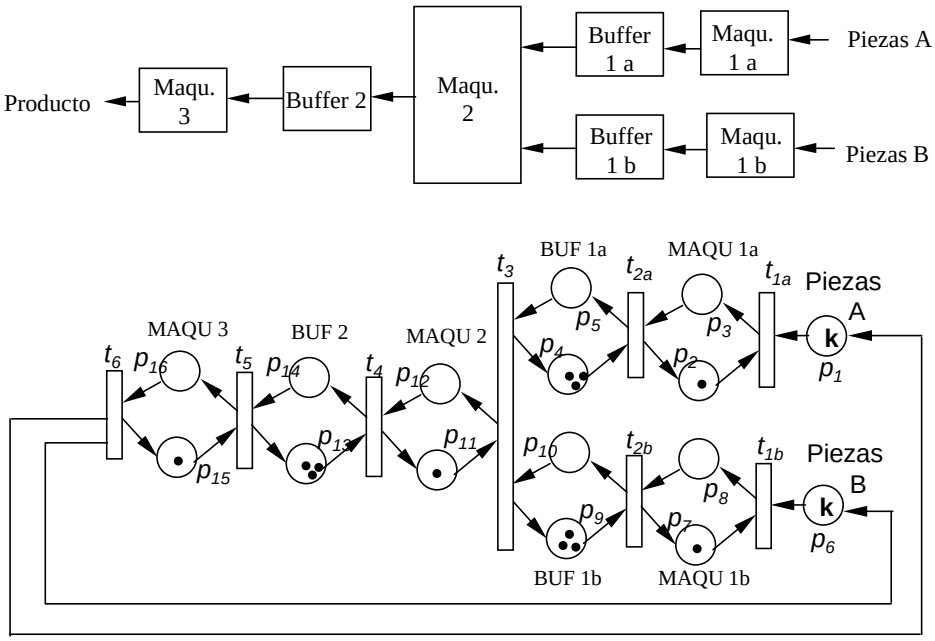


Figura 8-a: Sistema de producción representado mediante una RdP totalmente temporizada

Lo primero que hay que determinar es el tipo de retardos que tenemos (si en el disparo o en la sensibilización). Como la transición representa el tiempo que tarda la máquina en coger o dejar cada pieza, el retardo es en el disparo en todas las transiciones. De todas formas dado que los lugares que representan las máquinas, ya sea cargando o descargando (p2, p3, p7, p8, p11, p12, p15, p16), son lugares binarios, en este caso vuelve a ser indiferente el tipo de retardo (son equivalentes). Esto es algo bastante frecuente, y posiblemente esa sea la razón de que muchas veces no se tenga en cuenta el tipo de retardo, por lo que cuando no es indiferente se cometen errores en el modelado e implementación.

Entonces, según lo visto hasta ahora, el método de implementación sí es correcto para esta red, y el programa resultante se muestra en la Figura 8-b. En dicho programa se ha incluido alguna característica adicional (frecuencia de salida, parametrización de marcado inicial, etc.) a la que haremos referencia más adelante, pero que se ha incluido aquí para no tener que repetir el listado del programa.

```
t0=0;%el tiempo inicial
tf=input('¿Cuántos segundos quieres evaluar?');%el tiempo final
paso=0.1; %cuanto menor paso más exacto el cálculo y la gráfica
tspan=t0:(tf/paso)-1;
clf; %borra la figura actual
hold on; %mantiene el dibujo actual
buffer=3;
k=input('¿Parámetro inicial del sistema?');
Y(1,:)=[k 1 0 buffer 0 k 1 0 buffer 0 1 0 buffer 0 1 0];
trans=[1 1 1 1 1 1 1 1];
produc=0;%lo utilizo para calcular la frecuencia de salida
```

```
for i=2:(tf/paso)  %i es el contador del tiempo
   Y(i,:)=Y(i-1,:);
   if i-trans(1) >= 10 & all(Y(i-10:i,1)) & all(Y(i-10:i,2))
      Y(i,1)=Y(i,1)-1;Y(i,2)=Y(i,2)-1;Y(i,3)=Y(i,3)+1;trans(1)=i;
   end
   if i-trans(2) >= 10 & Y(i-10:i,6)>0 & Y(i-10:i,7)>0
      Y(i,6)=Y(i,6)-1;Y(i,7)=Y(i,7)-1;Y(i,8)=Y(i,8)+1;trans(2)=i;
   end
   if i-trans(3) >= 2 & Y(i-2:i,3)>0 & Y(i-2:i,4)>0
      Y(i,3)=Y(i,3)-1;Y(i,4)=Y(i,4)-
1;Y(i,2)=Y(i,2)+1;Y(i,5)=Y(i,5)+1;trans(3)=i;
   end
   if i-trans(4) >= 2 & Y(i-2:i,8)>0 & Y(i-2:i,9)>0
      Y(i,8)=Y(i,8)-1;Y(i,9)=Y(i,9)-
1;Y(i,7)=Y(i,7)+1;Y(i,10)=Y(i,10)+1;trans(4)=i;
   end
   if i-trans(5)>= 10 & Y(i-10:i,5)>0 & Y(i-10:i,10)>0 & Y(i-
10:i,11)>0
      Y(i,5)=Y(i,5)-1;Y(i,10)=Y(i,10)-1;Y(i,11)=Y(i,11)-1;
      Y(i,4)=Y(i,4)+1;Y(i,9)=Y(i,9)+1;Y(i,12)=Y(i,12)+1;trans(5)=i;
   end
   if i-trans(6)>= 2 & Y(i-2:i,12)>0 & Y(i-2:i,13)>0
      Y(i,12)=Y(i,12)-1;Y(i,13)=Y(i,13)-
1;Y(i,11)=Y(i,11)+1;Y(i,14)=Y(i,14)+1;trans(6)=i;
   end
   if i-trans(7)>= 10 & Y(i-2:i,14)>0 & Y(i-2:i,15)>0
      Y(i,14)=Y(i,14)-1;Y(i,15)=Y(i,15)-
1;Y(i,13)=Y(i,13)+1;Y(i,16)=Y(i,16)+1;trans(7)=i;
   end
   if i-trans(8)>= 2 & Y(i-2:i,16)>0
      Y(i,16)=Y(i,16)-
1;Y(i,15)=Y(i,15)+1;Y(i,1)=Y(i,1)+1;Y(i,6)=Y(i,6)+1;trans(8)=i;
      produc=produc+1;instante(produc)=i;
   end
end
for k=1:produc-1
   periodo(k)=(instante(k+1)-instante(k))/10;%periodo en segundos de
salida de pieza
   frecuencia(k)=10/(instante(k+1)-instante(k));%frecuencia en
segundos de salida de pieza
end
plot(tspan/10,Y(:,16),'r'); %ploteo el lugar 16 en función del tiempo
[filasY,columY]=size(Y);
Yf=[Y(filasY,:)];
disp('El flujo en régimen permanente es');frecuencia
disp('i.e., sale una pieza cada estos segundos');periodo
disp('El último valor encontrado de Y es');Yf
```

Figura 8-b: Programa de implementación del ejemplo.

4.4. Redes mixtas

Como comentábamos en el apartado anterior, todo lo visto hasta ahora en cuanto a las redes temporizadas es válido en sistemas en los que todas las transiciones son temporizadas. Con el término Redes Mixtas nos estamos refiriendo a redes con transiciones tanto sin temporizar como temporizadas (incluso con temporizaciones de

diversos tipos). Cuando es así, hay que tener en cuenta que con lo realizado hasta el momento no estamos garantizando que no "transcurra tiempo" en el disparo de transiciones sin temporizar.

Supongamos una red secuencial de dos lugares y dos transiciones en la que la única transición con retardo sea la que tenemos marcada (Figura 9).

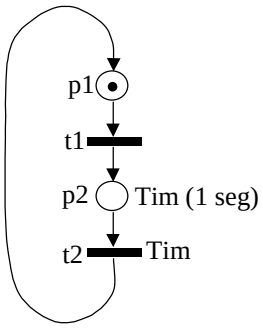


Figura 9: Sistema mixto, con una transición temporizada y otra no temporizada

La simulación de esta red, realizada su implementación tal cual hemos visto hasta ahora, nos da la gráfica que indicamos en la Figura 11. También se incluye el programa que la implementa, en la Figura 10, a modo de ejemplo de sistema con pesos unitarios, de cuyo tipo no se había incluido aún ninguno.

```
close;
hold off;
t0=0;
tf=10;
paso=0.1;
tspan=t0:tf/paso-1;
tim=1;
Y=[1 0];
for i=2:tf/paso
   Y(i,:)=Y(i-1,:);
   if Y(i,1)>=1
      Y(i,1)=Y(i,1)-1; Y(i,2)=Y(i,2)+1;
   end
   if i>=tim/paso & all(Y(i-tim/paso:i, 2))
      Y(i,2)=Y(i,2)-1; Y(i,1)=Y(i,1)+1;
   end
end
subplot(2,1,1), plot(tspan/10,Y(:,1),'r');
subplot(2,1,2), plot(tspan/10,Y(:,2),'b');
```

Figura 10: Implementación de red con arcos incidentes de peso unitario.

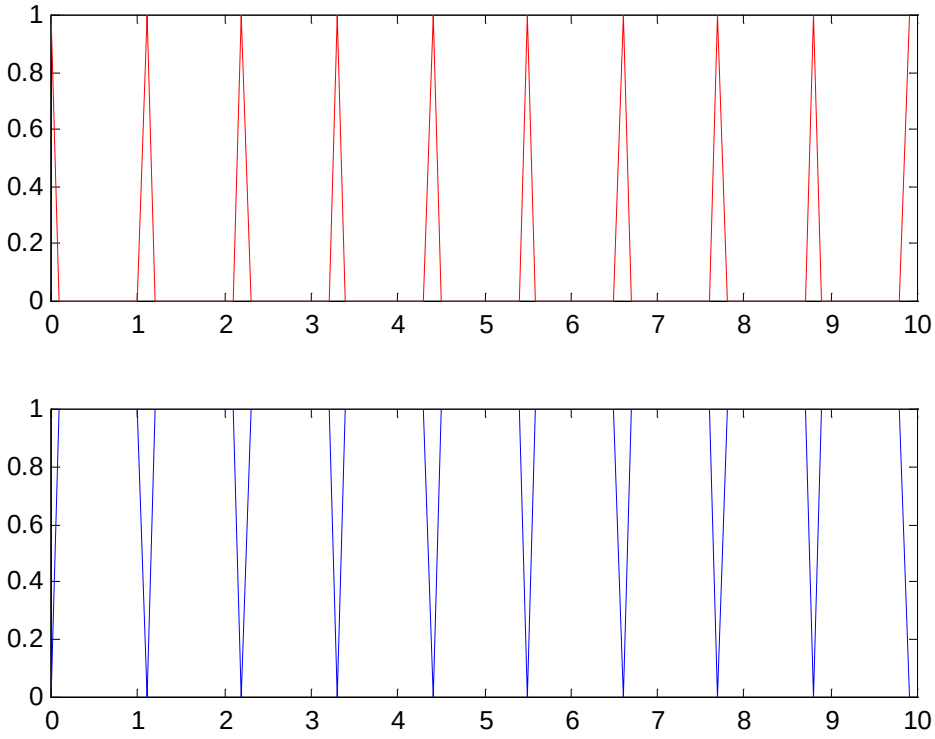


Figura 11: Gráfica del marcado de p1 (arriba) y p2 (abajo) en la simulación del programa de la Figura 10.

Vemos que la marca no aparece constantemente en p2, tal como debería ocurrir puesto que t1 es una transición sin temporizar. Sin embargo, las iteraciones que el programa emplea para mover las marcas a través de transiciones no temporizadas son consideradas como un paso de tiempo. El disparo, que se tenía que producir cada segundo, se produce cada 1,1 segundos.

Cuanto menor sea el paso en comparación con el tiempo de la temporización menor será también el error relativo cometido, pero a costa de multiplicar el esfuerzo computacional para lograr el mismo tiempo de simulación. Por ejemplo, esa misma simulación, con paso de 0.01 (la décima parte del que había) nos da el resultado de la figura 12, en el que se obtiene un periodo de 1.01 segundos frente a 1 segundo que es el valor exacto (error del 1%). Y si lo simulamos con el paso igual al tiempo de simulación se obtiene el de la figura 13, en el que el error es del 100%.

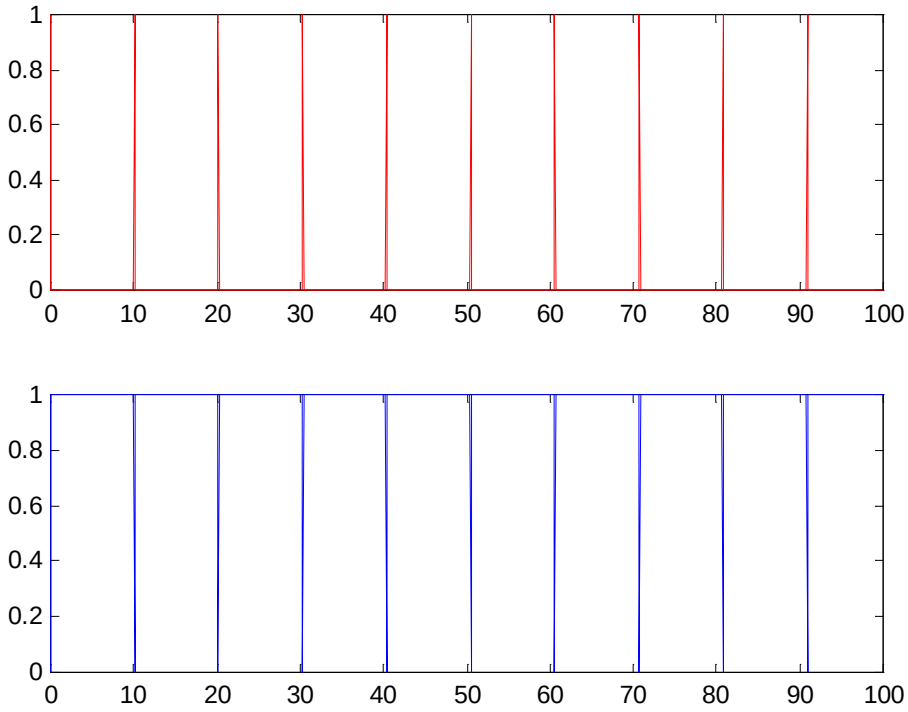


Figura 12: Gráfica con paso 0.01. El error relativo del periodo, frente al teórico es igual a un paso, es decir el 1%.

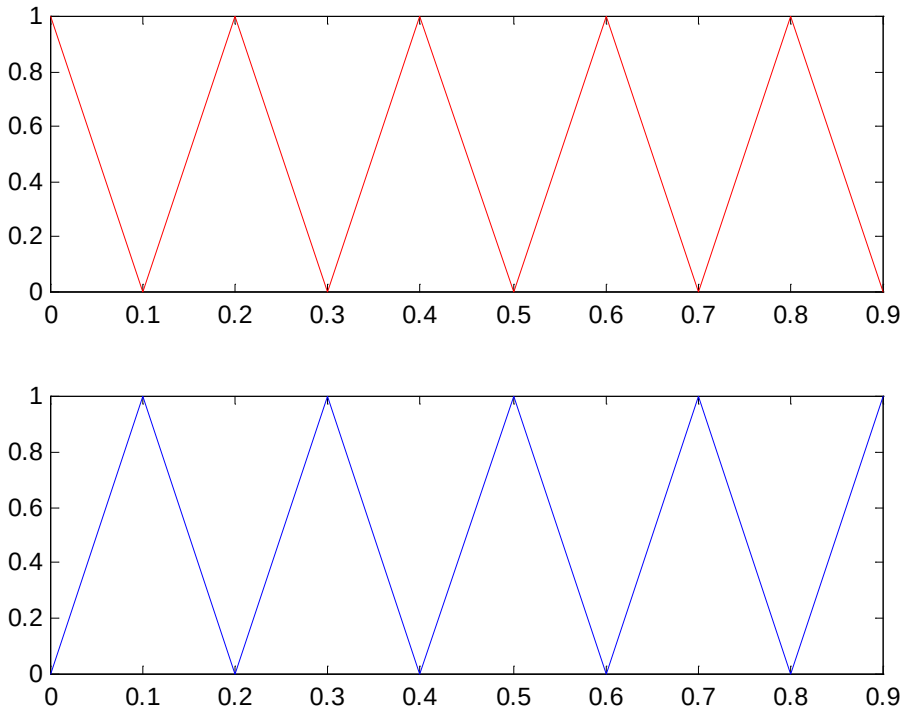


Figura 13: Gráfica con paso 0.1. El error relativo del periodo, frente al teórico es igual a un paso, es decir el 100%.

4.4.1. Retardo en el disparo de la transición

Para corregir el defecto que acabamos de estudiar en las redes mixtas (con transiciones temporizadas y sin temporizar), se debe realizar la implementación de la red siguiendo estos pasos:

- Se hace evolucionar la red sin avanzar en la iteración que computa el tiempo, durante sucesivas veces

- Cuando se recorren todas las transiciones sin que ninguna de ellas se dispare, entonces se avanza en la iteración del tiempo y se vuelve al paso anterior

La forma de implementar esos pasos consiste en incluir dentro del bucle *for i* que computa el tiempo, otro bucle con *control final* del mismo (repeat...until, o do...while). Dado que en Matlab no existen bucles de control final, en la siguiente implementación se construirá a partir de uno con control al principio (un while...do), lo cual no constituye ninguna complicación importante, pero conviene tenerlo en cuenta para realizarlo de forma más sencilla cuando se realice la implementación en otro tipo de lenguajes que sí que dispongan de ese tipo de estructuras (Pascal, Java, etc.).

Con todo ello el algoritmo queda tal cual se indica en la Figura 14.

```
definir el tiempo inicial(t0), el tiempo final (tf) y el paso (paso)
definir los instantes de simulación: tspan=t0:tf/paso-1;
definir el marcado inicial: Y=[marcado inicial];
inicializar los disparos iniciales: disparo=ones(1,num_temporizac);
for i=1:tf/paso
   seguir=1;
   while seguir ~=0
      seguir=0;
      ....
      %comienzo de una transición cualquiera con retardo
      if i-disparo(num_transic)>=τtr/paso & condiciones & evento
         acciones;disparo(1)=i;seguir=1;
      end %del if
      %fin de una transición cualquiera con retardo
      ....
   end %del while
   if i<tf/paso
      Y(i+1,:)=Y(i,:);
   end %del if
end %del for
crear las gráficas: plot(tspan*paso,Y(:,lugar_a_plotear),'color');
```

Figura 14: Algoritmo para la implementación de redes mixtas con retardos en el disparo de la transición.

Hay que prestar atención a que ahora se itera desde i=1, y se amplía la matriz *Y* en una fila más al final de la iteración, en vez de al principio. Esto es porque ahora sólo se

almacenan los marcados finales en cada instante de tiempo, por lo cual ni siquiera el marcado inicial necesariamente coincide con el primero que aparezca en las gráficas, ya que puede evolucionar antes de que pase tiempo. Por ejemplo esto es lo que ocurre en el último ejemplo (el de la Figura 9), en el que el marcado inicial es [1 0] y sin embargo el marcado almacenado para el primer instante (valor 1 de la matriz, pero que equivale a tiempo cero) es [0 1].

4.4.1.1. Ejemplo

Como ejemplo vamos a implementar según el anterior algoritmo la RdP de la figura 15. El programa que se obtiene está representado en la Figura 16.

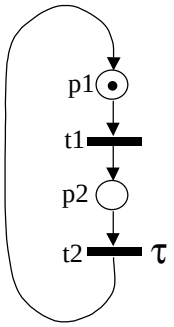


Figura 15: Ejemplo de sistema mixto
en cuanto al tipo de transiciones

```
close;
hold off;
t0=0;
tf=10;
paso=0.1;
tspan=t0:tf/paso-1;
tim=1;
Y=[1 0];
disparo(1)=1;
for i=1:tf/paso
   seguir=1;
   while seguir ~=0
      seguir=0;
      if Y(i,1)>=1
         Y(i,1)=Y(i,1)-1; Y(i,2)=Y(i,2)+1;seguir=1;
      end %del if
      if i-disparo(1)>=tim/paso & all(Y(i-tim/paso:i, 2))
         Y(i,2)=Y(i,2)-1; Y(i,1)=Y(i,1)+1;disparo(1)=i;seguir=1;
      end %del if
   end %del while
   if i<tf/paso
      Y(i+1,:)=Y(i,:);
   end %del if
end %del for
subplot(2,1,1), plot(tspan/10,Y(:,1),'r');
subplot(2,1,2), plot(tspan/10,Y(:,2),'b');
```

Figura 16: Programa que implementa el sistema de la Figura 15.

La Figura 17 muestra la gráfica correspondiente a la simulación, que efectivamente corresponde al resultado teórico que se debía obtener.

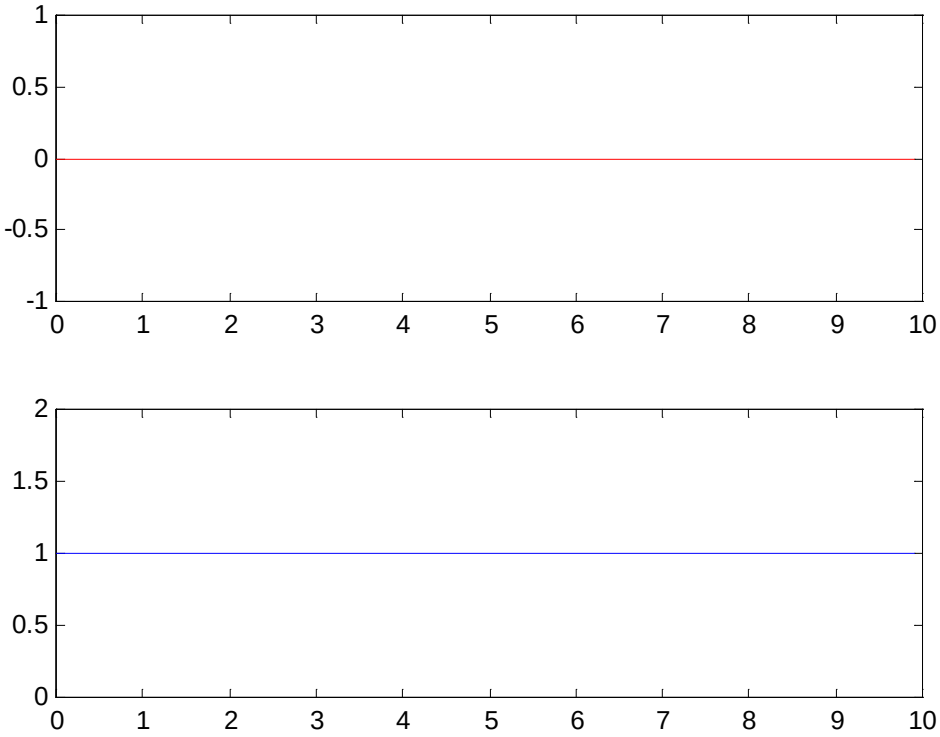


Figura 17: Simulación del sistema de la Figura 15. Marcado de p1 (arriba) y p2 (abajo).

Aunque en las gráficas del marcado no se aprecia, puesto que son constantes, si representamos la gráfica del disparo de la transición temporizada vemos que efectivamente el periodo es de 1 segundo exacto, como se aprecia en la gráfica de la Figura 18.

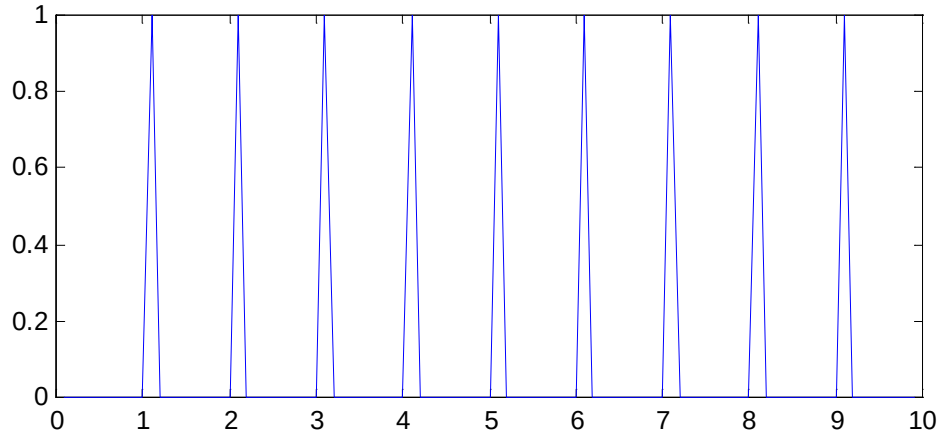


Figura 18: Gráfica del disparo de la transición con temporización según la implementación de la Figura 16. Los lugares donde la gráfica vale 1

indican el instante del disparo de la transición. La otra transición se dispara exactamente en los mismos instantes

4.4.2. Retardo en la sensibilización de la transición

Sin embargo, el algoritmo que hemos visto (Figura 14) no es correcto cuando tenemos retardos en la sensibilización de la transición. Implementando el sistema de la Figura 9 según el algoritmo de la Figura 14 seguimos obteniendo una simulación incorrecta. La razón es que ahora cuando el lugar p2 está sin marca no se tiene en cuenta el hecho de que no la tiene, ya que en el mismo instante de tiempo (antes de que el tiempo avance) volverá a tenerla. Entonces lo que hay que hacer es simplemente tener en cuenta desde qué instante está con marcado suficiente e imponer que haya pasado el tiempo de temporización desde entonces, muy similar a lo que se hacía con las transiciones con retardo en el disparo. Esa actualización de los instantes en los que se sensibiliza la transición (y por tanto empieza a temporizar) se realiza como una acción más, asociada al disparo de cualquier transición (incluso transiciones sin temporización) que pueda afectar a la transición temporizada en cuestión, siempre que haya sido dicho disparo el que haya producido la nueva sensibilización. Por lo tanto, en estos casos, el algoritmo queda como se muestra en la Figura 19.

```
definir el tiempo inicial(t0), el tiempo final (tf) y el paso (paso)
definir los instantes de simulación: tspan=t0:tf/paso-1;
definir el marcado inicial: Y=[marcado inicial];
inicializar instantes de sensibilización: sensib=ones(1,num_temporiz);
for i=1:tf/paso
   seguir=1;
   while seguir ~=0
      seguir=0;
      ....
      %comienzo de una transición cualquiera
      if i-sensib(num_transic)>=tim_tr/paso & condiciones & evento
         acciones;disparo(1)=i;seguir=1;
         if tr_sensib
            sensib(num_transic)=i;
         end
      end %del if
      %fin de una transición cualquiera
      ....
   end %del while
   if i<tf/paso
      Y(i+1,:)=Y(i,:);
   end %del if
end %del for
crear las gráficas: plot(tspan*paso,Y(:,lugar_a_plotear),'color');
```

Figura 19: Algoritmo de implementación de redes mixtas con transiciones con retardo en la sensibilización

4.4.2.1. Ejemplo

Y siguiendo dicho algoritmo, la implementación del problema de la Figura 9 queda como se muestra a continuación (Figura 20), y su simulación coincide con las de las Figuras 17 y 18, que efectivamente ahora sí corresponde con el análisis teórico.

```
close;
hold off;
t0=0;
tf=10;
paso=0.1;
tspan=t0:tf/paso-1;
tim=1;
Y=[1 0];
sensib(1)=1;
for i=1:tf/paso
   seguir=1;
   while seguir ~=0
      seguir=0;
      if Y(i,1)>=1
         Y(i,1)=Y(i,1)-1; Y(i,2)=Y(i,2)+1;seguir=1;
         if Y(i,2)==1 %sensibilización tr2, que es la nº1
            sensib(1)=i;
         end %del if
      end %del if
      if i-sensib(1)>=tim/paso & all(Y(i-tim/paso:i-1, 2))
          Y(i,2)=Y(i,2)-1; Y(i,1)=Y(i,1)+1;disparo(1)=i;seguir=1;
      end %del if
   end %del while
   if i<tf/paso
      Y(i+1,:)=Y(i,:);
   end %del if
end %del for
subplot(2,1,1), plot(tspan/10,Y(:,1),'r');
subplot(2,1,2), plot(tspan/10,Y(:,2),'b');
```

Figura 20: Implementación del ejemplo de la Figura 9 según el algoritmo de la Figura 19.

Por lo tanto cuando tengamos redes mixtas son éstas las formas de implementación en Matlab, o en cualquier otro lenguaje de propósito general, en función del tipo de transición con retardo que tengamos.

El paso máximo que se debe emplear es el máximo común divisor de los retardos de todas las temporizaciones, en cualquier unidad de tiempo en la que se obtengan valores enteros. Por ejemplo, si hay una temporización de 0.2 segundos y otra de 0.5, el máximo común divisor de 2 décimas y 5 décimas es 1 décima de segundo, que es el valor del paso máximo (es decir, 0.1 segundos). Además, es ese paso el que se debe emplear siempre (el máximo), porque cualquier otro paso divisor de éste dará el mismo resultado pero multiplicando el esfuerzo computacional.

4.4.3. Retardo en la sensibilización y en el disparo de la transición

Hasta aquí se ha analizado por separado las peculiaridades a la hora de traducir cada uno de los tipos de transición. Pero nada impide aplicar los algoritmos a sistemas con transiciones de los dos tipos, simplemente empleando en cada transición el método que le corresponda.

4.5. Redes mixtas con información total

4.5.1. Información de disparos con retardo y sin retardo

El método que se ha empleado en la simulación de las redes que incluyen transiciones temporizadas y sin temporizar ha consistido en hacer evolucionar todas las transiciones sin temporizar hasta que se queden bloqueadas, sobre una misma fila de las matrices de simulación, y en ese momento disparar las transiciones con retardo en una nueva fila, que de nuevo será modificada si se puede evolucionar sin que transcurra tiempo (si hay transiciones sin retardo vivas).

Con esa forma de realizar la simulación guardamos la información de todos los marcados que se producen en cada instante de tiempo, pero se pierden las evoluciones de las transiciones sin temporizar. Normalmente es mejor así, puesto que de otra forma puede ser desbordante la cantidad de información almacenada.

Sin embargo en otras ocasiones se necesita conocer la evolución de "todos" los pasos en la evolución de la red, normalmente para detectar errores que no se sabe por qué ocurren. Entonces necesitaremos realizar la simulación de distinta manera, para guardar toda esa información.

En primer lugar se tendrá que almacenar información del instante de tiempo que corresponde a cada uno de los pasos de la simulación, ya que en general habrá varios pasos dentro del mismo instante de tiempo (los que deriven del disparo de transiciones no temporizadas). Otra opción es guardar en una matriz el paso que corresponde a cada instante de tiempo, pero de esta manera luego se complica la representación gráfica.

Siguiendo con el análisis del algoritmo de implementación, guardando la información de las evoluciones sin tiempo, vemos que ahora no va a servir la forma que teníamos de comprobar que había transcurrido el tiempo necesario para la temporización, debido a que ahora comprobar el marcado en tres lugares anteriores no quiere decir que sean tres unidades de tiempo. Por eso se va a construir un algoritmo que rompe con la línea de los anteriores, y en el que se va a almacenar el valor de los temporizadores a medida que se avance en la simulación, para comprobarlo con el valor de disparo. De esta manera ya no necesitaremos la información que antes guardábamos en las variables sensibilización y disparo.

Este algoritmo se asemeja más a la forma de evolución de las RdP, y se aleja algo de la forma de implementarlo en los PLCs (dónde no se escribe sobre los temporizadores). La

ventaja que tenía una implementación similar a la de los PLCs es que muchas veces esta simulación se emplea para detectar errores en la implementación sobre el propio autómata programable, y de esta manera se encontraban más fácilmente.

Con todo lo comentado el algoritmo queda como se muestra en la figura 21.

```
definir el tiempo inicial(t0), el tiempo final (tf) y el paso (paso)
definir el marcado inicial: Y=[marcado inicial];
definir los tiempos de retardo de las transiciones: Tim=[tiempos de
retardo];
inicializar los tiempos de los retardos: T=zeros(1,num_temporiz);
tiempo= 0;i=1;tspan(1)=0;

while tiempo<=tf
%actualización de los parámetros de la iteración
i=i+1;
Y(i,:)=Y(i-1,:);
tspan(i)=tiempo
seguir=0; %indica que no hay disparos si no se aumenta el tiempo
%fin de actualización de los parámetros de la iteración

%test de disparo de las transiciones
...
      %se repite lo siguiente para cada transición
      if T(tr_actual)>= Tim(tr_actual)& %esto si es temporizada
      condiciones & evento
         acciones;
         T(tr_actual)=0;%esto si era un retardo en el disparo
         seguir=1;
      end %del if
      %se repite lo anterior para cada transición
...
%fin del test de disparo de las transiciones

% actualización del tiempo
if seguir=0
   tiempo=tiempo+paso;
   i=i-1; %elimino la última fila, que no ha cambiado
end
% fin de actualización del tiempo

% actualización de los temporizadores
...
% se repite lo siguiente por cada transición con retardo
if condiciones
   if seguir=0 & T(tr_actual)< Tim(tr_actual)
      T(tr_actual)=min([T(tr_actual)+paso Tim(tr_actual)]);
   end
else
T(tr_actual)=0;
end %del if
% se repite lo antrior por cada transición con retardo
...
% fin de actualización de los temporizadores
```

```
end %del while

crear las gráficas: plot(tspan,Y(:,lugar_a_plotear),'color');
```

Figura 21: Algoritmo de implementación de redes mixtas guardando información de disparos con retardo y sin retardo.

Las estructuras empleadas son:

Y:	Matriz con tantas filas como marcados almacenados, y tantas columnas como marcados
tspan:	Vector que indica el tiempo de simulación de cada marcado que se ha almacenado. Tiene tantos elementos como filas Y
T:	Vector con una columna para cada transición con retardo. Indica el tiempo que lleva temporizando cada temporizador
Tim:	Vector que almacena el tiempo de retardo de cada transición temporizada
tiempo:	Variable con la que seguimos el tiempo de simulación
i:	Variable con la que iteramos los diversos marcados del sistema, es decir, el número de columnas de Y

4.5.2. Ejemplo

Como ejemplo vamos a analizar el sistema que ya ha sido analizado en la figura 15. Como se trata de un sistema binario, es también exactamente equivalente al de la figura 9, es decir, la transición puede ser considerada con cualquiera de los dos tipos de retardo y el resultado no varía.

Según el algoritmo de la Figura 21, el programa resultante es el que se muestra en la Figura 22, y el marcado de los lugares queda como se muestra en la Figura 23.

```
close;
hold off;
t0=0;
tf=10;
paso=1;
Tim=[1];
Y=[1 0];
T=[0];
tiempo=0;
i=1;
tspan(1)=0;

while tiempo<=tf
   i=i+1;
   Y(i,:)=Y(i-1,:);
   tspan(i)=tiempo;
   seguir=0; %indica que no hay disparos si no se aumenta el tiempo
   if Y(i,1)>=1
      Y(i,1)=Y(i,1)-1; Y(i,2)=Y(i,2)+1;seguir=1;
   end
   if T(1)>=Tim(1)& Y(i,2)>=1
      Y(i,2)=Y(i,2)-1; Y(i,1)=Y(i,1)+1;disparo(1)=i;seguir=1;
   end %del if
   if seguir==0
```

```
      tiempo=tiempo+paso;
      i=i-1; %elimino la última fila, que no ha cambiado
   end
   if Y(i,2)>=1
      if seguir==0 & T(1)<Tim(1)
         T(1)=min([T(1)+paso Tim(1)]);
      end
   else
   T(1)=0;
   end %del if
end %del while

subplot(2,1,1), plot(tspan,Y(:,1),'r');
subplot(2,1,2), plot(tspan,Y(:,2),'b');
```

Figura 22: Programa para implementar el sistema de las figuras 9 ó 15 según el algoritmo de la Figura 21.

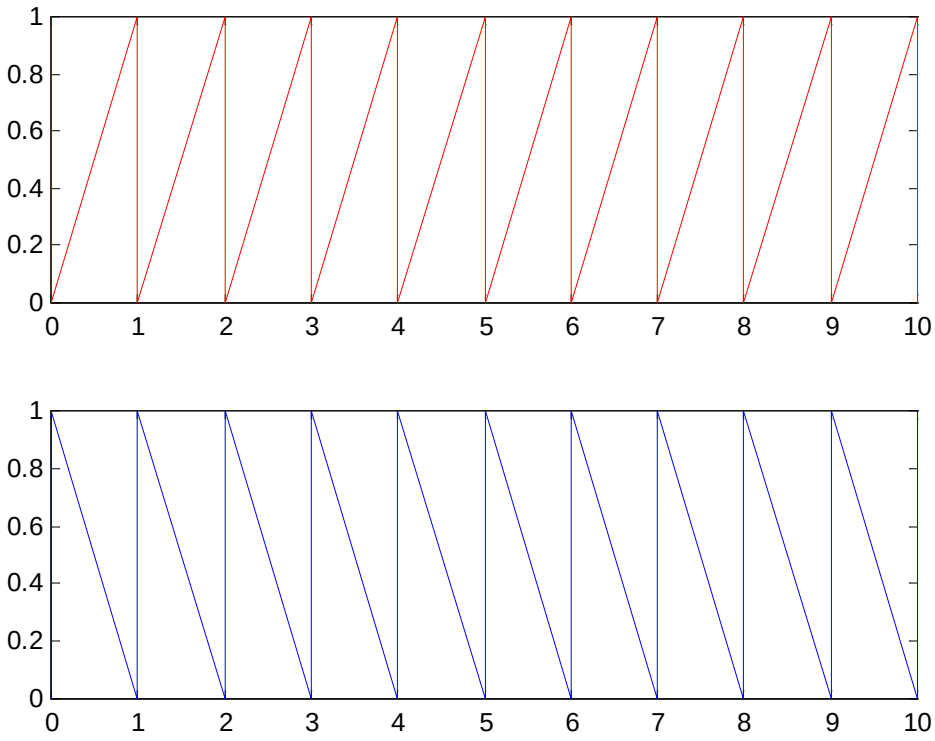


Figura 23: Marcado de los lugares del ejemplo de las Figuras 9 ó 15 según el programa de la Figura 22.

Vemos que, ahora guardamos toda la información de la evolución, es decir, sabemos que en el instante 1 segundo el lugar 1 comenzó teniendo una marca que perdió en ese mismo instante, y el lugar 2 comenzó sin marcas y ganó una en ese mismo instante. Además vemos que mediante este tipo de algoritmo no hace falta compensar el efecto de los índices en las matrices ya que las gráficas se realizan según el tiempo, en vez de según los pasos.

También podemos fijarnos que si se sustituye el paso ideal (1 segundo en este caso) por un divisor suyo (0.5, 0.2, etc.) el espacio en el que se almacena la información es

exactamente el mismo (aunque se realizarán más iteraciones, los resultados repetidos se eliminan).

4.5.3. Información total

Pero hay que tener en cuenta que cuando decimos que se guarda la información de todos los pasos eso no es del todo cierto. En realidad en el algoritmo propuesto (Figura 21) se guarda el análisis dentro de cada paso de tiempo como si fuese una red no temporizada, tal cual la simulábamos al principio (algoritmo de la Figura 2). Sin embargo hay que tener en cuenta que en una iteración del algoritmo se recorren secuencialmente todas las transiciones. Por lo tanto, dependiendo del orden en que se hayan situado las transiciones en el programa, puede ocurrir que en una secuencia de varios lugares consecutivos una marca evolucione desde el primero hasta el último en una sola iteración. En ese caso no se está guardando la información de todas las evoluciones de la red.

Normalmente esto se realiza así porque suele ser suficiente con la información que se recoge de esta manera, y se ahorra mucha información, pero si se desea realizar una sola evolución cada iteración, y por tanto guardar absolutamente toda la información sobre la evolución del sistema, tan sólo hay que cambiar la secuencia de estructuras IF...END que testean el disparo de las transiciones por una única estructura de IF...ELSEIF...ELSEIF...END. Es decir, en el algoritmo de la Figura 21 habría que hacer la sustitución que se indica en la Figura 24.

```
%test de disparo de las transiciones
...
      %se repite lo siguiente para cada transición
      if T(tr_actual)>= Tim(tr_actual)& %esto si es temporizada
      condiciones & evento
         acciones;
         T(tr_actual)=0;%esto si era un retardo en el disparo
         seguir=1;
      end %del if
      %se repite lo anterior para cada transición
...
%fin del test de disparo de las transiciones
```

Figura 24: Modificación en el algoritmo de la Figura 21 para la implementación de redes mixtas guardando toda la información posible.

4.5.4. Ejemplo

Veamos ahora otro claro ejemplo para distinguir los efectos de simular según el algoritmo de la Figura 21 ó el de la Figura 24. Supongamos que tenemos el sistema de la Figura 25. Se trata de un simple sistema secuencial con un temporizador en la tercera transición. De nuevo se trata de un sistema binario para que sea indiferente el tipo de retardo que se emplee (si no lo fuese bastaría con aplicar el tipo que corresponda, según está indicado en el algoritmo de la Figura 21). La simulación de este ejemplo según los métodos anteriores da un resultado exactamente igual al visto en los ejemplos de las

figuras 9 y 15, puesto que es el mismo ejemplo con un lugar de comportamiento "instantáneo" en medio de los que había.

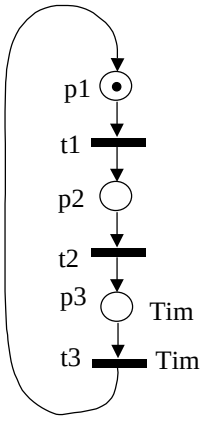


Figura 25: Ejemplo de sitema con diferente comportamiento en simulación según algoritmo de la Fig. 21 o según el de la 24.

Si analizamos el sistema como hemos hecho en el ejemplo anterior según el algoritmo de la Figura 21, se va a guardar la información de lo que ocurre al principio de la secuencia, y antes del disparo, pero la secuencia se realiza de un solo paso, y por lo tanto se pierde la información de la evolución del lugar p2 (Figura 26). Sin embargo si realizamos la simulación guardando toda la información (algoritmo de la Fig. 24), se mantiene la información de todo lo que ocurre en p2, es decir, que pasa de cero a uno y de nuevo a cero en el mismo instante (Figura 27). La figura 28 representa una tabla con los valores que almacena cada uno de los programas. Vemos que en la tabla de la derecha en todos los instantes de simulación (distintos valores de la primera columna) se conoce que p2 ha tenido marcado 1.

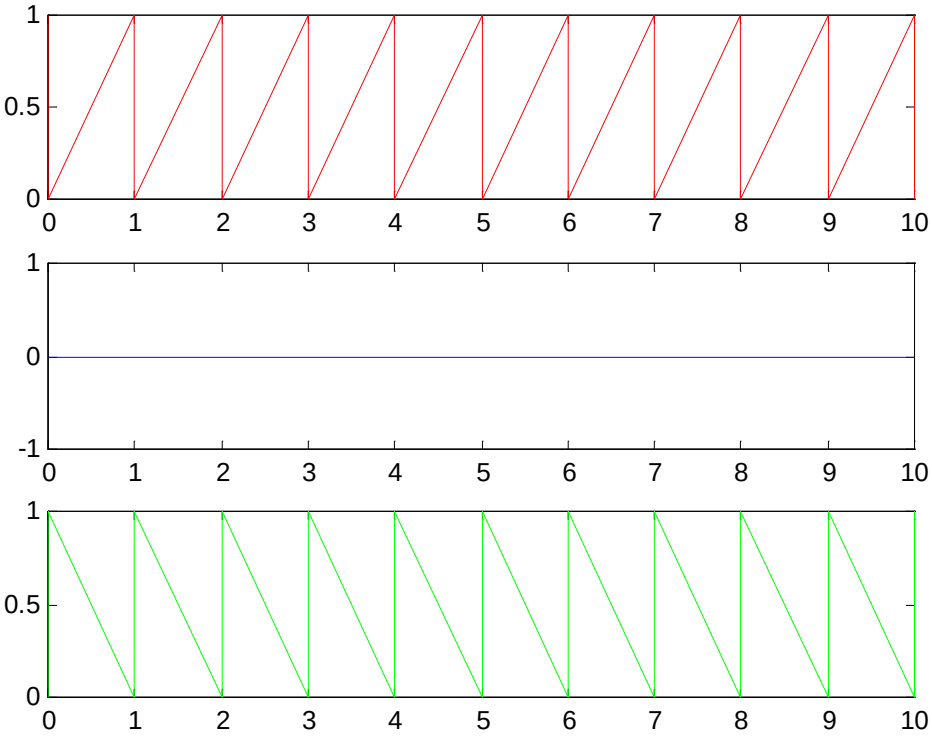


Figura 26: Simulación del sistema del ejemplo según el algoritmo de la Fig. 21.

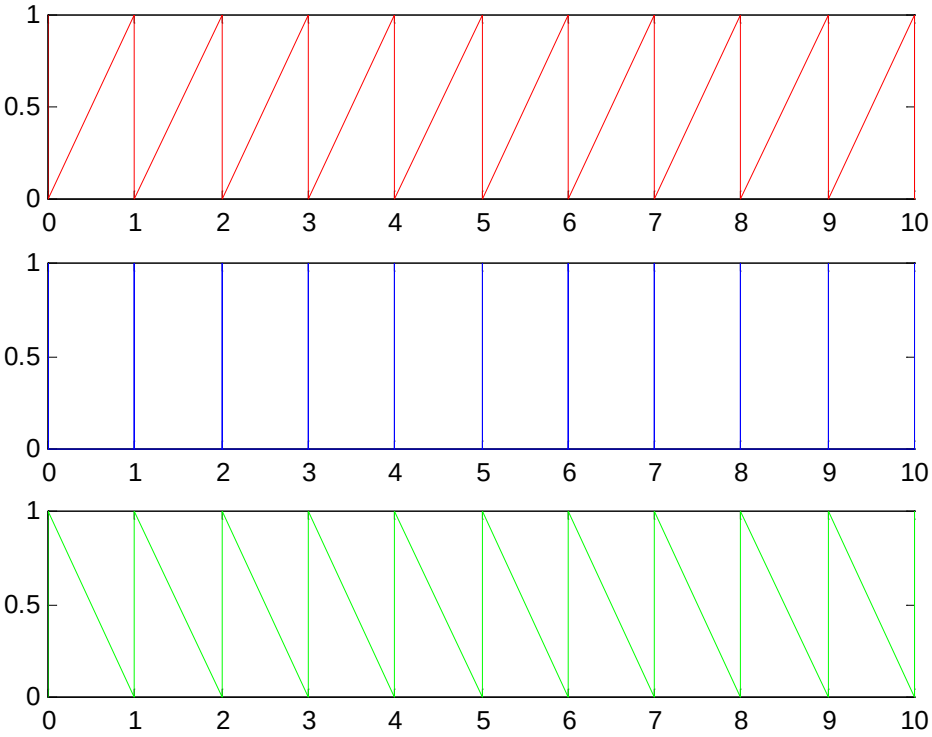


Figura 27: Simulación del sistema del ejemplo guardando toda la información.

Tiempo simulac	p1	p2	p3
0	1	0	0
0	0	0	1
1	1	0	0
1	0	0	1
2	1	0	0
2	0	0	1
3	1	0	0
3	0	0	1
4	1	0	0
4	0	0	1
5	1	0	0
5	0	0	1
6	1	0	0
6	0	0	1
7	1	0	0
7	0	0	1
8	1	0	0
8	0	0	1
9	1	0	0
9	0	0	1
10	1	0	0
10	0	0	1
10	0	0	1

Tiempo simulac	p1	p2	p3
0	1	0	0
0	0	1	0
0	0	0	1
1	1	0	0
1	0	1	0
1	0	0	1
2	1	0	0
2	0	1	0
2	0	0	1
3	1	0	0
3	0	1	0
3	0	0	1
4	1	0	0
4	0	1	0
4	0	0	1
5	1	0	0
5	0	1	0
5	0	0	1
6	1	0	0
6	0	1	0
6	0	0	1
7	1	0	0
7	0	1	0
7	0	0	1
8	1	0	0
8	0	1	0
8	0	0	1
9	1	0	0
9	0	1	0
9	0	0	1
10	1	0	0
10	0	1	0
10	0	0	1
10	0	0	1

Figura 28: Tabla comparativa de la información guardada en la simulación del ejemplo según los dos algoritmos distintos: Figura 21 (izquierda) y Figura 24 (derecha).

Por lo tanto el algoritmo de la Fig. 24 es el más completo para casos de transiciones mixtas y de varios tipos de retardos. Como ejemplo se incluye el programa de este ejemplo según dicho algoritmo (figura 29).

```
close;
hold off;
t0=0;
tf=10;
paso=1;
Tim=[1];
Y=[1 0 0];
T=[0];
```

```
tiempo=0;
i=1;
tspan(1)=0;
while tiempo<=tf
   i=i+1;
   Y(i,:)=Y(i-1,:);
   tspan(i)=tiempo;
   seguir=0; %indica que no hay disparos si no se aumenta el tiempo
   if Y(i,1)>=1
      Y(i,1)=Y(i,1)-1; Y(i,2)=Y(i,2)+1;seguir=1;
   elseif Y(i,2)>=1
      Y(i,2)=Y(i,2)-1; Y(i,3)=Y(i,3)+1;seguir=1;
   elseif T(1)>=Tim(1)& Y(i,3)>=1
      Y(i,3)=Y(i,3)-1; Y(i,1)=Y(i,1)+1;disparo(1)=i;seguir=1;
   end %del if
   if seguir==0
      tiempo=tiempo+paso;
      i=i-1; %elimino la última fila, que no ha cambiado
   end
   if Y(i,3)>=1
      if seguir==0 & T(1)<Tim(1)
         T(1)=min([T(1)+paso Tim(1)]);
      end
   else
   T(1)=0;
   end %del if
end %del while
subplot(3,1,1), plot(tspan,Y(:,1),'r');
subplot(3,1,2), plot(tspan,Y(:,2),'b');
subplot(3,1,3), plot(tspan,Y(:,3),'g');
```

Figura 29: Programa del ejemplo, según el algoritmo de la Figura 24.

4.6. Salidas

Una vez que hemos visto todas las posibles formas de simular la RdP que modeliza nuestro sistema, en función del tipo de red de que se trate, no debemos olvidar las salidas del sistema. Las salidas están asociadas a los lugares. En realidad también pueden asociarse al disparo de las transiciones, pero no es un caso habitual en sistemas para la industria. En realidad también pueden asociarse a combinaciones o a funciones de los lugares, es decir, a estados internos del sistema (puesto que el estado interno viene definido por el marcado global).

Pero además hay que tener en cuenta que esas salidas suelen ser a su vez entradas del sistema, es decir, eventos asociados a las transiciones. De esta manera, empleando salidas del sistema como entradas en otras transiciones conseguimos interrelacionar las redes entre sí.

Si las salidas fuesen siempre combinación lineal de los estados internos, dado que el estado lo tenemos en forma de matriz, o más bien vector (la última fila de Y), podríamos conseguir el vector de salida S como una simple operación matricial,

$$S=A*Y \qquad (1)$$

donde A es la matriz que define las salidas en función del estado interno.

Pero al igual que ocurría con la evolución de las transiciones, que en la práctica era muy complicado de implementar por métodos matriciales, con las salidas ocurre lo mismo, debido a varias razones:

- en muchas ocasiones se trata de funciones lógicas del estado, en vez de combinaciones lineales
- es muy frecuente encontrar salidas condicionadas, e igualmente las condiciones pueden ser funciones no lineales
- La matriz A es una matriz que ocupa gran cantidad de información, pues tiene tantos lugares como el producto de los lugares de la red por las salidas; de todos esos valores la mayoría suelen ser cero (pues muchas salidas coinciden con 1 ó dos lugares), y aunque Matlab trata eficientemente este tipo de matrices (matrices dispersas) otros tipos de lenguajes no lo hacen, y por lo tanto resulta poco rentable
- muchas veces interesa como salida el flanco del lugar (paso de activo a desactivo -flanco descendente-, o viceversa -flanco ascendente-), que tampoco se adapta bien al tratamiento matricial.
- el método matricial suele ser útil para salidas tipo *out*, pero no para las tipo *set-reset* que se dan a menudo.

Por todo esto se suele implementar la activación de las salidas en función del estado interno de una forma similar a como se hacía con la evaluación del disparo de las transiciones, de una en una.

4.6.1. Salidas tipo Out y Set-Reset

Las salidas tipo OUT, en la que la salida constituya un test lógico de marcado mayor que cero, que son las más habituales, se implementan asignando en cada iteración un valor 1 (true) a la salida si se da dicha condición, y 0 (false) en caso contrario. Por ejemplo:

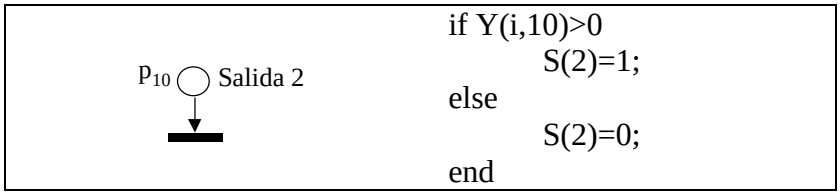


Figura 30: Salida tipo Out

El tipo visto es con diferencia el más frecuente, pero si la condición es de otro tipo, o la salida no es booleana se procede exactamente igual.

También son frecuentes las salidas tipo set o reset, en las cuales la salida se activa o desactiva si se cumple la condición pero no se hace nada si no es así. Por ejemplo:

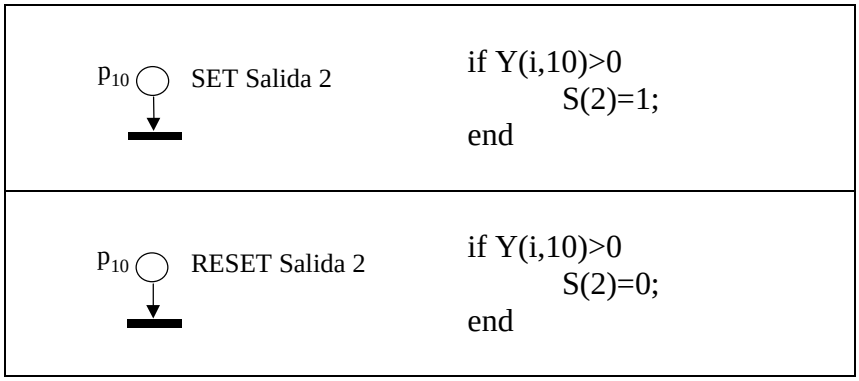


Figura 31: Salidas tipo Set-Reset

Podemos fijarnos que el tratamiento de actualización de los temporizadores que realizábamos en los algoritmos previos (Figura 21) consiste en una salida tipo Out sobre ellos en la cual en vez de poner a 1 cuando se da la condición, se incrementan en una unidad.

4.6.2. Salidas condicionadas

Las salidas condicionadas no aportan nada nuevo ni conllevan ninguna diferencia. Tan sólo hay que incluir en la condición para el tratamiento de la salida que se cumpla la condición incluida en el lugar. Se pueden emplear salidas condicionadas tanto en salidas tipo Out como Set-Reset o incluso en las temporizaciones. Por ejemplo:

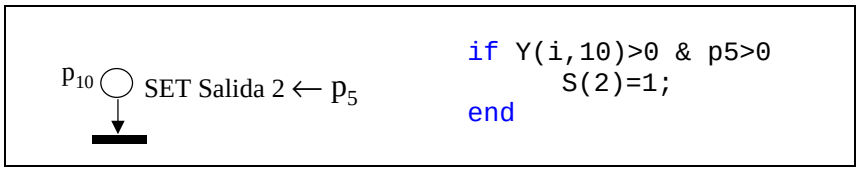


Figura 32: Salida condicionada

Y en el caso de condición en los temporizadores, se procede exactamente igual, en la zona del programa de actualización de los temporizadores.

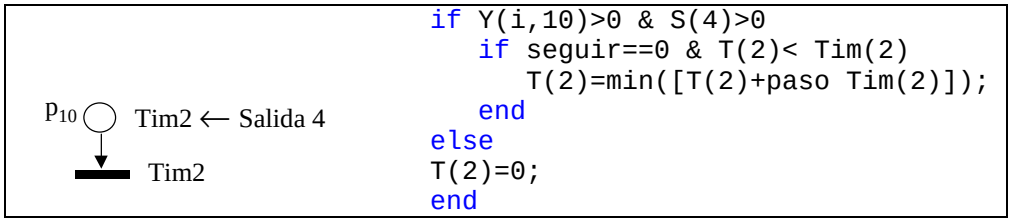
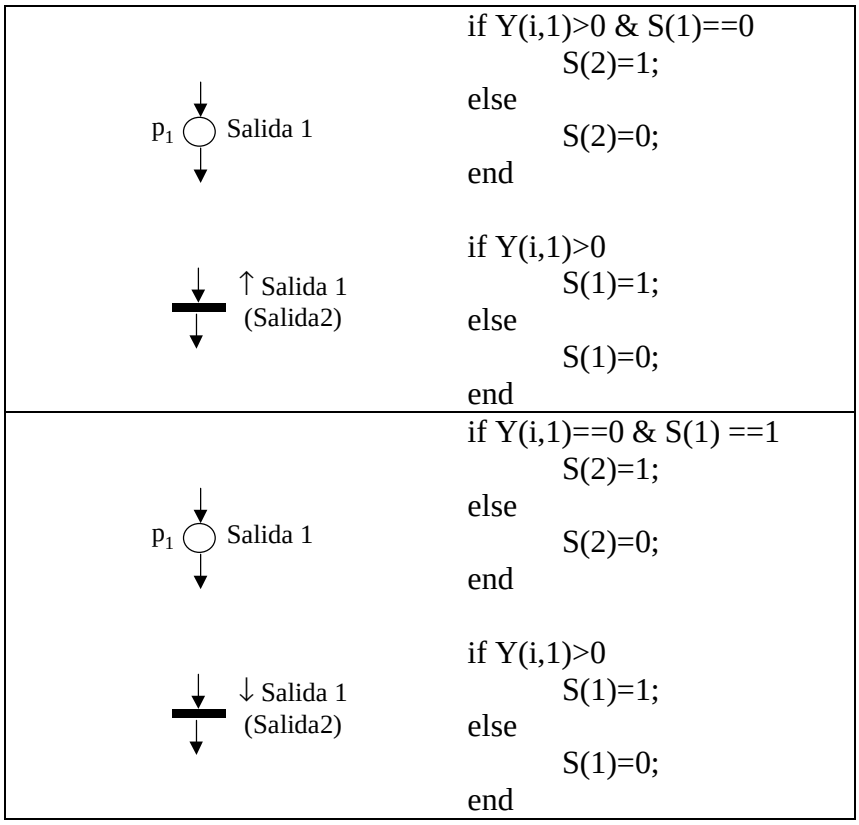


Figura 33: Temporización condicionada

4.6.3. Flancos

Otro tipo de señal muy empleada es la de los flancos, de subida o de bajada, de una señal. Un flanco es simplemente la transición de un valor (0 ó 1) al otro. Cuando de una salida lo que se vaya a emplear es uno de sus flancos para activar una transición, la forma de conseguir ese flanco es idéntica a como la realiza un autómata programable: se trabaja con dos señales distintas, la señal en cuestión y otra que es el flanco deseado. Y el tratamiento que se les da a las dos en la actualización de las salidas es como se muestra en el ejemplo de la figura siguiente: la señal se trata con normalidad, y antes que ella se actualiza su flanco, simplemente testeando el valor que le corresponde a la actualización de la señal y el que conserva aún (que es el de la iteración anterior).




```
if Y(i,1)>0 & S(1)==0
        S(2)=1;
else
        S(2)=0;
end

if Y(i,1)>0
        S(1)=1;
else
        S(1)=0;
end
```

```
if Y(i,1)==0 & S(1) ==1
        S(2)=1;
else
        S(2)=0;
end

if Y(i,1)>0
        S(1)=1;
else
        S(1)=0;
end
```

Figura 34: Flanco ascendente y flanco descendente

4.7. Throughput

Uno de los datos más interesantes a la hora de simular una RdP que modeliza un sistema industrial es el throughput, que es el término en inglés con el que se denomina producción o rendimiento. Se ha considerado oportuno mantener el término por dar idea de que consiste en conocer la cantidad que pasa a través de una transición, las marcas que evolucionan hasta los lugares de incidencia posterior. Conocida además la evolución de dicho dato se puede conocer la frecuencia de producción.

En realidad para conocer el throughput bastaría con incluir en la red un lugar que vaya integrando el paso de material que se produce a través de la transición. Por ejemplo en

el sistema de producción de la Figura 8-a se puede incluir el lugar que se muestra en la figura 35, y que no es otra cosa que la producción total (o la integración de los disparos de la transición t6). Sin embargo ese tipo de lugares, que por su carácter de integradores están no acotados en general, suele dar problemas en muchos simuladores.

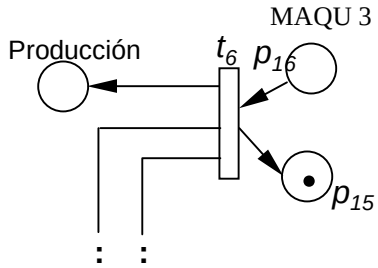


Figura 35: Lugar que integra la producción.

Una forma sencilla de conocer tanto la producción total en una transición, como la frecuencia (o el periodo), puede consistir simplemente en emplear un contador del número de disparos de la transición en cada instante, o bien en emplear un vector que indique en qué instantes se produce un elemento nuevo.

La primera de dichas opciones se emplea para casos más simples, y se puede realizar incluyendo dentro de las acciones que derivan del disparo de la transición esta nueva:

disparo(i)=1;.

Con ello luego se puede plotear el vector booleano que nos indica en qué instantes (pasos) ha habido disparo: *plot(disparo);* .

En la segunda opción, empleada para casos más complejos, se debe incluir en el código del programa las instrucciones que se muestran a continuación. Además todo ello ya está incluido en código del ejemplo previo de producción correspondiente a la Figura 8-b.

- Antes de las iteraciones: **produc=0;**
- Dentro de las iteraciones, en las acciones correspondientes al disparo de la transición:

```
produc=produc+1;instante(produc)=i;
```

- Despues de las iteraciones:

```
for k=1:produc-1
   periodo(k)=(instante(k+1)-instante(k))/(1/paso);
     %periodo en segundos de salida de pieza
   frecuencia(k)=(1/paso)/(instante(k+1)-instante(k));
     %frecuencia en segundos de salida de pieza
end
if produc>1
   disp('El flujo en régimen permanente es');frecuencia
   disp('i.e., sale una pieza cada estos segundos');periodo
end
```

Figura 36: Implementación del Throughput de una transición, y obtención del periodo y la frecuencia.

Hay que tener en cuenta que pueden darse varios disparos en un mismo instante de tiempo (redes mixtas) por lo cual la frecuencia puede tener valor infinito. En esos casos basta con trabajar sólo con el periodo (de no ser así el problema lógicamente dará error).

4.8. Variaciones en los parámetros temporales

4.8.1. Horizonte temporal variable

En muchas ocasiones, y sobre todo durante las primeras simulaciones, no se sabe el tiempo que tardará en estabilizarse la respuesta del modelo que se está simulando, para llegar al régimen estacionario. En esos casos puede ser interesante incluir el programa en un bucle que se repita hasta que se le indique lo contrario, y en el que el tiempo (o los pasos) de simulación se vayan incrementando en vez de empezar desde cero de nuevo. Esto puede ser muy interesante para ahorrar tiempo hasta que determinemos cual es el rango de tiempo en el que se va a trabajar.

Si no se ha realizado lo anterior, y se quiere seguir simulando desde donde se había dejado, se puede realizar sustituyendo los valores iniciales por los de la última fila de la simulación recién realizada. Es lo mismo pero realizado por el usuario, en vez de programado en el listado del programa.

Por ejemplo, el sistema de producción de la Figura 8-a, cuyo código se representaba en la Figura 8-b, se puede modificar como se indica a continuación para realizar el horizonte temporal variable

```
      INICIALIZACIÓN IGUAL QUE EN FIG 8-b
repetir='si';
iteraciones=1;

while repetir~='no'
tiempo=input('¿Cuántos segundos quieres evaluar?');
tf=tf+tiempo
tspan=t0:(tf/paso)-1;
for i=iteraciones+1:(tf/paso)  %i es el contador del tiempo
      IGUAL QUE EN FIG 8-b
end
plot(tspan/10,Y(:,16),'r'); %ploteo el lugar 16 en función del tiempo
repetir=input('¿Quieres seguir simulando? ','s');
iteraciones=tf/paso;
end %del while
```

Figura 37-a: Simulación continuada en el tiempo del ejemplo de producción de la Figura 8-a.

4.8.2. Simulación continuada en el tiempo

Sin embargo, muchas veces es otro el problema que nos lleva a tener que modificar los parámetros temporales y a tener que hacerlo de diferente manera. Un sistema que

modelice un proceso industrial real, aun huyendo de precisiones innecesarias, se sitúa en los cientos de lugares, o incluso miles. Si en el sistema hay algún retardo de tiempo relativamente muy inferior al tiempo total que queremos simular, nos encontramos con que no es posible realizar la simulación por problema del tamaño de la matriz Y. Ello es especialmente problemático si empleamos el algoritmo de las Figuras 21 ó 24, en los que se guardaba toda la información disponible, pero que emplea toda una fila de Y (cientos o miles de valores) para cada disparo de la transición.

En esos casos, que en la práctica de simulación de plantas industriales completas se dan muy frecuentemente, lo que se hace es llegar hasta el horizonte de tiempo deseado a base de realizar simulaciones de tiempo más breves. Cuando se alcanza este tiempo, que permite manejar Y aceptables, se almacena el estado del sistema y el tiempo de simulación, y se procede a realizar la siguiente sub-simulación pero sobreescribiendo los valores anteriores. De esa manera el sistema trabajará con matrices pequeñas que se irán empleando y desechando, y con paciencia se puede llegar a cualquier horizonte temporal de simulación sin posibilidad de desbordar el sistema informático.

```
      INICIALIZACIÓN
repetir='si';
iteraciones=1;
t0=0;tf=0;
definir paso, Y inicial y trans;
repetir='si';
iteraciones=1;
while repetir~='no'
   tiempo=input('¿Cuántos segundos quieres evaluar?');
   tf=tf+tiempo
   tspan=t0/paso:(tf/paso)-1;
      for i=2:(tiempo/paso)
         Y(i,:)=Y(i-1,:);
         ITERACIONES DE TRANSICIONES, TEMPORIZADORES Y SALIDAS
      end
   %sacar gráficas del tramo de evaluación realizado
   disp('El último valor encontrado de Y es');Yf
   repetir=input('¿Quieres seguir simulando? ','s');
   clear Y;Y=Yf;t0=tf;
end %del while
```

Figura 37-b: Modificación del programa 8-b para ampliar el horizonte de simulación.

En el capítulo correspondiente a la aplicación práctica industrial veremos que empleando Matlab como sistema de simulación se puede obtener la gráfica correspondiente a cada tramo simulado, pero también se puede ir realizando las gráficas a partir de las anteriores, de tal forma que aunque en cada nueva sub-simulación se pierden los datos de la anterior, la gráfica representa el proceso de simulación completo. Evidentemente en este último caso (de gráfica del proceso total) sí se puede correr el riesgo de desbordamiento del sistema en horizontes muy grandes (si bien las gráficas optimizan los valores a medida que los van recibiendo, y varios valores consecutivos iguales ocupan menos información en el sistema que en la matriz Y).

Sin embargo hay que tener cuidado con el método de iteración que se está utilizando para la simulación cuando se emplee el horizonte temporal variable. Si se está analizando redes temporizadas, y el método por el que se comprueba si ha transcurrido el tiempo necesario es analizando las últimas filas de la matriz Y, (como es el caso del ejemplo anterior) al comenzar una nueva iteración se está reinicializando automáticamente los temporizadores. Ello no ocurre en simulaciones de sistemas sin temporizar (Figura 2), o en simulaciones con temporización en las que se almacena el tiempo transcurrido en un vector (Figuras 21 y 24). Este problema puede evitarse iniciando cada nuevo periodo temporal con una nueva matriz Y con las filas suficientes para abarcar el tiempo del mayor retraso, pero aun así es más recomendable emplear un algoritmo de simulación en el que ello no sea necesario, tal como se hará en la aplicación industrial que se muestra en capítulos posteriores.

4.9. Sistemas continuos y continuizados

Pese a la utilidad demostrada por la aplicación Matlab en las simulaciones discretas desarrolladas hasta el momento, su gran utilidad reside en su manejo como herramienta de simulación de variables continuas, gobernadas por ecuaciones diferenciales.

En muchas ocasiones el sistema a automatizar no es discreto, sino que tiene partes continuas y partes discretas. Especialmente se da el caso de necesitar variables continuas en la simulación del proceso, más que en su automatización. Por ejemplo supongamos que queremos controlar el nivel de un depósito en el que constantemente está entrando agua, abriendo o cerrando una válvula que funciona a modo de sobradero. La automatización consiste tan sólo en abrir o cerrar (en función del nivel), es decir, es discreta, pero para simular la altura del depósito debemos trabajar sobre una variable continua.

Sin embargo el uso de las variables continuas (o híbridas) en automatizaciones industriales tiene un campo, ahora mismo en expansión, que es el de las redes continuizadas. Continuizar las redes discretas consiste en transformar la red discreta en una red continua que la modelice, para determinar prestaciones de esa la red discreta (y por lo tanto del sistema) trabajando de forma continua y con variables continuas.

Cuando la red discreta es muy grande y soporta grandes marcados (gran número de marcas), ocurre la explosión del sistema como discreto, que complica su manejo. Además debido al gran número de marcas en la continuización de una variable discreta en otra continua se comete un error relativo pequeño.

Por supuesto no siempre puede realizarse esa transformación, y además puede realizarse de varias formas distintas (finitos servidores, infinitos servidores, etc.). Además la interpretación de los resultados del modelo continuo y su exportación al discreto es complicada y está aún en fase de estudio y formalización. En aplicaciones industriales reales aún no se han implantado estas técnicas, pero sin duda se pueden obtener resultados muy buenos si se emplean, por supuesto teniendo en cuenta qué es lo que se está haciendo.

Con este tipo de modelos continuizados, la simulación en cualquier lenguaje, y especialmente en Matlab (por la comentada facilidad en el manejo de ecuaciones diferenciales), permite obtener simulaciones de sistemas más complejos y con mayor horizonte de simulación empleando menor esfuerzo computacional. En el ejemplo que veremos a continuación lo comprobaremos de forma cuantitativa.

Dado ese bajo coste computacional de este tipo de simulaciones continuas, pueden ser muy interesantes para hacer una primera evaluación en el ajuste de los parámetros del sistema, cuando hay muchos parámetros (muchas direcciones sobre las que evolucionar), y por tanto el espacio multidimensional sobre el que hay que buscar la solución óptima es grande y complejo (ver apartado siguiente). Una vez que se han encontrado los valores ideales de los parámetros y se han determinado las regiones de actuación más propicias, se puede comprobar el correcto funcionamiento con la simulación continua. Lo que es evidente es que para obtener resultados más precisos y fiables se requiere mayor esfuerzo de computación.

4.9.1. Ejemplo

Como ejemplo de sistema continuizado veremos el ejemplo tratado en la sección 4.3.3.1 (Figura 8-a). Como ya hemos comentado el proceso de continuización es complejo y requiere de grandes fundamentos. La continuización de dicho sistema puede llevar a un sistema igual al de la Figura 8-a pero considerando los lugares y transiciones como continuos, donde las velocidades a las que pasan las marcas por las transiciones son proporcionales al mínimo de los marcados incidentes (lo que se denomina infinitos servidores o velocidad variable), o bien puede llevar a un sistema como el representado en la Figura 38, en el que las velocidades a las que pasan las marcas por las transiciones están acotadas por un valor constante (lo que se denomina finitos servidores o velocidad constante).

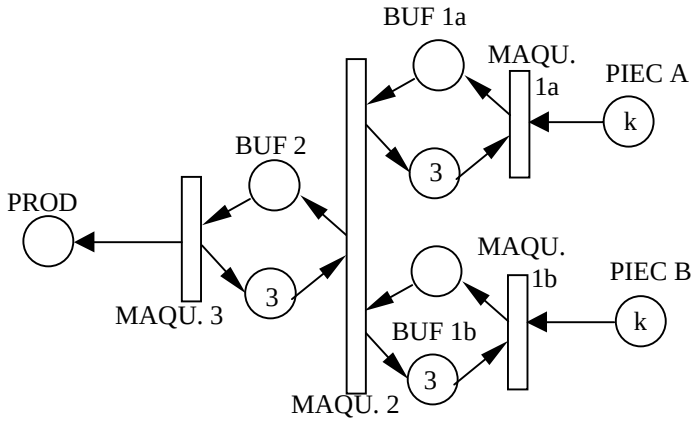


Figura 38: Modelo continuo correspondiente al problema de la Figura 8-a

No se va a entrar en cómo se ha llegado a esos modelos, sino en cómo poder realizar su simulación de una forma sencilla, para posteriormente poder aplicarlo a sistemas complejos, como se hará en próximos capítulos. Para el manejo de las ecuaciones diferenciales Matlab emplea un método Runge-Kutta (en cualquier otro lenguaje de

programación se puede programar fácilmente), para el que hay que indicarle en un fichero aparte cuáles son las ecuaciones diferenciales que puede resolver. Por eso cada simulación presenta dos ficheros: el del programa de simulación y el de las ecuaciones diferenciales.

La simulación del primer sistema se muestra en las gráficas 39-a y 39-b, y la del segundo (el de la Figura 38) se muestra en las 41-a y 41-b. No es necesario darle al programa el paso para el método Runge-Kutta (en este caso de orden 2/3), pues él toma uno apropiado, pero se le puede indicar uno menor (como se ha hecho) para lograr mayor precisión si es que así se requiere.

La Figura 40 muestra la simulación de la evolución de los marcados del primer sistema continuo presentado (infinitos servidores). La del segundo sistema (finitos servidores) no se muestra porque en este ejemplo va a ser constante el marcado de cada lugar, como puede preverse analizando la Figura 38 (teniendo en cuenta que las transiciones tienen una velocidad de paso de marcado constante e igual entre ellas).

```
t0=0;%el tiempo inicial para el runge-kutta
tf=input('¿Cuántos segundos quieres evaluar?');%el tiempo final para
el runge-kutta
paso=0.01; %cuanto menor paso más exacto el cálculo y la gráfica
k=input('¿Parámetro inicial del sistema?');
Y0=[k 1 0 3 0 k 1 0 3 0 1 0 3 0 1 0];
tspan=t0:paso:tf;
[t,Y]=ode23('ec_infinite',tspan,Y0);
plot(tspan,Y(:,1),tspan,Y(:,3),tspan,Y(:,5),tspan,Y(:,12),tspan,Y(:,14
),tspan,Y(:,16));
hold on; plot(tspan,Y(:,16)*5,'r')%plotea 10 * el throughput en rojo
```

Figura 39-a: Programa de simulación de un sistema continuo a velocidad variable.

```
function deriv=ec_infinite(t,Y)
lambdacarga=1;
lambdadescarga=5;
l11=lambdacarga;
l12=lambdacarga;
l21=lambdadescarga;
l22=lambdadescarga;
l3=lambdacarga;
l4=lambdadescarga;
l5=lambdacarga;
l6=lambdadescarga;
t11=l11*min(Y(1),Y(2));
t12=l12*min(Y(6),Y(7));
t21=l21*min(Y(3),Y(4));
t22=l22*min(Y(8),Y(9));
t3=l3*min(Y(5),min(Y(10),Y(11)));
t4=l4*min(Y(12),Y(13));
t5=l5*min(Y(14),Y(15));
t6=l6*Y(16);
deriv=zeros(16,1);
deriv(1)=t6-t11;
deriv(2)=t21-t11;
deriv(3)=t11-t21;
deriv(4)=t3-t21;
deriv(5)=t21-t3;
deriv(6)=t6-t12;
deriv(7)=t22-t12;
deriv(8)=t12-t22;
deriv(9)=t3-t22;
deriv(10)=t22-t3;
deriv(11)=t4-t3;
deriv(12)=t3-t4;
deriv(13)=t5-t4;
deriv(14)=t4-t5;
deriv(15)=t6-t5;
deriv(16)=t5-t6;
```

Figura 39-b: Programa ec_infinite, que describe las ecuaciones diferenciales del sistema de la Figura anterior.

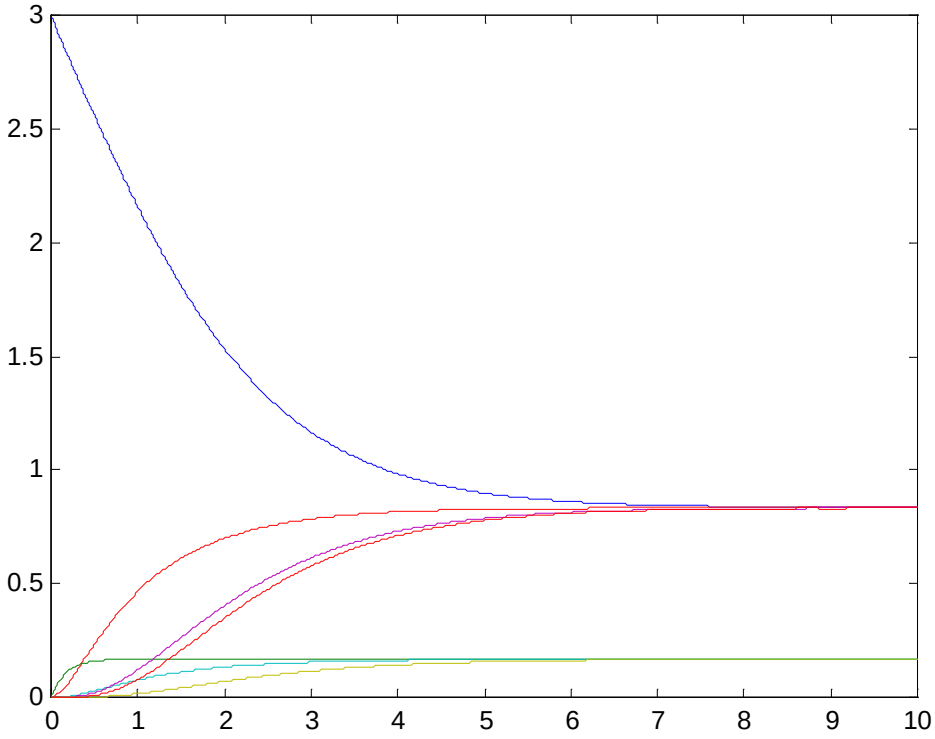


Figura 40: Simulación de la evolución del marcado de los lugares de un sistema continuo a velocidad variable.

```
t0=0;%el tiempo inicial para el runge-kutta
tf=input('¿Cuántos segundos quieres evaluar?');%el tiempo final para
el runge-kutta
paso=0.01; %cuanto menor paso más exacto el cálculo y la gráfica
k=input('¿Parámetro inicial del sistema?');
Y0=[k 3 0 k 3 0 3 0];
tspan=t0:paso:tf;
[t,Y]=ode23('ec_finite',tspan,Y0);
plot(tspan,Y(:,1),tspan,Y(:,3),tspan,Y(:,8));
hold on; plot(tspan,Y(:,8)*(1/1.2),'r')%plotea el throughput en rojo
```

Figura 41-a: Programa de simulación de un sistema continuo a velocidad constante.

```
function deriv=ec_finite(t,Y)
lambda=1/1.2;
l11=lambda;
l12=lambda;
l2=lambda;
l3=lambda;
if min(Y(1),Y(2))>0
   t11=l11;
else
   t11=0;
end
if min(Y(4),Y(5))>0
   t12=l12;
if min(min(Y(3),Y(6)),Y(7))>0
   t2=l2;
else
   t2=0;
end;
if Y(8)>0 t3=l3;
else
   t3=0;
end;
deriv=zeros(8,1);
deriv(1)=t3-t11;
deriv(2)=t2-t11;
deriv(3)=t11-t2;
```

else
  t12=0;
end
	deriv(4)=t3-t12;
	deriv(5)=t2-t12;
	deriv(6)=t12-t2;
	deriv(7)=t3-t2;
	deriv(8)=t2-t3;

Figura 41-b: Programa ec_finite, que describe las ecuaciones diferenciales del sistema de la Figura anterior.

La simulación del sistema discreto, eliminando todas las instrucciones superfluas y prescindibles, en un computador medio para un tiempo de 100 segundos con paso de 0.1 cuesta 4.17 segundos. La misma simulación con las mismas características en el sistema continuo cuesta tan sólo 1.65 segundos. Si se amplía el horizonte de simulación hasta los 1000 segundos, el discreto tarda 412.55 segundos, por los 18.67 del continuo.

4.10. Parametrización de los sistemas. Métodos de búsqueda para optimización del proceso.

Como ya hemos adelantado anteriormente, una de las ventajas de la simulación en el computado del proceso productivo, tanto de la automatización como de la planta, consiste en poder comprobar el efecto que tendrá la variación de los parámetros para conseguir mejores resultados.

Pongamos como ejemplo el sistema productivo que hemos tratado en todo el capítulo, y que se encuentra representado en la figura 8-a. Deliberadamente se puso el parámetro k como el máximo número de piezas de cada tipo (o de contenedores para las piezas) que tiene el sistema. En las simulaciones se ha empleado con valor igual a 3, pero se dejó indicado así para analizar el efecto de modificar el número de contenedores en el sistema. Si k vale 1, la frecuencia de producción es de 0.278 s^{-1}, con k igual a 2 la frecuencia sube a 0.555 s^{-1}, y para valores de k iguales o superiores a 3 la frecuencia es de 0.833 s^{-1}. Por lo tanto vemos que no se obtendría beneficio de aumentar el número de recipientes, si no se aumenta también la capacidad de los buffers.

Ese análisis de la producción según el valor del parámetro k se puede programar fácilmente en la simulación por computador para obtener la curva correspondiente. Si tenemos dos parámetros (por ejemplo número de bandejas y tiempo de la transición 2) tendremos que movernos a lo largo de una superficie para determinar el punto o la zona que optimiza la frecuencia de salida. Y en general lo que tendremos es n parámetros y por tanto un espacio n-dimensional.

Además, se puede incluir en la programación aspectos como el coste de los parámetros, y por lo tanto, no habrá que intentar optimizar la producción a cualquier precio sino una cierta relación producción-coste que se habrá indicado mediante una función objetivo. En este aspecto puede resultar muy interesante emplear técnicas de inteligencia artificial para maximizar esa función objetivo mediante diversos métodos de búsqueda. En el capítulo 5 se muestran diversos algoritmos de búsqueda de soluciones a un problema mediante inteligencia artificial, y se aplican para resolver un problema concreto. Esos métodos podrían igualmente incluirse en este apartado. También resulta interesante en muchas ocasiones para optimizar el sistema el empleo de métodos de programación

lineal y de programación no lineal, si bien a veces el empleo de los sistemas discretos y continuos con parámetros se vale de estos métodos de forma implícita.

4.11. Conclusiones

La aportación de este capítolo dentro de la investigación de la tesis consiste en el desarrollo de la metodología apropiada para la simulación de cualquier sistema que hayamos modelado mediante RdP (por tanto también mediante GRAFCET) en un lenguaje de programación. Además, dada la difusión, sobre todo en el ámbito universitario, de la aplicación informática Matlab, así como su potencia y simplicidad, se particulariza dicha metodología para dicho lenguaje de programación. La adaptación a cualquier lenguaje de alto nivel de propósito general es bastante sencilla. En el análisi se tiene además muy en cuenta los puntos que posteriormente serán interesantes en las aplicaciones industriales reales, como son el throughput, el tipo de salida, los tipos de retardos, la variación de parámetros temporales, etc.

Toda esta aportación se empleará además posteriormente en la simulación de una planta industrial real.

4.12. Referencias

[1] M. Ajmone Marsan, editor. Application and Theory of Petri Nets 1993, volume 691 of Lecture Notes in Computer Science. Springer, 1993.

[2] M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. ACM Trans. on Computer Systems, 2 (2) : 93-122, 1984.

[3] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. Modelling with Generalized Stochastic Petri Nets. Wiley, 1995.

[4] H. Alla and R. David. Continuous and hybrid Petri nets. Journal of Circuits, Systems, and Computers, 8 (1): 159-188, 1998.

[5] J. Aracil. Introduction à la Dynamic des Systèmes. Presses Universitaires de Lyon, 1984.

[6] P. Brandimarte and A. Villa. Advanced Models for Manufacturing Systems Management. Mathematical modelling series. CRC Press, 1995.

[7] G. Ciardo, C. Nicol, and K. S. Trivedi. Discrete-event simulation of fluid stochastic petri nets. In Procs. Of the 8rd Int. Workshop on Petri Nets and Performance Models (PNPM'97). IEEE-Computer Society Press, 1997.

[8] R. David and H. Alla. Continuous Petri nets. In Proc. Of the 8th European Workshop on Application and Theory of Petri Nets, pages 275-294, Zaragoza, Spain, 1987.

[9] R. David and H. Alla. Autonomous and timed continuous Petri nets. In G. Rozenberg, editor, Advances in Petri Nets 1993, volume 674 of Lecture Notes in Computer Science, pages 71-90. Springes, 1993.

[10] S. B. Gershwin. Manufacturing Systems Engineering. Prentice-Hall, 1994.

[11] L. E. Holloway, B. H. Krogh, and A. Giua. A survey of peri net methods for controlled discrete event systems. Journal of Discrete Event Dynamic Systems, 7: 151-190, 1997.

Tema 5

IMPLEMENTACIÓN DE TÉCNICAS AVANZADAS EN LA AUTOMATIZACIÓN

5.1. Introducción

En este capítulo se analiza la utilización de técnicas más avanzadas que se integran dentro de las RdP empleadas en las automatizaciones. En concreto la investigación se enfoca sobre dos puntos: la automatización mediante RdP adaptativas, para procesos de producción altamente cambiantes (apartado 5.2), y el empleo de técnicas de inteligencia artificial, más concretamente de técnicas de búsqueda, en la automatización del proceso (apartado 5.3).

En este segundo apartado se pretende más abrir el abanico de posibilidades que esas técnicas de búsqueda ofrecen para automatizaciones on-line que presentar unos resultados concretos. Por ello se presenta una aplicación diseñada para resolver un pequeño problema en un almacén, partiendo de una base teórica que se presenta al principio de la correspondiente sección. Esa base teórica no se ha incluido en el capítulo 1, dedicado precisamente a tal tipo de contenidos, puesto que se aplica tan sólo en este capítulo, pero hay que tener en cuenta que los apartados del 5.3.1 al 5.3.4 dan una visión de las técnicas de búsqueda, y que por tanto los expertos en el tema pueden obviarlos y pasar directamente a la aplicación.

Respecto a estas técnicas, tanto las correspondientes a automatizaciones adaptativas como las de inteligencia artificial, cabría destacar que se prestan muy a menudo a ser usadas sobre sistemas robóticos. La razón es que se trata de sistemas habitualmente complejos y por esa razón suelen requerir de aplicaciones especiales.

5.2. Automatización adaptativa en procesos de producción flexibles

5.2.1. Introducción

Los robots son ampliamente usados en los sistemas de producción. Algunos de esos sistemas corresponden a procesos flexibles, y entonces la automatización también debe ser tan flexible como sea posible. Cuando hay muchos robots en un proceso industrial la automatización puede ser compleja, y es mucho más complejo si el proceso es flexible (si los robots pueden agregarse o pueden quitarse, o si sus comportamientos pueden modificarse)[4]. En esta sección veremos cómo se han desarrollado algunas aplicaciones robóticas en los procesos flexibles de una manera simple pero eficaz para procesos industriales automatizados.

Hemos visto en capítulos anteriores que las RdP son una herramienta poderosa para tratar procesos industriales en todas sus fases (modelado, control, automatización,

simulación, supervisión, etc.), y también cómo pueden traducirse directamente al lenguaje de los PLCs. En la aplicación que se propone, se emplea un programa de ordenador para generar automáticamente, a partir de una tabla de datos, la RdP de la automatización, que depende de los elementos del sistema, sus posibles estados, los procesos, y los requisitos. Esto hace posible que cualquiera familiarizado con el proceso pueda generar la RdP pertinente, incluso siendo profano en temas de robótica o automatización. En este caso el tamaño o el grado de sofisticación del sistema es irrelevante.

Por otra parte, la generación automática de la RdP para el proceso presenta muchas otras ventajas igualmente interesantes [5, 6]. Programar la automatización in situ es normalmente complicado, en parte porque los sistemas de control de automatización más ampliamente usados, los autómatas programables, generalmente no tienen un lenguaje de alto nivel adaptado a los procesos secuenciales y concurrentes. Pero la traducción de una RdP a lenguaje de programación para PLC es un proceso que está resuelto según la metodología presentada en el capítulo 3. Por consiguiente, es posible generar, en tiempo real y de una manera simple, el programa de control para el proceso con una aplicación de computador. Esta aplicación genera la RdP desde las especificaciones, y el programa del PLC desde la RdP, según se aprecia en la Figura 1.

Figura 1: Generación del Programa.

También es posible generar el programa del PLC directamente a partir de la descripción del proceso, sin llevar a cabo el paso intermedio de la RdP, pero no es muy aconsejable, puesto que se perderían todas las ventajas que esta herramienta ofrece:

- Las RdP tienen una extensa teoría que las avala, y cuyos resultados están implementados en aplicaciones de computador que permiten detectar muchos de los errores que pueden ocurrir en el diseño. Así, pueden corregirse antes de transferir el programa al PLC y al proceso.
- Otras herramientas basadas en la teoría de RdP optimizan la red en sí misma. Así, se transfiere al PLC un programa equivalente pero más eficaz con un menor tiempo de respuesta. En cualquier caso, esto no es normalmente relevante en los procesos de producción, y la RdP original, equivalente a la perfeccionada, también puede usarse, porque suele ser más sencilla de entender.
- También tenemos la ventaja de los simuladores de RdP. Con ellas puede verificarse si el funcionamiento corresponde a las expectativas. Y esta verificación puede llevarse a cabo de una manera gráfica, con un entorno mucho más agradable y cómodo que el código mnemónico o el ladder [3].

No es muy importante ser capaz de desarrollar instantáneamente el programa de control, pero sí la posibilidad de modificarlo en tiempo real, puesto que las condiciones del proceso pueden cambiar en cualquier momento, para adaptarse a la operación más conveniente en un estado particular de trabajo. Es posible agregar o quitar nuevas

restricciones cuando se requiera, así como aumentar el número de elementos o sus estados en el proceso, etc. Estos cambios se modifican automáticamente en la RdP e incluso en el programa del proceso. Este problema es muy interesante en procesos que frecuentemente se ven modificados (por ejemplo, el empaquetado automático de tornillos o piezas). Más adelante se expondrá un ejemplo.

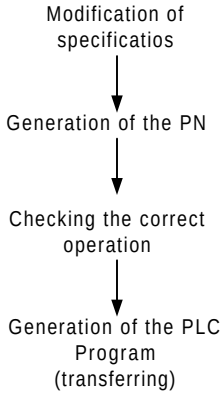


Figura 2: Modificación de las especificaciones

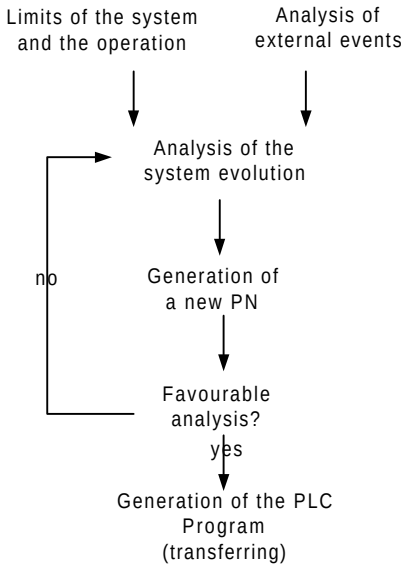


Figura 3: Evolución del sistema.

Pero es posible ir un paso más allá y permitir al propio sistema negociar y decidir estos cambios de funcionamiento en el proceso, dependiendo de la evolución de los eventos externos (las entradas del sistema) y los requisitos de las salidas. De esta manera, se realiza un control adaptativo, por el cual se modifica el funcionamiento (la asignación de recursos a los diferentes procesos). Esta modificación no ocurre escogiendo diversas opciones prefijadas, sino generando la más conveniente por medio de la evolución del programa de funcionamiento dentro de los límites establecidos. Éste es un tipo de optimización del proceso por medio de una evolución constante (muy similar a las técnicas de algoritmos genéticos). Lógicamente, esto tiene sentido en procesos que cambian frecuentemente, donde una optimización off-line no puede llevarse a cabo de una manera efectiva.

5.2.2. Generación y Modificación de la RdP

Las consideraciones anteriores pueden verse en un ejemplo simple en la Figura 4-a. Hay dos bandas transportadoras como entradas (S1, S2), otras dos como salidas (S3, S4), y dos robots (R1, R2) que pueden transportar piezas desde las bandas de entrada a las de salida. La RdP que controla el proceso es la de la Figura 5, con la interpretación mostrada en la tabla de la figura 4-b. R1 y R2 son los robots, S1... S4 son las bandas transportadoras, s1...s4 son los sensores que detectan piezas en sus correspondientes bandas y $\uparrow$ indica un flanco ascendente (el paso de no-detección a detección), y $\downarrow$ lo contrario.

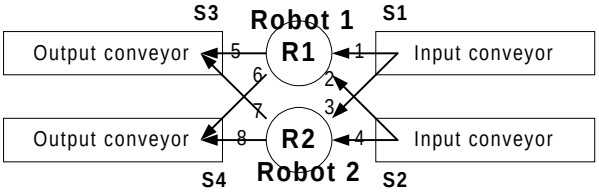


Figura 4-a: Proceso industrial simple con dos robots

t1	↑s1
t2	↑s2
t19	↓s3
t20	↓s4
p2	S1 (cargada)
p4	S2 (cargada)
p17	S3 (libre)
p18	S4 (libre)
p7	proceso1 (R1 coge de S1)
p8	proceso2 (R1 coge de S2)
p9	proceso3 (R2 coge de S1)
p10	proceso4 (R2 coge de S2)
p13	proceso5 (R1 deja en S3)
p14	proceso6 (R1 deja en S4)
p15	proceso7 (R2 deja en S3)
p16	proceso8 (R2 deja en S4)
p5	R1 libre
p6	R2 libre
p11	R1 cargada
p12	R2 cargada

Figura 4-b: Tabla de interpretación de la RdP de la Fig. 5, correspondiente al proceso de la Figura 4.

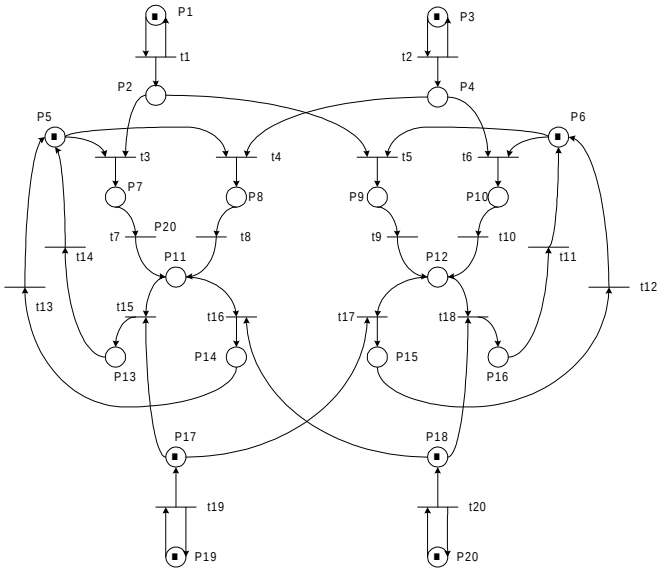


Figura 5: RdP del proceso de la Figura 4-a.

La red de la Figura 5 es muy simple, pero no tiene en cuenta ningún conflicto potencial entre los dos robots: cuando ambos van por el mismo lado o cuando se cruzan. Tampoco tiene en cuenta la prioridad de uno u otro proceso (parece lógico para cada robot tener un proceso prioritario, cuando pueden llevar a cabo varios). La introducción de estas condiciones (como se muestra en la tabla de la figura 6-b) lleva a una RdP más sofisticada, que se presenta en la Figura 6-a.

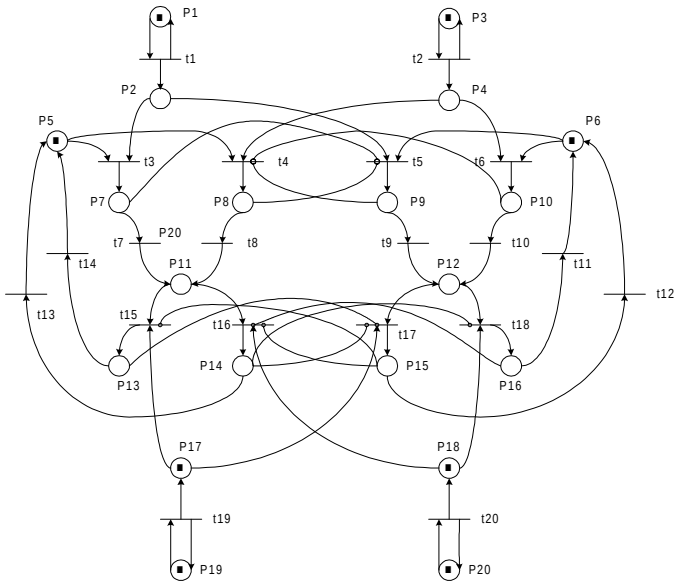


Figura 6-a: RdP con restricciones

RESTRICTIONS:	PRIORITIES:	
t4	$\overline{p9} \cdot \overline{p10}$	Priority(t3,t6) > Priority(t4,t5)
t5	$\overline{p7} \cdot \overline{p8}$	Priority (t15) > Priority (t16)
t15	$\overline{p15}$	Priority (t18) > Priority (t17)
t16	$\overline{p15} \cdot \overline{p16}$	
t17	$\overline{p13} \cdot \overline{p14}$	
t18	$\overline{p14}$	

Figura 6-b: Tabla de restricciones y prioridades

Supongamos que el proceso se modifica, por ejemplo introduciendo una nueva banda transportadora como entrada y permitiendo a cada robot tomar piezas de la banda del medio o de la banda más próxima a sí mismo. Entonces el resultado es el mostrado en la Figura 7. La RdP para este proceso es muy similar a la RdP que corresponde al proceso mostrado en la Figura 4-a, pero es diferente. Para obtener esta RdP desde la primera (Figura 6-a) se necesita entender la red total. Una pequeña modificación en cualquier parte de la RdP podría mejorar el comportamiento en esa parte pero estropear el proceso

global u otras partes. El resultado se muestra en la Figura 8. Las restricciones y las prioridades son las mismas que en el proceso precedente.

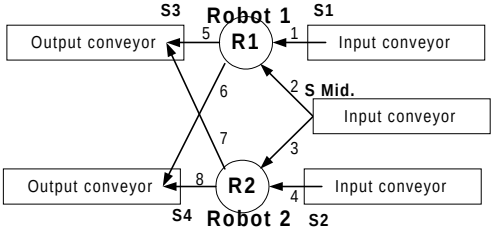


Figura 7: Modificaciones en el proceso de producción.

elements : *stable states*

processes
- *beginning*
  - *input states*
  - *output states*
  - *condition events*
  - *priority*
- *ending*
  - *input states*
  - *output states*
  - *condition events*
  - *priority*

initial conditions

Figura 7-b: Datos de descripción del proceso.

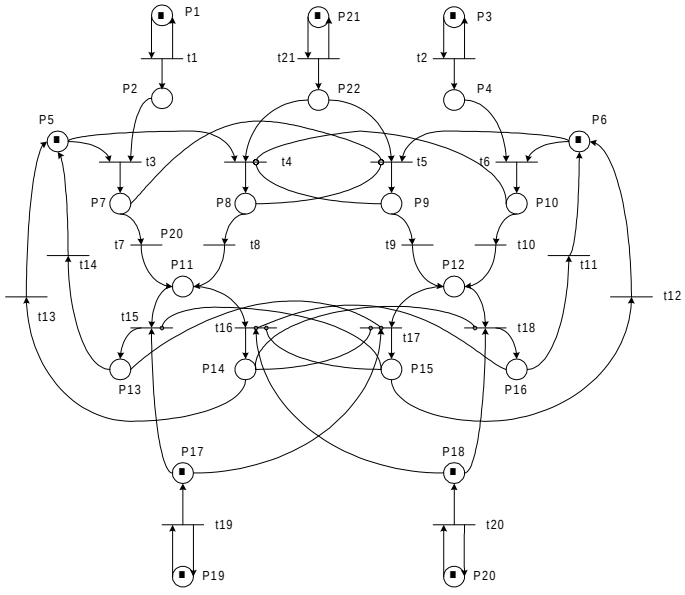


Figura 8: RdP del proceso modificado

Si algún nuevo robot se introduce ahora, el resultado es mucho más complejo, y podría ser necesario empezar el proceso entero de nuevo. Lo mismo sucede si se cambian prioridades o restricciones. Sin embargo, computacionalmente este no es un gran problema. Es necesario simplemente cambiar el mapa de definición del proceso. Este

mapa tiene que incluir los datos de la de la figura 7-b, y estos datos son simplemente los de la tabla de la figura 4-b, con un cierto orden, de modo que el programa pueda entenderlos y traducirlos en la RdP.

La tabla de la figura 8-b corresponde al proceso de la Figura 4-a, y genera la RdP de la Figura 6-a. Para conseguir la RdP correspondiente al proceso modificado (Figura 8), se necesitaba simplemente cambiar algunos parámetros en esta tabla.

ELEMENT	STABLE STATES
Input conveyor 1 (S1)	S1
Input conveyor 2 (S2)	S2
Output conveyor 1 (S3)	S3
Output conveyor 2 (S4)	S4
Robot 1	R1-loaded / R1-free
Robot 2	R2-loaded / R2-free

processes activ/disact	consumed states	produced states	conditions	prio-rity	comments
pr1 activat.	R1free.S1	-	-	1	R1 catch from S1
disact.	-	R1load	end_pr1	0	
pr2 activat.	R1free.S2	-	$\overline{pr3} \cdot \overline{pr4}$	0	R1 catch from S2
disact.	-	R1load	end_pr2	0	
pr3 activat.	R2free.S1	-	$\overline{pr1} \cdot \overline{pr2}$	0	R2 catch from S1
disact.	-	R2load	end_pr3	0	
pr4 activat.	R2free.S2	-	-	1	R2 catch from S2
disact.	-	R2load	end_pr4	0	
pr5 activat.	R1load.S3	-	$\overline{pr7}$	1	R1 leave in S1
disact.	-	R1free	end_pr5	0	
pr6 activat.	R1load.S4	-	$\overline{pr7} \cdot \overline{pr8}$	0	R1 leave in S2
disact.	-	R1free	end_pr6	0	
pr7 activat.	R2load.S3	-	$\overline{pr5} \cdot \overline{pr6}$	0	R2 leave in S1
disact.	-	R2free	end_pr7	0	
pr8 activat.	R2load.S4	-	$\overline{pr6}$	1	R2 leave in S2
disact.	-	R2free	end_pr8	0	
pr9 act	-	S1	$\uparrow s1$	0	come piece to S1
pr10 act	-	S2	$\uparrow s2$	0	come piece to S2
pr11 act	-	S3	$\downarrow s3$	0	leave piece from S3
pr12 act	-	S4	$\downarrow s4$	0	leave piece from S4

Figura 8-b: Tabla de datos para describir el proceso de la Figura 4-a

5.2.3. Modificación de prioridades y restricciones

Como se ha visto previamente, puede ser necesario modificar las prioridades o las restricciones en las transiciones de la red durante su ciclo de operación. Las restricciones intrínsecas dentro de una red son muy usuales cuando aparecen procesos incompatibles o combinaciones incompatibles de procesos, o en los casos de sincronismo o concurrencia en los procesos. Supongamos que un proceso asignado a p_1 debe ser activado solamente cuando una combinación lógica de otro lugar $f(p_1.. p_n)$ es cierta. Entonces la condición lógica f simplemente debería ser incluida como la condición de activación de la transición (o transiciones) previos a p_1 (*p1). Estos tipos de condiciones están muy bien adaptadas a la traducción al lenguaje del PLC (en lenguaje ladder). Pero no están tan bien adaptados a su traducción en nuevos elementos de la RdP (realizarlos a base de arcos, lugares y transiciones). Para hacer estas últimas traducciones, las restricciones intrínsecas deberían eliminarse (como funciones que dependen de los lugares p_i) y su información lógica debería convertirse en los nuevos elementos en la red que proporciona la misma información lógica. Según el tipo de condición lógica, esto puede ser más fácil o más complejo, como se muestra en las funciones de las Figuras 9 a 11, dónde las funciones lógicas de los lugares son asignadas a la transición *t*.

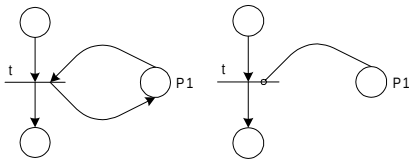


Figura 9: Restricciones lógicas en las transiciones: $p1$, $\overline{p1}$

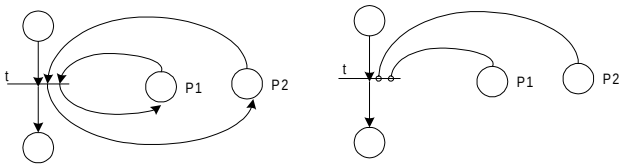


Figura 10: Restricciones lógicas en las transiciones: $p1 \cdot p2$, $\overline{p1} \cdot \overline{p2}$

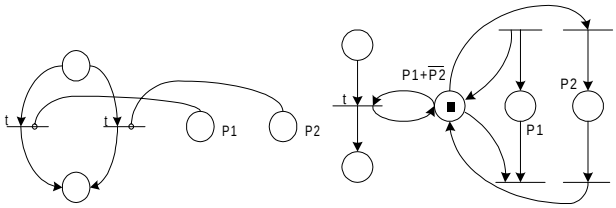


Figura 11: Restricciones lógicas en las transiciones: $\overline{p1} + \overline{p2}$, $p1 + \overline{p2}$

Esto es muy similar a lo que ocurre con los lugares redundantes, pero ahora las funciones son lógicas en vez de algebraicas. Incluso pueden existir varias maneras de llevarlo a cabo. Por ejemplo, para conseguir una función lógica p1+p2 es posible: a)

duplicar la red existente (en figura 12a), b) hacer la duplicación a parte (Figura 12b), c) no duplicar la red, usando las transiciones de entrada y salida de los elementos de la función lógica (Figura 12c).

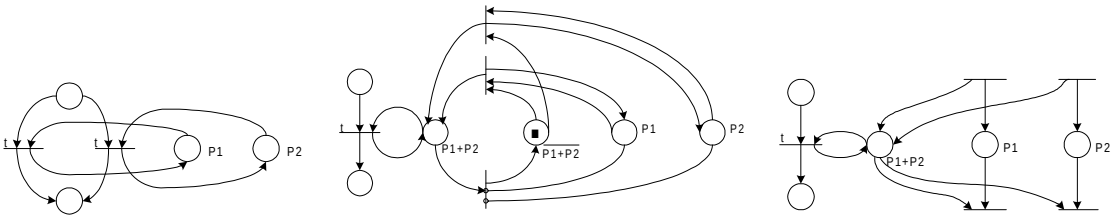


Figura 12 (a, b, y c): Restricción lógica p1+p2.

Por otro lado, las prioridades en las transiciones se adaptan mejor a los programas y simuladores con RdP que a la programación en PLCs. De cualquier modo, estas prioridades pueden ser traducidas a restricciones lógicas. En este caso, las funciones lógicas están compuestas por las transiciones con mayor prioridad (una función lógica "y" de todas ellas invertidas), y cada una de estas transiciones es otra función lógica de sus lugares de la entrada. Por ejemplo, en el proceso con la RdP de la Figura 6, poner *prioridad(t15)>prioridad(t16)* es lo mismo que añadir la condición lógica $\overline{t15} = \overline{p11 \cdot p17 \cdot \overline{p15}} = \overline{p11} + \overline{p17} + p15$ en t16. Además es posible reducir estas condiciones, como en este ejemplo dónde $\overline{p11}$ puede eliminarse de la red sin restricciones (Figura 5). En la red con restricciones (Figura 6) p15 también puede ser eliminado. Entonces la prioridad puede adquirirse añadiendo la condición $\overline{p17}$ en t16. También pueden introducirse prioridades con una frecuencia relativa (por ejemplo con una razón de 2/3 entre ellos), o cualesquiera otras.

5.2.4. Reasignación de recursos y tolerancia a fallos.

Las prioridades y restricciones asignadas a un sistema pueden llevar a una manera de trabajar completamente diferente. Un enfoque erróneo del problema lleva a un comportamiento no deseado. Consideremos por ejemplo el sistema de la Figura 13. Hay tres robots y cada robot lleva a cabo su tarea cuando hay una pieza que espera en su lugar de origen. R1 hace el proceso 1, es decir, mueve una pieza de A hasta A2, y los otros robots llevan a cabo una tarea análoga.

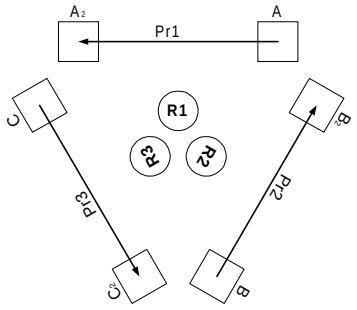


Figura 13: Tres robots que realizan tres procesos

Time required for moving a piece
In its process: 2 seconds
In the next robot's process: 3 seconds
Arrival times of pieces to places
A: 0, 1 + 3 n n=0,1,...
B: 2 + 3 n n=0,1,...
C: 3 + 3 n n=0,1,...

Figura 14: Tabla de Distribución temporal

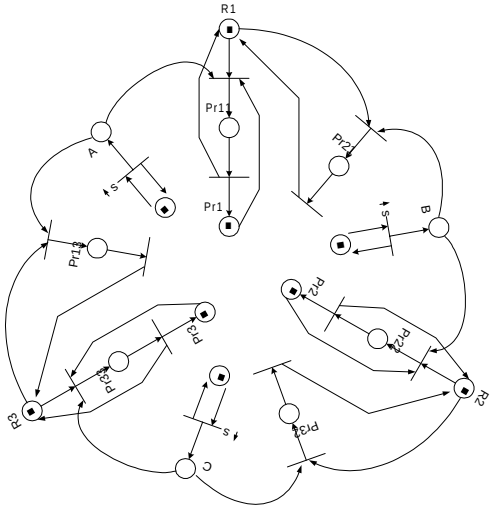


Figura 15: RdP de los procesos de la Figura 13

Es más, cada robot puede llevar a cabo los procesos del robot siguiente (R1 puede hacer el proceso 2, moviendo piezas desde B hasta B2, y así sucesivamente con R2 y R3). Esto sucede si no hay ninguna pieza esperando en su propio lugar, si los otros robots están ocupados y si hay una pieza esperando. Esto se muestra con una RdP en la Figura 15. R_i indica que el robot i está libre. P_{ri} indica que el proceso i no está siendo realizado por ningún robot. A (o B o C) indica que hay una pieza en el lugar A (o B o C). Y $\uparrow s$ indica que el sensor detecta que una nueva pieza ha venido al lugar, y está esperando para que el robot la mueva.

Parece una buena política para manejar los recursos, pero no obstante, si el material llega como se muestra en la tabla de la Figura 14, entonces cada robot llevará a cabo el trabajo del siguiente, y ninguno hará su trabajo.

La reasignación de recursos simplemente consiste en una modificación de la tabla de generación de la RdP. En este caso, se aumentan los grados de libertad de la evolución de la RdP. Pueden evolucionar según las restricciones y prioridades, pero también según la estructura de la red. Como previamente se ha mostrado, esto puede llevarse a cabo manualmente, después de un análisis off-line, o puede programarse para que se lleve a cabo automáticamente. En este último caso, las condiciones de evolución de los elementos de la tabla deberían programarse dependiendo de la evolución del sistema de entrada y del estado interno. El sistema debería saber qué elementos son susceptibles de ser modificados (añadidos o eliminados) y las condiciones para hacerlo. Por ejemplo, en

el último proceso, la opción de modificar el programa puede programarse para que cada robot pueda cumplir los otros dos procesos (en lugar de sólo el siguiente). De hecho esta posibilidad de modificar la red, limitada por las opciones programadas, puede interpretarse como un caso de modificación de restricciones y prioridades de la red, abarcando todas las posibilidades programadas [1, 2] (todos los robots tienen arcos a todos los procesos, pero siempre hay algunas restricciones).

Lo importante en este tipo de programación se centra en la reasignación de recursos: se han usado técnicas de lógica difusa (Fuzzy) y redes neuronales en varios procesos industriales similares a los de los ejemplos con resultados excelentes.

Todas las consideraciones anteriores se refieren a la modificación del trabajo del sistema para conseguir una mejora por medio de la adaptación a las condiciones cambiantes. De la misma manera, el funcionamiento puede adaptarse a un fallo en cualquier subsistema. Después de todo, un fallo en un elemento del sistema es lo mismo que quitar uno de los recursos. Por ejemplo, en el último caso, si un robot se estropea, o es retirado, el efecto es igual que quitar este robot de la RdP (y por consiguiente sus arcos de entrada y de salida) así como las condiciones lógicas. O también es lo mismo que añadir un 0 lógico a sus transiciones.

5.3. Aplicaciones de Inteligencia Artificial en la optimización del control automático

5.3.1. Introducción

Empleando RdP se puede integrar el control de los elementos más complicados del proceso (como pueden ser los robots) dentro del proceso global. Si dicho proceso global es totalmente determinista, se puede dejar evolucionar y ver su evolución a lo largo de los diversos estados por los que pasa. Pero también podemos emplear modelos no deterministas, en los que se controle la evolución del sistema mediante sus eventos, en los cuales hay que buscar la secuencia de eventos que consigue llevar al sistema desde el estado inicial al que nos interesa. En esos casos la búsqueda de dichas secuencias, del camino que lleva al sistema al estado deseado, puede ser una tarea complicada, y la utilización de técnicas de inteligencia artificial (IA), empleando algoritmos de búsqueda, puede constituir una manera eficaz de resolver el problema.

Por eso en esta sección se realiza un análisis de los métodos más útiles para las aplicaciones que integran esta tesis, y se aplican esos métodos para la resolución de un problema en particular. Sin embargo, estas técnicas pueden ser empleadas en cualquier parte del proceso en donde haya que buscar una solución que cumpla ciertas condiciones sin saber por qué camino se puede conseguir. En particular, estas técnicas pueden ser ciertamente útiles en el control de las propias RdP empleadas para el modelado y automatización del proceso, cuando sea una aplicación externa la que las controle mediante la secuencia de eventos necesarios.

En este estudio, que principalmente está centrado en sistemas de eventos discretos, las técnicas de IA han sido empleadas en las aplicaciones que manejan el almacén automático de bloques de elementos cerámicos fabricados. En dicho almacén se depositan los bloques de los distintos materiales a medida que se van fabricando, y se van retirando cuando abandonan la fábrica. Pero cuando se producen cambios de elemento fabricado, cosa que es relativamente frecuente (del orden de una vez por semana), puede hacer falta recuperar algún bloque que no sea directamente accesible, en cuyo caso hay que ir desplazando los bloques hasta traer el que nos interesa a una posición accesible. Esta tarea puede realizarse de varias maneras, y según la disposición de los elementos en el almacén puede ser complicado encontrar una solución, o puede ocurrir que la solución encontrada sea poco eficiente.

Precisamente para realizar la labor que acabamos de mencionar, de cálculo de la solución óptima, se ha recurrido a las técnicas de IA, implementando un algoritmo de búsqueda en un programa informático que se muestra a continuación y que se ha integrado dentro de la aplicación completa de control de la planta automatizada.

En esta sección se muestra en primer lugar un breve análisis de los métodos de búsqueda, tanto los métodos sin información como los de búsqueda heurística. Posteriormente se analizan las peculiaridades del problema concreto del almacén que se está tratando, así como el método de búsqueda implementado, y se muestra la heurística que se ha encontrado en las investigaciones para resolver el problema mejorando los resultados (convergiendo más rápidamente en las condiciones habituales de trabajo). Finalmente se muestra la aplicación informática que se ha implementado en lenguaje VB, para monitorizar el problema y el resultado, y que se integra dentro del conjunto de aplicaciones de control avanzado del proceso.

5.3.2 Conceptos teóricos sobre inteligencia artificial. Definiciones y clasificaciones

Para poder abordar de forma teórica el problema, mediante una metodología que facilite la solución del mismo, previamente se deben definir ciertos conceptos:

Agente: Es un elemento que percibe su *entorno* (la realidad que le afecta), mediante unos *sensores*, y que es capaz de actuar sobre él mediante los *actuadores*.

Agente racional: Es un agente que es capaz de cumplir el objetivo para el que ha sido diseñado. La racionalidad depende de los factores siguientes:

- La medida de actuación con la que se mide el objetivo alcanzado (grado de éxito).
- Las percepciones del entorno.
- Los conocimientos del agente acerca del entorno.
- Las acciones que puede realizar.

Agente racional ideal: Es aquel que realiza cualquier acción esperada para maximizar su medida de actuación, que se basa en evidencias obtenidas por secuencias de percepciones y en conocimientos incorporados.

El objetivo de la inteligencia artificial es el diseño de un programa agente que sea capaz de implementar una función que asocie a cada secuencia percibida una acción adecuada. Dicho programa se ejecuta dentro de un sistema de cómputo, que tenga una serie de sensores y actuadores, además de una estructura, a la que se le llama arquitectura. Por lo tanto un agente es la suma de la arquitectura más el programa.

Se definen varios tipos de estructuras de agentes racionales:

1. Agentes con tabla percepción - acción:
   Tienen una tabla de búsqueda en la que almacenan la totalidad de secuencias de percepciones. Dada una secuencia percibida, se localiza en la tabla la acción apropiada. Habitualmente no se puede llevar a la práctica porque la tabla suele ser extensa y no es autónomo, ya que si cambiase el entorno, la tabla no serviría.
2. Agentes de reflejo simple:
   Presentan reglas de *condición-acción* que resumen porciones de la tabla *percepción-acción*, de forma que se puedan implementar eficientemente. El inconveniente es que su rango de aplicación es muy pequeño.
3. Agente con estado interno:
   Es capaz de almacenar información de los estados del entorno en momentos anteriores utilizando un estado interno. Para definir cual es la situación del entorno en el momento en curso y actualizar la información del estado interno ha de saberse cómo evoluciona el *mundo* y qué es lo que producen las acciones.
4. Agentes basados en objetivos:
   Adicionalmente a la descripción del entorno, para decidir las acciones que ha de realizar, el agente ha de conocer cierto tipo de información acerca de sus *objetivos*, siendo esta información la que detalla las situaciones deseables.
   Existen dos campos de la I.A. dedicados a encontrar secuencias de acciones conducentes a alcanzar los objetivos del agente, estos son búsqueda y planificación.
5. Agentes basados en la utilidad:
   La utilidad es el grado de satisfacción que se ha alcanzado, medido mediante una función que asocia a un estado un número real. Esta especificación completa de la función de utilidad permite tomar decisiones racionales cuando existen múltiples objetivos que pueden ser conflictivos. Cada uno de los operadores puede ser caracterizado como poseedor de una función de utilidad.

Entorno: Los agentes inteligentes que acabamos de definir deben actuar de manera que el entorno experimente una serie de estados que permitan maximizar el rendimiento. Los entornos pueden clasificarse según los siguientes tipos:

1. Accesibles o no accesibles: Cuando los sensores del agente permiten el acceso a la totalidad del estado del entorno, se dice que es accesible a tal agente. Para entornos accesibles no hay necesidad de que el agente mantenga un estado interno.
2. Deterministas y no deterministas: Cuando el estado siguiente de un entorno se determina totalmente mediante el estado actual y las acciones seleccionadas por el agente, se dice que el entorno es determinista.

3. Episódicos y no episódicos: Se dice que un entorno es episódico cuando la experiencia del agente se divide en episodios, el cual consta de una percepción y una acción.
4. Estáticos y dinámicos: Cuando el entorno no sufre modificaciones mientras el agente está deliberando, se dice que es estático. Si tal entorno no cambia, pero el tiempo afecta a la medida de actuación del agente, se dice que el entorno es semi-dinámico.
5. Discretos y continuos: Cuando existe un número limitado de percepciones y acciones distintas, se dice que el número es discreto.

Problema: es la información que utiliza el agente para decidir qué va a hacer. Se define mediante dos conceptos básicos: los estados y las acciones. El *espacio de estados* del problema es el conjunto de estados que pueden alcanzare desde el estado inicial mediante cualquier secuencia de acciones.

Para definir formalmente el problema se necesita:
1. El estado inicial
2. Conjunto de operadores (posibles acciones)
3. El test del objetivo: la prueba que aplica el agente a un estado para decidir si se trata de un estado objetivo. Se lleva a cabo mediante un conjunto explícito de estados, o mediante una propiedad abstracta.
4. El coste del camino (en tiempo o en memoria): una función que asigna un coste a cada camino determinado. Ese coste es la suma de todas las acciones individuales que componen el camino.
5. Una solución: camino que va desde el estado inicial a un estado que satisface el test del objetivo.

Los problemas se pueden clasificar en los siguientes grupos:
1. Problemas de estado único (entorno accesible): Son aquellos en que el estado siempre puede determinarse con certeza.
2. Problemas de estado múltiple (entorno no accesible o no determinista): Se conoce un conjunto de estados en los que puede estar.
3. Problemas de contingencia (entorno dinámico): No puede encontrarse una secuencia de acciones garantizando la solución. El agente ha de calcular todo un árbol de acciones. Cada rama del árbol supone una posible contingencia que pudiera surgir. Esto ocurre en la mayoría de los problemas físicos reales.
4. Problemas de exploración: El agente no pose información acerca de las acciones ni acerca de qué tipos de estados existen. Ha de aprender el efecto de las acciones.

5.3.3 Resolución de problemas mediante búsqueda

Para la resolución de problemas mediante búsqueda se utiliza un *agente resolutor de problemas* que se puede definir como una clase de agente con *objetivo* que es capaz de decidir qué hacer, mediante la búsqueda de secuencias de operaciones (*acciones*), que llevan a estados deseados (*objetivos*).

Una vez que se ha definido el problema, se debe buscar la solución. Para esto se debe definir una estructura simbólica y las operaciones necesarias para resolver problemas y desarrollar estrategias, para así buscar eficiente y correctamente las posibles soluciones para estas estructuras y operaciones.

Para realizar la medida de la eficiencia de la actuación en la búsqueda de soluciones se debe tener en cuenta el coste de la búsqueda, tanto en tiempo como en memoria, el éxito (logro de la solución), y el coste total, que corresponde al del camino más el de la búsqueda.

La búsqueda de la solución se lleva a cabo a través de un espacio de estados, mediante los siguientes pasos:

- Se toma el estado inicial y se comprueba si es el estado objetivo.
- Se expande el estado actual, generando nuevos estados aplicando operadores.
- Se elige el siguiente estado a expandir, guardando los demás para después en el caso de que la primera elección no llegue al objetivo. La elección del estado que se desea expandir primero depende de la *estrategia de búsqueda*
- Se repiten los pasos de elección, prueba y expansión hasta alcanzar una solución, o hasta que no haya más estados que expandir.

Este proceso de búsqueda se puede entender como la construcción de un *árbol de búsqueda*. No se debe confundir este *árbol de búsqueda* (que se crea en el proceso de búsqueda) con el *espacio de estados* (un grafo con estados y operadores).

Para construir los árboles de búsqueda se emplean como nodos estructuras de datos de cinco componentes:

1. el estado en el espacio de estados al que corresponde el nodo, denominado nodo padre
2. el nodo del árbol de búsqueda que generó ese nodo
3. el operador al que se aplicó el nodo
4. el número de nodos de la ruta que hay desde la raíz hasta dicho nodo (la profundidad del nodo)
5. el coste de la ruta que va del estado inicial al nodo.

Cualquier nodo (salvo el inicial) tiene un nodo padre. Además cada nodo no-final tiene unos nodos que se generan a partir de él, a través de una *función expandir* (función que aplica a un nodo todos los operadores posibles, dándonos sus nodos rama). El grupo de nodos que están a la espera de ser expandidos se conoce como *margen o frontera* (también llamado *lista de nodos abiertos o de espera*). La representación más sencilla sería la de un conjunto de nodos, donde la *estrategia de búsqueda* es una función que escoge el siguiente nodo que se va a expandir. Según esto se supondrá que el grupo de nodos se implanta como si se tratara de una *lista de espera*.

Las operaciones con una lista de espera son las siguientes:

1. hacer lista de espera (Elementos): crea una lista de espera con base en los elementos dados.
2. vacía (Lista de espera): contestará afirmativamente sólo cuando no haya más elementos en la lista.

3. quitar frente (Lista de espera): elimina el elemento que encabeza la lista, y lo devuelve.
4. función poner en lista (Elementos, Lista de espera): pone un conjunto de elementos en la lista.

Cada variedad en la lista de espera generará distintas variedades del algoritmo de búsqueda.
Basándonos en las definiciones anteriores ya es posible dar una versión más formal del algoritmo general de búsqueda:

En el algoritmo general de búsqueda (Figura 16), obsérvese que Función_Lista_de_espera es una variable cuyo valor se convertirá en función.

Función Búsqueda_General (Problema, Función-lista de espera) responde con solución o fallo
Nodos ← Hacer-Lista de espera (Hacer-nodo(Estado-Inicial[problema]))
Bucle hacer
Si nodos está vació, **contestar** con falla
Nodo← Eliminar_del_frente(nodos)
Si Prueba-meta[problema] se aplica a Estado(nodo) y se tiene éxito, **contestar** con nodo
Nodos←Func.-Lista de espera(nodos,Expandir(nodo,Operadores[problema]))
fin

Figura 16: algoritmo general de búsqueda

5.3.4. Estrategias de búsqueda. Algoritmos para la resolución de problemas mediante búsqueda

Los algoritmos de búsqueda pueden clasificarse en dos tipos:

- Búsqueda ciega o no informada: No tienen información sobre el número de pasos o el coste del camino desde el estado actual al objetivo, sólo distinguen un estado objetivo de uno no objetivo. Las distintas estrategias se distinguen por el orden en el que son expandidos los nodos.
- Búsqueda heurística o informada: Utilizan conocimiento específico para un problema determinado. Permiten encontrar soluciones con más eficiencia.

5.3.4.1 Busqueda ciega o sin información

Veamos los seis algoritmos de búsqueda que pueden emplearse en búsqueda sin contar con información:

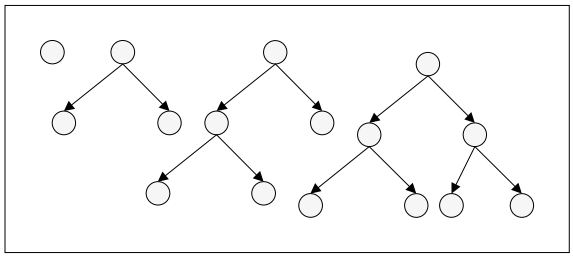
Búsqueda preferente por amplitud
Se trata de una de las estrategias más sencillas**.** En este caso, primero se expande el nodo raíz, y después todos los nodos generados por él; luego, sus sucesores, y así

sucesivamente. De modo general, todos los nodos que están en la profundidad d del árbol de búsqueda se expanden antes de los nodos que están en la profundidad d+1. Para realizar una búsqueda preferente por amplitud, se utiliza un algoritmo Búsqueda-General con una función de lista de espera mediante la cual se van poniendo los estados recién llegados a la cola de la lista, a continuación de todos los estados generados previamente:

Función Búsqueda_preferente_por_amplitud (problema) **responde** con una solución o falla
Responde con Búsqueda_general (problema) En-Lista de espera al final.

La estrategia de la búsqueda preferente por amplitud es bastante sistemática, ya que primero toma en cuenta todas las rutas de amplitud 1, luego las de longitud 2, etc.

En la siguiente figura puede verse el avance de un sencillo árbol binario.



Caso de existir solución, con seguridad que ésta se encontrará mediante la búsqueda preferente por amplitud; si son varias las soluciones, tal tipo de búsqueda preferente por amplitud permitirá siempre encontrar el estado meta más próximo. En función de los criterios, la búsqueda preferente por amplitud es muy completa y óptima, siempre y cuando el coste de la ruta sea una función que no disminuya al aumentar la profundidad del nodo, (por lo general, esta condición se da sólo cuando el coste de los operadores es el mismo).
Hasta aquí, la opción de la búsqueda preferente por amplitud parece atractiva. Pero, para mostrar por que no es siempre la estrategia de opción, considérese la cantidad de tiempo y memoria que son necesarias para realizar una búsqueda. Ha de tenerse en cuenta el espacio de estados hipotético, en el que la expansión de cada uno de los estados genera b nuevos estados. Se dice que el *factor de ramificación* de tales estados (y del árbol de búsqueda) es b. La raíz del árbol de búsqueda genera *b* nodos en el primer nivel, cada uno de los cuales a su vez genera b nodos más, dando un total de b^2 en el segundo nivel, y así sucesivamente. Supongamos ahora que en la solución de este problema, hay una ruta de longitud *d*. Así pues, la máxima cantidad de nodos expandidos antes de poder encontrar una solución es:

$1+b^2+.....+b^d$

Esta es por tanto la cantidad máxima, aunque la solución puede aparecer en cualquiera de los puntos del d-avo nivel. En el mejor de los casos, la cantidad será menor a éste.
Por lo tanto, dependiendo de los casos, el coste en memoria o en tiempo, puede llegar a ser prohibitivo, por tratarse de un *problema de complejidad exponencial* irresoluble en la mayoría de los casos, salvo que se trate de problemas poco complejos.

Búsqueda de coste uniforme

Aplicando la búsqueda preferente por amplitud se llega a encontrar el estado meta más próximo a la superficie, no obstante, éste estado no siempre es la solución de coste mínimo de la función general de coste de ruta. En el caso de *búsqueda de coste uniforme* se modifica la estrategia preferente por amplitud en el sentido de expandir siempre el nodo de menor coste en el margen (medido por el coste de la ruta g(n)) en vez del nodo de menor profundidad. No es difícil observar que la búsqueda preferente por amplitud no es sino una búsqueda de coste uniforme en la cual g(n)=PROFUNDIDAD(n).

El caso de coste de ruta no uniforme es similar al caso de encontrar la ruta más corta para ir de una ciudad a otra, donde las carreteras que unen distintas ciudades, tienen distinta longitud.

Si se cumplen determinadas condiciones, hay seguridad de que la primera solución encontrada es la más barata, dado que si hubiera una ruta más barata que fuese solución, ya se habría expandido anteriormente y ya habría sido encontrada.

Mediante la búsqueda por coste uniforme puede encontrarse la solución más barata, siempre y cuando se satisfaga un muy sencillo requisito: el coste de la ruta nunca debe ir disminuyendo mientras avanzamos por la ruta. En otras palabras, es importante que:
g(SUCESOR(n)) >= g(n), en todos los nodos n.

Búsqueda preferente por profundidad:

En este tipo de *búsqueda preferente por profundidad* siempre se expande uno de los nodos que se encuentre en lo más profundo del árbol. Solamente si la búsqueda conduce a un callejón sin salida (un nodo sin meta que no tiene expansión), se revierte la búsqueda y se expanden los nodos de niveles menos profundos.

La implantación de tal estrategia se realiza mediante Búsqueda-General, con una función de lista de espera que va colocando los estados recién generados a la cabeza de tal lista. Puesto que el nodo expandido fue el más profundo, los respectivos sucesores estarán a profundidades cada vez mayores. El proceso de la búsqueda puede verse en la siguiente figura.

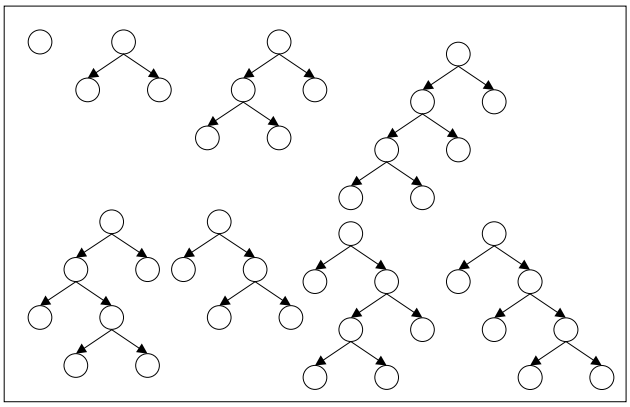


Figura 17: árboles de búsqueda preferente por profundidad de un árbol de búsqueda binario.

Para la búsqueda preferente por profundidad las necesidades de memoria son bastante modestas. Como puede verse en la figura, sólo es necesario guardar la ruta que va del

nodo raíz al nodo hoja, junto con los nodos restantes no expandidos, por cada nodo de la ruta. Si un espacio de estados tiene un factor de ramificación b y una profundidad máxima m, la cantidad de memoria que se requiere en una búsqueda preferente por profundidad es sólo de *O(bm)* nodos, en contraste con la cantidad de $O(b^d)$ necesaria en una búsqueda preferente por amplitud.

El inconveniente de la búsqueda preferente por profundidad es la posibilidad de que se quede estancada al avanzar por una ruta equivocada. En muchos problemas, los árboles de búsqueda son muy profundos, o hasta infinitos, de modo que en una búsqueda preferente por profundidad nunca será posible recuperarse de alguna desafortunada opción en uno de los nodos cercanos a la parte superior del árbol. La búsqueda proseguirá siempre en sentido descendente, sin ir hacia atrás, aún en el caso de que exista una solución próxima. Por lo tanto, en estos problemas, este tipo de búsqueda o se queda atrapada en un bucle infinito y nunca es posible regresar al encuentro de una solución, o a la larga encontrará una ruta de solución más larga que una solución óptima. Ello quiere decir que la búsqueda preferente por profundidad no es ni la más completa, ni la óptima. Por tanto, *cuando existan árboles de búsqueda con profundidades máximas prolongadas o infinitas ha de evitarse el empleo de la búsqueda preferente por profundidad.*
Es sencillo implantar la búsqueda preferente por profundidad mediante búsqueda general.

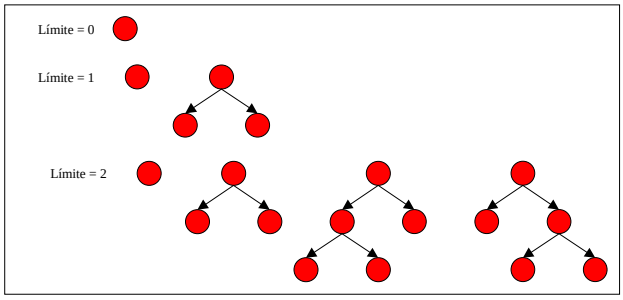
Función Búsqueda-preferente-por- profundidad (Problema) **responde** con una solución o falla. Búsqueda-general (problema) en-lista de espera al-frente

Búsqueda limitada por profundidad
Con el método de *búsqueda limitada por profundidad* se eliminan las dificultades expuestas en la búsqueda preferente por profundidad al imponer un límite a la profundidad máxima de una ruta. Para implantar tal límite ha de utilizarse un algoritmo especial de búsqueda limitada por profundidad, o utilizar el algoritmo general de búsqueda con operadores que se informan constantemente de la profundidad. A modo de símil podemos plantearnos el problema de encontrar el camino más corto entre 2 ciudades, teniendo en el mapa 20 ciudades. Sabemos que en el caso de que exista solución, ésta deberá tener como máximo 19 ciudades. La implementación al límite en profundidad se realiza utilizando operadores del tipo: “Si se está en la ciudad A y hasta allí se ha recorrido una ruta que incluye menos de 19 pasos, proceder a generar un nuevo estado en la ciudad B con una longitud de ruta que sea una unidad mayor”. Este nuevo conjunto de operadores garantiza que, caso de existir, se encontrará la solución; pero lo que no se garantiza es que la primera solución encontrada sea necesariamente la más breve: la búsqueda limitada por profundidad es completa pero no óptima. Además, en el caso de elegir un límite de profundidad excesivamente pequeño, la búsqueda limitada por profundidad ni siquiera será completa. La complejidad espacio-temporal de la búsqueda limitada por profundidad es comparable a la búsqueda preferente por profundidad. Requiere un tiempo de $O(b^l)$ y un espacio *O(bl)*, en donde l es el límite de profundidad.

Búsqueda por profundización iterativa

La difícultad en una búsqueda limitada por profundidad reside en la elección de un límite adecuado. En el caso del ejemplo propuesto (problema de las ciudades) el límite de 19 se escogió por tratarse de un límite "obvio", aunque puede darse el caso de que pueda realizarse en menos nodos, si se tiene una topología de mapa dada. Sin embargo, en la mayoría de los problemas, no resulta posible determinar un límite adecuado de profundidad hasta no haber resuelto el problema.

La *búsqueda por profundización iterativa* es una estrategia que elude el tema de la elección del mejor límite de la profundidad a base de probar todos los límites de profundidad posibles: Primero la profundidad 0, luego la profundidad 1, luego la profundidad 2, etcétera. El algoritmo utilizado se muestra en la figura siguiente:



En efecto, con la búsqueda de profundización iterativa se combinan las ventajas de las búsquedas preferente por profundidad y preferente por amplitud. Resulta óptima y completa, como la búsqueda preferente por amplitud, al tiempo que la memoria necesaria es sólo la de la búsqueda preferente por profundidad. El orden de expansión de los estados es similar al de la búsqueda preferente por amplitud, excepto que algunos de los estados se expanden varias veces. En la figura anterior pueden observarse las primeras de 3 iteraciones de la Búsqueda-por-profundización-iterativa en un árbol de búsqueda binario.

La búsqueda por profundización iterativa puede dar la imagen de desperdicio de tiempo, al expansionar tantos nodos tantas veces. Sin embargo, en la mayoría de los problemas, el exceso de tal expansión múltiple resulta ser en realidad bastante pequeña. La explicación intuitiva de lo anterior es que en un árbol de búsqueda exponencial, casi todos los nodos están en el nivel inferior, por lo que no importa demasiado que los niveles superiores se expandan varias veces.

Función Búsqueda-por-profundidad-iterativa (problema) **responde** con una secuencia solución.
Entradas: *problema*, un problema.
Para profundidad ← 0 **a** infinito **hacer**
    Si Búsqueda-limitada-por-profundidad(problema, profundidad) tiene éxito, **entregue** el resultado obtenido.
Fin
Responda falla

Algoritmo de búsqueda por profundidad iterativa.

Hay que recordar que la cantidad de expansiones de una búsqueda limitada por profundidad hasta una profundidad d con factor de ramificación b es:

$1+ b + b^2+...+b^{d-2}+b^{d-1}+b^d$
Para dar una idea de su valor, supóngase que b=10 y d=5; la cantidad correspondiente es:

1 + 10 + 100 + 1.000 + 10.000 + 100.000 = 111.111

En el caso de una búsqueda por profundización iterativa, los nodos que están en el nivel inferior se expanden una vez y los que aparecen a continuación del nivel inferior se expanden dos veces, etc., hasta llegar a la raíz del árbol de búsqueda, que se expande d+1 veces. Así que la cantidad total de expansiones en una búsqueda por profundización iterativa es:

$(d+1)*1+(d)*b+(d-1)*b^2+...+3*b^{d-2}+2*b^{d-1}+b^d$

En este caso, para los valores anteriores de b=10 y d=5, la cantidad de expansiones es:
6+50+400+3.000+20.000+100.000=123.456

A efectos de comparación. una búsqueda por profundización iterativa que comience por la profundidad 1 y continué hasta llegar a la profundidad d expande solo el 11 % más nodos en una búsqueda preferente por amplitud o por profundidad hasta la profundidad d, cuando b=10.
Cuanto mayor sea el factor de ramificación menor será el exceso de repetición de estados expandidos; incluso cuando el factor de ramificación es 2, la búsqueda por profundización iterativa sólo consume el doble de la búsqueda preferente por amplitud completa. Es decir que la complejidad temporal de la profundización iterativa sigue siendo $O(b^d)$ y la complejidad espacial O(bd). *Por lo general, la profundización iterativa es el método idóneo para aquellos casos donde el espacio de estados es grande y se ignora la profundidad de la solución.*

Búsqueda bidireccional
La búsqueda bidireccional es en esencia una búsqueda simultánea que avanza a partir del estado inicial, que retrocede a partir de la meta y que se detiene cuando ambas búsquedas se detienen en algún punto intermedio. En el caso de los problemas cuyo factor de ramificación es b en ambas direcciones, la búsqueda bidireccional se muestra muy útil. Si, como anteriormente se supone que existe una solución cuya profundidad es d, entonces la solución estará a $O(2b^{d/2})=O(b^{d/2})$ pasos, ya que en las búsquedas hacia delante y hacia atrás solo se recorre la mitad del trayecto.
Para un caso concreto con b=10 y d=6, una búsqueda preferente por amplitud genera 1.111.111 nodos, mientras que una búsqueda bidireccional tiene éxito cuando cada una de las direcciones de búsqueda están a profundidad 3, en cuyo caso generan 2.222 nodos. En teoría esto parece maravilloso, pero antes de poder implementar el algoritmo correspondiente han de resolverse varias cuestiones:

- Lo más importante es: ¿qué significa tiene buscar hacia atrás a partir de la meta?. Se definen los *predecesores* de un nodo n como todos aquellos nodos cuyo sucesor es n. La búsqueda hacia atrás implica pues la sucesiva generación de predecesores a partir del nodo meta.

- Si todos los operadores son reversibles, los conjuntos predecesor y sucesor son idénticos; pero en algunos problemas, sin embargo, el cálculo de los predecesores puede resultar muy difícil.
- ¿Qué se hará cuando son varios los posibles estados meta? En el caso de contar con una lista *explícita* de los estados meta, podemos aplicar una función de predecesor al conjunto de estado exactamente como se aplicó la función del sucesor en una búsqueda de estado múltiple. Si sólo contamos con una descripción del conjunto, aunque es posible darse una idea de los "conjuntos de estados que podrían generar el conjunto meta", resulta un procedimiento engañoso. Por ejemplo, en un juego de ajedrez ¿cuáles son los estados que se consideran predecesores a la meta jaque mate?.
- Es necesario contar con una herramienta eficiente para verificar cada uno de los nodos nuevos a fin de saber si ya están en el árbol de búsqueda de la otra mitad de la búsqueda.
- Ha de definirse que búsqueda se realizará en cada una de las dos mitades.

En la cifra de complejidad $O(b^{d/2})$ se supone que el procedimiento para probar la intersección de las dos fronteras puede efectuarse en tiempo constante (es decir, es independiente de la cantidad de estados). Para ello se emplea una embrollada tabla. A fin de que ambas búsquedas lleguen a encontrarse en algún momento, los nodos de al menos uno de ellos deberá quedar retenido en la memoria (como búsqueda preferente por amplitud). Es decir, la complejidad espacial de una búsqueda bidireccional que no cuenta con información es $O(b^{d/2})$.

Comparación de las diversas estrategias de búsqueda

En la siguiente tabla se comparan las seis estrategias de búsqueda de acuerdo a los cuatro criterios de selección:

Criterio	Preferente por ampli-tud	Coste uniforme	Preferente por prof.	Limitada en prof.	Profundi-zación iterativa	Bidirec-cional
Tiempo	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Espacio	b^d	b^d	bm	bl	bd	$b^{d/2}$
Óptima	Si	Si	No	No	Si	Si
Completa	Si	Si	No	Si cuando l > = d	SI	Si

Análisis de los estados repetidos

Hasta el momento no se ha dicho nada acerca de una de las más importantes complicaciones en el proceso de búsqueda: la posibilidad de la pérdida de tiempo al expandir los estados que ya se encontraron y ya fueron expandidos anteriormente, en alguna otra ruta. En el caso de algunos problemas, esta posibilidad no existe; solo hay una forma de llegar a cada uno de los estados.

Pero para muchos otros estados no es inevitable la repetición de estados. Este es el caso de todos aquellos problemas donde los operadores son reversibles, como para los problemas de determinación de rutas, y el problema de los misioneros y los caníbales. Aunque los árboles de búsqueda sean infinitos, si se podan algunos de los estados repetidos, el árbol quedará podado a un tamaño finito, y se generará sólo aquella porción del árbol que cubre la gráfica del espacio de estados. Aún en el caso de que el

árbol sea finito, evitar la repetición de estados produce una reducción exponencial del coste de búsqueda.

Existen tres formas de manejar los estados repetidos. Por orden de eficiencia y exceso de cómputo son:
- No ha de regresarse al estado del que acaba de llegar. Ha de instruirse a la función de expansión (o al conjunto del operador) para que se niegue a generar un sucesor cuyo estado sea el mismo que el del nodo padre.
- No han de crearse rutas que contengan ciclos. Ha de instruirse a la función de expansión (o al conjunto de operadores) a que se niegue a generar sucesores de un nodo idénticos a cualquiera de los ancestros del nodo.
- No ha de generarse ningún estado que se haya generado alguna vez anteriormente.

Para ello, han que guardarse en la memoria todos los estados generados, lo que implica potencialmente una complejidad espacial de $O(b^d)$. Es mejor considerar lo anterior como O(s), en donde s es la cantidad de estados en la totalidad del espacio de estados.
Para implantar esta última opción, los algoritmos de búsqueda utilizan una confusa tabla en la que se guardan todos los nodos generados. De este modo la verificación de estados repetidos resulta ser una opción razonablemente eficiente. El compromiso entre el coste de almacenar y verificar y el coste de la búsqueda adicional dependerá del problema: cuantos más bucles tenga el espacio de estados, mayor será la posibilidad de que la verificación rinda dividendos.

5.3.4.2 Búsqueda respaldada con información o heurística

Todos los métodos de resolución de problemas mediante búsqueda que no cuentan con información (los vistos anteriormente), resultan increíblemente ineficientes en la mayoría de los casos. Si el problema propuesto está perfectamente definido, se pueden reducir las opciones. Generalmente, el conocimiento en el que se apoya una decisión se obtiene mediante una *función de evaluación*, la cual arroja un número que sirve para representar lo deseable (o indeseable) que sería la expansión de un nodo. Cuando los nodos se ordenan de modo tal que se expande primero aquél con mejor evaluación, entonces se trata de una estrategia denominada *búsqueda preferente por lo mejor*.

Función Búsqueda-preferente-por-lo-mejor (Problema,Función-Evaluación) **responde con una** secuencia de solución.
Entradas: problema, un problema
Función-Eval, una función de evaluación
Función-Lista de espera ← una función que ordena todos los nodos mediante Función-Eval
Responde con Búsqueda-general(problema,Función-Lista de espera)

Del mismo modo que existe toda una familia de algoritmos Búsqueda-general, con distintas funciones de ordenamiento, también existe una familia de algoritmos Búsqueda-preferente-por-lo-mejor, que esta formada por diversas funciones de evaluación. Puesto que su objetivo es encontrar soluciones de bajo coste, por lo general en estos algoritmos se utiliza alguna medida estimada del coste de la solución, y se

hacen esfuerzos por reducir este coste al mínimo. Anteriormente ya se ha presentado un ejemplo de tal medida: el empleo del coste de ruta g para decidir qué ruta emplear. Sin embargo, esta medida no es una búsqueda directa dirigida a la meta. Para enfocar la búsqueda, en tal medida debe figurar algún tipo de cálculo del coste de la ruta que va de un estado al estado más cercano a la meta. Se trata por tanto de dos tipos de aproximación básicos: el primero trata de expandir el nodo más cercano a la meta; el segundo, el correspondiente a la ruta de la solución menos costosa.

Búsqueda avara. Reducción del coste estimado al mínimo para alcanzar la meta

Para la mayoría de los problemas, aunque puede estimarse el coste que implica llegar a una meta desde un estado determinado, no es posible hacerlo con precisión. La función que se utiliza para calcular tales estimaciones de coste es conocida como *función heurística*, y se simbolizada generalmente con la letra h:

h(n) = coste estimado de la ruta de menor coste que une el estado del nodo n con un estado meta.

A la búsqueda preferente por lo mejor que utiliza la función h(n) para elegir cuál es el siguiente nodo que se va a expandir se la denomina *búsqueda avara*, por razones que resultarán evidentes. Dada una función heurística h, el código correspondiente a una búsqueda avara es el siguiente:

Función Búsqueda-avara (problema) **responde con** solución o falla
Responde con Búsqueda-preferente-por-lo-mejor(problema,h)

Desde el punto de vista formal, h puede ser cualquier función, habiendo de cumplir el único requisito es que h(n) = 0 cuando n es una meta.

Por otra parte, también es deseable que la función heurística aporte una idea de lo cerca que está un estado de la solución, en el sentido de que su valor sea menor cuando más cerca esté de esta.

Búsqueda A^*. Reducción del coste de ruta total al mínimo

Con la búsqueda avara, como permite reducir al mínimo el coste de la meta, h(n), se reduce en forma considerable el coste de la búsqueda. Por desgracia, este tipo de búsqueda no es óptimo ni tampoco completo. De otra parte, la búsqueda de coste uniforme, reduce al mínimo el coste de la ruta, g(n) y es óptima y completa, aunque puede ser muy ineficiente. Sería muy interesante utilizar ambas estrategias para así combinar las ventajas que ofrecen, lo cual es por suerte posible hacerlo, combinando las dos funciones de evaluación mediante una suma:

$$f(n)=g(n)+h(n)$$

Dado que con g(n) se calcula el coste desde la ruta que va del nodo de partida al nodo n, y h(n) es es coste estimado de la ruta de menos coste que va de n a la meta, tenemos que:

f(n) = coste estimado de la solución de menos coste, pasando por n.

En otras palabras, si de lo que se trata es de encontrar la solución de menos coste, entonces es razonable probar primero en el nodo cuyo valor de f sea el más bajo. Lo interesante de esta estrategia es que no sólo es razonable, sino que puede demostrarse que es completa y óptima, dada una sencilla restricción de la función h.
Tal restricción consiste en escoger una función h que nunca sobreestime el coste que implica alcanzar la meta. A dicha función h se le conoce como *heurística admisible*. Por naturaleza, este tipo de funciones heurísticas son optimistas, pues consideran que el coste para resolver un problema es siempre inferior a lo que en realidad es. Tal optimismo se transfiere a la función *f: Si h es aceptable, f(n) nunca sobreestimará el coste real de la mejor solución que pase por n*. A esta búsqueda preferente por lo mejor, en la que se utiliza f como una función de evaluación y una función h aceptable, se le conoce como búsqueda A^*.

Función Búsqueda-A^* (problema) **responde con** una solución o falla
Responde con Búsqueda-preferente por lo mejor (problema,g+h)

Propiedades de A^*:

- óptima

Si G es un estado meta óptimo, cuyo coste de ruta es f^*, y G_2 un estado meta subóptimo, es decir un estado meta cuyo coste de ruta es $g(G_2) > f^*$, supóngamos que A^* escogió a G_2 de la lista de espera. Dado que G_2 es un estado meta, la búsqueda habría concluido al escoger la solución subóptima. Se trata de demostrar que eso no es posible.
En efecto, considérese un nodo n que en un momento dado es un nodo hoja de una ruta óptima para G (debe haber algún nodo así, a menos que ya se haya expandido toda la ruta, en cuyo caso el algoritmo ya habría respondido con G). Supuesto que h es aceptable, tendremos:

$$f^*.>= f(n)$$

Además, si n no es seleccionado para expansión sobre G_2 , debemos tener

$$f(n)>=f(G_2)$$

Combinando juntas esas dos desigualdades, se tiene:

$$f^*.>= f(G_2)$$

Pero como G_2 es un estado meta, tenemos que $h(G_2)=0$, por lo tanto $f(G_2)=g(G_2)$. Y así, con base en lo supuesto se demuestra que:

$$f^*.>= g(G_2)$$

Lo anterior contradice la suposición de que G_2 es estado de meta subóptimo, y por lo tanto, A^* nunca seleccionará para expandir un estado meta subóptimo. Por lo tanto, puesto que sólo se produce una solución después de seleccionarla para efectuar una expansión, A^* es un algoritmo óptimo.

- completa

Anteriormente se comprobó que dado que A^* expande los nodos por orden de f creciente, al final hará una expansión que le permita llegar al estado de meta. Esto es

cierto, desde luego, salvo que haya una cantidad infinita de nodos con $f(n)<f^*$. La única forma de que haya una cantidad infinita de nodos es:

a) que haya un nodo con factor de ramificación infinito o
b) que haya una ruta con coste de ruta finito, pero con un número infinito de nodos a lo largo de ella.

Los dos casos son infrecuentes en los problemas normales.

Por lo tanto, el enunciado correcto consiste en afirmar que A^* es completa en *gráficas localmente finitas* (gráficas con un factor de ramificación finito), suponiendo que exista una constante positiva δ, de modo que todos los operadores cuesten por lo menos δ.

5.3.5. Control del almacén mediante IA

5.3.5.1. El problema del almacén

Básicamente el objetivo de la aplicación consiste en la resolución de un problema de movimiento automático de bloques de ladrillos o elementos cerámicos depositados en un almacén para conseguir situarlos donde nos interese. Dicha necesidad deriva de la posibilidad de tener distintos tipos de elementos en el almacén, en cuyo caso puede interesar acceder a uno que esté situado detrás de los de otro tipo.

El almacén recibe los bloques de material de la planta por la parte inferior (ver Figura 18) y desde el almacén se cargan en los camiones de salida por la parte superior. Los bloques están numerados, de tal forma que se conocen todos los datos de cada uno (tipo de material, fecha de fabricación, etc.) y se tienen que mover empleando los huecos libres que quedan, para conseguir huecos a la entrada y para tener los bloques que nos interesen a la salida.

Pero además de todo eso, puede ocurrir que interese que en el proceso, alguno de los bloques permanezca fijo (porque es un material más delicado, porque se ha bloqueado el vagón o porque se está trabajando sobre él, o porque nos interesa que permanezca en su posición previendo necesidades inmediatas). Por eso el problema debe tener en cuenta también la necesidad de bloquear los bloques que nos interese en sus posiciones para buscar el camino sin emplear esos lugares.

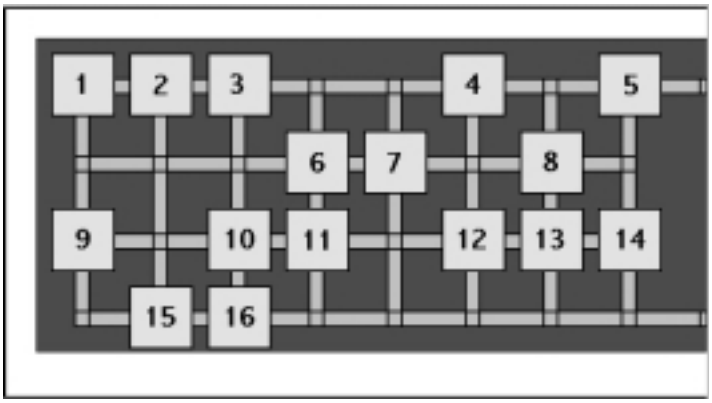


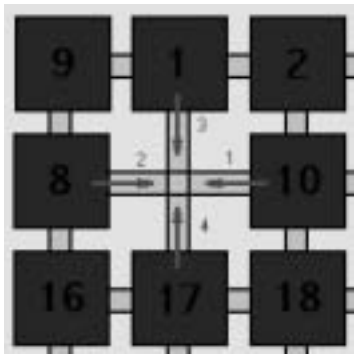
Figura 18: Almacén automático de bloques de elementos producidos.

5.3.5.2. Planteamiento formal del problema

Para la resolución del problema, se ha utilizado el algoritmo de búsqueda A*, ya que es el que mejores expectativas ofrece para este problema en cuanto a tiempo de resolución, gasto de memoria, y solución óptima.

Estado inicial:
En este problema, el estado es la disposición de los bloques, y es representado por medio de una matriz de 8 x 4, que coincide con la geometría del problema. Esta matriz indica la disposición de los bloques, con su número del 1 al 31 (capacidad máxima). Los lugares que no tengan bloques (huecos), se representan con un cero.

Conjunto de operadores



Existe un grupo de cuatro operaciones sencillas:

1- Mover el bloque de la derecha del hueco, hacia la izquierda.
2- Mover el bloque de la izquierda del hueco hacia la derecha.
3- Mover el bloque de arriba hacia abajo.
4- Mover el bloque de abajo hacia arriba.

Además para estos movimientos existen limitaciones, que son:

1- Limitación geométrica, ya que en los bordes del recinto, no se pueden dar los cuatro tipos de movimiento.
2- Restricciones en el planteamiento del problema, ya que puede que uno de los bloques que rodee el hueco, sea precisamente uno de los que no se pueden mover.

Test del objetivo
Para definir el objetivo, a la vez que las restricciones del problema, se ha definido una *matriz objetivo* de 8x4 enteros. En ella si aparece un número del 1 al 31 nos indica que en esa posición debe acabar el bloque de ese número, si hay un 100 es indiferente qué bloque acabe en esa posición, y si hay un 1000, ese bloque no se debe mover en el proceso, por lo que no debemos preocuparnos de si cumple o no cumple las expectativas (por definición las cumplirá, al no moverse de su sitio en todo el proceso).

Entonces se compara el estado del nodo en curso con la matriz objetivo, y si se cumplen todas las condiciones entonces el objetivo se ha cumplido.

Coste del camino
Afortunadamente, el coste del camino en este caso coincide con la profundidad del nodo en el árbol, ya que todos los operadores tardan en ejecutarse el mismo tiempo (porque el bloque al desplazarse por entre dos lugares cualesquiera tarda el mismo tiempo). Esto es porque el almacén es homogéneo en la distribución de sus lugares. Si no fuese así, entonces habría que implementar una función coste, que nos indicara la longitud del recorrido.

5.3.5.3. Solución y heurística empleada

Solución
Va a ser cualquiera que cumpla las condiciones de la matriz objetivo descrita anteriormente.

Heurística utilizada
Normalmente el coste de una solución exacta para un problema menos restrictivo es una buena heurística para el problema original. En general, cuanto más informada es una heurística, menor es el número de nodos expandidos para alcanzar la solución óptima, Sin embargo, se debe tener cuidado con los cálculos necesarios para obtener el valor heurístico, ya que puede ocurrir que el cálculo cueste más que la expansión de nodos que ahorra.

La Heurística que se utilizó en un principio para resolver este problema fue la suma de las distancias Manhatan (es decir, las distancias no se recorren en dirección diagonal, sino solo en horizontal y vertical) de cada una de las casillas que estaban contenidas en la función objetivo. De esta forma, si el movimiento es favorable, mejorará la heurística.

Pero con esa heurística se comprobó que no se cumplían las expectativas de nodos expandidos para llegar a la solución, ya que un ejemplo sencillo casi agotaba la memoria del ordenador. Por eso se tuvo que buscar otra solución. Esta solución se basa en el siguiente razonamiento:

Supongamos que el hueco que aprovechamos para realizar el movimiento está alejado de la posición del bloque que queremos mover. Ningún movimiento de los iniciales nos aportaría información de cómo de bueno es ese movimiento, ya que la heurística sería la misma, al no haberse cambiado de posición ningún bloque. Es más, al principio, al sumarse la heurística a la profundidad, se expandiría el árbol todo lo posible en los primeros nodos, ya que los *hijos* serían todos de *peor heurística* (+ coste del camino que el padre). La solución consiste en aplicarle un coeficiente a cada término de la suma de la heurística anterior, que se haga mínima cuando el bloque al que le vamos a medir su distancia Manhatan, tiene al lado el hueco del movimiento. De esta manera, al tratarse de una multiplicación, sigue cumpliendo la condición de que en el objetivo la heurística se anule (valor 0), y favorece los caminos que tienen más cerca el hueco para poder mover el bloque, que los que no los tienen, solucionando así el problema.

Esta solución, en uno de los ejemplos más sencillos, logró un camino correcto con una expansión de tan sólo 78 nudos, frente a los 357 del caso anterior. Y en los casos más complicados posibles siempre se logra una solución sin agotar la memoria del ordenador, cosa que no se lograba anteriormente.

Puesta en práctica del método descrito

Para la realización práctica de la resolución de este problema, se han realizado varias versiones (programas de Visual Basic 5), hasta llegar a una solución admisible tanto en tiempo de ejecución de la búsqueda de la solución, como en número de pasos para resolver el problema. A continuación se describen por qué se han realizado estos programas de esta forma, qué inconvenientes se les ha encontrado y por qué han sido cambiados. Esto puede dar una idea de, sobre todo, que no hacer cuando se empleen estas técnicas en aplicaciones similares:

Versión 1:
En cada nodo se guarda la ruta en una cadena de texto, por lo que consume mucha memoria al tener que guardar en cada nodo una cadena de caracteres, por tanto solamente podemos tener 940 nodos, lo que limita mucho el tipo de problemas que se pueden resolver.

Versión 2:
En un intento de resolver el problema anterior, en lugar de guardar la ruta para llegar a ese nodo, se guarda el nodo padre, y aprovechando que en los nodos se guarda el operador que lo ha generado, podemos reconstruir la ruta solamente con esta información, y así pasamos de tener 940 nodos a 1400 nodos como máximo. Pero aun así, hay algunos problemas “sencillos” (colocación de 2 boxes en un determinado lugar) que requieren mas de 1400 nodos.

Versión 3:
Para intentar paliar el problema de memoria, se realiza un “paginado” de las listas de nodos, colocando solamente las paginas activas en memoria, con eso conseguimos multiplicar por mucho la capacidad de memoria de las listas. El problema, es que andar trabajando con disco duro en lugar de memoria, hace intratable el tiempo de ejecución del algoritmo, y para un problema real, esta solución, sin ser acompañada de ninguna mejora, no es viable.

Versión 4:
Para mejorar los tres métodos anteriores en la tarea de encontrar una solución óptima en tiempo y que dé solución aceptables, se ha intentado buscar otro tipo de soluciones. En primer lugar se puede observar que la heurística utilizada era buena cuando queríamos llevar dos bloques a una posición que estuviese separada; pero si la posición destino era contigua, se veía que el método no era bueno al colocarlos en la posición destino (no al acercarlos a ella). También puede observarse que si la posición donde se quieren colocar las cajas es cercana a su posición original, la heurística propuesta en la bibliografía clásica (la de distancia de cada casilla a su posición final) es sensiblemente mejor. Así que se puede deducir que el problema no depende tanto de la cantidad de memoria, sino de la heurística utilizada. Entonces, se ha optado por utilizar una heurística que detecta

en cuál de los dos casos nos encontramos. Es decir, si se puede encontrar una solución trabajando en una pequeña zona del almacén, se utiliza la heurística de la bibliografía, y si no, la que se ha descrito anteriormente. Esta nueva heurística ha conseguido dar buenos resultados en tiempos de cálculo cortos y con memoria dentro de los límites.

Heurística mejorada

Como hemos dicho en el apartado anterior, cuando queremos colocar varios bloques en una zona contigua, y estos están en una zona reducida, la heurística que llamaremos Heurística1 funciona mejor, y cuando se quieren llevar a una zona alejada y que no estén contiguos, funciona mejor la heurística que llamaremos Heurística2. Por otra parte, cuando queremos expandir un nodo, si tiene en el Objetivo, un bloque marcado como no movible, la función expandir, no lo expandirá.

Con lo dicho anteriormente, se nos puede ocurrir que según se van expandiendo los nodos, si en lugar de hacerlo con el objetivo inicial, lo vamos haciendo en zonas mas pequeñas, restringiéndolo con un nuevo Objetivo2, que tenga delimitada la zona por medio de bloques inamovibles, se podrá encontrar, al ser mas pequeña la zona, la solución mas rápidamente. Además, si dependiendo del tamaño de esa zona (mas de 3 filas o 3 columnas) se utiliza la heurística 2 o la 1, entonces se podrá encontrar la solución de forma más eficiente tanto en tiempo como en espacio de memoria.

Esto tiene un inconveniente, que es que se puedan quedar zonas muertas por tener un bloque inamovible dentro de esa zona, pero estos ya han sido tenidos en cuenta, y en ese caso se desplazará la columna de bloques inamovibles de forma que no se ahogue y se pueda encontrar una solución. Todo lo dicho anteriormente se explica con los siguiente ejemplos:

Tenemos el siguiente estado en la disposición de los bloques de un nodo:

1		2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

Y queremos llegar al siguiente nodo objetivo:

X	X	X	X	X	X	X	X
X	X	X	9	X	F13	X	X
X	X	X	X	4	5	X	X
X	X	X	X	X	X	X	X

Es decir, queremos que los bloques números 4, 5 y 9 (y el hueco) se queden en esa posición sin tocar el bloque 13. Si definimos ”Área de trabajo” como la mínima serie de columnas consecutivas del estado del nodo que engloban bloques objetivo, nuestra área de trabajo será:

1		2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

También podemos definir como "Área de objetivo" la mínima serie de columnas consecutivas del estado objetivo, que engloban a todos los bloques objetivo (y el hueco) que deben ser movidos.

X	X	X	X	X	X	X	X
X	X	X	9	X	F13	X	X
X	X	X	X	4	5	X	X
X	X	X	X	X	X	X	X

Entonces, nuestro nuevo objetivo, tendrá una columna de bloques fijos a la izquierda de la columna que esté mas a la izquierda del área de trabajo o del área de objetivos. Lo mismo pasará por la derecha. Entonces, sin tener en cuenta la casilla fija F13, nuestro objetivo 2 sería:

X	F	X	X	X	X	F	X
X	F	X	9	X	F13	F	X
X	F	X	X	4	5	F	X
X	F	X	X	X	X	F	X

Reflexionando un poco, vemos que en esta disposición del objetivo2, si tenemos algún elemento fijo en las casillas sombreadas, podremos tener problemas a la hora de solucionar el problema, porque tendremos casillas muertas para mover. En nuestro caso, tenemos fija la casilla que pone F13, por lo tanto debemos mover a la derecha la columna de elementos fijos de la derecha, quedándonos al final el objetivo2:

X	F	X	X	X	X	X	F
X	F	X	9	X	F13	X	F
X	F	X	X	4	5	X	F
X	F	X	X	X	X	X	F

Como vemos, tenemos reducido el área donde se van a expandir los nodos, en este caso, como es de 5 columnas se utilizará una Heurística2, y si fuesen 3 columnas (tamaño mínimo limitado), entonces utilizaríamos la Heurística1.

5.2.5.4. Aplicación informática.

La aplicación informática sobre la que se ha aplicado el algoritmo de resolución del problema, y que monitoriza y simula el proceso, está implementada en lenguaje Visual Basic.

Es una aplicación muy sencilla de manejo, en la que inicialmente aparece el esquema del almacén sobre el que podemos indicar cual es la disposición inicial de bloques (Figura 19). Debajo del esquema del almacén hay unas casillas que representan todos los lugares, en las que tenemos que indicar la disposición final del almacén, es decir, la solución buscada. Para indicar dicha solución se debe rellenar con una 'f' las posiciones que queremos que permanezcan fijas, poner el número de los bloques en las posiciones en las que queremos que terminen, y dejar una 'x' en las posiciones que no importa con que bloque terminen, tal como muestra la pantalla de ayuda. Tras indicar la disposición final, si se pulsa el botón 'Calcula' el algoritmo resuelve el problema y genera un archivo de texto con los pasos de la solución calculada. Posteriormente podemos representar los movimientos calculados mediante el botón 'Animación'. Además hay otros dos botones, para visualizar el estado y para inicializar el proceso.

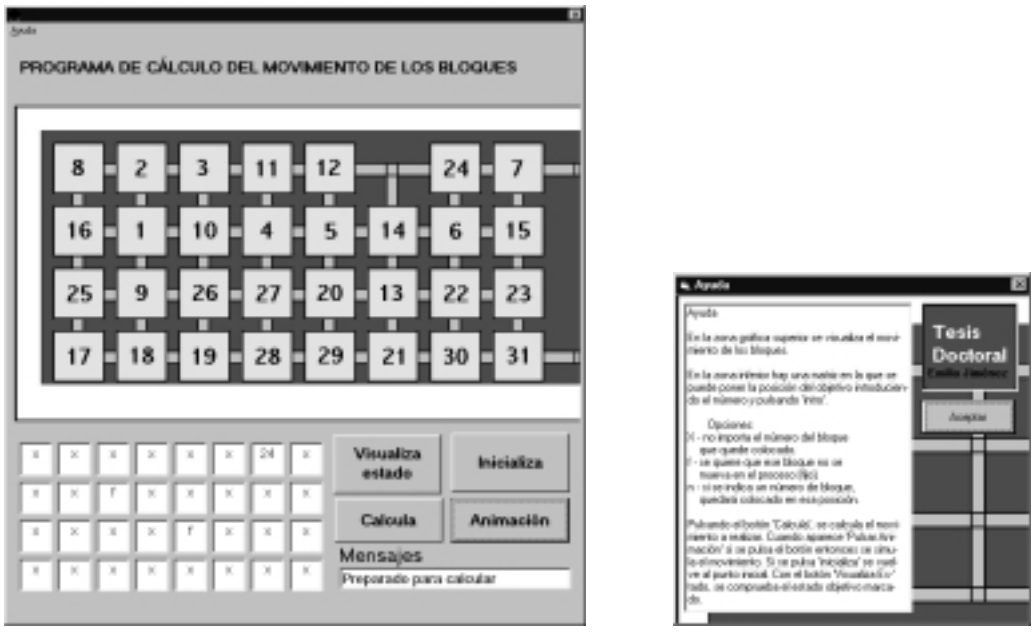


Figura 19: Aplicación informática para el problema del almacén, y pantalla de ayuda.

5.4. Conclusiones

La aportación de este capítulo se centra en dos puntos: el primero en la utilización de RdP dinámicas (con estructura variable) que se adapten a la producción en sistemas flexibles altamente variables, y el segundo el empleo de algoritmos de búsqueda para encontrar solución a problemas en la automatización.

En el primero de los puntos se han mostrado algunos ejemplos de robots manejados en aplicaciones con características especiales: Procesos flexibles de producción con robots compartidos entre varios procesos y robots usados en procesos altamente variables. Se consigue la modificación de programas complejos en las plantas industriales automáticas fácilmente. Eso se logra con un programa que guarda las especificaciones en una tabla y los convierte en la RdP correspondiente, y después la RdP en el programa de PLC. Esto también permite al sistema llevar a cabo automáticamente los cambios en los parámetros del funcionamiento, según los requisitos y el sistema de trabajo. Estos cambios son realizados en tiempo real (usando técnicas Fuzzy y redes neuronales), de modo que la RdP (y los programas de control del proceso) evolucionan para conseguir una mejora por medio de la adaptación a las condiciones cambiantes. Estas aportaciones también han sido estudiadas sobre procesos reales.

También se ha mostrado dentro de ese primer punto del tema que esa información lógica, que cambia dependiendo de las entradas y los estados internos, puede ser usada como restricciones o condiciones (funciones del estado interno), o prioridades (que a su vez pueden interpretarse como condiciones). Ambas pueden traducirse en nuevos elementos dentro de la RdP (arcos, lugares y transiciones), para que de esta manera pueda realizarse el análisis de la actuación del proceso con las restricciones simplemente a través de la RdP modificada.

La segunda parte del capítulo muestra como pueden integrarse en todas estas aplicaciones métodos de inteligencia artificial para buscar la mejor solución cuando haya varias posibilidades o cuando no sepamos cómo conseguirla. Puesto que de IA no se ha tratado nada en el primer capítulo, que es el que introducía a los temas principales de automatización, se comienza haciendo un repaso de los métodos de búsqueda empleados en IA, al estilo de un Estado del Arte. Tras analizar los métodos de búsqueda se ha implementado una aplicación que emplea el más apropiado para el movimiento automático de los paquetes producidos y almacenados, mediante puentes grúa o mediante otros tipos de robots. Esos métodos pueden también emplearse para controlar las propias RdP cuando éstas no sean deterministas (cuando la automatización controle parte de los eventos de su evolución) y haya que buscar un camino para conseguir llevarlas al estado deseado.

5.5 Referencias

1. Belli, F., Grosspietch, KE.: Specification of fault-tolerant systems issues by predicate/transition nets and regular expressions: Approach and case study. IEEE Trans on Software Engineering (1991) 17 (6), 513-526

2. Burns, A., Wellings, A.: Real-Time Systems and Programming Languages. Addison-Wesley, California (1996)

3. David, R.: Grafcet: A powerful tool for specification of logic controllers. IEEE Trans on Control Systems Technology (1995) 3 (3) , 253-268

4. Jiménez, E., Miruri, JM., Martínez de Pisón, J.F., Gil, M.: Supervised Real-Time Control with PLCs and SCADA in Ceramic Plant. 6th IFAC Workshop on Algorithms and Architectures for Real-Time Control (2000) 1 , 221-226

5. Jiménez, E.: Redes de Petri de Supervisión y Simulación de Procesos Industriales Automatizados. XXI Jornadas de Automática CEA-IFAC (2000)

6. Morriss, B.: Programmable Logic Controllers. Prentice Hall, New Jersey (1999)

7. N. Bhandari, D.K. Rollins. Superior Semi-Empirical Dynamic Predictive Modeling that Addresses Interactions. IASTED Intelligent Systems and Control ISC'99 (1999)

8. N. Konstas, S. Lloyd, H. Yu, C. Chatwin. Generic Net Modelling Framework for Petri Nets. IASTED Intelligent Systems and Control ISC'99 (1999)

9. Z. Bingul, A.S. Sekmen, S. Palaniappan, S. Sabatto. An Application of Multi Dimensional Optimization Problems using Genetic Algorithms. IASTED Intelligent Systems and Control ISC'99 (1999)

10. W.T. Goh, Z. Zhang. Autonomous Petri-Net for Manufacturing System Modelling in an Agile Manufacturing Environment. IASTED International Confer. Robotics and Applications 1999.

11. Evolutionary algorithms in engineering and computer science : recent advances in genetic algorithms, evolution strategies, evolutionary programming, genetic programming and industrial apllications / edited by K. Miettinen... [et al.] (1999)

12. Momoh, James A., Electric systems, dynamics, and stability with artificial intelligence applications / New York ; Basel : Marcel Dekker, [2000] ISBN 0-8247-0233-6

13. Bender, Edward A., Mathematical methods in artificial intelligence / Edward A. Bender (2000)

14. Vas, Peter, Artificial-intelligence-based electrical machines and drivers : application of fuzzy, neural, fuzzy-neural, and genetic-algorithm-based techniques / Peter Vas (1999)

15. National Conference on Artificial Intelligence (16th. 1999. Orlando), Proceedings : Sixteenth National Conference on Artificial Intelligence (AAAI-99) : Eleventh Innovative Apllications of Artificial Intelligence Conference (IAAI-99) : [July 18-22, 1999, Orlando, Florida] / [sponsored by the American Association for Artificial Intelligence] (1999)

16. Inteligencia artificial y realidad virtual (2000) Madrid : Instituto de Cooperación Iberoamericana, 2000. Cuadernos hispanoamericanos, ISSN 1131-6438 ; 596. Ejemplar monográfico de la revista Cuadernos hispanoamericanos, n. 596 (2000)

17. Artificial intelligence in perspective / edited by Daniel g. Bobrow (1994)

18. Artificial intelligence: methodology, systems, applications / edited by A.M. Ramsay (1996)

19. Artificial intelligence techniques in power systems / edited by Kevin Warwick, Aarthur Ekwue and Raj Aggarwal (1997)

Tema 6

APLICACIÓN EN PLANTA INDUSTRIAL DE ÚLTIMA GENERACIÓN.

6.1. Introducción

En este capítulo se presenta la aplicación de todo lo comentado en los temas precedentes sobre una planta industrial real de última generación. La planta es una de las mayores del mundo de fabricación de ladrillos, y también una de las más modernas. En concreto, la última de las naves, que es a la que nos referiremos, aún está en fase de prueba, aplicando muchas de estas técnicas, para conseguir ajustar los parámetros y mejorar en lo posible el ritmo productivo. Veremos que las características de esta planta la hacen especialmente propicia para muchas de las técnicas desarrolladas en esta línea de investigación: se trata de un proceso complejo y extenso, compuesto por múltiples subsistemas fuertemente interconectados. Debido a las características del producto siempre se debe estar modificando las características de producción (se fabrica el tipo de ladrillo que demanda el mercado), el proceso no es crítico por exigencias de tiempos de respuesta, pero es frecuente producir material defectuoso por causas de variación de los parámetros tanto internos (tiempo en horno o secadero de las piezas, etc.) como externos (temperatura y humedad ambiente, etc.).

Una de las dificultades comunes a todos los temas, que es la de resumir en no demasiadas hojas las aportaciones del trabajo de investigación, en este capítulo será especialmente dificultoso, debido a la cantidad de material obtenido de dicho trabajo y que debería ser incluido (y posiblemente debido también al entusiasmo de quien lo redacta, que después de tanto tiempo de investigación sobre dicho proceso, con buenos y malos resultados, considera la planta como una obra propia de la que se siente ciertamente orgulloso).

Por tanto en este tema comenzaremos con un análisis metodológico de los sistemas concurrentes formados por varios susbsistemas cíclicos a través de los que va pasando el producto para seguir el proceso de producción. Esta aportación es novedosa respecto al resto del trabajo. Posteriormente se explica detalladamente cómo es el proceso productivo al que nos estamos refiriendo, y a partir de ahí se van aplicando sobre dicho proceso las técnicas que hemos analizado en los capítulos anteriores para analizar sus resultados y comprobar la eficacia real de toda la investigación.

6.2. Modelado y control de procesos con subsistemas cíclicos

Un tipo de sistema relativamente usual en el modelado de plantas industriales es el que incluye varios subsistemas cíclicos, concurrentes y sincronizados entre sí y alguno incluso integrado dentro de otros. El material principal de producción pasa sucesivamente a través de ciclos para seguir una cierta ruta a lo largo de la planta. Por otro lado, hemos analizado previamente algunas herramientas para el modelado y

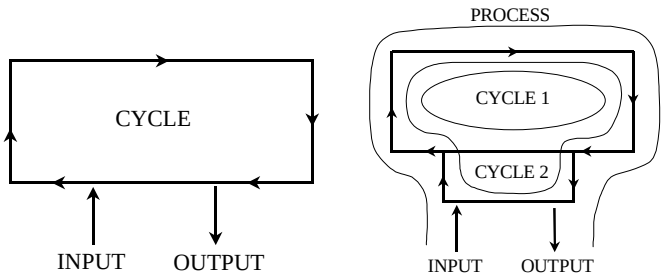
control automático de sistemas concurrentes de eventos discretos. El modelado y la automatización de esta clase de sistemas desarrollados usando RdP posibilitan el análisis del sistema desde varios puntos de vista (elementos cíclicos, áreas de la planta, producción de material) proporcionando altas prestaciones al control automático. Este apartado describe el análisis del comportamiento de las RdP y la implementación en el modelado y control automático de aquellos sistemas que incluyen un flujo concurrente con subsistemas cíclicos.

6.2.1 Introducción

En el modelado de plantas industriales es usual encontrar sistemas y relaciones entre ellos, muy similar a otros que ya se han diseñado. Por tanto, la experiencia en las relaciones del modelado entre los subsistemas diferentes puede ser muy importante cuando se modelan grandes sistemas formados por muchos subsistemas no excesivamente complicados. Uno de estos sistemas podría ser el compuesto de varios subsistemas cíclicos relacionados con cada otro por relaciones autorizadas, sincronizaciones, inclusiones y concurrencias, cuando el principal material de producción pasa sucesivamente a través de cada ciclo para seguir una cierta ruta. En esos sistemas el modelado del proceso industrial, y por consiguiente el control automático, pueden ser considerados desde varios puntos de vista: desde los ciclos individuales, desde las diversas áreas de la planta a lo largo de las cuales pasan los ciclos, o incluso desde el propio material de producción (piezas). Los tres puntos de vista llevan al mismo resultado, puesto que la automatización es la misma, pero algunos resultados pueden ser analizados más apropiadamente dependiendo de la solución escogida. Cuando se desarrollan el modelado y el análisis de la planta por medio de RdP [7], dependiendo del punto de vista, los lugares y las señales tendrán significados diferentes. Todo esto se podrá comprobar mediante un proceso industrial, muy completo e interesante que incluye muchos subsistemas en varios niveles y con procesos diferentes, pero que en esencia tienen mucho en común.

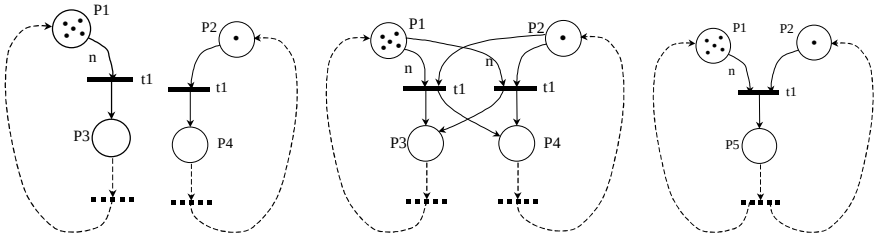
6.2.2 Sistemas que Incluyen Subsistemas Cíclicos. Marcas e Interpretación de Valores

Como ya se ha comentado, un tipo muy usual de sistemas especiales lo forman los aquí compuestos por un flujo, bien de una sustancia o bien de piezas (continuo o discreto), a través de los diferentes subsistemas cíclicos del sistema. Esto se esboza en la Figura 1 (Figura 1a con un subsistema simple, y Figura 1b con un ciclo embebido en otro). Esto es usual en sistemas donde el material a procesar se introduce en contenedores para seguir el proceso, e incluso, esos contenedores pueden englobarse en otros para llevar a cabo un proceso diferente. El modelado de estos sistemas puede desarrollarse de una manera muy eficaz por medio de RdP, pero es necesario tener presente algunas consideraciones.



Figuras 1a y 1b. Diagrama de flujo de material (o piezas) a través de ciclos y de los subciclos que incluyen.

Supongamos un sistema simple donde un contenedor debe llenarse de n piezas para seguir un proceso. El ciclo del contenedor y el flujo de las piezas son considerados en la Figura 2a que muestra ambos ciclos. Hay una clara sincronización: mientras no hay ningún contenedor ni n elementos, ni p_1 ni p_2 evolucionarán. Esto se muestra en Figura 2b donde la RdP modela el sincronismo de ambos ciclos. Esta RdP puede sustituirse por la de la Figura 2c que es similar, cuando p3 y p4 han sido combinadas en p5. Sin embargo, en Figura 2c podría ser interesante prestar atención al significado de las marcas. Cada marca en el p1 representa un elemento, cada marca en el p2 representa un contenedor, y cada marca en el p3 representa un contenedor cargado con n elementos. Así, podría interpretarse que las Figuras 2a y 2b representan cada subsistema, y la Figura 2c el sistema entero.



Figuras 2 a, b, y c. Relaciones entre los ciclos, o entre el ciclo y el flujo

Este modelo del sistema entero es correcto, y puede implementarse en un dispositivo de la manera mencionada (como se muestra en la Figura 2c). Pero normalmente se requiere saber en tiempo real qué contenedor está en cada área del proceso, o tener un conocimiento histórico del contenedor dónde cada elemento se ha producido (tracking). En este caso el análisis es realizado desde el punto de vista de las piezas (o el material). Entonces esto se debe desarrollar a través de alguna función adicional cuando se implemente la RdP. Esto puede ser modelado con RdP coloreadas (CRdP), dónde cada marca (o grupo de marcas) tiene un color que la diferencia del resto de las marcas (o de los grupos). El problema es implementar la CRdP en el dispositivo de control (por ejemplo PLCs [2,4] o PCs[1]).

Si el problema se resuelve por medio de un programa desarrollado con un lenguaje de programación, una buena solución puede ser implementar las marcas que corresponden a los contenedores como registros, con un campo que es un vector de n espacios de tipo "elemento". Como esos contenedores puede englobarse en otros más grandes (Figura 3),

estos últimos podrían ser registros de m registros, y podría ser posible proceder de la misma manera a más niveles.

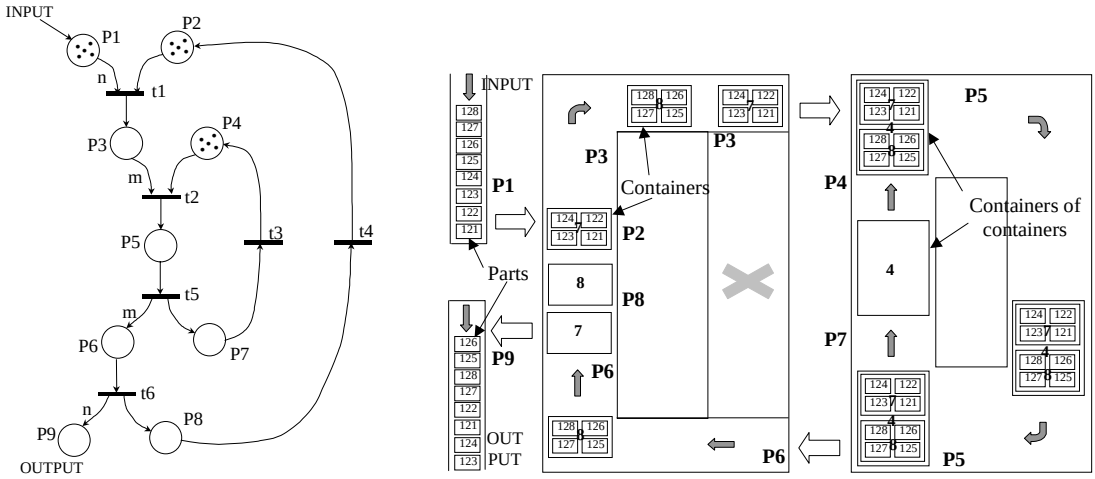


Fig. 3. Flujo a través de los ciclos a varios niveles. RdP e interpretación en un proceso industrial.

Cuando la aplicación se lleva a cabo con PLCs, que normalmente no incluyen registros, puede realizarse por medio de tablas: se asigna una tabla con n valores a cada elemento de tipo contenedor. Igualmente, se asigna una tabla con m valores a cada contenedor de contenedores tipo elemento. Finalmente, es necesario guardar una tabla asociada a los elementos que fluyen a través del ciclo con la información sobre los contenedores y los contenedores de contenedores en los que las piezas se han guardado durante su paso a lo largo del proceso. Esta última tabla debe tener tantas casillas como el número de procesos entrelazados que siguen las piezas.

6.2.3 Aplicaciones complementarias de las herramientas gráficas

En el caso anterior, bien sea desarrollado por medio de PC o por PLC, es necesario agregar algunas aplicaciones complementarias a la RdP para rellenar correctamente las tablas (o los registros). En el disparo de cada transición correspondiente a una " convergencia-y" de piezas y contenedores (t1 en la Figura 3), debería asociarse una aplicación a la transición. Esta aplicación tiene que rellenar la tabla que corresponde al recipiente que participa en la convergencia, Tabla 2, así como llenar las casillas de la tabla que corresponde a las piezas con el número del contenedor en que han estado contenidas (Tabla 1). Esta última tabla constituirá la tabla de datos históricos, y es la única que debe conservarse (no debe borrarse cíclicamente). Las otras deben borrarse al dispararse la correspondiente " divergencias-y" (t6 en la Figura 3), y volver a escribirse en el siguiente ciclo. La convergencia/divergencia entre los contenedores y contenedores de contenedores son similares: la aplicación debe rellenar la tabla de las piezas (Tabla 1) y la tabla de los contenedores de contenedores (Tabla 3) en el disparo de t2 (convergencia), y en el disparo de t5 (divergencia).

Y finalmente se necesita una tabla con la información de qué elementos están en cada área, (Tabla 4) el tercer tipo de tabla. De esta manera, si el contenedor en cada área del proceso (red coloreada) es conocido, así como la tabla correspondiente a dicho

contenedor, entonces también son conocidas las piezas que hay en cada área. Cada disparo de transición debe modificar estas Tablas de Áreas.

Así, todas estas tablas representan los tres puntos de vista de la información:
- Las áreas en la planta: Las Tablas de Areas
- Los elementos cíclicos: Tabla de Contenedores y Tabla de Contenedores de Contenedores
- Material de producción o piezas: La Tabla de las piezas

También es necesario tener presente que los valores no pueden ser implementados como simples variables enteras, puesto que se trata de una red coloreada [6,7] (los elementos son diferentes). Es necesario implementar estos valores como una tabla para registrar la posición exacta de los elementos (o los contenedores o los contenedores de contenedores). Para ello, cada elemento (la pieza, el contenedor, etc.) tiene un número que lo identifica (Tablas 1-4).

Y estas diferentes aplicaciones asociadas a los disparos de la transición rellenarán las tablas dependiendo de los datos de las tablas anteriores, pero también del tipo de carga/descarga. Por ejemplo, si un grupo de piezas son movidas de un contenedor a otro, el orden final será diferente dependiendo de la forma en que se han movido (de una en una, de dos en dos, en grupos, etc.), y además debe ser tenida en cuenta la naturaleza de los apilados en las estaciones (FIFO, LIFO, el etc).

Parts	Container	Container of containers
.....	.....	.....
121	7	4
122	7	4
123	7	4
124	7	4
125	8	4
126	8	4
127	8	4
128	8	4
.....	.....	.....

Tabla 1: Tabla de piezas (histórico de datos).

Recip.	part 1	part 2	part 3	part 4=n
1	....	....	....	....
2	....	....	....	....
3	....	....	....	....
4	....	....	....	....
5	....	....	....	....
6	....	....	....	....
7	121	122	123	124
8	125	126	127	128

Tabla 2: Tabla de contenedores.

Contain. of cont.	cont. 1	cont. 2=m
1	....	....
2	....	....
3	....	....
4	7	8

Tabla 3: Tabla de conten. de conten.

Area	Parts	Containers	Containers of containers
P1	129,130,131	———	———
P2	———	1,2,3,4,5,6	———
P3		———	———
P4	———	———	1,2,3
P5	———	———	1
P6	———		———
P7	———	———	
P8	———		———
P9		———	———

Tabla 4: Tabla de áreas

Figura4:
Tablas del Proceso

6.3. Descripción de la planta

Éste es un ejemplo del análisis llevado a cabo con la implementación de RdP para este tipo de procesos [3], que contienen un flujo de piezas a través de subsistemas recurrentes. La Figura 4 representa un sistema con un flujo de bloques sobre bandejas que se introducen en los vagones para llevar a cabo el primer proceso. Después, los elementos se amontonan para introducirlos directamente en otro tipo de contenedor y realizar otro proceso. Todo esto, junto con los recursos comunes, sincronizaciones, retardos, concurrencias, etc., dan la RdP que modela y controla el sistema entero.

6.3.1. Descripción del proceso industrial

El proceso industrial del ejemplo tiene todas las características previamente mencionadas. El proceso puede parecer simple, puesto que está compuesto por varios ciclos que individualmente no tienen excesiva dificultad, aunque la automatización es extremadamente compleja, ya que es necesario coordinar cada uno de esto ciclos para conseguir un funcionamiento correcto. Y todavía puede ser más difícil optimizarlo para las condiciones de trabajo del sistema. Éste es un ejemplo de proceso donde el aspecto más interesante no es el de las piezas producidas (los ladrillos cerámicos), ni las operaciones que rodean al material, sino los procesos y las relaciones entre ellos [5]. Aun así, se va a dar una breve descripción del proceso físico industrial, para tener en todo momento la referencia de los que se está tratando.

El sistema sobre el que se ha aplicado la automatización es una moderna planta de producción de elementos cerámicos para la construcción: ladrillo, ladri-yeso, bloques de termoarcilla, etc. La planta está totalmente automatizada, por lo que es evidente que se requerirá de diversos tipos y modelos de controles PID a lo largo de toda ella, pero son controles que en general no precisan de grandes requerimientos dinámicos, como veremos enseguida, pero sí es muy recomendable su seguridad y fiabilidad.

El sistema productivo a grandes rasgos consiste en el amasado de la arcilla, la conformación en elementos cerámicos en la galletera, el secado de dichos elementos en el secadero y la posterior cocción en el horno. Todo ello con los consiguientes procesos de apilado y desapilado, carga y descarga en las vagonetas, empaquetado, etc.

En el secadero los elementos de barro recién conformados se introducen para que pierdan parte del agua que contienen. A lo largo del recorrido por el secadero deben transcurrir por zonas con ciertos parámetros de temperatura, humedad y velocidad del aire, variables a lo largo del recorrido, siguiendo una curva predefinida. El calor y la humedad se consiguen introduciendo de forma forzada aire calentado como consecuencia del calor residual que se produce en los gases de escape y en el agua de refrigeración de sistemas de cogeneración -en nuestro caso de gas- que producen energía eléctrica a partir del combustible y recuperan el calor residual para este secado.

En el horno los productos ya secados se cuecen para la formación del ladrillo o elemento cerámico. El calor necesario se consigue quemando gas en el interior del horno refractario, y haciendo circular los gases por entre los ladrillos. Igualmente el

proceso a lo largo del horno debe mantener una curva de temperatura a lo largo del recorrido. El recorrido se divide en tres zonas: zona de precalentamiento, zona de cocción y zona de enfriamiento rápido.

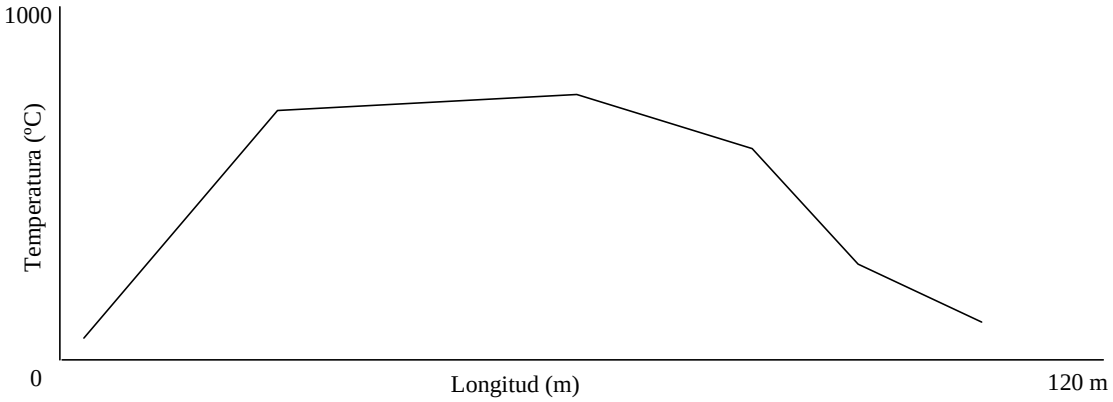


Figura 5: Típica Curva de Temperatura de Cocción en Horno

Estas dos automatizaciones son las principales de la planta, y tienen la especial característica de contar con unos parámetros (temperatura, humedad y velocidad del aire) que varían a lo largo del recorrido pero que deben permanecer constantes (al menos en cada ciclo) en cada punto del recorrido, siguiendo la consigna de las curvas de funcionamiento preestablecidas. Estas curvas de funcionamiento son de especial importancia, puesto que una desviación en alguno de los puntos podría ocasionar la fractura del material. Además son curvas en cuyo control no es necesario una excesiva velocidad (no es importante optimizar el tiempo de respuesta), pero sin embargo sí es importante que no existan sobreoscilaciones en ciertos puntos y mucho más que se siga fielmente la consigna (es decir, que el error sea cero).

Una sobreoscilación en la temperatura de la zona de cocción podría llevarnos a fundir el material. En la zona de enfriamiento se debe tener especial cuidado en realizar una transición suave por la temperatura de 573°C , ya que es la zona de transición entre la cristalización α y la cristalización β, por lo que en esa zona lo importante es anular totalmente el error. En el secadero, un mal seguimiento de las consignas durante el primer 5% de perdida, que es cuando se produce la contracción, ocasionaría el agrietamiento del material o su rotura.

Por lo tanto comprobamos que en los controles de esta parte de la planta la principal condición es el fiel seguimiento de la consigna, intentando eliminar el error. Además el quemado en el horno se realiza a través de inyectores situados a unos 1,5 m de distancia durante más de 40 m. y en cada una de los lugares de inyección se dispone de sensores de todos los parámetros requeridos, y en todas las direcciones. Por lo tanto cuando hablamos de una curva de temperatura o de humedad no estamos hablando de un sólo control sino de un control para cada punto de inyección.

Dichas curvas son constantes a lo largo de cada ciclo de producción, pero se deben modificar en función de la carga (la masa que pase por unidad de tiempo), el producto (el tipo de elemento cerámico que se esté fabricando en ese momento) y de la materia prima (la calidad y tipo de la arcilla empleada). Por tanto se debe memorizar las curvas

de referencia en función de dichas características, así como su comportamiento. Se hace evidente la necesidad de contar con control adaptativo, y de disponer de sistemas de almacenamiento de la gran cantidad de parámetros que componen las curvas de consigna.

Pero existen muchos otros puntos, además de los del secadero y el horno, que precisan de controles con distintas características (menores tiempos de respuesta, sobreamortiguados, etc.), por lo que nuestro sistema de control debe ser capaz de adaptarse a diversos modelos y configuraciones.

6.3.2. Descripción previa del sistema global y de los ciclos

La descripción detallada del proceso productivo puede analizarse en un primer acercamiento al modelado según el dibujo de la Figura 6, donde la ruta de paso del material a su través se ha representado por flechas, desde el momento que entra en la planta hasta que lo deja como producto final. Este proceso puede resumirse como sigue:

En el paso 1, el material llega como continuo y, tras el proceso de la extrusión y troceado, sale como una secuencia de piezas. Éstas se mueven por una cinta transportadora (2), y un robot (3) los deposita de forma ordenada en bandejas. Cuando una bandeja está completa, otra cinta transportadora (4) la transporta a un puente grúa que la deposita (5) en un carro de bandejas. Cuando este el vagón de bandejas está completo, se mueve (6) hasta el primer proceso, dónde lleva a cabo una ruta (7,8,9) a través de un área (para ser preciso, esta área es un secadero, pero este hecho no es relevante), donde debe permanecer durante un tiempo dado. Cuando el carro de las bandejas ha terminado el proceso, es llevado (10) a un área de descarga donde otro puente grúa (11) descarga las bandejas desde el carro de bandejas. Estas bandejas avanzan (12) por otro transbordador. Un robot (13) apila las piezas desde las bandejas antes de que cada bandeja vacía siga su ruta cíclica. Las piezas que han sido apiladas se cargan de nuevo por otro robot (14) en otro carro para piezas. Cuando el carro está lleno, avanza (15) hasta una plataforma móvil que lo transporta (16) perpendicularmente a su ruta de avance. Una vez transportado, sigue una ruta (17) a través del área del segundo proceso (en este caso, un horno) durante un tiempo fijo, y cuando ha terminado, otra plataforma móvil lo transporta de nuevo (18). Entonces los carros de piezas avanzan (19) a un área de descarga donde un robot (20) desapila las piezas individuales y otro robot (21) las coloca sobre palets, que son llevados (22) al almacén. Así, la arcilla se ha transformado en paquetes de ladrillos cerámicos [3].

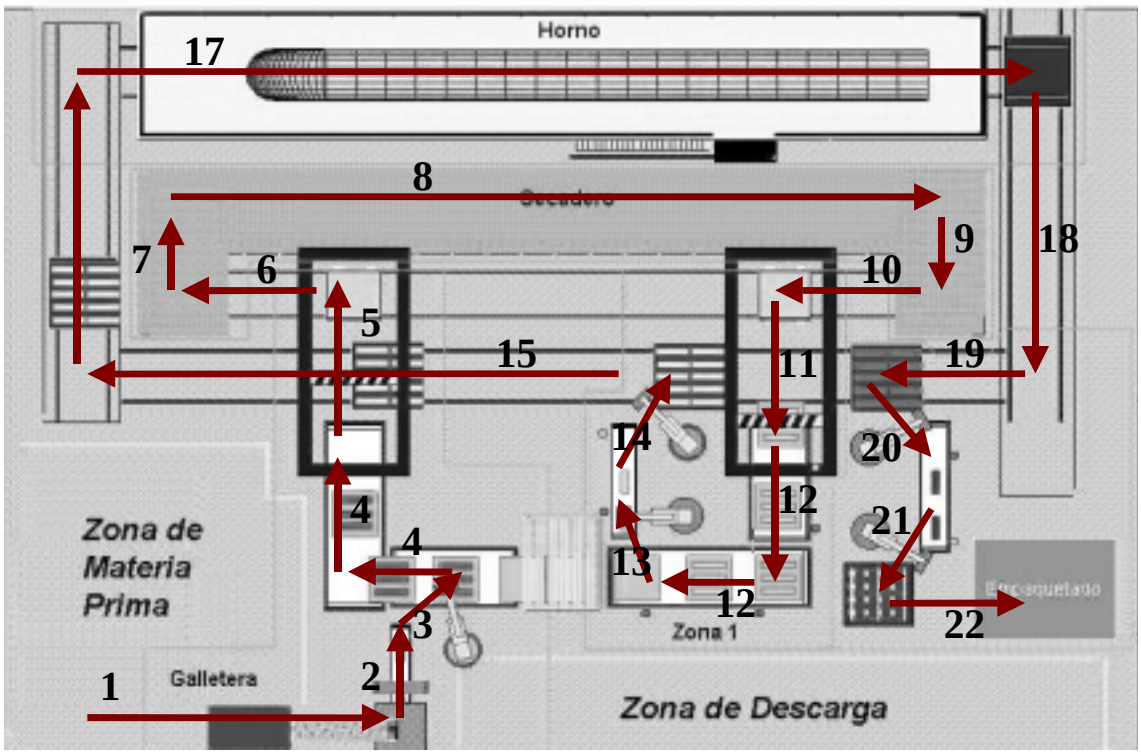


Figura 6: Proceso productivo e indicación del recorrido del material.

6.3.3. Ciclos individuales

Este proceso que acabamos de estudiar está basado en muchos otros ciclos que pueden analizarse perfectamente en el esquema del proceso. Hay diez ciclos singulares simples en el proceso. Los ciclos 1 a 7 pertenecen a robots, que simplemente cogen un objeto, lo ponen en otro lugar, y repiten el proceso de nuevo. Su mando es muy simple, como puede verse en la Figura 7.

Lógicamente no es necesario aquí prestar atención a los detalles del funcionamiento, como la separación de los robots después de dejar las piezas, la forma de evitar colisiones si las piezas depositadas se ponen en el movimiento, u otros detalles que se consideran dentro "lugares genéricos". Más adelante se trata todo ello en profundidad. También es evidente que los programas responsables de "coger" y "dejar" tendrán presente el tipo del producto con el que el sistema trabaja en ese momento, la cantidad de piezas que ya se han apilado o desapilado, etc.

En esta RdP se ha seguido una política de avanzar el trabajo siempre que sea posible. Es decir, si el robot puede coger una pieza, aunque no haya ningún hueco para depositarla todavía, la coge (adelantando el trabajo) y espera hasta que haya hueco. Esta política es válida para sistemas dónde cada dispositivo sólo puede llevar a cabo una tarea (robot 1, la tarea 1, robot 2 la tarea 2, etc.). Pero esto no es posible cuando un dispositivo puede llevar a cabo más de una tarea dependiendo de los eventos.

Por tanto en esta planta puede ser más apropiado que los robots 2, 3, 4 y 5 también puedan llevar a cabo las tareas de cualquiera de los dos robots adyacentes. Así pueden obtenerse dos resultados: el sistema puede continuar incluso trabajando con dos robots

dañados, y si hay un cuello de botella en cualquiera de los cuatro procesos, los robots vecinos pueden ayudarle. Para conseguir esto, las cuatro RdP que corresponden al ciclo del robot (Figura 7) se relacionan entre sí como se muestra en la Figura 8 (sólo es mostrado el cuarto robot porque el proceso es completamente simétrico).

Es lógico dar prioridad a las transiciones "T_{ab}, con a=b" puesto que cada robot es el más cercano a su proceso. Con respecto a las otras transiciones, "T_{ab} con a=b", pueden aplicarse varias políticas: la igualdad de prioridades, prioridad a un lado (derecho o izquierdo), o prioridad dependiente del estado del proceso en ese momento. Precisamente, se han obtenido muy buenos resultados en el sistema real cuando la prioridad se maneja por medio de un sistema adaptativo basado en entrenamiento a través de redes neuronales.

El resto de ciclos simples son las plataformas móviles (8 y 9 en Fig. 10) que se encargan del transporte de los carros de piezas desde una vía a la otra, y el sistema para transportar los paquetes finales al almacén (10 en Fig. 10). Su funcionamiento se muestra en la Figura 9.

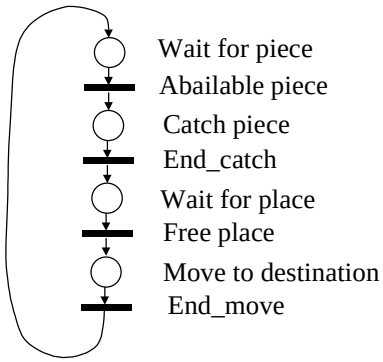


Figura 7: Funcionamiento de las RdP de los robots.

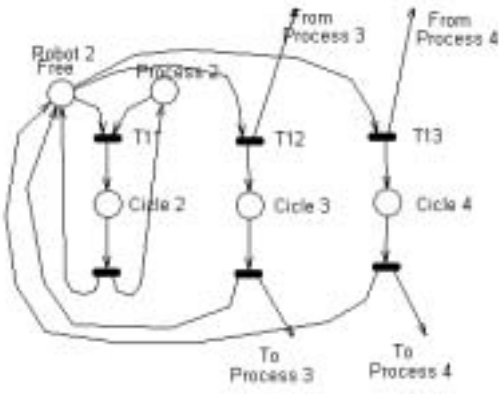


Fig. 8. Control coordinado de los cuatro robots
T_{ab} = Transición "Robot a en proceso b"

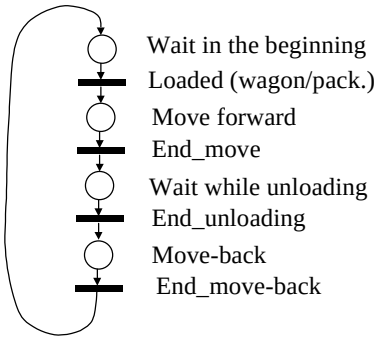


Fig. 9 Ciclo de los transbordadores.

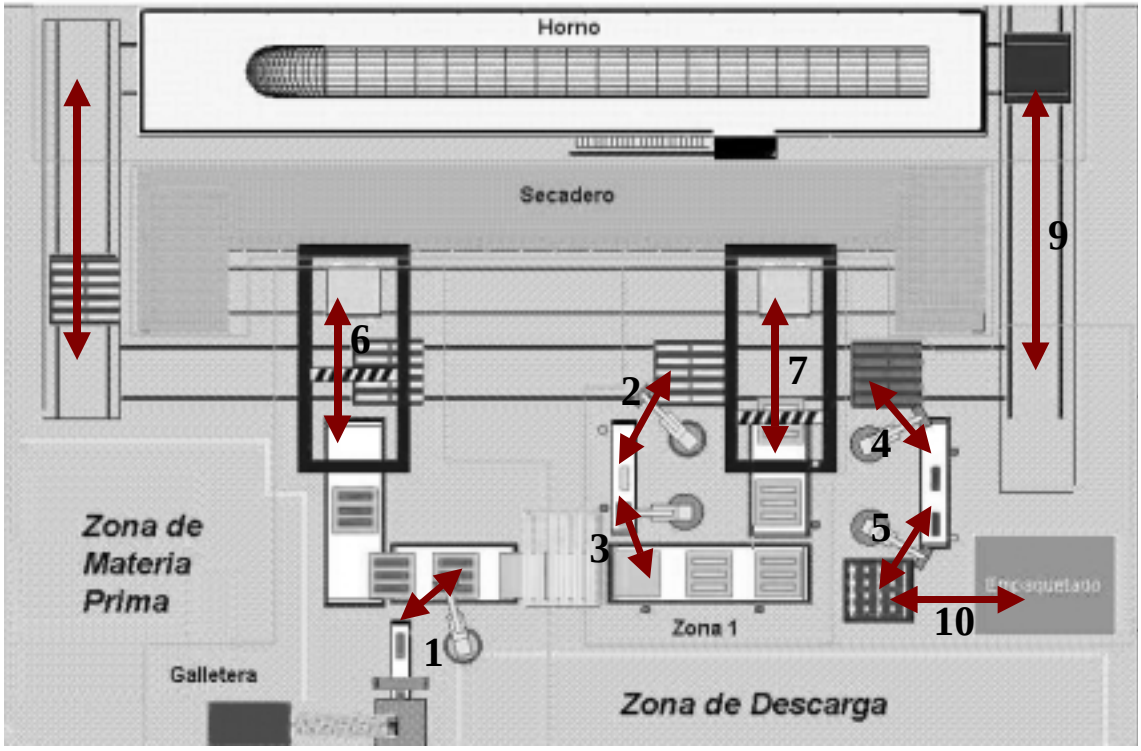


Figura 10: Representación de los 10 ciclos individuales

6.3.4. Ciclos Complejos

Además de los diez ciclos simples, hay otros tres ciclos más complejos: el de los carros de las bandejas, el de los carros de las piezas, y el de las bandejas. Las RdP de estos tres ciclos pueden interpretarse como la operación de uno de los elementos (carros de las piezas, carros de las bandejas, o bandejas), si tenemos una RdP binaria. Pero pueden interpretarse como el funcionamiento global de ese tipo de elementos si es incluido un token para representar cada carro. De tal manera, si queremos tener un conocimiento preciso del elemento concreto que hay en cada área, es necesario usar una RdP coloreada con tantos colores como elementos. De esta manera evitamos usar una RdP individual para cada carro.

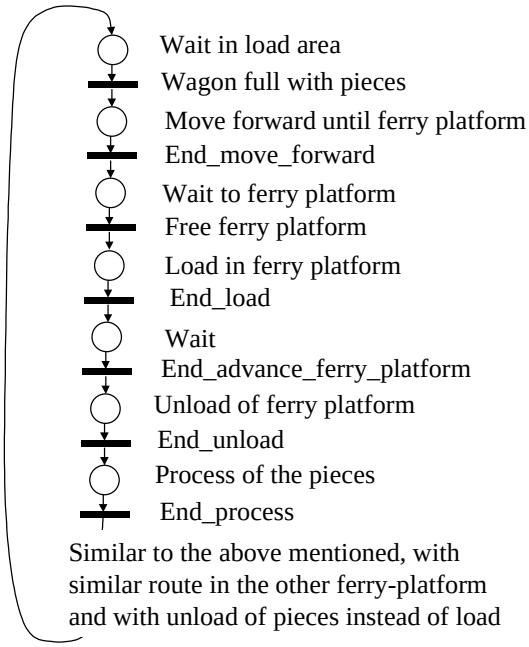


Fig. 11. Ciclo de trabajo de los vagones de piezas.

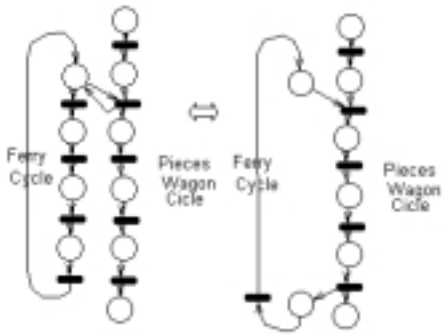


Fig. 12. Coordinación de los ciclos de transbordador y de vagón de piezas.

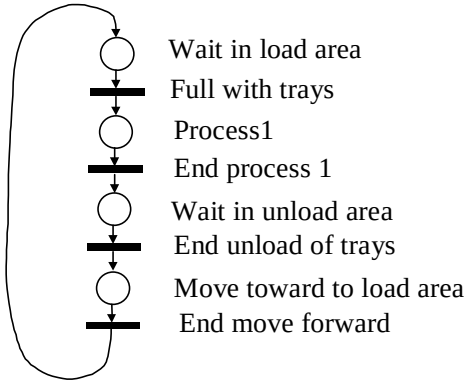


Fig. 13. Ciclo de los vagones de bandejas.

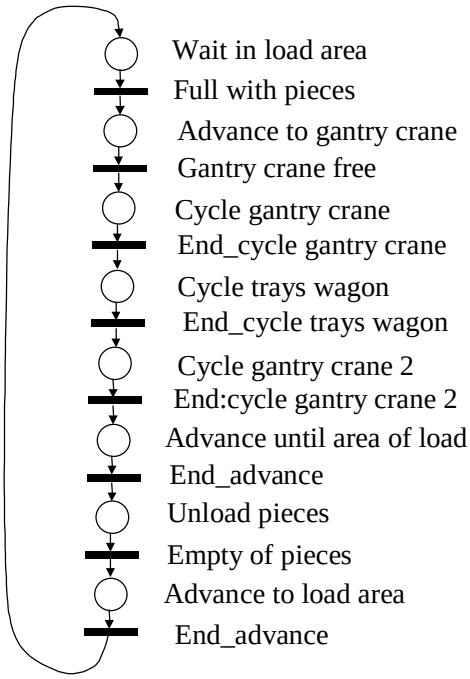


Fig. 14. Ciclo de las bandejas

En estos ciclos de trabajo, pueden usarse varias políticas. Por ejemplo, si en el ciclo de los carros de las bandejas hay más carros que los justos en el proceso más uno, entonces en el área correspondiente habrá un carro esperando, a modo de buffer (también conocido como pulmón), y la RdP será ligeramente diferente. Hay también buffers para las bandejas y para los carros de las piezas.

El ciclo de trabajo de los carros de las piezas consiste en una sucesión compuesta de 1, 2, 3, 4 y 5 en la Figura 15, y su RdP se muestra en Figura 11. Con el ciclo de los carros de las bandejas conseguimos los mismos resultados que en el ciclo anterior (6, 7, 8, 9, 10 y 11 en Figura 15 y RdP en Figura 13). También es necesario coordinarlo con otros ciclos; en este caso, con el ciclo de las bandejas. Y el ciclo de las bandejas (1 a 11 en Figura 16) también debe ser modelado por medio de una RdP coloreada (Figura 14) si queremos saber qué bandeja está en un lugar dado o en un carro de bandejas concreto.

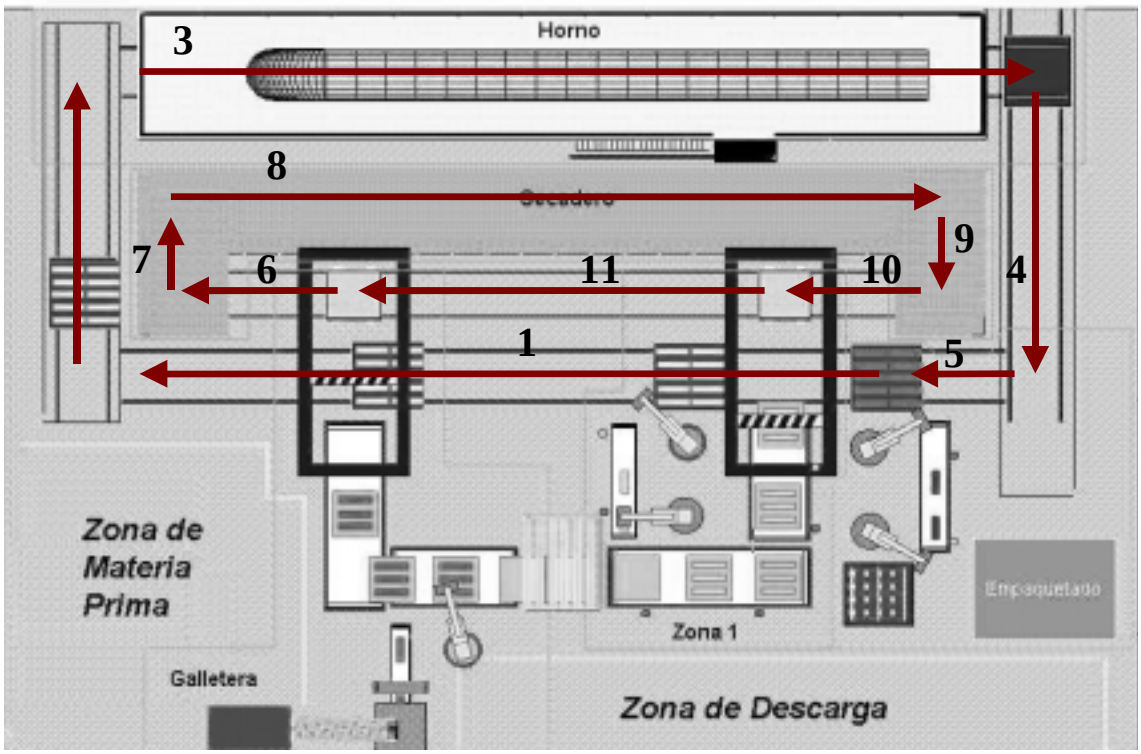


Figura 15: Ciclos complejos de vagones de bandejas y de vagones de piezas.

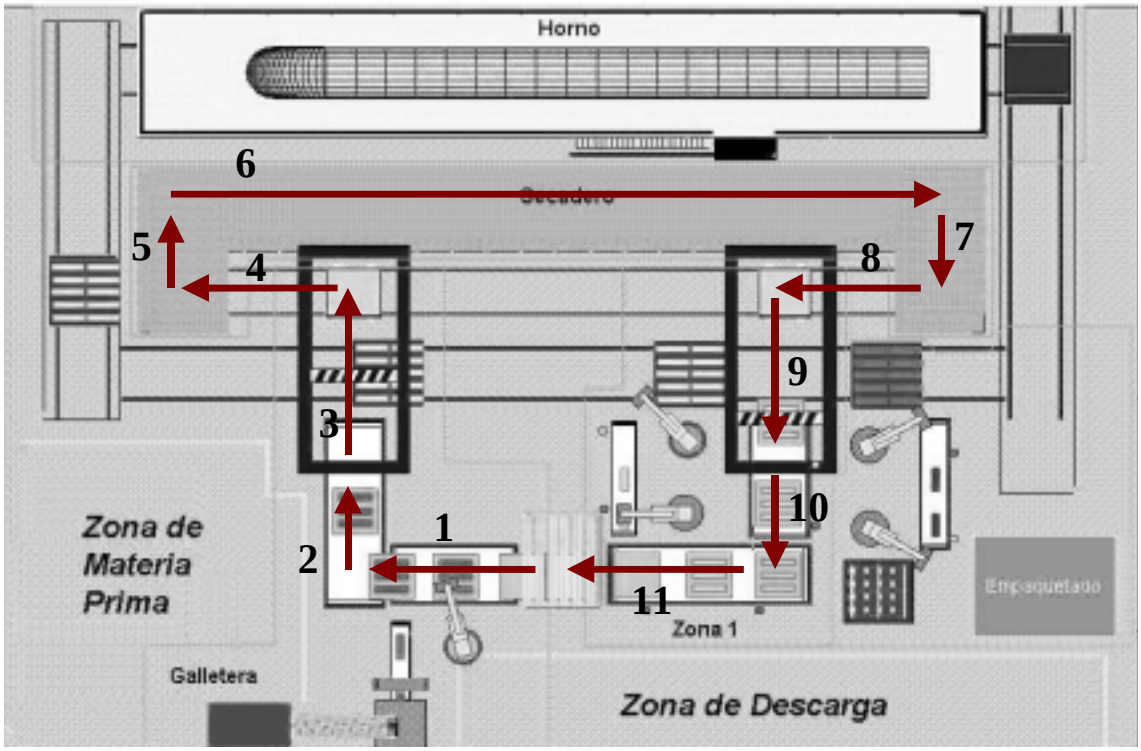


Figura 16: Ciclo complejo de bandejas.

6.3.5 Coordinación de los Ciclos

Una vez se han desarrollado todos los ciclos individuales, la conexión de todos ellos consiste en analizar las piezas que tienen en común: el ciclo de carros de las bandejas con el ciclo de bandejas, el ciclo de las bandejas con los ciclos del puente grúa y los de los robots 4 y 5, etc. De esta manera resulta que varios ciclos en la RdP global pueden reducirse prácticamente a los recursos comunes (Fig 12), porque el resto de sus lugares está implícitamente incluido en otros ciclos.

Muchas de las relaciones entre los ciclos diferentes son una conversión de elementos: m piezas entran en una bandeja, *n* bandejas entran en un carro de las bandejas, o *r* piezas entran en un paquete (palet) de producto final (Figs 17 y 18).

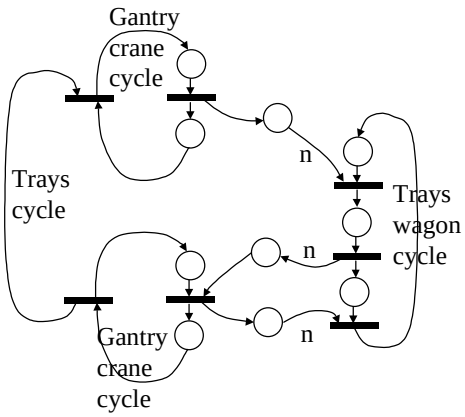


Fig. 17. Conversión de bandejas a carros de bandejas

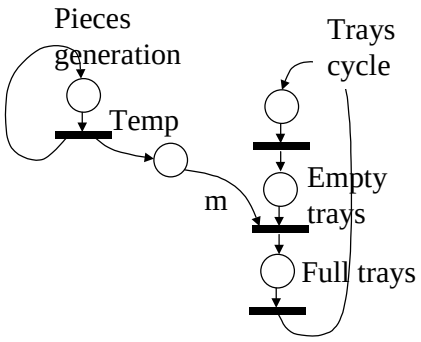


Fig. 18. Conversión de piezas a bandejas.

Con esta forma de representación global del sistema, la automatización opera como si cada ciclo trabajara independientemente. Pero puede perderse información interna sobre los elementos que son transformados en otros (las piezas en bandejas, en carros de piezas y en los paquetes del producto final; las bandejas en carros de las bandejas y en piezas, etc.). Si es necesario guardar toda la información sobre dónde ha estado cada elemento (la bandeja y los carros), hay dos maneras de hacerlo:

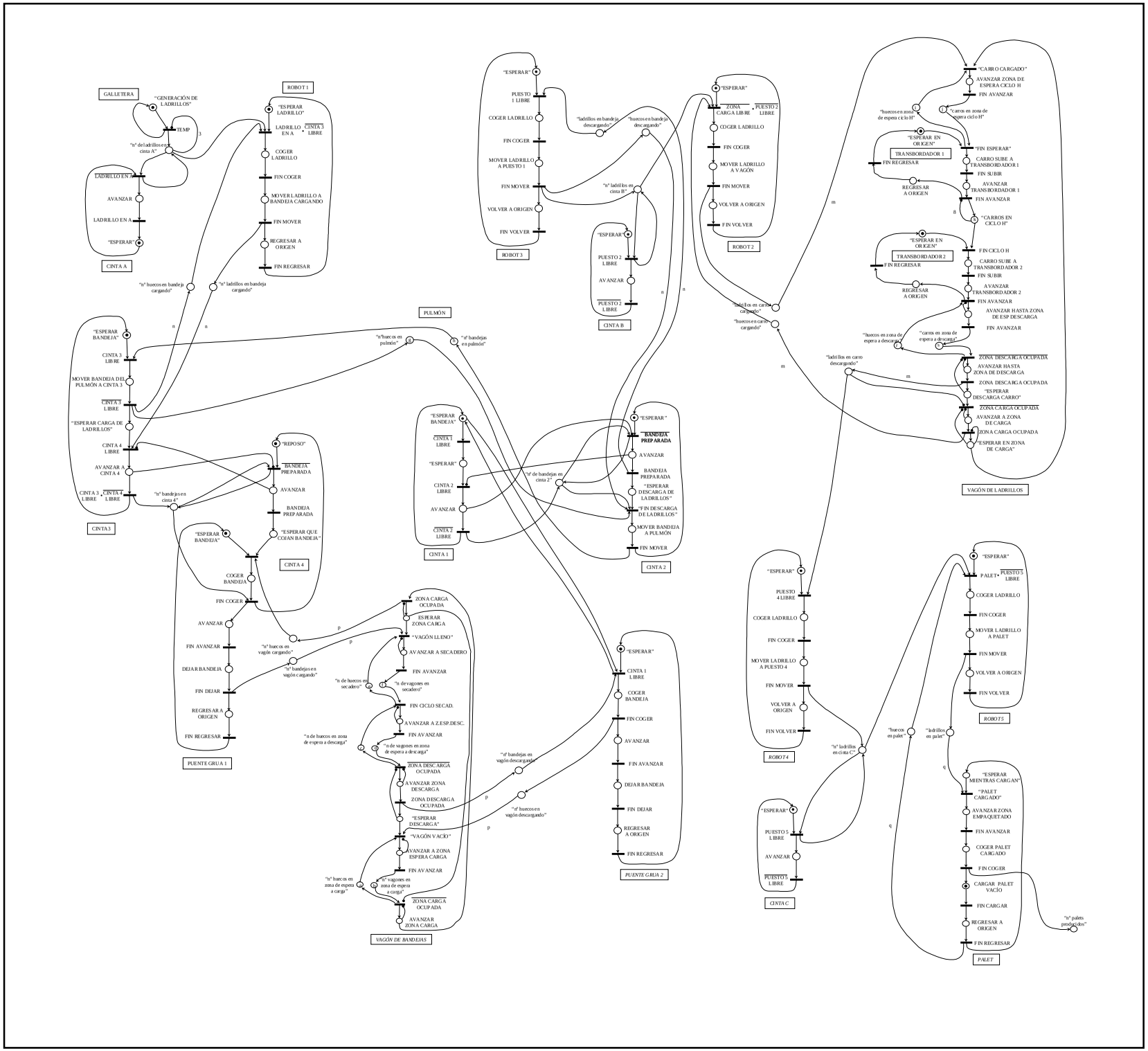
- Usando la RdP global del sistema para el funcionamiento de la planta, y otro sistema (un SCADA) para representar todos los ciclos y guardar su información.

- Obteniendo esta información directamente de la RdP global durante su evolución, pero usando como tokens en la implementación de la RdP un tipo de datos en forma de registro coloreado.

Este registro coloreado simplemente consiste en un dato similar a un token coloreado pero conteniendo la información sobre el subtokens de que está compuesto.

Uniendo todos los subsistemas anteriores, se obtiene la RdP entera para controlar el sistema y sus elementos, que es la que se muestra en la Figura 19 de la página siguiente.

(Figura 19: Coordinación de los subsistemas que componen la automatización) ⇒

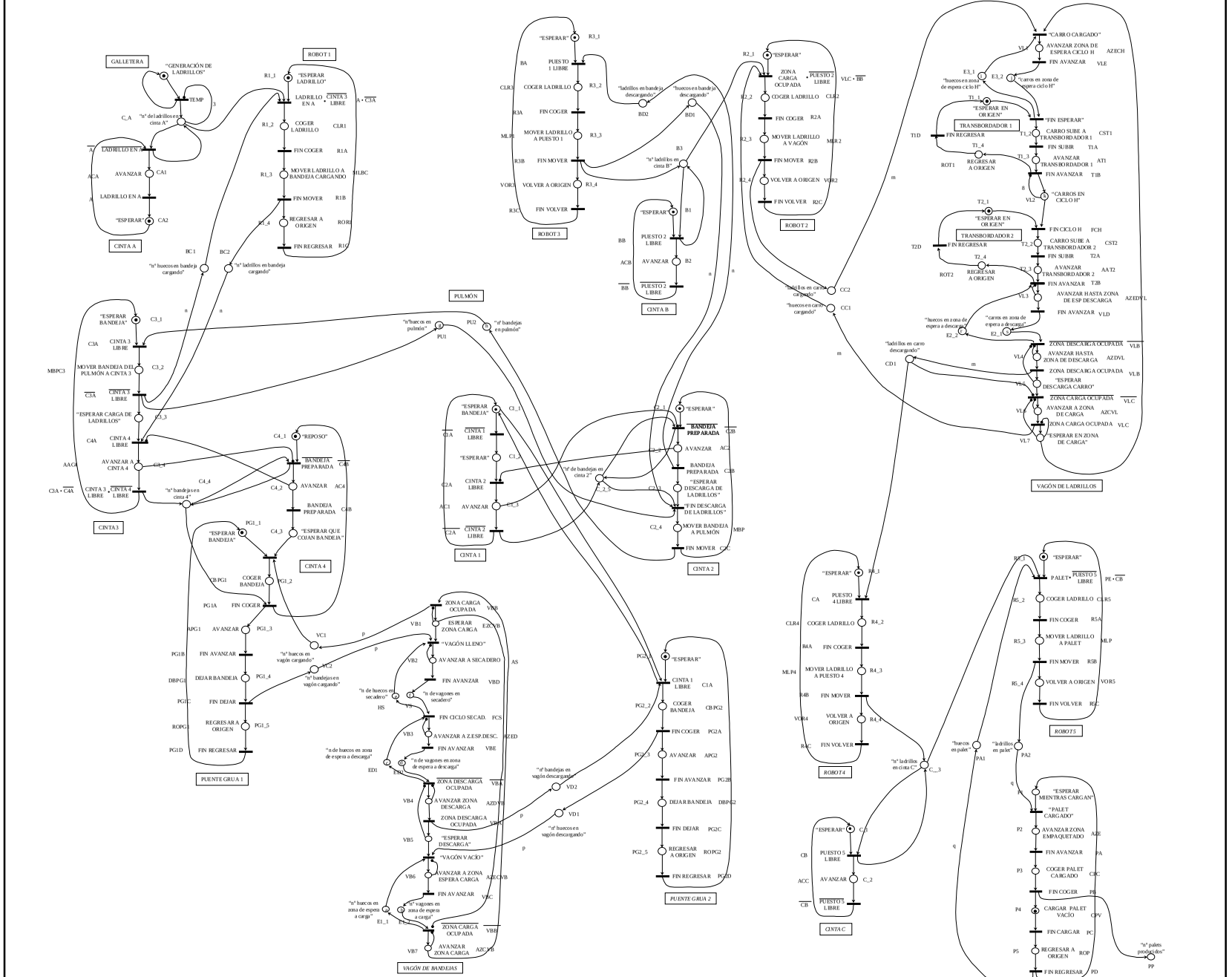

GALLETERA
"GENERACIÓN DE LADRILLOS"
TEMP
3
"n° de ladrillos en cinta A"
LADRILLO EN A
AVANZAR
LADRILLO EN A
"ESPERAR"
CINTA A
ROBOT 1
"ESPERAR LADRILLO"
LADRILLO EN A
CINTA 3 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A BANDEJA CARGANDO
FIN MOVER
REGRESAR A ORIGEN
FIN REGRESAR
"n° huecos en bandeja cargando"
"n° ladrillos en bandeja cargando"
"ESPERAR"
PUESTO 1 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PUESTO 1
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 3
"ladrillos en bandeja descargando"
"huecos en bandeja descargando"
"n° ladrillos en cinta B"
"ESPERAR"
PUESTO 2 LIBRE
AVANZAR
PUESTO 2 LIBRE
CINTA B
"ESPERAR"
ZONA CARGA LIBRE
PUESTO 2 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A VAGÓN
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 2
"ladrillos en carro cargando"
"huecos en carro cargando"
"CARRO CARGADO"
AVANZAR ZONA DE ESPERA CICLO H
FIN AVANZAR
"huecos en zona de espera ciclo H"
"carros en zona de espera ciclo H"
"ESPERAR EN ORIGEN"
TRANSBORDADOR 1
FIN REGRESAR
REGRESAR A ORIGEN
"FIN ESPERAR"
CARRO SUBE A TRANSBORDADOR 1
FIN SUBIR
AVANZAR TRANSBORDADOR 1
FIN AVANZAR
"CARROS EN CICLO H"
"ESPERAR EN ORIGEN"
TRANSBORDADOR 2
FIN REGRESAR
REGRESAR A ORIGEN
FIN CICLO H
CARRO SUBE A TRANSBORDADOR 2
FIN SUBIR
AVANZAR TRANSBORDADOR 2
FIN AVANZAR
AVANZAR HASTA ZONA DE ESP DESCARGA
FIN AVANZAR
"huecos en zona de espera a descarga"
"carros en zona de espera a descarga"
ZONA DESCARGA OCUPADA
AVANZAR HASTA ZONA DE DESCARGA
ZONA DESCARGA OCUPADA
"ESPERAR DESCARGA CARRO"
ZONA CARGA OCUPADA
AVANZAR A ZONA DE CARGA
ZONA CARGA OCUPADA
"ESPERAR EN ZONA DE CARGA"
VAGÓN DE LADRILLOS
"ladrillos en carro descargando"
PULMÓN
"n°huecos en pulmón"
"n° bandejas en pulmón"
"ESPERAR BANDEJA"
CINTA 3 LIBRE
MOVER BANDEJA DEL PULMÓN A CINTA 3
CINTA 3 LIBRE
"ESPERAR CARGA DE LADRILLOS"
CINTA 4 LIBRE
AVANZAR A CINTA 4
CINTA 3 LIBRE
CINTA 4 LIBRE
"n° bandejas en cinta 4"
CINTA3
"REPOSO"
BANDEJA PREPARADA
AVANZAR
BANDEJA PREPARADA
"ESPERAR QUE COJAN BANDEJA"
CINTA 4
"ESPERAR BANDEJA"
COGER BANDEJA
FIN COGER
AVANZAR
FIN AVANZAR
DEJAR BANDEJA
FIN DEJAR
REGRESAR A ORIGEN
FIN REGRESAR
PUENTE GRUA 1
"ESPERAR BANDEJA"
CINTA 1 LIBRE
"ESPERAR"
CINTA 2 LIBRE
AVANZAR
CINTA 2 LIBRE
CINTA 1
"n° de bandejas en cinta 2"
"ESPERAR"
BANDEJA PREPARADA
AVANZAR
BANDEJA PREPARADA
"ESPERAR DESCARGA DE LADRILLOS"
"FIN DESCARGA DE LADRILLOS"
MOVER BANDEJA A PULMÓN
FIN MOVER
CINTA 2
ZONA CARGA OCUPADA
ESPERAR ZONA CARGA
"VAGÓN LLENO"
AVANZAR A SECADERO
FIN AVANZAR
"n de huecos en secadero"
"n de vagones en secadero"
FIN CICLO SECAD.
AVANZAR A Z.ESP.DESC.
FIN AVANZAR
"n de huecos en zona de espera a descarga"
"n de vagones en zona de espera a descarga"
ZONA DESCARGA OCUPADA
AVANZAR ZONA DESCARGA
ZONA DESCARGA OCUPADA
"ESPERAR DESCARGA"
"VAGÓN VACÍO"
AVANZAR A ZONA ESPERA CARGA
FIN AVANZAR
"n° huecos en zona de espera a carga"
"n° vagones en zona de espera a carga"
ZONA CARGA OCUPADA
AVANZAR ZONA CARGA
VAGÓN DE BANDEJAS
"n° huecos en vagón cargando"
"n° bandejas en vagón cargando"
"n° bandejas en vagón descargando"
"n° huecos en vagón descargando"
"ESPERAR"
CINTA 1 LIBRE
COGER BANDEJA
FIN COGER
AVANZAR
FIN AVANZAR
DEJAR BANDEJA
FIN DEJAR
REGRESAR A ORIGEN
FIN REGRESAR
PUENTE GRUA 2
"ESPERAR"
PUESTO 4 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PUESTO 4
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 4
"n° ladrillos en cinta C"
"ESPERAR"
PUESTO 5 LIBRE
AVANZAR
PUESTO 5 LIBRE
CINTA C
"ESPERAR"
PALET
PUESTO 6 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PALET
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 5
"huecos en palet"
"ladrillos en palet"
"ESPERAR MIENTRAS CARGAN"
"PALET CARGADO"
AVANZAR ZONA EMPAQUETADO
FIN AVANZAR
COGER PALET CARGADO
FIN COGER
CARGAR PALET VACÍO
FIN CARGAR
REGRESAR A ORIGEN
FIN REGRESAR
PALET
"n° palets producidos"

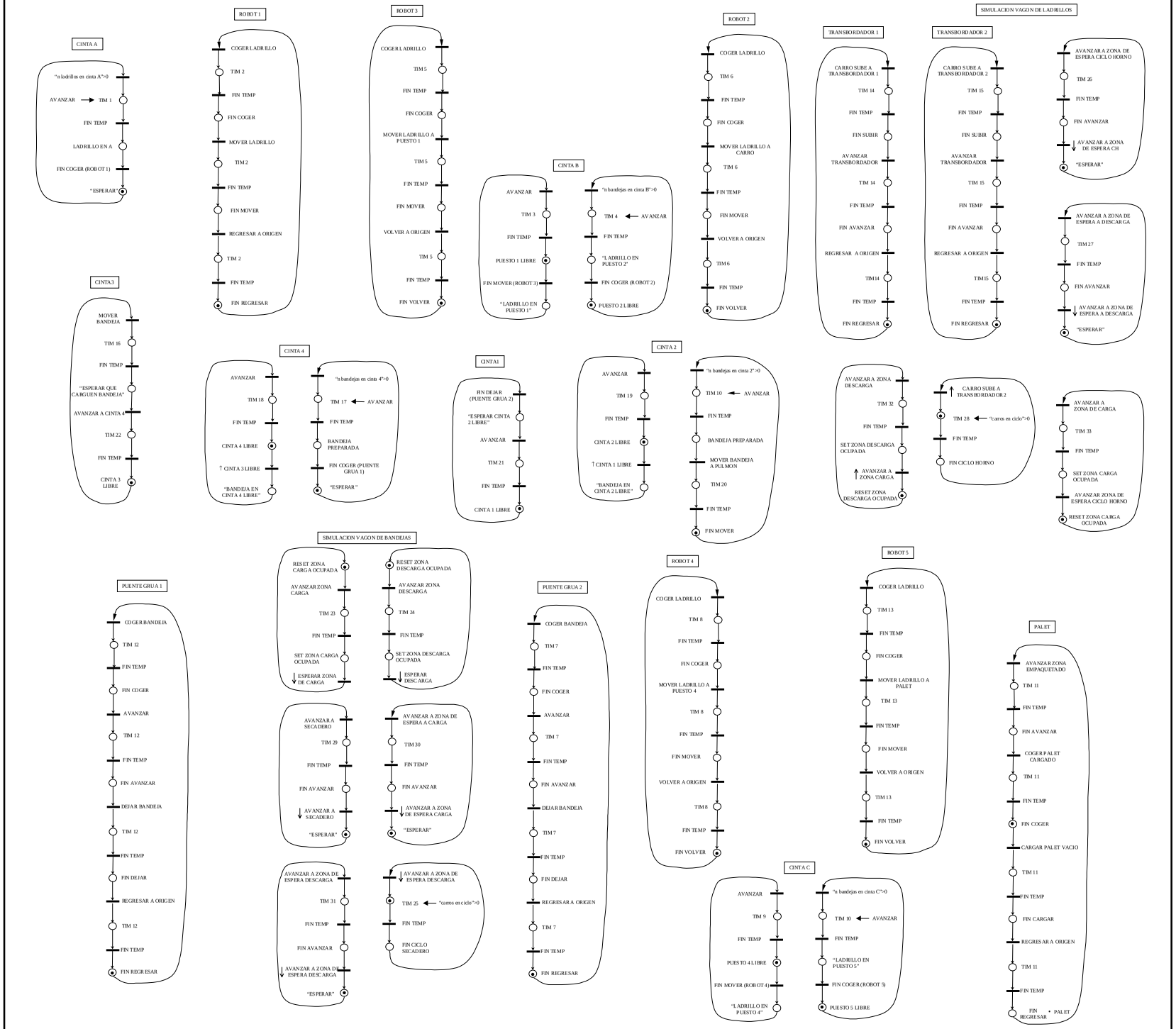
6.4. Automatización. Implementación sobre PLC.

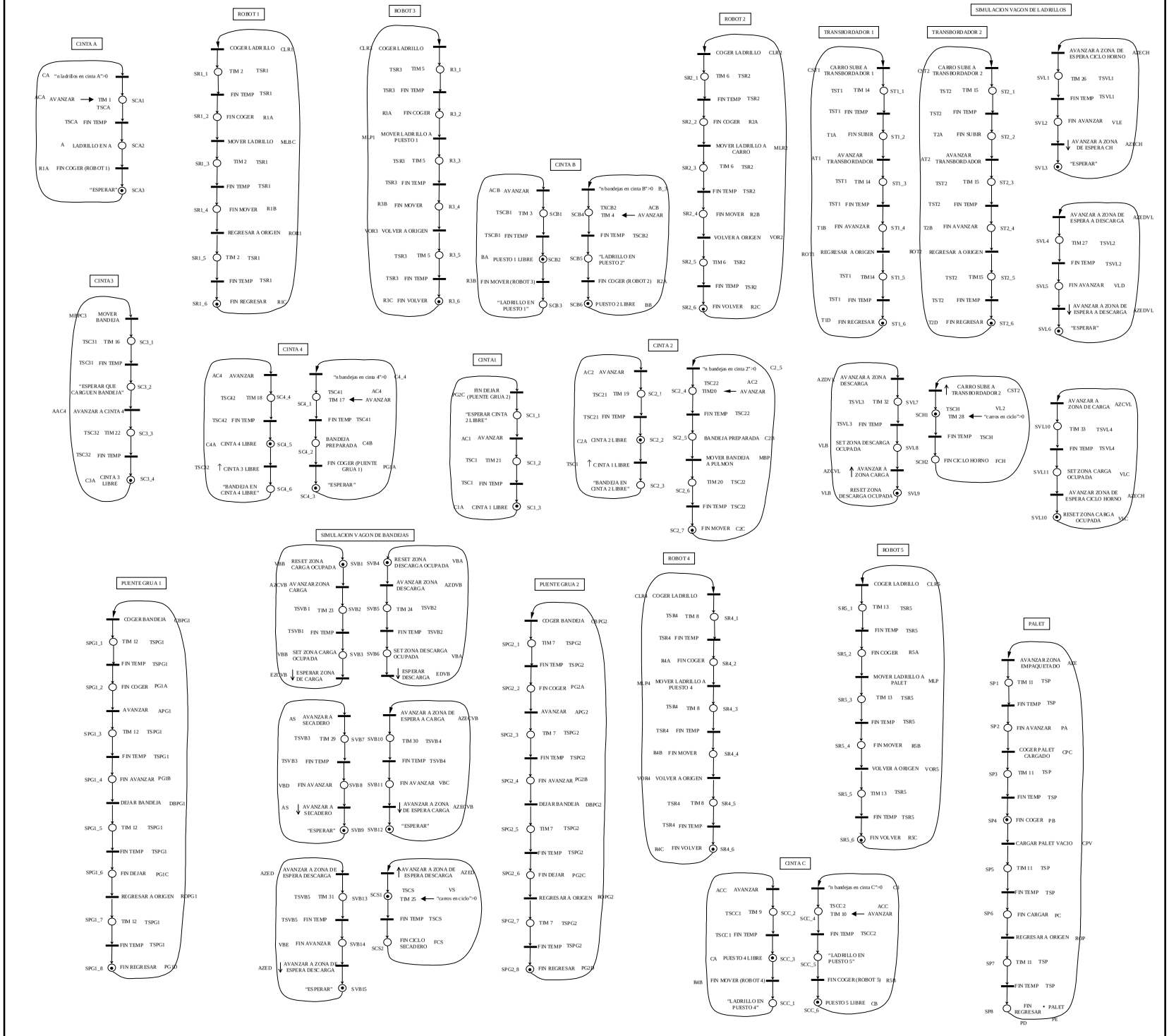
La realización de la automatización, una vez se tiene la figura 19, tan sólo consiste en asignar posiciones de memoria (realizar el mapeado de memoria) y traducirla como se ha visto en el capítulo 3. En la Figura 20 tenemos el mapeado de memoria realizado sobre la propia RdP, y la automatización completa, dada su extensión, se incluirá como una pequeña publicación independiente.

Pero en esa automatización también se ha incluido el modelo del sistema realizado sobre RdP, que se muestra en la Figura 21. La Figura 22 muestra lo mismo incluyendo el mapeado de memoria de ese modelo del sistema (sin automatizar).

(Figura 20: Mapeado de memoria de la automatización, sobre la RdP) ⇒
(Figura 21: RdP del modelo del sistema sin automatizar) ⇒
(Figura 22: Mapeado de memoria del modelo del sistema, sobre la RdP) ⇒


GALLETERA
"GENERACIÓN DE LADRILLOS"
TEMP
"nº de ladrillos en cinta A"
C_A
LADRILLO EN A
AVANZAR
CA1
LADRILLO EN A
"ESPERAR"
CA2
CINTA A
ROBOT 1
"ESPERAR LADRILLO"
LADRILLO EN A
CINTA 3 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A BANDEJA CARGANDO
FIN MOVER
REGRESAR A ORIGEN
FIN REGRESAR
"nº huecos en bandeja cargando"
"nº ladrillos en bandeja cargando"
"ESPERAR"
PUESTO 1 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PUESTO 1
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 3
"ladrillos en bandeja descargando"
"huecos en bandeja descargando"
"nº ladrillos en cinta B"
"ESPERAR"
PUESTO 2 LIBRE
AVANZAR
PUESTO 2 LIBRE
CINTA B
"ESPERAR"
ZONA CARGA OCUPADA
COGER LADRILLO
FIN COGER
MOVER LADRILLO A VAGÓN
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 2
"ladrillos en carro cargando"
"huecos en carro cargando"
"CARRO CARGADO"
AVANZAR ZONA DE ESPERA CICLO H
FIN AVANZAR
"huecos en zona de espera ciclo H"
"carros en zona de espera ciclo H"
"ESPERAR EN ORIGEN"
TRANSBORDADOR 1
FIN REGRESAR
REGRESAR A ORIGEN
"FIN ESPERAR"
CARRO SUBE A TRANSBORDADOR 1
FIN SUBIR
AVANZAR TRANSBORDADOR 1
FIN AVANZAR
"CARROS EN CICLO H"
"ESPERAR EN ORIGEN"
TRANSBORDADOR 2
FIN REGRESAR
REGRESAR A ORIGEN
FIN CICLO H
CARRO SUBE A TRANSBORDADOR 2
FIN SUBIR
AVANZAR TRANSBORDADOR 2
FIN AVANZAR
AVANZAR HASTA ZONA DE ESP DESCARGA
FIN AVANZAR
"huecos en zona de espera a descarga"
"carros en zona de espera a descarga"
"ladrillos en carro descargando"
ZONA DESCARGA OCUPADA
AVANZAR HASTA ZONA DE DESCARGA
ZONA DESCARGA OCUPADA
"ESPERAR DESCARGA CARRO"
ZONA CARGA OCUPADA
AVANZAR A ZONA DE CARGA
ZONA CARGA OCUPADA
"ESPERAR EN ZONA DE CARGA"
VAGÓN DE LADRILLOS
PULMÓN
"ESPERAR BANDEJA"
CINTA 3 LIBRE
MOVER BANDEJA DEL PULMÓN A CINTA 3
CINTA 3 LIBRE
"ESPERAR CARGA DE LADRILLOS"
CINTA 4 LIBRE
AVANZAR A CINTA 4
CINTA 3 • CINTA 4 LIBRE
"nº huecos en pulmón"
"nº bandejas en pulmón"
"nº bandejas en cinta 4"
CINTA3
"REPOSO"
BANDEJA PREPARADA
AVANZAR
BANDEJA PREPARADA
"ESPERAR QUE COJAN BANDEJA"
CINTA 4
"ESPERAR BANDEJA"
COGER BANDEJA
FIN COGER
AVANZAR
FIN AVANZAR
DEJAR BANDEJA
FIN DEJAR
REGRESAR A ORIGEN
FIN REGRESAR
PUENTE GRUA 1
"ESPERAR BANDEJA"
CINTA 1 LIBRE
"ESPERAR"
CINTA 2 LIBRE
AVANZAR
CINTA 2 LIBRE
CINTA 1
"nº de bandejas en cinta 2"
"ESPERAR"
BANDEJA PREPARADA
AVANZAR
BANDEJA PREPARADA
"ESPERAR DESCARGA DE LADRILLOS"
"FIN DESCARGA DE LADRILLOS"
MOVER BANDEJA A PULMÓN
FIN MOVER
CINTA 2
"ESPERAR"
ZONA CARGA OCUPADA
ESPERAR ZONA CARGA
"VAGÓN LLENO"
AVANZAR A SECADERO
FIN AVANZAR
"nº de vagones en secadero"
FIN CICLO SECAD.
AVANZAR A Z.ESP.DESC.
FIN AVANZAR
"nº de huecos en zona de espera a descarga"
"nº de vagones en zona de espera a descarga"
ZONA DESCARGA OCUPADA
AVANZAR ZONA DESCARGA
ZONA DESCARGA OCUPADA
"ESPERAR DESCARGA"
"VAGÓN VACÍO"
AVANZAR A ZONA ESPERA CARGA
FIN AVANZAR
"nº huecos en zona de espera a carga"
"nº vagones en zona de espera a carga"
ZONA CARGA OCUPADA
AVANZAR ZONA CARGA
VAGÓN DE BANDEJAS
"nº bandejas en vagón descargando"
"nº huecos en vagón descargando"
"nº huecos en vagón cargando"
"nº bandejas en vagón cargando"
"ESPERAR"
CINTA 1 LIBRE
COGER BANDEJA
FIN COGER
AVANZAR
FIN AVANZAR
DEJAR BANDEJA
FIN DEJAR
REGRESAR A ORIGEN
FIN REGRESAR
PUENTE GRUA 2
"ESPERAR"
PUESTO 4 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PUESTO 4
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT4
"nº ladrillos en cinta C"
"ESPERAR"
PUESTO 5 LIBRE
AVANZAR
PUESTO 5 LIBRE
CINTA C
"huecos en palet"
"ladrillos en palet"
"ESPERAR"
PALET • PUESTO 5 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PALET
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT5
"ESPERAR MIENTRAS CARGAN"
"PALET CARGADO"
AVANZAR ZONA EMPAQUETADO
FIN AVANZAR
COGER PALET CARGADO
FIN COGER
CARGAR PALET VACÍO
FIN CARGAR
REGRESAR A ORIGEN
FIN REGRESAR
"nº palets producidos"
PALET

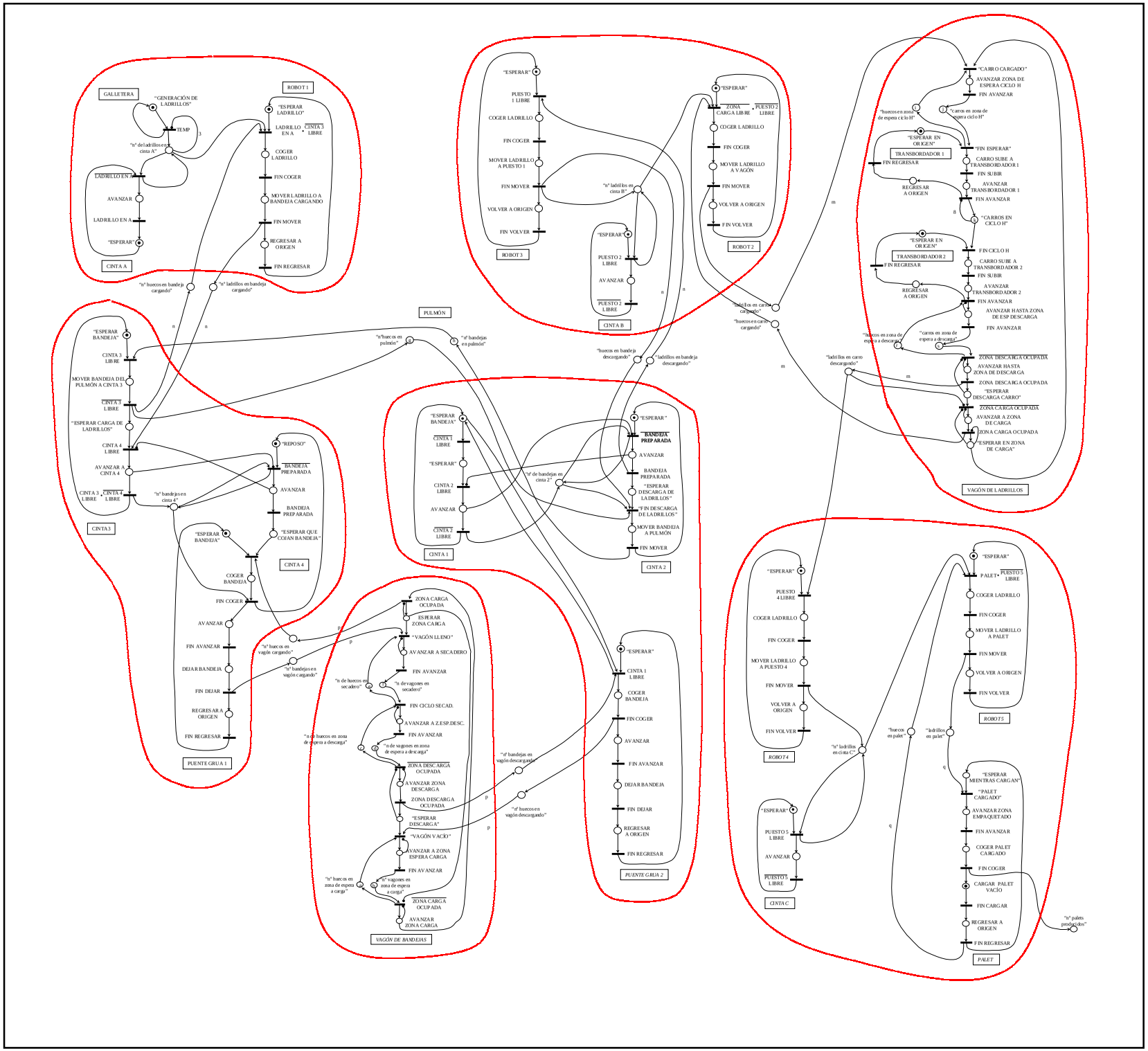

CINTA A
"n ladrillos en cinta A">0
AVANZAR
TIM 1
FIN TEMP
LADRILLO EN A
FIN COGER (ROBOT 1)
"ESPERAR"
ROBOT 1
COGER LADRILLO
TIM 2
FIN TEMP
FIN COGER
MOVER LADRILLO
TIM 2
FIN TEMP
FIN MOVER
REGRESAR A ORIGEN
TIM 2
FIN TEMP
FIN REGRESAR
ROBOT 3
COGER LADRILLO
TIM 5
FIN TEMP
FIN COGER
MOVER LADRILLO A PUESTO 1
TIM 5
FIN TEMP
FIN MOVER
VOLVER A ORIGEN
TIM 5
FIN TEMP
FIN VOLVER
CINTA B
AVANZAR
TIM 3
FIN TEMP
PUESTO 1 LIBRE
FIN MOVER (ROBOT 3)
"LADRILLO EN PUESTO 1"
"n bandejas en cinta B">0
TIM 4
AVANZAR
FIN TEMP
"LADRILLO EN PUESTO 2"
FIN COGER (ROBOT 2)
PUESTO 2 LIBRE
ROBOT 2
COGER LADRILLO
TIM 6
FIN TEMP
FIN COGER
MOVER LADRILLO A CARRO
TIM 6
FIN TEMP
FIN MOVER
VOLVER A ORIGEN
TIM 6
FIN TEMP
FIN VOLVER
SIMULACION VAGON DE LADRILLOS
TRANSBORDADOR 1
CARRO SUBE A TRANSBORDADOR 1
TIM 14
FIN TEMP
FIN SUBIR
AVANZAR TRANSBORDADOR
TIM 14
FIN TEMP
FIN AVANZAR
REGRESAR A ORIGEN
TIM14
FIN TEMP
FIN REGRESAR
TRANSBORDADOR 2
CARRO SUBE A TRANSBORDADOR 2
TIM 15
FIN TEMP
FIN SUBIR
AVANZAR TRANSBORDADOR
TIM 15
FIN TEMP
FIN AVANZAR
REGRESAR A ORIGEN
TIM15
FIN TEMP
FIN REGRESAR
AVANZAR A ZONA DE ESPERA CICLO HORNO
TIM 26
FIN TEMP
FIN AVANZAR
AVANZAR A ZONA DE ESPERA CH
"ESPERAR"
AVANZAR A ZONA DE ESPERA A DESCARGA
TIM 27
FIN TEMP
FIN AVANZAR
AVANZAR A ZONA DE ESPERA A DESCARGA
"ESPERAR"
CINTA3
MOVER BANDEJA
TIM 16
FIN TEMP
"ESPERAR QUE CARGUEN BANDEJA"
AVANZAR A CINTA 4
TIM 22
FIN TEMP
CINTA 3 LIBRE
CINTA 4
AVANZAR
TIM 18
FIN TEMP
CINTA 4 LIBRE
↑ CINTA 3 LIBRE
"BANDEJA EN CINTA 4 LIBRE"
"n bandejas en cinta 4">0
TIM 17
AVANZAR
FIN TEMP
BANDEJA PREPARADA
FIN COGER (PUENTE GRUA 1)
"ESPERAR "
CINTA1
FIN DEJAR (PUENTE GRUA 2)
"ESPERAR CINTA 2 LIBRE"
AVANZAR
TIM 21
FIN TEMP
CINTA 1 LIBRE
CINTA 2
AVANZAR
TIM 19
FIN TEMP
CINTA 2 LIBRE
↑ CINTA 1 LIBRE
"BANDEJA EN CINTA 2 LIBRE"
"n bandejas en cinta 2">0
TIM 10
AVANZAR
FIN TEMP
BANDEJA PREPARADA
MOVER BANDEJA A PULMON
TIM 20
FIN TEMP
FIN MOVER
AVANZAR A ZONA DESCARGA
TIM 32
FIN TEMP
SET ZONA DESCARGA OCUPADA
↑ AVANZAR A ZONA CARGA
RESET ZONA DESCARGA OCUPADA
↑ CARRO SUBE A TRANSBORDADOR 2
TIM 28
"carros en ciclo">0
FIN TEMP
FIN CICLO HORNO
AVANZAR A ZONA DE CARGA
TIM 33
FIN TEMP
SET ZONA CARGA OCUPADA
AVANZAR ZONA DE ESPERA CICLO HORNO
RESET ZONA CARGA OCUPADA
SIMULACION VAGON DE BANDEJAS
RESET ZONA CARGA OCUPADA
AVANZAR ZONA CARGA
TIM 23
FIN TEMP
SET ZONA CARGA OCUPADA
↓ ESPERAR ZONA DE CARGA
RESET ZONA DESCARGA OCUPADA
AVANZAR ZONA DESCARGA
TIM 24
FIN TEMP
SET ZONA DESCARGA OCUPADA
↓ ESPERAR DESCARGA
AVANZAR A SECADERO
TIM 29
FIN TEMP
FIN AVANZAR
↓ AVANZAR A SECADERO
"ESPERAR"
AVANZAR A ZONA DE ESPERA A CARGA
TIM 30
FIN TEMP
FIN AVANZAR
↓ AVANZAR A ZONA DE ESPERA CARGA
"ESPERAR"
AVANZAR A ZONA DE ESPERA DESCARGA
TIM 31
FIN TEMP
FIN AVANZAR
↓ AVANZAR A ZONA DE ESPERA DESCARGA
"ESPERAR"
↓ AVANZAR A ZONA DE ESPERA DESCARGA
TIM 25
"carros en ciclo">0
FIN TEMP
FIN CICLO SECADERO
PUENTE GRUA 1
COGER BANDEJA
TIM 12
FIN TEMP
FIN COGER
AVANZAR
TIM 12
FIN TEMP
FIN AVANZAR
DEJAR BANDEJA
TIM 12
FIN TEMP
FIN DEJAR
REGRESAR A ORIGEN
TIM 12
FIN TEMP
FIN REGRESAR
PUENTE GRUA 2
COGER BANDEJA
TIM 7
FIN TEMP
FIN COGER
AVANZAR
TIM 7
FIN TEMP
FIN AVANZAR
DEJAR BANDEJA
TIM 7
FIN TEMP
FIN DEJAR
REGRESAR A ORIGEN
TIM 7
FIN TEMP
FIN REGRESAR
ROBOT 4
COGER LADRILLO
TIM 8
FIN TEMP
FIN COGER
MOVER LADRILLO A PUESTO 4
TIM 8
FIN TEMP
FIN MOVER
VOLVER A ORIGEN
TIM 8
FIN TEMP
FIN VOLVER
ROBOT 5
COGER LADRILLO
TIM 13
FIN TEMP
FIN COGER
MOVER LADRILLO A PALET
TIM 13
FIN TEMP
FIN MOVER
VOLVER A ORIGEN
TIM 13
FIN TEMP
FIN VOLVER
CINTA C
AVANZAR
TIM 9
FIN TEMP
PUESTO 4 LIBRE
FIN MOVER (ROBOT 4)
"LADRILLO EN PUESTO 4"
"n bandejas en cinta C">0
TIM 10
AVANZAR
FIN TEMP
"LADRILLO EN PUESTO 5"
FIN COGER (ROBOT 5)
PUESTO 5 LIBRE
PALET
AVANZAR ZONA EMPAQUETADO
TIM 11
FIN TEMP
FIN AVANZAR
COGER PALET CARGADO
TIM 11
FIN TEMP
FIN COGER
CARGAR PALET VACIO
TIM 11
FIN TEMP
FIN CARGAR
REGRESAR A ORIGEN
TIM 11
FIN TEMP
FIN REGRESAR
• PALET

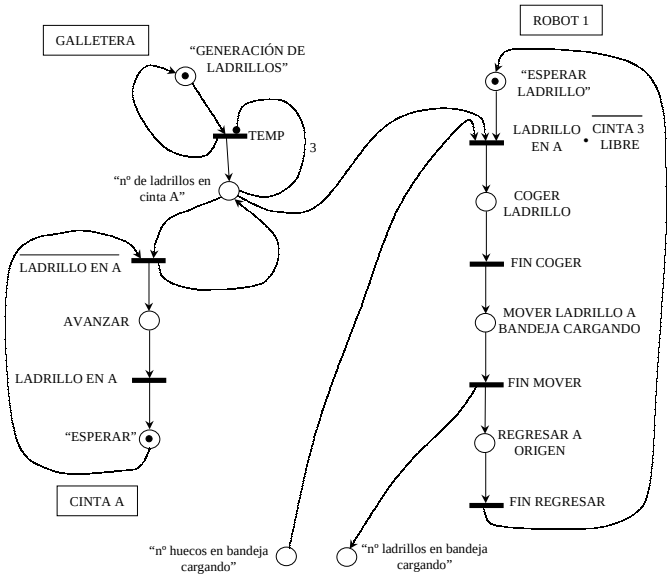

SIMULACION VAGON DE LADRILLOS
CINTA A
CA "n ladrillos en cinta A">0
ACA AVANZAR TIM 1 TSCA SCA1
TSCA FIN TEMP SCA2
A LADRILLO EN A SCA2
R1A FIN COGER (ROBOT 1) SCA3
"ESPERAR"
ROBOT 1
COGER LADRILLO CLR1
SR1_1 TIM 2 TSR1
FIN TEMP TSR1 SR1_2
FIN COGER R1A
MOVER LADRILLO MLB1C
SR1_3 TIM 2 TSR1
FIN TEMP TSR1 SR1_4
FIN MOVER R1B
REGRESAR A ORIGEN ROR1
SR1_5 TIM 2 TSR1
FIN TEMP TSR1 SR1_6
FIN REGRESAR R1C
ROBOT 3
CLR3 COGER LADRILLO R3_1
TSR3 TIM 5
TSR3 FIN TEMP R3A FIN COGER R3_2
MLP3 MOVER LADRILLO A PUESTO 1
TSR3 TIM 5 R3_3
TSR3 FIN TEMP R3B FIN MOVER R3_4
VOR3 VOLVER A ORIGEN
TSR3 TIM 5 R3_5
TSR3 FIN TEMP R3C FIN VOLVER R3_6
CINTA B
ACB AVANZAR TSCB1 TIM 3 SCB1
TSCB1 FIN TEMP BA PUESTO 1 LIBRE SCB2
R3B FIN MOVER (ROBOT 3) "LADRILLO EN PUESTO 1" SCB3
"n bandejas en cinta B">0 B_3
SCB4 TXCB2 TIM 4 ACB AVANZAR
FIN TEMP TSCB2
SCB5 "LADRILLO EN PUESTO 2"
SCB6 PUESTO 2 LIBRE BB FIN COGER (ROBOT 2) R2A
ROBOT 2
COGER LADRILLO CLR2
SR2_1 TIM 6 TSR2
FIN TEMP TSR2 SR2_2
FIN COGER R2A
MOVER LADRILLO A CARRO MLR2
SR2_3 TIM 6 TSR2
FIN TEMP TSR2 SR2_4
FIN MOVER R2B
VOLVER A ORIGEN VOR2
SR2_5 TIM 6 TSR2
FIN TEMP TSR2 SR2_6
FIN VOLVER R2C
TRANSBORDADOR 1
CST1 CARRO SUBE A TRANSBORDADOR 1 ST1_1
TST1 TIM 14
TST1 FIN TEMP T1A FIN SUBIR ST1_2
AT1 AVANZAR TRANSBORDADOR
TST1 TIM 14 ST1_3
TST1 FIN TEMP T1B FIN AVANZAR ST1_4
ROT1 REGRESAR A ORIGEN
TST1 TIM14 ST1_5
TST1 FIN TEMP T1D FIN REGRESAR ST1_6
TRANSBORDADOR 2
CST2 CARRO SUBE A TRANSBORDADOR 2 ST2_1
TST2 TIM 15
TST2 FIN TEMP T2A FIN SUBIR ST2_2
AT2 AVANZAR TRANSBORDADOR
TST2 TIM 15 ST2_3
TST2 FIN TEMP T2B FIN AVANZAR ST2_4
ROT2 REGRESAR A ORIGEN
TST2 TIM15 ST2_5
TST2 FIN TEMP T2D FIN REGRESAR ST2_6
AVANZAR A ZONA DE ESPERA CICLO HORNO AZECH
SVL1 TIM 36 TSVL1
FIN TEMP TS VL1 SVL2
FIN AVANZAR VLE
AVANZAR A ZONA DE ESPERA CH AZECH
SVL3 "ESPERAR"
AVANZAR A ZONA DE ESPERA A DESCARGA AZEDVL
SVL4 TIM 27 TSVL2
FIN TEMP TSVL2 SVL5
FIN AVANZAR VLD
AVANZAR A ZONA DE ESPERA A DESCARGA AZEDVL
SVL6 "ESPERAR"
CINTA3
MBPC3 MOVER BANDEJA
TSC31 TIM 16 SC3_1
TSC31 FIN TEMP
"ESPERAR QUE CARGUEN BANDEJA" SC3_2
AAC4 AVANZAR A CINTA 4
TSC32 TIM 22 SC3_3
TSC32 FIN TEMP
C3A CINTA 3 LIBRE SC3_4
CINTA 4
AC4 AVANZAR TSC42 TIM 18 SC4_4
TSC42 FIN TEMP C4A CINTA 4 LIBRE SC4_5
TSC32 CINTA 3 LIBRE "BANDEJA EN CINTA 4 LIBRE" SC4_6
"n bandejas en cinta 4">0 C4_4
SC4_1 TSC41 TIM 17 AC4 AVANZAR
FIN TEMP TSC41
SC4_2 BANDEJA PREPARADA C4B
FIN COGER (PUENTE GRUA 1) PG1A
"ESPERAR" SC4_3
CINTA1
FIN DEJAR (PUENTE GRUA 2) PG2C
"ESPERAR CINTA 2 LIBRE" SC1_3
AC1 AVANZAR TSC1 TIM 21 SC1_2
TSC1 FIN TEMP C1A CINTA 1 LIBRE SC1_3
CINTA 2
AC2 AVANZAR TSC21 TIM 19 SC2_1
TSC21 FIN TEMP C2A CINTA 2 LIBRE SC2_2
TSC1 CINTA 1 LIBRE "BANDEJA EN CINTA 2 LIBRE" SC2_3
"n bandejas en cinta 2">0 C2_5
SC2_4 TSC22 TIM20 AC2 AVANZAR
FIN TEMP TSC22
SC2_5 BANDEJA PREPARADA C2B
MOVER BANDEJA A PULMON MBP
SC2_6 TIM 20 TSC22
FIN TEMP TSC22
SC2_7 FIN MOVER C2C
AVANZAR A ZONA DE DESCARGA AZDVL
TSVL3 TIM 32 SVL7
TSVL3 FIN TEMP VLB SET ZONA DESCARGA OCUPADA SVL8
AZCVL AVANZAR A ZONA CARGA
VLB RESET ZONA DESCARGA OCUPADA SVL9
CARRO SUBE A TRANSBORDADOR 2 CST2
SCH1 TSCH TIM 28 VL2 "carros en ciclo">0
FIN TEMP TSCH
SCH2 FIN CICLO HORNO FCH
AVANZAR A ZONA DE CARGA AZCVL
SVL10 TIM 33 TSVL4
FIN TEMP TS VL4
SVL11 SET ZONA CARGA OCUPADA VLC
AVANZAR ZONA DE ESPERA CICLO HORNO AZECH
S VL10 RESET ZONA CARGA OCUPADA VLC
SIMULACION VAGON DE BANDEJAS
VBB RESET ZONA CARGA OCUPADA SVB1 SVB4 RESET ZONA DESCARGA OCUPADA VBA
AZCVB AVANZAR ZONA CARGA
AVANZAR ZONA DESCARGA AZDVB
TSVB1 TIM 23 SVB2 SVB5 TIM 24 TSVB2
TSVB1 FIN TEMP
FIN TEMP TSVB2
VBB SET ZONA CARGA OCUPADA SVB3 SVB6 SET ZONA DESCARGA OCUPADA VBA
EZCVB ESPERAR ZONA DE CARGA
ESPERAR DESCARGA EDVB
AS AVANZAR A SECADERO
AVANZAR A ZONA DE ESPERA A CARGA AZECVB
TSVB3 TIM 29 SVB7 SVB10 TIM 30 TSVB4
TSVB3 FIN TEMP
FIN TEMP TSVB4
VBD FIN AVANZAR SVB8 SVB11 FIN AVANZAR VBC
AS AVANZAR A SECADERO
AVANZAR A ZONA DE ESPERA CARGA AZECVB
"ESPERAR" SVB9 SVB12 "ESPERAR"
AZED AVANZAR A ZONA DE ESPERA DESCARGA
AVANZAR A ZONA DE ESPERA DESCARGA AZED
TSVB5 TIM 31 SVB13 SCS1 TSCS TIM 25 VS "carros en ciclo">0
TSVB5 FIN TEMP
FIN TEMP TSCS
VBE FIN AVANZAR SVB14 SCS2 FIN CICLO SECADERO FCS
AZED AVANZAR A ZONA DE ESPERA DESCARGA
"ESPERAR" SVB15
PUENTE GRUA 1
COGER BANDEJA CBPG1
SPG1_1 TIM 12 TSPG1
FIN TEMP TSPG1 SPG1_2
FIN COGER PG1A
AVANZAR APG1 SPG1_3
TIM 12 TSPG1
FIN TEMP TSPG1 SPG1_4
FIN AVANZAR PG1B
DEJAR BANDEJA DBPG1 SPG1_5
TIM 12 TSPG1
FIN TEMP TSPG1 SPG1_6
FIN DEJAR PG1C
REGRESAR A ORIGEN ROPG1 SPG1_7
TIM 12 TSPG1
FIN TEMP TSPG1 SPG1_8
FIN REGRESAR PG1D
PUENTE GRUA 2
COGER BANDEJA CBPG2
SPG2_1 TIM 7 TSPG2
FIN TEMP TSPG2 SPG2_2
FIN COGER PG2A
AVANZAR APG2 SPG2_3
TIM 7 TSPG2
FIN TEMP TSPG2 SPG2_4
FIN AVANZAR PG2B
DEJAR BANDEJA DBPG2 SPG2_5
TIM 7 TSPG2
FIN TEMP TSPG2 SPG2_6
FIN DEJAR PG2C
REGRESAR A ORIGEN ROPG2 SPG2_7
TIM 7 TSPG2
FIN TEMP TSPG2 SPG2_8
FIN REGRESAR PG2D
ROBOT 4
CLR4 COGER LADRILLO SR4_1
TSR4 TIM 8
TSR4 FIN TEMP B4A FIN COGER SR4_2
MLP4 MOVER LADRILLO A PUESTO 4
TSR4 TIM 8 SR4_3
TSR4 FIN TEMP B4B FIN MOVER SR4_4
VOR4 VOLVER A ORIGEN
TSR4 TIM 8 SR4_5
TSR4 FIN TEMP B4C FIN VOLVER SR4_6
CINTA C
ACC AVANZAR TSCC1 TIM 9 SCC_2
TSCC1 FIN TEMP CA PUESTO 4 LIBRE SCC_3
B4B FIN MOVER (ROBOT 4) "LADRILLO EN PUESTO 4" SCC_1
"n bandejas en cinta C">0 CC
SCC_4 TSCC2 TIM 10 ACC AVANZAR
FIN TEMP TSCC2
SCC_5 "LADRILLO EN PUESTO 5"
FIN COGER (ROBOT 5) R5B
SCC_6 PUESTO 5 LIBRE CB
ROBOT 5
COGER LADRILLO CLR5
SR5_1 TIM 13 TSR5
FIN TEMP TSR5 SR5_2
FIN COGER R5A
MOVER LADRILLO A PALET MLP
SR5_3 TIM 13 TSR5
FIN TEMP TSR5 SR5_4
FIN MOVER R5B
VOLVER A ORIGEN VOR5
SR5_5 TIM 13 TSR5
FIN TEMP TSR5 SR5_6
FIN VOLVER R5C
PALET
AVANZAR ZONA EMPAQUETADO AZE
SP1 TIM 11 TSP
FIN TEMP TSP SP2
FIN AVANZAR PA
COGER PALET CARGADO CPC
SP3 TIM 11 TSP
FIN TEMP TSP SP4
FIN COGER PB
CARGAR PALET VACIO CPV
SP5 TIM 11 TSP
FIN TEMP TSP SP6
FIN CARGAR PC
REGRESAR A ORIGEN ROP
SP7 TIM 11 TSP
FIN TEMP TSP SP8
FIN REGRESAR * PALET
PD PE

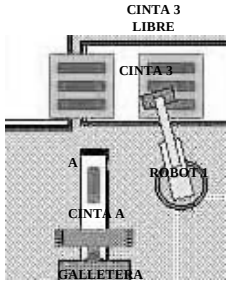
Pero de la automatización que tenemos se puede hacer un estudio mucho más profundo. Por ejemplo, un análisis detallado revela que en realidad la automatización consiste en una serie de grupos o bloques que van pasando tokens desde el principio hasta el final, desde los buffers de entrada hasta los de salida. Los grupos se muestran en la Figura 23, y esta observación resulta muy interesante, puesto que para obtener mayor producción vemos que lo que hay que hacer es simplemente mejorar el grupo que sea más lento. De esta manera, podemos simular (de cualquiera de las formas estudiadas a lo largo de esta tesis) cada uno de los grupos por separado, para ver la velocidad de cada uno.

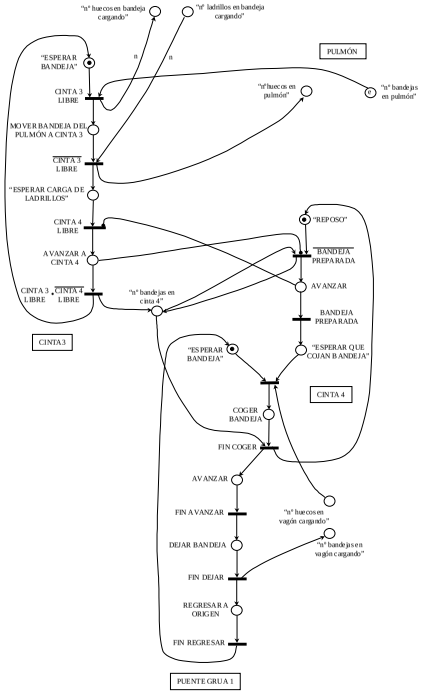
Como experiencia en automatizaciones industriales de varios procesos fuertemente interconectados, debo indicar que muchas veces se quiere mejorar la producción pero no se sabe qué hay que hacer para ello. Es posible que aumentar el número de vagones no mejore en nada el proceso, si el cuello de botella estaba en otro sitio que no se ve afectado. Las siguientes figuras corresponden a cada uno de los grupos por separado, en la que se ha incluido una imagen de cómo son físicamente, para indicar dónde están los sensores y demás elementos de que constan.

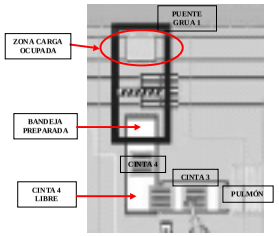
(Figura 23: Grupos o bloques de la automatización) ⇒
(Figura 24: Grupo Galletera, Robot 1 y Cinta A) ⇒
(Figura 25: Grupo Cinta 3, Cinta 4 y Puente Grúa 1) ⇒
(Figura 26: Grupo Vagón de Bandejas) ⇒
(Figura 27: Grupo Cinta 1, Cinta 2 y Puente Grúa 2) ⇒
(Figura 28: Grupo Robot 2, Robot 3 y Cinta B) ⇒
(Figura 29: Grupo Vagón de ladrillos) ⇒
(Figura 30: Grupo Robot 4, Robot 5, Cinta C, Palets) ⇒

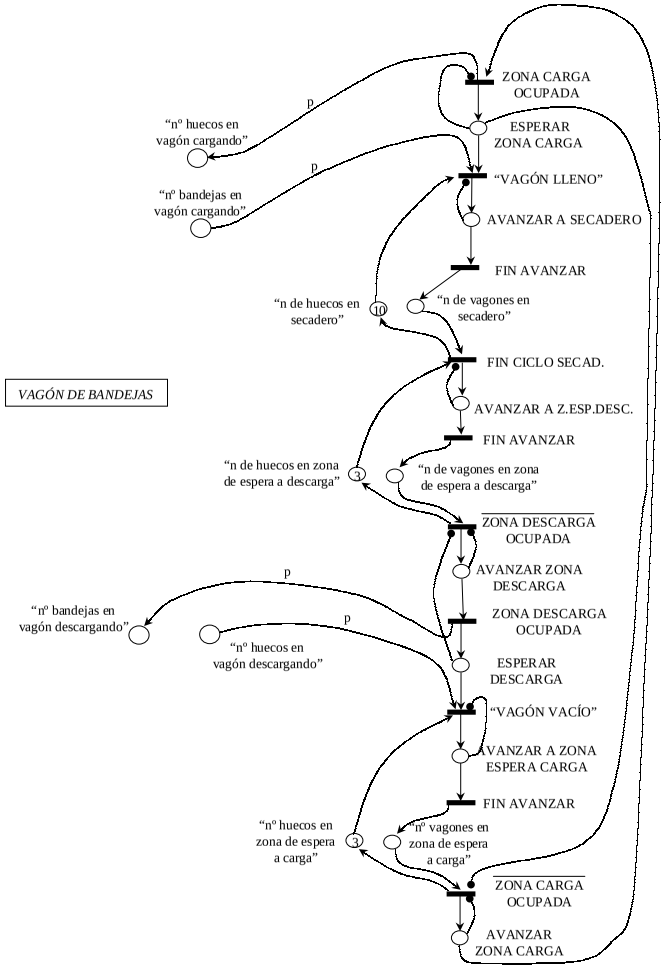

GALLETERA
"GENERACIÓN DE LADRILLOS"
TEMP
3
"n° de ladrillos en cinta A"
LADRILLO EN A
AVANZAR
LADRILLO EN A
"ESPERAR"
CINTA A
ROBOT 1
"ESPERAR LADRILLO"
LADRILLO EN A
CINTA 3 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A BANDEJA CARGANDO
FIN MOVER
REGRESAR A ORIGEN
FIN REGRESAR
"n° huecos en bandeja cargando"
"n° ladrillos en bandeja cargando"
"ESPERAR"
PUESTO 1 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PUESTO 1
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 3
"n° ladrillos en cinta B"
"ESPERAR"
PUESTO 2 LIBRE
AVANZAR
PUESTO 2 LIBRE
CINTA B
"ESPERAR"
ZONA CARGA LIBRE
PUESTO 2 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A VAGÓN
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 2
"CARRO CARGADO"
AVANZAR ZONA DE ESPERA CICLO H
FIN AVANZAR
"huecos en zona de espera ciclo H"
"carros en zona de espera ciclo H"
"ESPERAR EN ORIGEN"
TRANSBORDADOR 1
FIN REGRESAR
REGRESAR A ORIGEN
"FIN ESPERAR"
CARRO SUBE A TRANSBORDADOR 1
FIN SUBIR
AVANZAR TRANSBORDADOR 1
FIN AVANZAR
"CARROS EN CICLO H"
"ESPERAR EN ORIGEN"
TRANSBORDADOR 2
FIN REGRESAR
REGRESAR A ORIGEN
FIN CICLO H
CARRO SUBE A TRANSBORDADOR 2
FIN SUBIR
AVANZAR TRANSBORDADOR 2
FIN AVANZAR
AVANZAR HASTA ZONA DE ESP DESCARGA
FIN AVANZAR
"huecos en zona de espera a descarga"
"carros en zona de espera a descarga"
ZONA DESCARGA OCUPADA
AVANZAR HASTA ZONA DE DESCARGA
ZONA DESCARGA OCUPADA
"ESPERAR DESCARGA CARRO"
ZONA CARGA OCUPADA
AVANZAR A ZONA DE CARGA
ZONA CARGA OCUPADA
"ESPERAR EN ZONA DE CARGA"
VAGÓN DE LADRILLOS
"ladrillos en carro cargando"
"huecos en carro cargando"
"ladrillos en carro descargando"
PULMÓN
"n°huecos en pulmón"
"n° bandejas en pulmón"
"huecos en bandeja descargando"
"ladrillos en bandeja descargando"
"ESPERAR BANDEJA"
CINTA 3 LIBRE
MOVER BANDEJA DEL PULMÓN A CINTA 3
CINTA 3 LIBRE
"ESPERAR CARGA DE LADRILLOS"
CINTA 4 LIBRE
AVANZAR A CINTA 4
CINTA 3 LIBRE
CINTA 4 LIBRE
CINTA3
"n° bandejas en cinta 4"
"REPOSO"
BANDEJA PREPARADA
AVANZAR
BANDEJA PREPARADA
"ESPERAR QUE COJAN BANDEJA"
CINTA 4
"ESPERAR BANDEJA"
COGER BANDEJA
FIN COGER
AVANZAR
FIN AVANZAR
DEJAR BANDEJA
FIN DEJAR
REGRESAR A ORIGEN
FIN REGRESAR
PUENTE GRUA 1
"n° huecos en vagón cargando"
"n° bandejas en vagón cargando"
"ESPERAR BANDEJA"
CINTA 1 LIBRE
"ESPERAR"
CINTA 2 LIBRE
AVANZAR
CINTA 2 LIBRE
CINTA 1
"n° de bandejas en cinta 2"
"ESPERAR"
BANDEJA PREPARADA
AVANZAR
BANDEJA PREPARADA
"ESPERAR DESCARGA DE LADRILLOS"
"FIN DESCARGA DE LADRILLOS"
MOVER BANDEJA A PULMÓN
FIN MOVER
CINTA 2
ZONA CARGA OCUPADA
ESPERAR ZONA CARGA
"VAGÓN LLENO"
AVANZAR A SECADERO
FIN AVANZAR
"n de huecos en secadero"
"n de vagones en secadero"
FIN CICLO SECAD.
AVANZAR A Z.ESP.DESC.
FIN AVANZAR
"n de huecos en zona de espera a descarga"
"n de vagones en zona de espera a descarga"
ZONA DESCARGA OCUPADA
AVANZAR ZONA DESCARGA
ZONA DESCARGA OCUPADA
"ESPERAR DESCARGA"
"VAGÓN VACÍO"
AVANZAR A ZONA ESPERA CARGA
FIN AVANZAR
"n° huecos en zona de espera a carga"
"n° vagones en zona de espera a carga"
ZONA CARGA OCUPADA
AVANZAR ZONA CARGA
VAGÓN DE BANDEJAS
"n° bandejas en vagón descargando"
"n° huecos en vagón descargando"
"ESPERAR"
CINTA 1 LIBRE
COGER BANDEJA
FIN COGER
AVANZAR
FIN AVANZAR
DEJAR BANDEJA
FIN DEJAR
REGRESAR A ORIGEN
FIN REGRESAR
PUENTE GRUA 2
"ESPERAR"
PUESTO 4 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PUESTO 4
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT4
"n° ladrillos en cinta C"
"ESPERAR"
PUESTO 5 LIBRE
AVANZAR
PUESTO 5 LIBRE
CINTA C
"ESPERAR"
PALET
PUESTO 6 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PALET
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT5
"huecos en palet"
"ladrillos en palet"
"ESPERAR MIENTRAS CARGAN"
"PALET CARGADO"
AVANZAR ZONA EMPAQUETADO
FIN AVANZAR
COGER PALET CARGADO
FIN COGER
CARGAR PALET VACÍO
FIN CARGAR
REGRESAR A ORIGEN
FIN REGRESAR
PALET
"n° palets producidos"

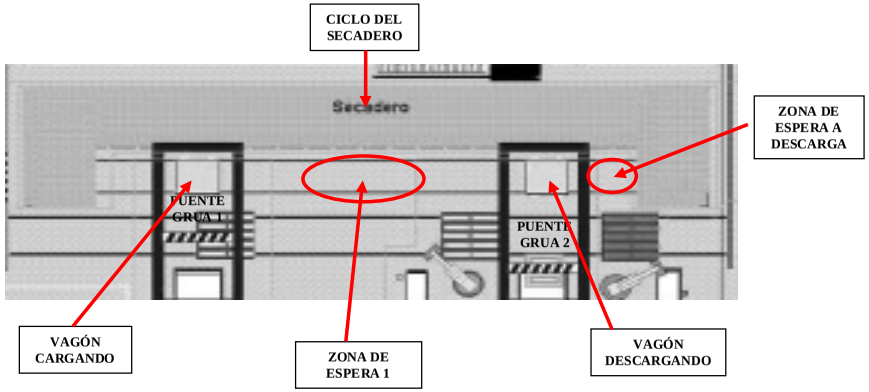

GALLETERA
"GENERACIÓN DE LADRILLOS"
TEMP
3
"nº de ladrillos en cinta A"
LADRILLO EN A
AVANZAR
LADRILLO EN A
"ESPERAR"
CINTA A
ROBOT 1
"ESPERAR LADRILLO"
LADRILLO EN A
CINTA 3 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A BANDEJA CARGANDO
FIN MOVER
REGRESAR A ORIGEN
FIN REGRESAR
"nº huecos en bandeja cargando"
"nº ladrillos en bandeja cargando"

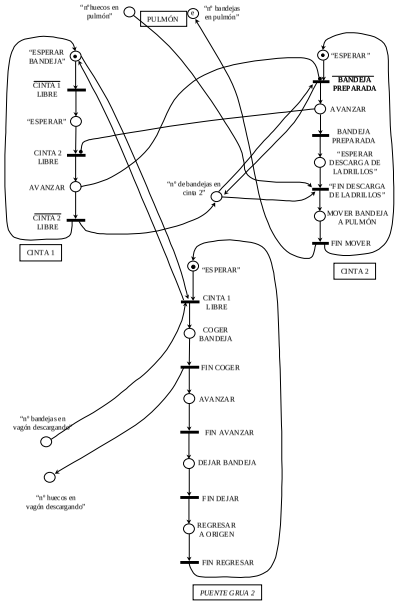

CINTA 3 LIBRE
CINTA 3
A
ROBOT 1
CINTA A
GALLETERA

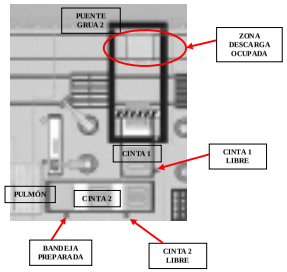

"n" huecos en bandeja cargando"
"n" ladrillos en bandeja cargando"
PULMÓN
"ESPERAR BANDEJA"
n
n
CINTA 3 LIBRE
"n" huecos en pulmón"
"n" bandejas en pulmón"
MOVER BANDEJA DEL PULMÓN A CINTA 3
CINTA 3 LIBRE
"ESPERAR CARGA DE LADRILLOS"
"REPOSO"
CINTA 4 LIBRE
BANDEJA PREPARADA
AVANZAR A CINTA 4
AVANZAR
CINTA 3 LIBRE
CINTA 4 LIBRE
"n" bandejas en cinta 4"
BANDEJA PREPARADA
CINTA 3
"ESPERAR BANDEJA"
"ESPERAR QUE COJAN BANDEJA"
CINTA 4
COGER BANDEJA
FIN COGER
AVANZAR
FIN AVANZAR
"n" huecos en vagón cargando"
DEJAR BANDEJA
"n" bandejas en vagón cargando"
FIN DEJAR
REGRESAR A ORIGEN
FIN REGRESAR
PUENTE GRUA 1

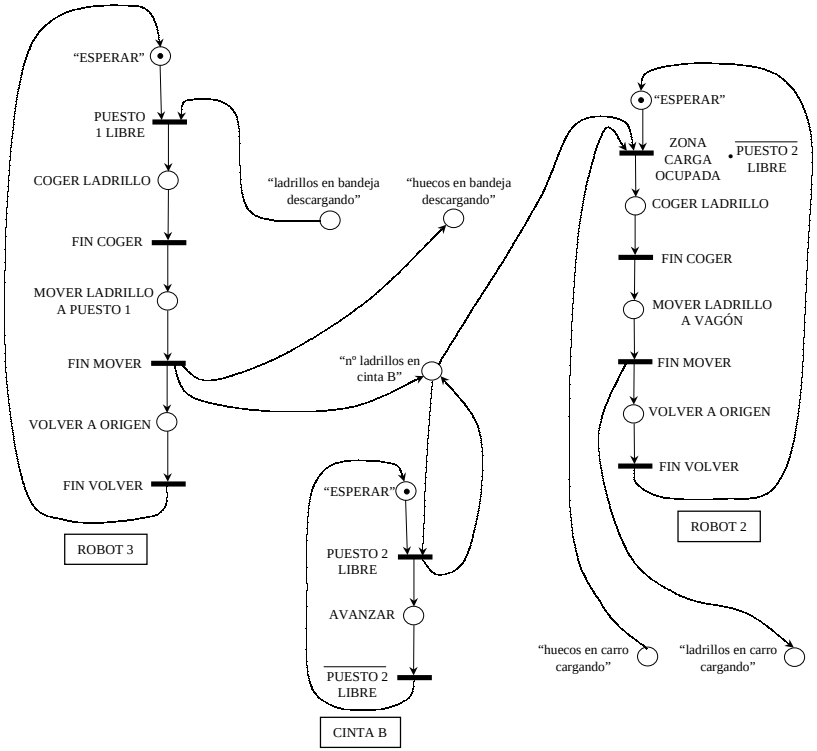

PUENTE GRUA 1
ZONA CARGA OCUPADA
BANDEJA PREPARADA
CINTA 4
CINTA 3
CINTA 4 LIBRE
PULMÓN

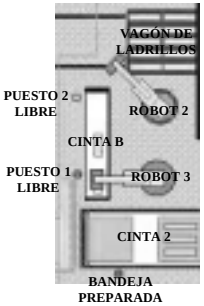

ZONA CARGA OCUPADA
"nº huecos en vagón cargando"
p
ESPERAR ZONA CARGA
p
"nº bandejas en vagón cargando"
"VAGÓN LLENO"
AVANZAR A SECADERO
FIN AVANZAR
"n de huecos en secadero"
10
"n de vagones en secadero"
FIN CICLO SECAD.
VAGÓN DE BANDEJAS
AVANZAR A Z.ESP.DESC.
FIN AVANZAR
"n de huecos en zona de espera a descarga"
3
"n de vagones en zona de espera a descarga"
ZONA DESCARGA OCUPADA
AVANZAR ZONA DESCARGA
p
"nº bandejas en vagón descargando"
ZONA DESCARGA OCUPADA
p
"nº huecos en vagón descargando"
ESPERAR DESCARGA
"VAGÓN VACÍO"
AVANZAR A ZONA ESPERA CARGA
FIN AVANZAR
"nº huecos en zona de espera a carga"
3
"nº vagones en zona de espera a carga"
ZONA CARGA OCUPADA
AVANZAR ZONA CARGA

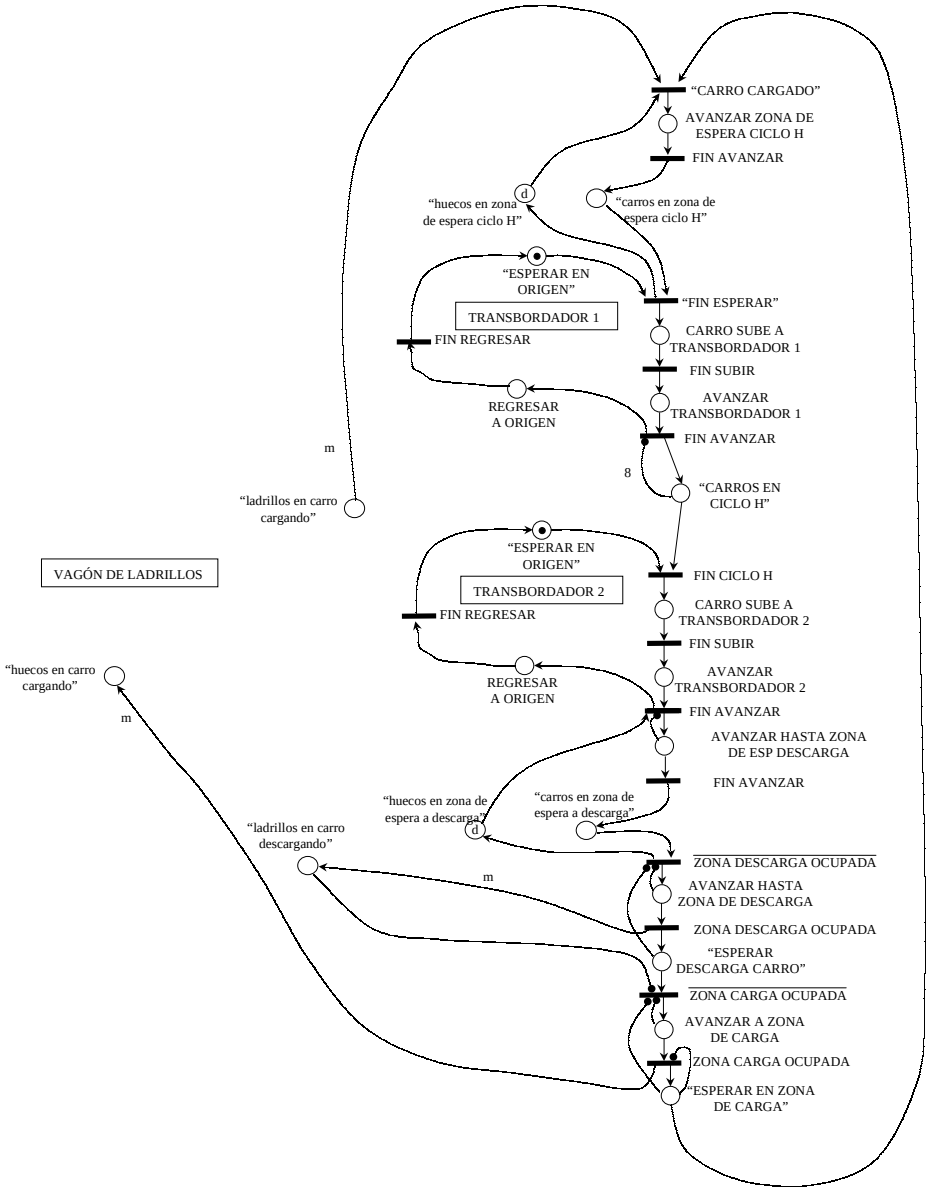

CICLO DEL SECADERO
Secadero
ZONA DE ESPERA A DESCARGA
PUENTE GRUA 1
PUENTE GRUA 2
VAGÓN CARGANDO
ZONA DE ESPERA 1
VAGÓN DESCARGANDO

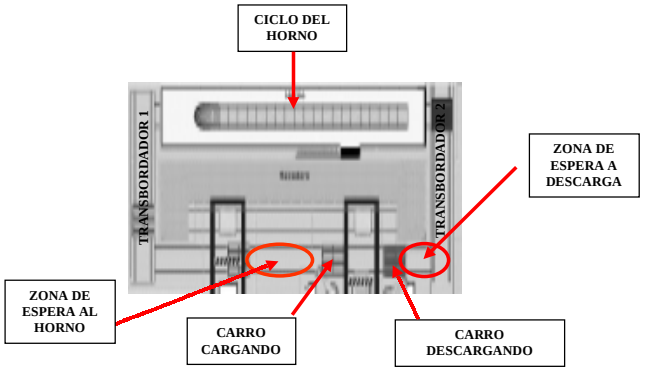

"nº huecos en pulmón"
PULMÓN
"nº bandejas en pulmón"
"ESPERAR BANDEJA"
CINTA 1 LIBRE
"ESPERAR"
CINTA 2 LIBRE
AVANZAR
CINTA 2 LIBRE
CINTA 1
"ESPERAR"
BANDEJA PREPARADA
AVANZAR
BANDEJA PREPARADA
"ESPERAR DESCARGA DE LADRILLOS"
"nº de bandejas en cinta 2"
"FIN DESCARGA DE LADRILLOS"
MOVER BANDEJA A PULMÓN
FIN MOVER
CINTA 2
"ESPERAR"
CINTA 1 LIBRE
COGER BANDEJA
FIN COGER
AVANZAR
FIN AVANZAR
DEJAR BANDEJA
"nº bandejas en vagón descargando"
"nº huecos en vagón descargando"
FIN DEJAR
REGRESAR A ORIGEN
FIN REGRESAR
PUENTE GRUA 2

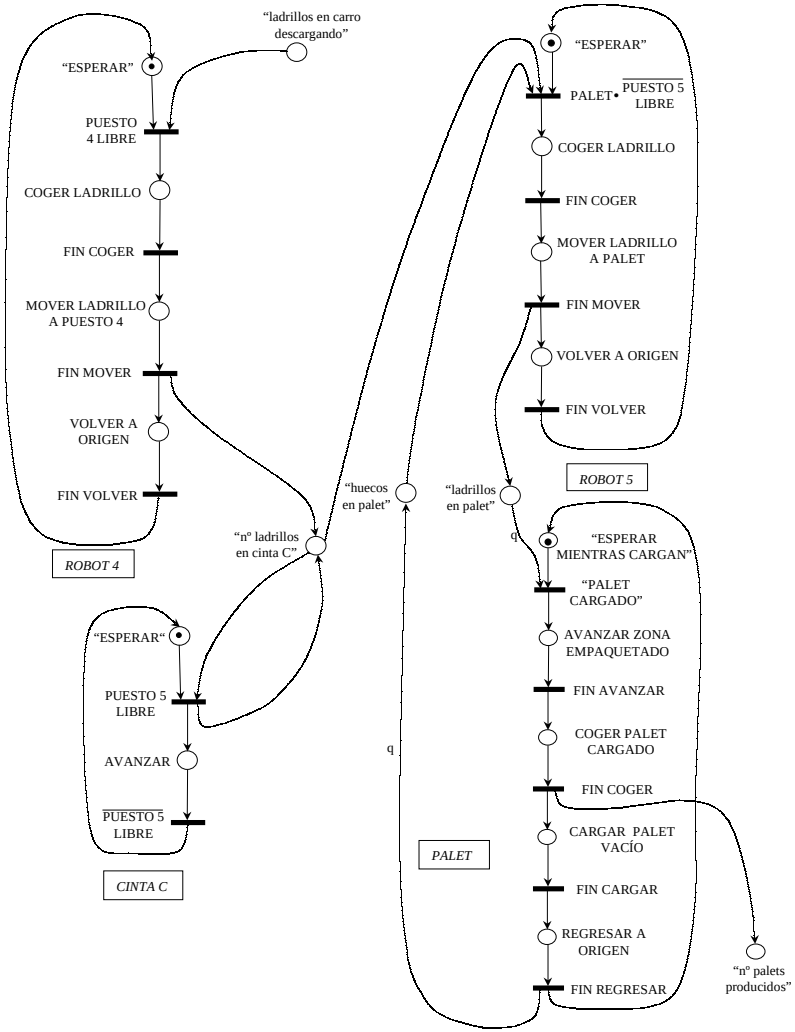

PUENTE GRUA 2
ZONA DESCARGA OCUPADA
CINTA 1
CINTA 1 LIBRE
PULMÓN
CINTA 2
BANDEJA PREPARADA
CINTA 2 LIBRE

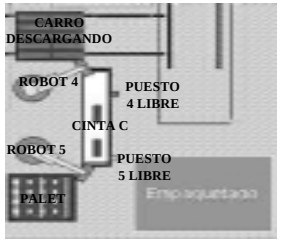

"ESPERAR"
PUESTO 1 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PUESTO 1
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 3
"ladrillos en bandeja descargando"
"huecos en bandeja descargando"
"nº ladrillos en cinta B"
"ESPERAR"
PUESTO 2 LIBRE
AVANZAR
PUESTO 2 LIBRE
CINTA B
"ESPERAR"
ZONA CARGA OCUPADA
PUESTO 2 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A VAGÓN
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 2
"huecos en carro cargando"
"ladrillos en carro cargando"


VAGON DE LADRILLOS
PUESTO 2 LIBRE
ROBOT 2
CINTA B
PUESTO 1 LIBRE
ROBOT 3
CINTA 2
BANDEJA PREPARADA


"CARRO CARGADO"
AVANZAR ZONA DE ESPERA CICLO H
FIN AVANZAR
"huecos en zona de espera ciclo H"
"carros en zona de espera ciclo H"
"ESPERAR EN ORIGEN"
"FIN ESPERAR"
TRANSBORDADOR 1
CARRO SUBE A TRANSBORDADOR 1
FIN REGRESAR
FIN SUBIR
REGRESAR A ORIGEN
AVANZAR TRANSBORDADOR 1
FIN AVANZAR
m
8
"CARROS EN CICLO H"
"ladrillos en carro cargando"
"ESPERAR EN ORIGEN"
VAGÓN DE LADRILLOS
TRANSBORDADOR 2
FIN CICLO H
FIN REGRESAR
CARRO SUBE A TRANSBORDADOR 2
FIN SUBIR
"huecos en carro cargando"
REGRESAR A ORIGEN
AVANZAR TRANSBORDADOR 2
m
FIN AVANZAR
AVANZAR HASTA ZONA DE ESP DESCARGA
FIN AVANZAR
"huecos en zona de espera a descarga"
"carros en zona de espera a descarga"
"ladrillos en carro descargando"
ZONA DESCARGA OCUPADA
m
AVANZAR HASTA ZONA DE DESCARGA
ZONA DESCARGA OCUPADA
"ESPERAR DESCARGA CARRO"
ZONA CARGA OCUPADA
AVANZAR A ZONA DE CARGA
ZONA CARGA OCUPADA
"ESPERAR EN ZONA DE CARGA"


CICLO DEL HORNO
TRANSBORDADOR 1
TRANSBORDADOR 2
ZONA DE ESPERA A DESCARGA
ZONA DE ESPERA AL HORNO
CARRO CARGANDO
CARRO DESCARGANDO


"ladrillos en carro descargando"
"ESPERAR"
PUESTO 4 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PUESTO 4
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 4
"nº ladrillos en cinta C"
"huecos en palet"
"ESPERAR"
PALET
PUESTO 5 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PALET
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 5
"ladrillos en palet"
q
"ESPERAR"
PUESTO 5 LIBRE
AVANZAR
PUESTO 5 LIBRE
CINTA C
q
"ESPERAR MIENTRAS CARGAN"
"PALET CARGADO"
AVANZAR ZONA EMPAQUETADO
FIN AVANZAR
COGER PALET CARGADO
FIN COGER
CARGAR PALET VACÍO
FIN CARGAR
REGRESAR A ORIGEN
FIN REGRESAR
PALET
"nº palets producidos"


CARRO DESCARGANDO
ROBOT 4
PUESTO 4 LIBRE
CINTA C
ROBOT 5
PUESTO 5 LIBRE
PALET

6.5. Simulación. Implementación en Matlab

Para realizar estas simulaciones que acabamos de ver no es necesario disponer del PLC. Podemos simularlas fácilmente sobre Matlab siguiendo la metodología expuesta en el capítulo 4. Una vez se analiza y sigue dicha metodología de simulación de las redes (temporizadas o no) en Matlab, resulta sencillo, aunque largo.

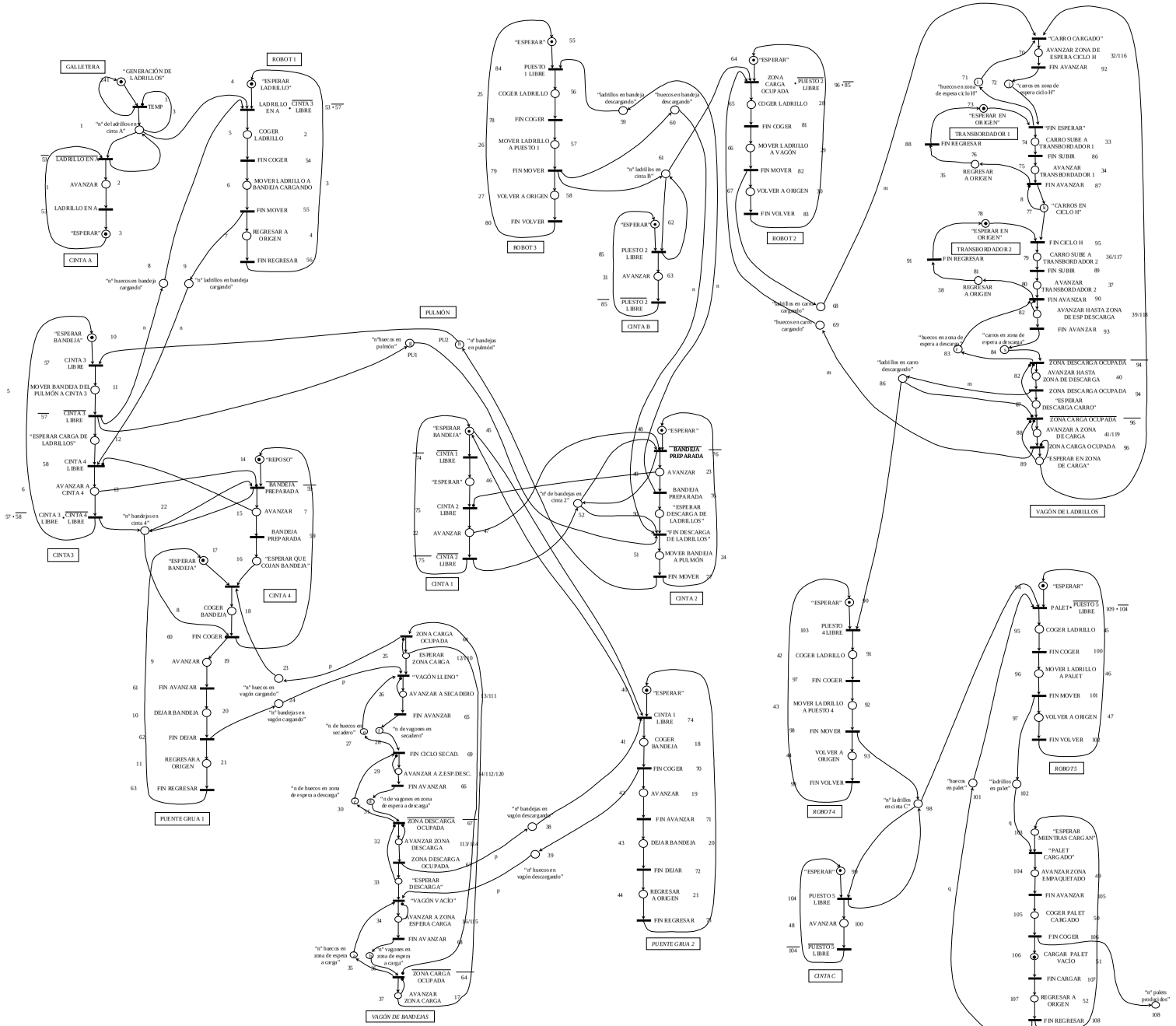
Lo primero que tenemos que hacer es asignar a las entradas y salidas, tanto de la automatización como del modelado del sistema, una asignación de variables (a modo del mapeado de memoria de los PLCs). Pero dado que Matlab trabaja muy cómodamente con matrices, lo mejor es asignarles simplemente el número de orden que ocupan dentro de una matriz. La asignación que se ha realizado se muestra en las figuras 31 y 32, sobre la propia RdP, como se muestran a continuación.

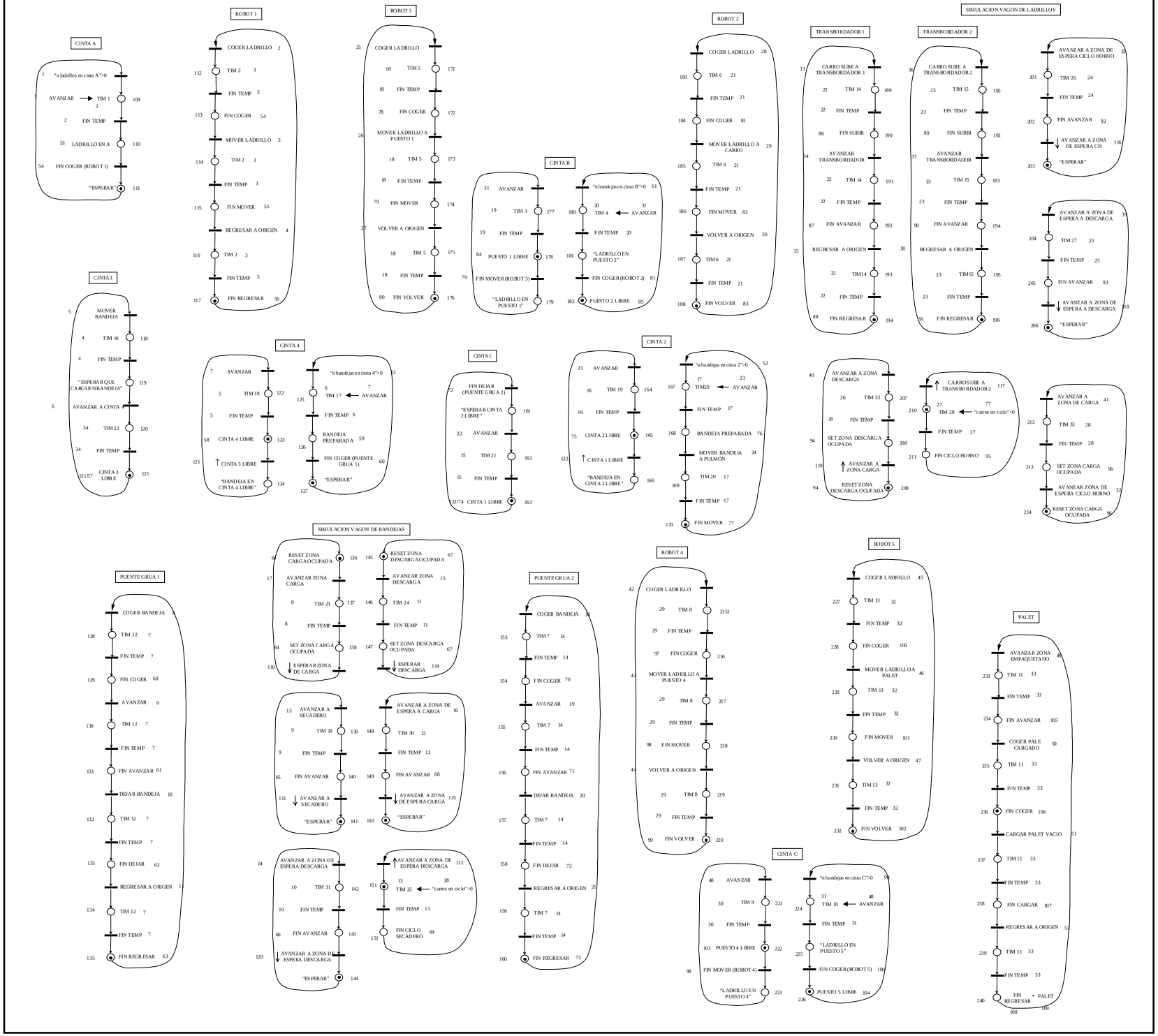
El resultado de los programas que se obtienen (el listado), dada su extensión, se presentarán como otra publicación independiente. En ella aparecen, tras las asignaciones de variables, los listados de cuatro programas: parametros, transiciones, temporizaciones y salidas. A esos cuatro programas se les llama desde la aplicación principal. En realidad se han generado 4 aplicaciones (muy simples) para hacer la simulación como más convenga:

fabrica1	hace una sola simulación
fabrica2	repite simulaciones en el tiempo, en gráficas nuevas, y desecha la información pasada
fabrica3	repite simulaciones, en la misma gráfica, y desecha la información pasada
fabrica 4	hace una simulación larga dividiéndola en varias pequeñas, para ahorrar tiempo y poder simular mayor periodo con menor esfuerzo

Los listados de estas aplicaciones corresponden a las Figuras 33 a 36, y en la gráfica 37 se muestra un resultado de la simulación de todas las variables. Estos gráficos son los que permiten optimizar el proceso productivo.

(Figura 31: Mapeado de variables de automatización en Matlab sobre la RdP) ⇒
(Figura 32: Mapeado de variables del modelado en Matlab sobre la RdP) ⇒


GALLETERA
"GENERACIÓN DE LADRILLOS"
TEMP
"n° de ladrillos en cinta A"
LADRILLO EN A
AVANZAR
LADRILLO EN A
"ESPERAR"
CINTA A
ROBOT 1
"ESPERAR LADRILLO"
LADRILLO EN A
CINTA 5 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A BANDEJA CARGANDO
FIN MOVER
REGRESAR A ORIGEN
FIN REGRESAR
"n° huecos en bandeja cargando"
"n° ladrillos en bandeja cargando"
"ESPERAR"
PUESTO 1 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PUESTO 1
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 3
"ladrillos en bandeja descargando"
"huecos en bandeja descargando"
"n° ladrillos en cinta B"
"ESPERAR"
PUESTO 2 LIBRE
AVANZAR
PUESTO 2 LIBRE
CINTA B
"ESPERAR"
ZONA CARGA OCUPADA
PUESTO 2 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A VAGÓN
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 2
"ladrillos en carro cargando"
"huecos en carro cargando"
"CARRO CARGADO"
AVANZAR ZONA DE ESPERA CICLO H
FIN AVANZAR
"huecos en zona de espera ciclo H"
"carros en zona de espera ciclo H"
"ESPERAR EN ORIGEN"
TRANSBORDADOR 1
FIN REGRESAR
REGRESAR A ORIGEN
"FIN ESPERAR"
CARRO SUBE A TRANSBORDADOR 1
FIN SUBIR
AVANZAR TRANSBORDADOR 1
FIN AVANZAR
"CARROS EN CICLO H"
"ESPERAR EN ORIGEN"
TRANSBORDADOR 2
FIN REGRESAR
REGRESAR A ORIGEN
FIN CICLO H
CARRO SUBE A TRANSBORDADOR 2
FIN SUBIR
AVANZAR TRANSBORDADOR 2
FIN AVANZAR
AVANZAR HASTA ZONA DE ESP DESCARGA
FIN AVANZAR
"huecos en zona de espera a descargar"
"carros en zona de espera a descargar"
ZONA DESCARGA OCUPADA
AVANZAR HASTA ZONA DE DESCARGA
ZONA DESCARGA OCUPADA
"ESPERAR DESCARGA CARRO"
ZONA CARGA OCUPADA
AVANZAR A ZONA DE CARGA
ZONA CARGA OCUPADA
"ESPERAR EN ZONA DE CARGA"
"ladrillos en carro descargando"
VAGÓN DE LADRILLOS
"ESPERAR BANDEJA"
CINTA 3 LIBRE
MOVER BANDEJA DEL PULMÓN A CINTA 3
CINTA 4 LIBRE
"ESPERAR CARGA DE LADRILLOS"
CINTA 4 LIBRE
AVANZAR A CINTA 4
CINTA 3 LIBRE
CINTA 4 LIBRE
CINTA3
PULMÓN
"n°huecos en pulmón"
PU1
PU2
"n° bandejas en pulmón"
"REPOSO"
BANDEJA PREPARADA
AVANZAR
BANDEJA PREPARADA
"ESPERAR QUE COJAN BANDEJA"
CINTA 4
"n° bandejas en cinta 4"
"ESPERAR BANDEJA"
COGER BANDEJA
FIN COGER
AVANZAR
FIN AVANZAR
DEJAR BANDEJA
FIN DEJAR
REGRESAR A ORIGEN
FIN REGRESAR
PUENTE GRUA 1
"ESPERAR BANDEJA"
CINTA 1 LIBRE
"ESPERAR"
CINTA 2 LIBRE
AVANZAR
CINTA 2 LIBRE
CINTA 1
"n° de bandejas en cinta 2"
"ESPERAR"
BANDEJA PREPARADA
AVANZAR
BANDEJA PREPARADA
"ESPERAR DESCARGA DE LADRILLOS"
"FIN DESCARGA DE LADRILLOS"
MOVER BANDEJA A PULMÓN
FIN MOVER
CINTA 2
"ESPERAR"
ZONA CARGA OCUPADA
ESPERAR ZONA CARGA
"VAGÓN LLENO"
AVANZAR A SECADERO
FIN AVANZAR
"n° de huecos en secadero"
"n° de vagones en secadero"
FIN CICLO SECAD.
AVANZAR A Z.ESP.DESC.
FIN AVANZAR
"n° de huecos en zona de espera a descarga"
"n° de vagones en zona de espera a descarga"
ZONA DESCARGA OCUPADA
AVANZAR ZONA DESCARGA
ZONA DESCARGA OCUPADA
"ESPERAR DESCARGA"
"VAGÓN VACÍO"
AVANZAR A ZONA ESPERA CARGA
FIN AVANZAR
"n° huecos en zona de espera a carga"
"n° vagones en zona de espera a cargo"
ZONA CARGA OCUPADA
AVANZAR ZONA CARGA
VAGÓN DE BANDEJAS
"n° huecos en vagón cargando"
"n° bandejas en vagón cargando"
"n° bandejas en vagón descargando"
"n° huecos en vagón descargando"
"ESPERAR"
CINTA 1 LIBRE
COGER BANDEJA
FIN COGER
AVANZAR
FIN AVANZAR
DEJAR BANDEJA
FIN DEJAR
REGRESAR A ORIGEN
FIN REGRESAR
PUENTE GRUA 2
"ESPERAR"
PUESTO 4 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PUESTO 4
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 4
"n° ladrillos en cinta C"
"ESPERAR"
PUESTO 5 LIBRE
AVANZAR
PUESTO 5 LIBRE
CINTA C
"ESPERAR"
PALET
PUESTO 6 LIBRE
COGER LADRILLO
FIN COGER
MOVER LADRILLO A PALET
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
ROBOT 5
"huecos en palet"
"ladrillos en palet"
"ESPERAR MIENTRAS CARGAN"
"PALET CARGADO"
AVANZAR ZONA EMPAQUETADO
FIN AVANZAR
COGER PALET CARGADO
FIN COGER
CARGAR PALET VACÍO
FIN CARGAR
REGRESAR A ORIGEN
FIN REGRESAR
"n° palets producidos"
PALET


CINTA A
"n ladrillos en cinta A">0
AVANZAR
TIM 1
FIN TEMP
LADRILLO EN A
FIN COGER (ROBOT 1)
"ESPERAR"
ROBOT 1
COGER LADRILLO
TIM 2
FIN TEMP
FIN COGER
MOVER LADRILLO
FIN MOVER
REGRESAR A ORIGEN
FIN REGRESAR
ROBOT 3
COGER LADRILLO
MOVER LADRILLO A PUESTO 1
VOLVER A ORIGEN
FIN VOLVER
CINTA B
"n bandejas en cinta B">0
FIN COGER (ROBOT 2)
PUESTO 2 LIBRE
"LADRILLO EN PUESTO 1"
ROBOT 2
COGER LADRILLO
MOVER LADRILLO A CARRO
FIN MOVER
VOLVER A ORIGEN
FIN VOLVER
SIMULACION VAGON DE LADRILLOS
TRANSBORDADOR 1
CARRO SUBE A TRANSBORDADOR 1
AVANZAR TRANSBORDADOR
REGRESAR A ORIGEN
FIN REGRESAR
TRANSBORDADOR 2
CARRO SUBE A TRANSBORDADOR 2
AVANZAR TRANSBORDADOR
REGRESAR A ORIGEN
FIN REGRESAR
AVANZAR A ZONA DE ESPERA CICLO HORNO
AVANZAR A ZONA DE ESPERA CH
"ESPERAR"
AVANZAR A ZONA DE ESPERA A DESCARGA
AVANZAR A ZONA DE ESPERA A DESCARGA
"ESPERAR"
CINTA3
MOVER BANDEJA
"ESPERAR QUE CARGUEN BANDEJA"
AVANZAR A CINTA 4
CINTA 3 LIBRE
CINTA 4
AVANZAR
CINTA 4 LIBRE
CINTA 3 LIBRE
"BANDEJA EN CINTA 4 LIBRE"
"n bandejas en cinta 4">0
FIN COGER (PUENTE GRUA 1)
"ESPERAR"
CINTA 1
FIN DEJAR (PUENTE GRUA 2)
"ESPERAR CINTA 2 LIBRE"
AVANZAR
CINTA 1 LIBRE
CINTA 2
AVANZAR
CINTA 2 LIBRE
CINTA 1 LIBRE
"BANDEJA EN CINTA 2 LIBRE"
"n bandejas en cinta 2">0
FIN MOVER
AVANZAR A ZONA DESCARGA
SET ZONA DESCARGA OCUPADA
RESET ZONA DESCARGA OCUPADA
AVANZAR A ZONA CARGA
CARRO SUBE A TRANSBORDADOR 2
"carros en ciclo">0
FIN CICLO HORNO
AVANZAR A ZONA DE CARGA
SET ZONA CARGA OCUPADA
AVANZAR ZONA DE ESPERA CICLO HORNO
RESET ZONA CARGA OCUPADA
SIMULACION VAGON DE BANDEJAS
RESET ZONA CARGA OCUPADA
AVANZAR ZONA CARGA
SET ZONA CARGA OCUPADA
ESPERAR ZONA DE CARGA
RESET ZONA DESCARGA OCUPADA
AVANZAR ZONA DESCARGA
SET ZONA DESCARGA OCUPADA
ESPERAR DESCARGA
AVANZAR A SECADERO
"ESPERAR"
AVANZAR A ZONA DE ESPERA A CARGA
"ESPERAR"
AVANZAR A ZONA DE ESPERA DESCARGA
"ESPERAR"
AVANZAR A ZONA DE ESPERA DESCARGA
"carros en ciclo">0
FIN CICLO SECADERO
PUENTE GRUA 1
COGER BANDEJA
FIN COGER
AVANZAR
DEJAR BANDEJA
REGRESAR A ORIGEN
FIN REGRESAR
PUENTE GRUA 2
COGER BANDEJA
FIN COGER
AVANZAR
DEJAR BANDEJA
REGRESAR A ORIGEN
FIN REGRESAR
ROBOT 4
COGER LADRILLO
MOVER LADRILLO A PUESTO 4
VOLVER A ORIGEN
FIN VOLVER
CINTA C
AVANZAR
PUESTO 4 LIBRE
"LADRILLO EN PUESTO 4"
"n bandejas en cinta C">0
"LADRILLO EN PUESTO 5"
FIN COGER (ROBOT 5)
PUESTO 5 LIBRE
ROBOT 5
COGER LADRILLO
MOVER LADRILLO A PALET
VOLVER A ORIGEN
FIN VOLVER
PALET
AVANZAR ZONA EMPAQUETADO
COGER PALE CARGADO
CARGAR PALET VACIO
FIN CARGAR
REGRESAR A ORIGEN
FIN REGRESAR
PALET

```
clear; clear all;
parametros; %programa con los parámetros del proceso
close;
hold off;
tf=10;
tiempo=0;
i=1;
tspan(1)=tiempo;
while tiempo<=tf
   i=i+1;
   Y(i,:)=Y(i-1,:);
   tspan(i)=tiempo;
   seguir=0; %indica que no hay disparos si no se aumenta el tiempo

   transiciones;%test y disparo de transiciones

   if seguir==0
      tiempo=tiempo+paso;
      i=i-1; %elimino la última fila, que no ha cambiado
   end

   tempor;%actualización de los temporizadores

   salidas;%actualización de las salidas

end %del while
tiempo=tiempo-paso;

subplot(3,1,1), plot(tspan,Y(:,241),'r');
subplot(3,1,2), plot(tspan,Y(:,108),'b');
subplot(3,1,3), plot(tspan,Y(:,8),'g');
```

Figura 33: Programa "Fabrica1" sobre Matlab

```
clear; clear all;
parametros; %programa con los parámetros del proceso
close;
%hold on;
%tf=1000;

tiempo=0;
repetir='si';
while repetir~='no' %While de repetir otro periodo de simulación

tf=input('Periodo de tiempo a evaluar');
tf=tf+tiempo;
i=1;
tspan(1)=tiempo;
while tiempo<=tf %While de iterar para la simulación
   i=i+1;
   Y(i,:)=Y(i-1,:);
   tspan(i)=tiempo;
   seguir=0; %indica que no hay disparos si no se aumenta el tiempo

   transiciones;%test y disparo de transiciones

   if seguir==0
      tiempo=tiempo+paso;
      i=i-1; %elimino la última fila, que no ha cambiado
```

```
   end

   tempor;%actualización de los temporizadores

   salidas;%actualización de las salidas

end %del while
tiempo=tiempo-paso;
%subplot(3,1,1), plot(tspan,Y(:,241),'r');
%subplot(3,1,2), plot(tspan,Y(:,108),'b');
%subplot(3,1,3), plot(tspan,Y(:,8),'g');

figure;
subplot(1,1,1), plot(tspan,Y);


repetir=input('¿Quieres seguir simulando? ','s');
%preparo la nueva Y para la siguiente iteración
if  repetir ~= 'no'
   [filasY,columY]=size(Y);
   Yf=[Y(filasY,:)];
   clear Y;clear tspan;
   Y=Yf;
end % del if

end %del while
```

Figura 34: Programa "Fabrica2" sobre Matlab

```
clear; clear all;
parametros; %programa con los parámetros del proceso
close;
hold on;
%tf=1000;

tiempo=0;
repetir='si';
while repetir~='no' %While de repetir otro periodo de simulación

tf=input('Periodo de tiempo a evaluar');
tf=tf+tiempo;
i=1;
tspan(1)=tiempo;
while tiempo<=tf %While de iterar para la simulación
   i=i+1;
   Y(i,:)=Y(i-1,:);
   tspan(i)=tiempo;
   seguir=0; %indica que no hay disparos si no se aumenta el tiempo

   transiciones;%test y disparo de transiciones

   if seguir==0
      tiempo=tiempo+paso;
      i=i-1; %elimino la última fila, que no ha cambiado
   end

   tempor;%actualización de los temporizadores

   salidas;%actualización de las salidas
```

```
end %del while
tiempo=tiempo-paso;
%subplot(3,1,1), plot(tspan,Y(:,241),'r');
%subplot(3,1,2), plot(tspan,Y(:,108),'b');
%subplot(3,1,3), plot(tspan,Y(:,8),'g');

%figure;
subplot(1,1,1), plot(tspan,Y);


repetir=input('¿Quieres seguir simulando? ','s');
%preparo la nueva Y para la siguiente iteración
if  repetir ~= 'no'
   [filasY,columY]=size(Y);
   Yf=[Y(filasY,:)];
   clear Y;clear tspan;
   Y=Yf;
end % del if

end %del while
```

Figura 35: Programa "Fabrica3" sobre Matlab

```
clear; clear all;
parametros; %programa con los parámetros del proceso
close;
hold on;
%tf=1000;

tiempo=0;
repetir='si';
while repetir~='no' %While de repetir otro periodo de simulación

   %tf=input('Periodo de tiempo a evaluar');
tf=50;
tf=tf+tiempo;
i=1;
tspan(1)=tiempo;
while tiempo<=tf %While de iterar para la simulación
   i=i+1;
   Y(i,:)=Y(i-1,:);
   tspan(i)=tiempo;
   seguir=0; %indica que no hay disparos si no se aumenta el tiempo

   transiciones;%test y disparo de transiciones

   if seguir==0
      tiempo=tiempo+paso;
      i=i-1; %elimino la última fila, que no ha cambiado
   end

   tempor;%actualización de los temporizadores

   salidas;%actualización de las salidas

end %del while
tiempo=tiempo-paso;
%subplot(3,1,1), plot(tspan,Y(:,241),'r');
```

```
%subplot(3,1,2), plot(tspan,Y(:,108),'b');
%subplot(3,1,3), plot(tspan,Y(:,8),'g');

%figure;
subplot(1,1,1), plot(tspan,Y);


%repetir=input('¿Quieres seguir simulando? ','s');
if tiempo <=400
   repetir='si';
else
   repetir='no';
end

%preparo la nueva Y para la siguiente iteración
if  repetir ~= 'no'
   [filasY,columY]=size(Y);
   Yf=[Y(filasY,:)];
   clear Y;clear tspan;
   Y=Yf;
end % del if

end %del while
```

Figura 36: Programa "Fabrica4" sobre Matlab

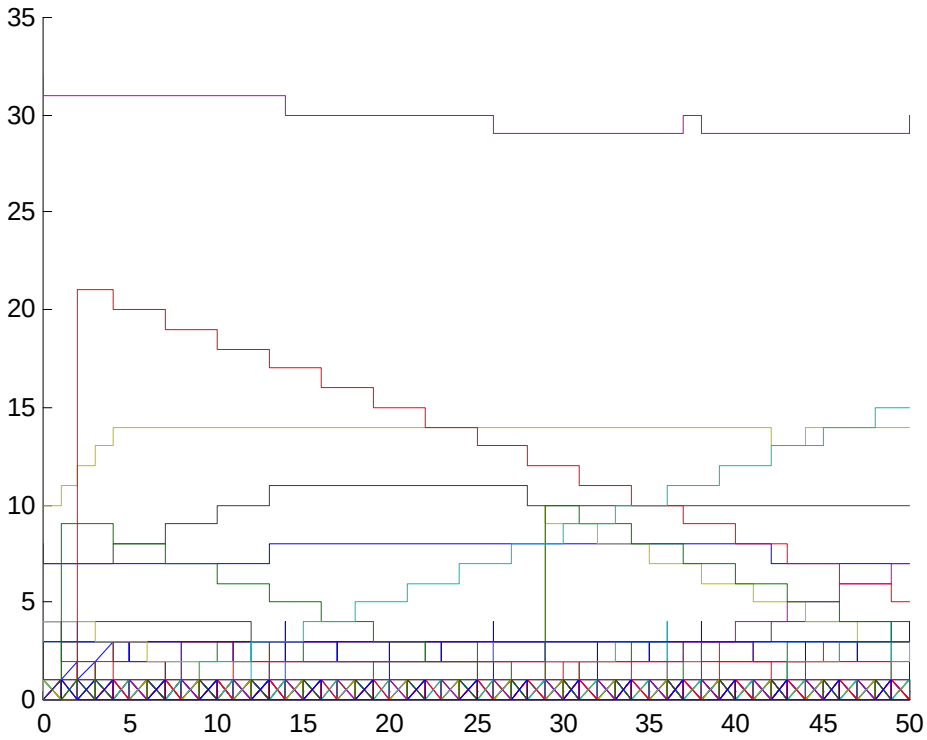


Figura 37: Resultado de la simulación en Matlab del sistema y la automatización

6.6 Implementación en SCADA

La implementación en el SCADA de toda la automatización que acabamos de ver, incluida la simulación, permite operar con mucha mayor comodidad y eficacia en la fase de producción, pero no sólo eso, sino que además se convierte en una herramienta imprescindible para la elaboración y sobre todo la depuración de la automatización: (tanto del modelado como de la implementación en el PLC).

Resulta imposible incluir todo el listado de las aplicaciones generadas en el SCADA empleado (en este caso FIX D-MAX, de OMRON), incluso en un anexo, por la gran extensión que tiene. Por ello se tratará de indicar simplemente algunas partes que resulten de interés.

6.6.1. Las aplicaciones en el SCADA en general

Las aplicaciones en el SCADA, como se ha visto en algún ejemplo sencillo anteriormente, se basan en la *Base de Datos* de la aplicación, y a partir de ahí se construyen los *Bloques* (que vienen a corresponder con las variables de un lenguaje de programación).

Un resumen de los bloques empleados en una de las partes es el siguiente:

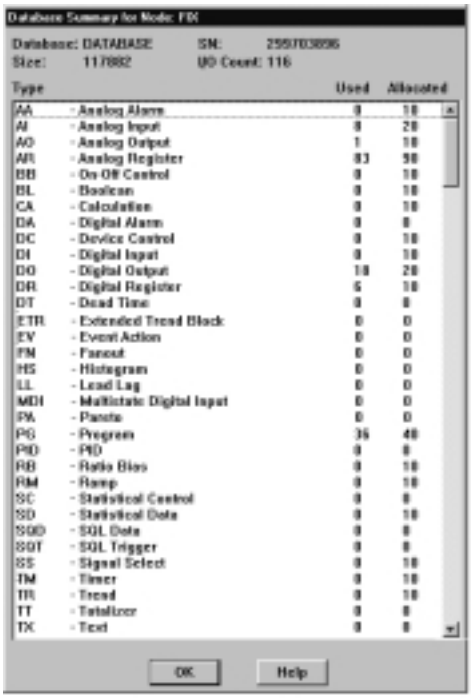


Figura 38: Resumen de la Base de Datos de una aplicación SCADA.

Tan sólo el listado de dicha base de datos ocupa más de 10 hojas (el resto muchos cientos). La forma de programarlo es muy similar a los lenguajes orientados a objetos, rellenando las casillas de las propiedades de los bloques. Incluso las programaciones se realizan rellenando un bloque tipo programa (Figura 39)

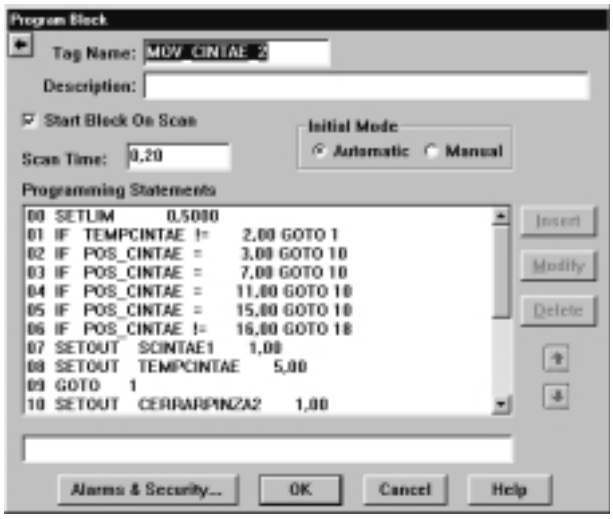


Figura 39: Ventana de bloque tipo programa.

6.6.2. Monitorización con SCADA

Aunque una de las aportaciones de esta investigación consiste en emplear los SCADAS para automatizar (automatización redundante supervisora), no hay que olvidar todos los cometidos que tienen en su funcionamiento habitual (ver tema 1). A tal respecto se muestran las figuras siguientes en las que se muestra, de la forma más resumida posible, algunas de las aplicaciones implementadas.

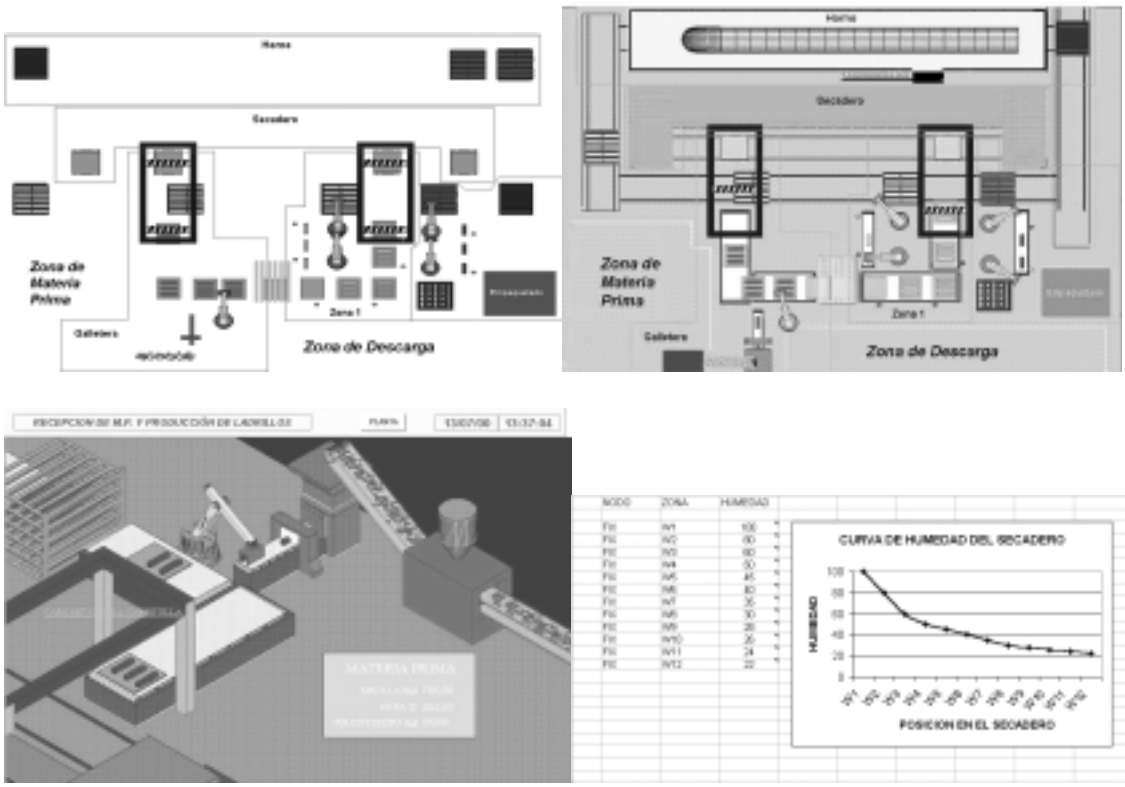


Figura 40: Pantallas de monitorización de datos con el SCADA.

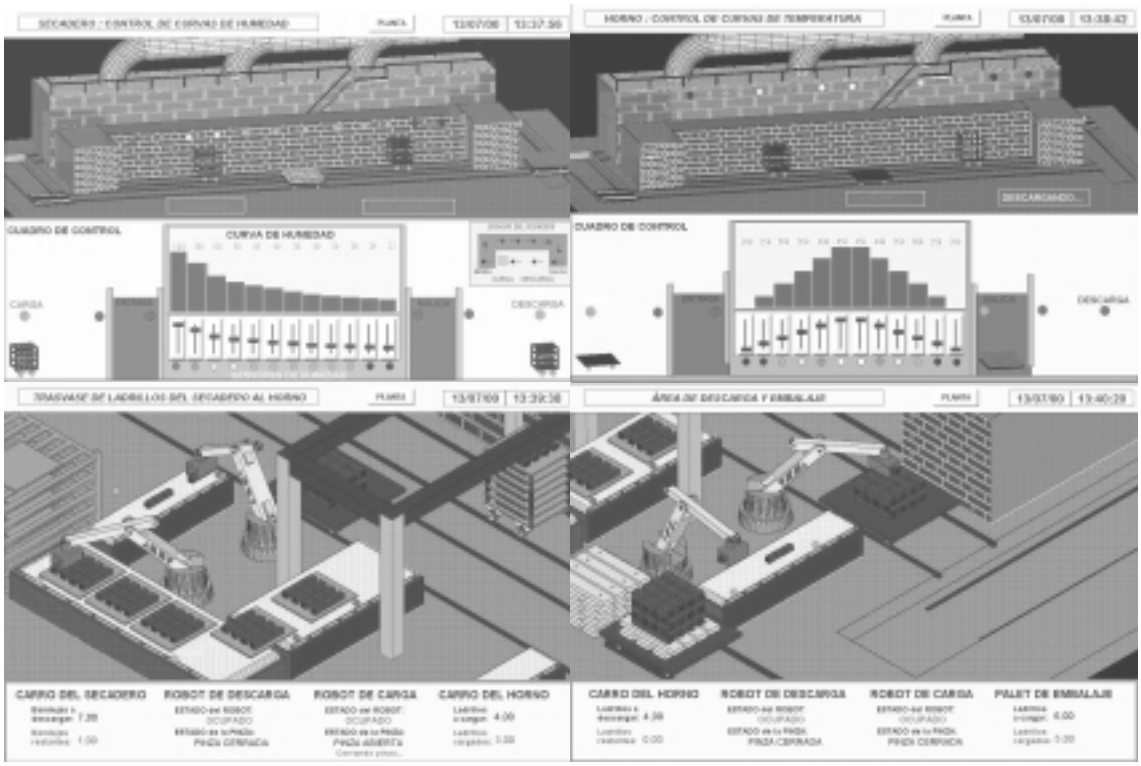


Figura 41: Pantallas de monitorización de zonas de trabajo

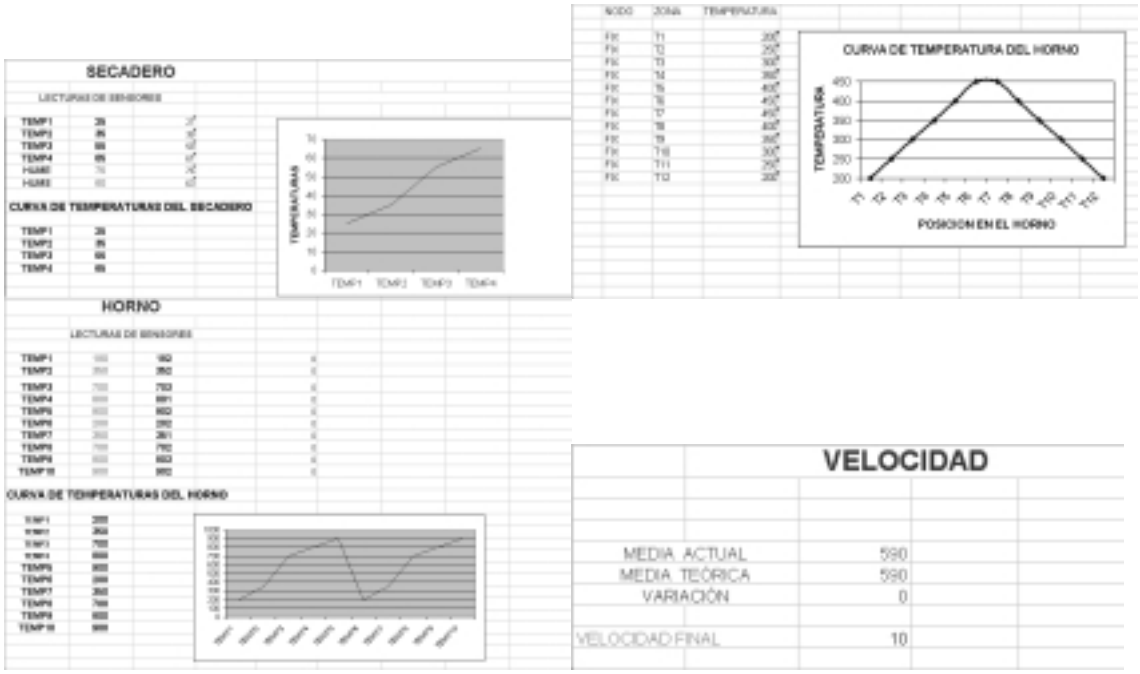
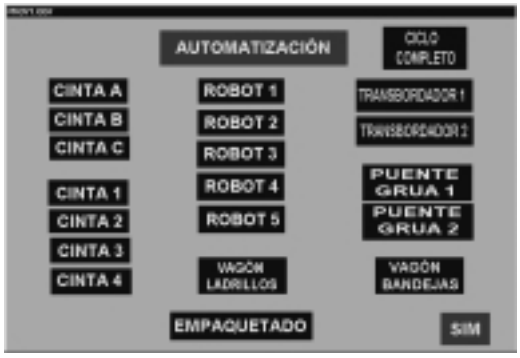
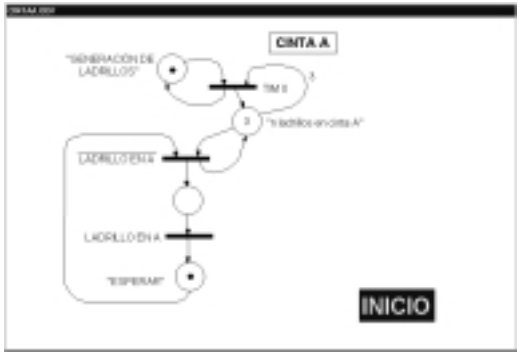


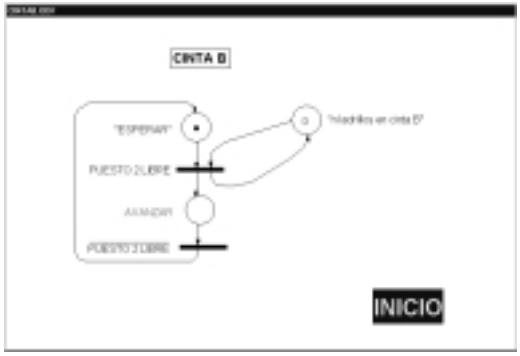
Figura 42: Pantallas de Hoja de Cálculo y Base de Datos conectadas a la aplicación SCADA

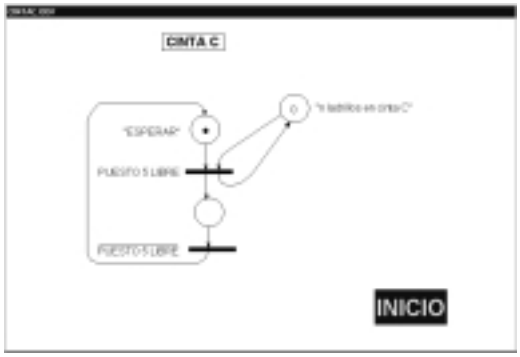
6.6.2. Control con el SCADA

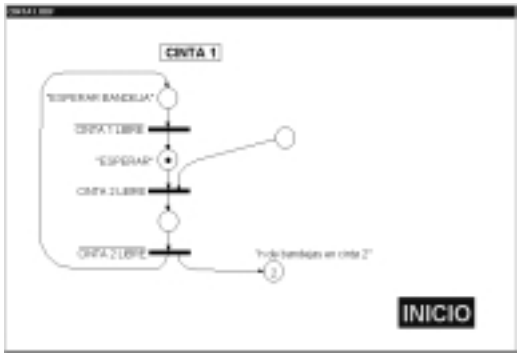
Y como también se ha indicado a lo largo de este trabajo, los SCADAS se pueden emplear para dotar al sistema de nuevas prestaciones adicionales. De nuevo sería inviable presentar las automatizaciones realizadas a tal fin, por lo que se presentan tan sólo las pantallas de las aplicaciones empleadas para ello (Figuras 42). En este caso se comprueba lo que se indicaba en el capítulo 2 de que suele ser preferible monitorizar y controlar el sistema mediante su RdP, y si se quiere se puede ampliar con pantallas gráficas que representen una vista de la automatización (como teníamos en la Figura 41).

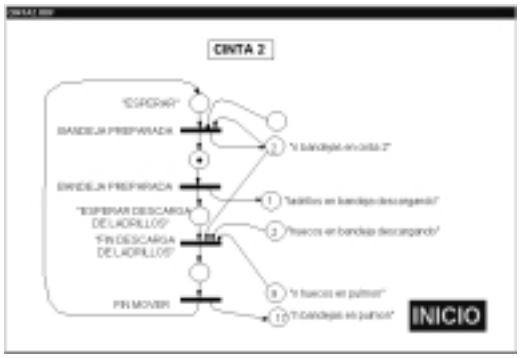


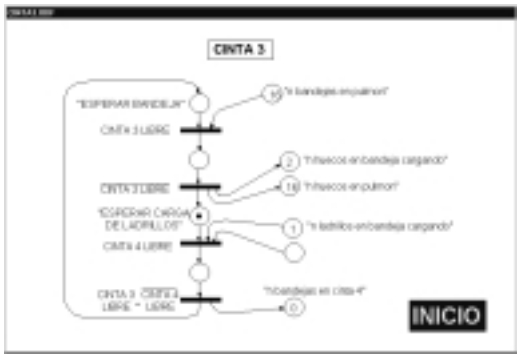


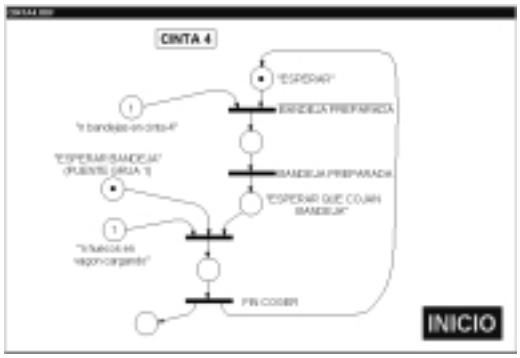


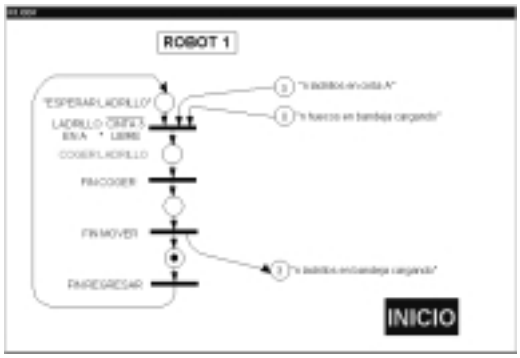


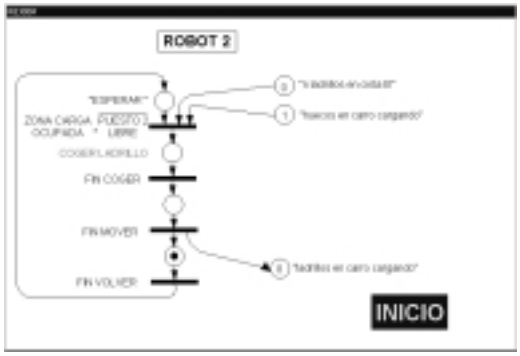


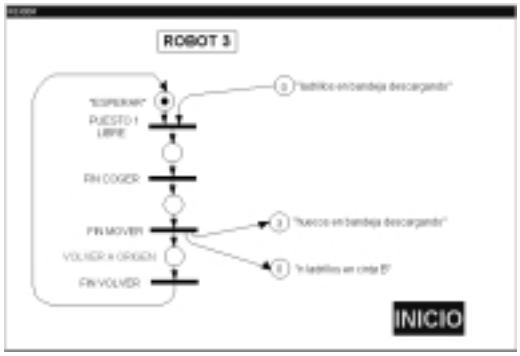

CINTA 2
"ESPERAR"
BANDEJA PREPARADA
"n bandejas en cinta Z"
BANDEJA PREPARADA
"ladrillos en bandeja descargando"
"ESPERAR DESCARGA DE LADRILLOS"
"huecos en bandeja descargando"
"FIN DESCARGA DE LADRILLOS"
"n huecos en pulmon"
FIN MOVER
"n bandejas en pulmon"
INICIO

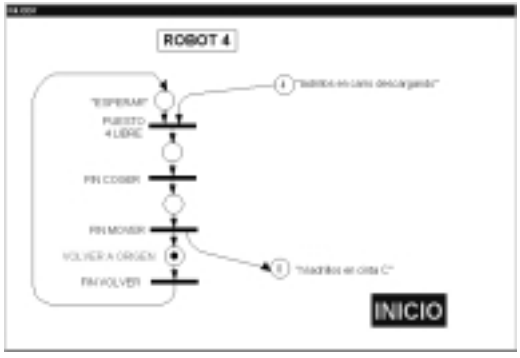

CINTA 3
"n bandejas en pulmon"
"ESPERAR BANDEJA"
CINTA 3 LIBRE
"huecos en bandeja cargando"
CINTA 3 LIBRE
"huecos en pulmon"
"ESPERAR CARGA DE LADRILLOS"
"n ladrillos en bandeja cargando"
CINTA 4 LIBRE
"n bandejas en cinta 4"
INICIO

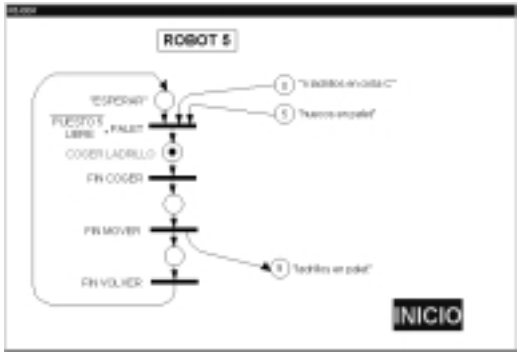

CINTA 4
"ESPERAR"
"n bandejas en cinta 4"
BANDEJA PREPARADA
"ESPERAR BANDEJA" (PUENTE GRUA 1)
BANDEJA PREPARADA
"ESPERAR QUE COJAN BANDEJA"
"n huecos en vagon cargando"
FIN COGER
INICIO


ROBOT 1
"ESPERAR LADRILLO"
"n ladrillos en cinta A"
"n huecos en bandeja cargando"
COGER LADRILLO
FIN COGER
FIN MOVER
"n ladrillos en bandeja cargando"
FIN REGRESAR
INICIO

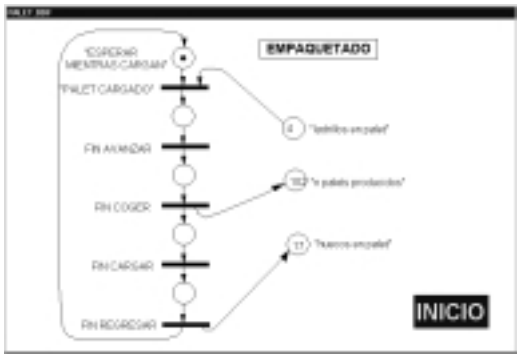

ROBOT 2
"ESPERAR"
"n ladrillos en cinta B"
"huecos en carro cargando"
COGER LADRILLO
FIN COGER
FIN MOVER
"ladrillos en carro cargando"
FIN VOLVER
INICIO

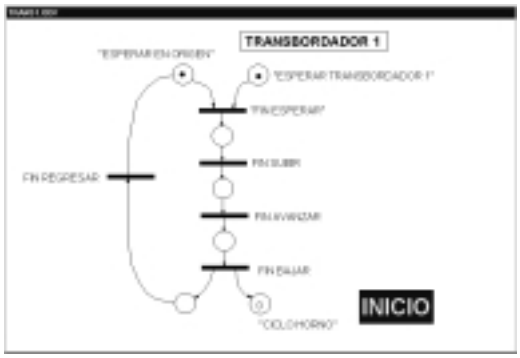

ROBOT 3
"ESPERAR"
"ladrillos en bandeja descargando"
FIN COGER
FIN MOVER
"huecos en bandeja descargando"
VOLVER A ORIGEN
"n ladrillos en cinta E"
FIN VOLVER
INICIO

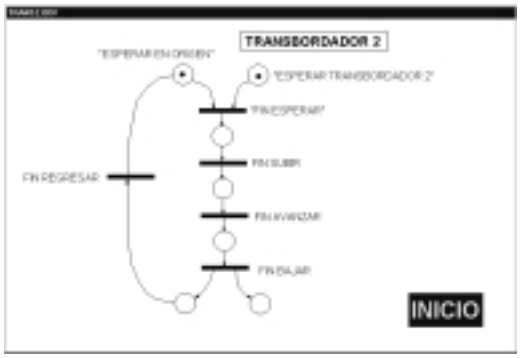

ROBOT 4
"ESPERAR"
"ladrillos en carro descargando"
FIN COGER
FIN MOVER
VOLVER A ORIGEN
"ladrillos en cinta C"
FIN VOLVER
INICIO

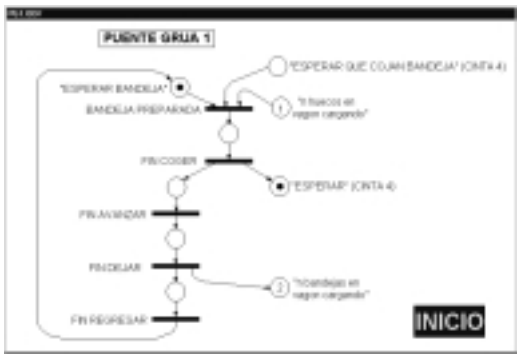

ROBOT 5
"ESPERAR"
"n ladrillos en cinta C"
"huecos en palet"
COGER LADRILLO
FIN COGER
FIN MOVER
"ladrillos en palet"
FIN VOLVER
INICIO

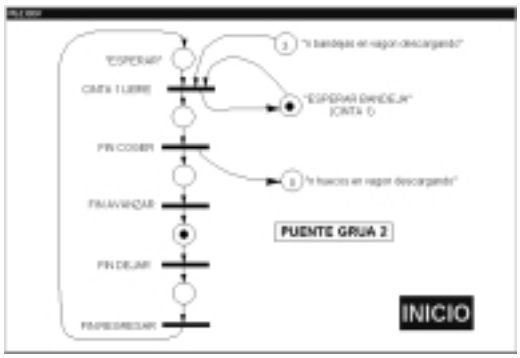

VAGON DE LADRILLOS
INICIO


EMPAQUETADO
INICIO

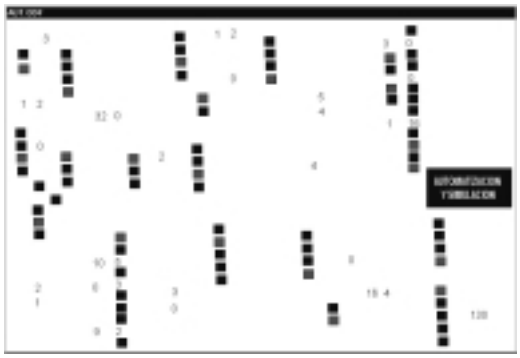

TRANSBORDADOR 1
INICIO

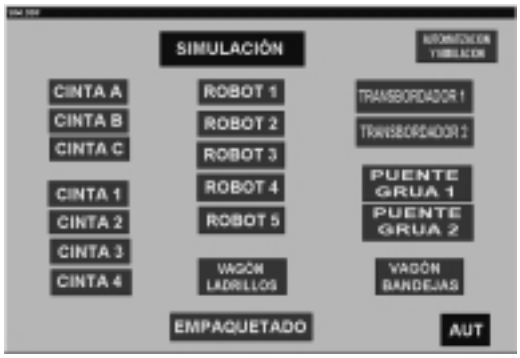

TRANSBORDADOR 2
INICIO

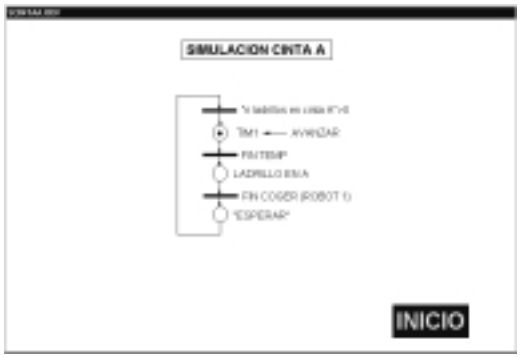

PUENTE GRUA 1
INICIO

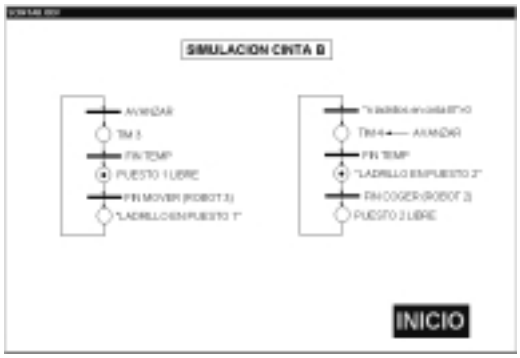

PUENTE GRUA 2
INICIO

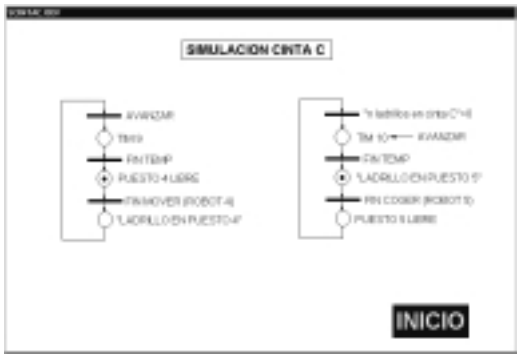

VAGON DE BANDEJAS
INICIO

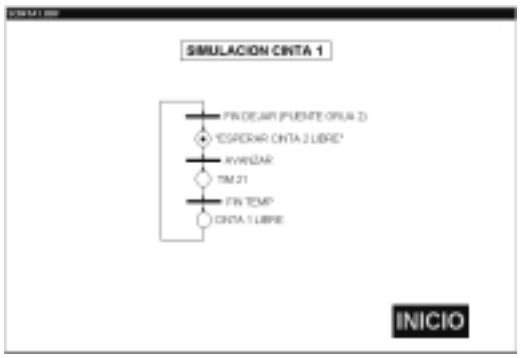


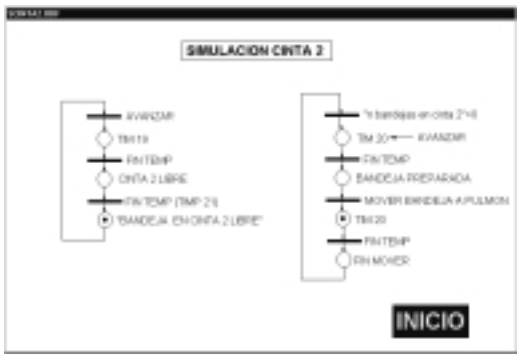

SIMULACIÓN
CINTA A
CINTA B
CINTA C
CINTA 1
CINTA 2
CINTA 3
CINTA 4
ROBOT 1
ROBOT 2
ROBOT 3
ROBOT 4
ROBOT 5
VAGÓN LADRILLOS
EMPAQUETADO
TRANSBORDADOR 1
TRANSBORDADOR 2
PUENTE GRUA 1
PUENTE GRUA 2
VAGÓN BANDEJAS
AUT

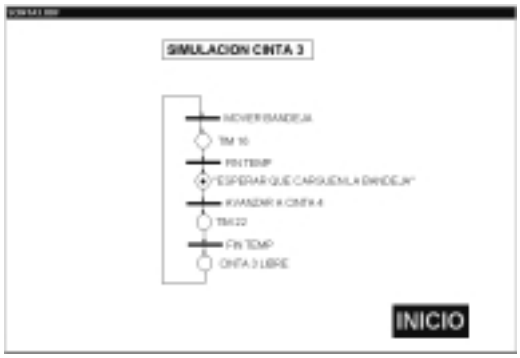

SIMULACION CINTA A
INICIO

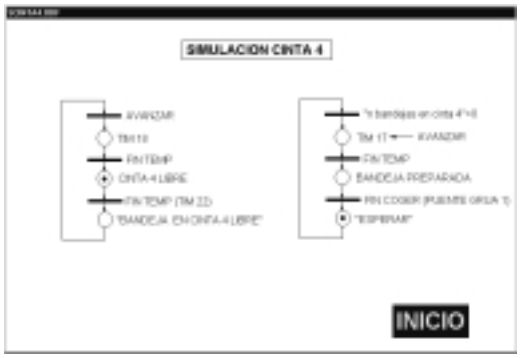

SIMULACION CINTA B
INICIO

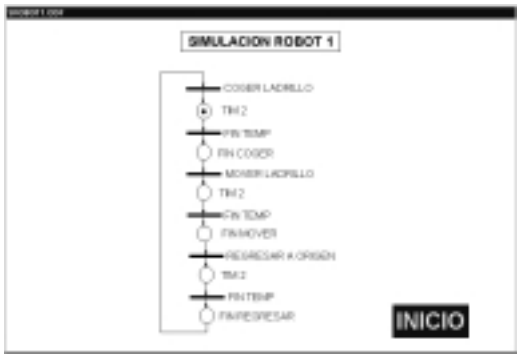

SIMULACION CINTA C
INICIO

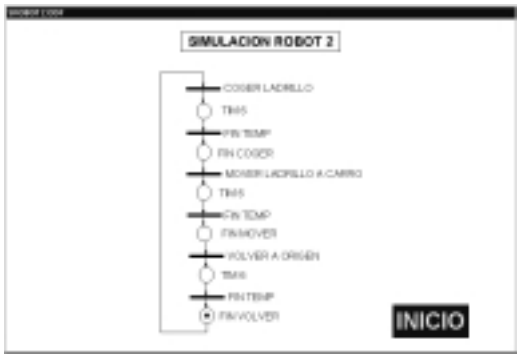

SIMULACION CINTA 1
INICIO

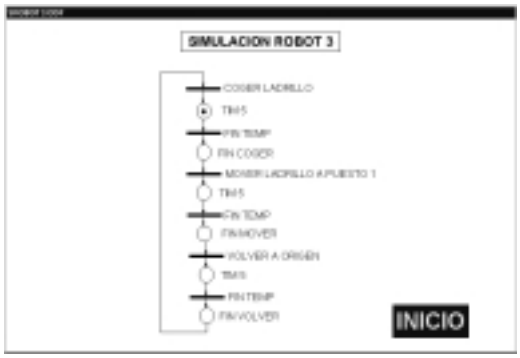

SIMULACION CINTA 2
INICIO

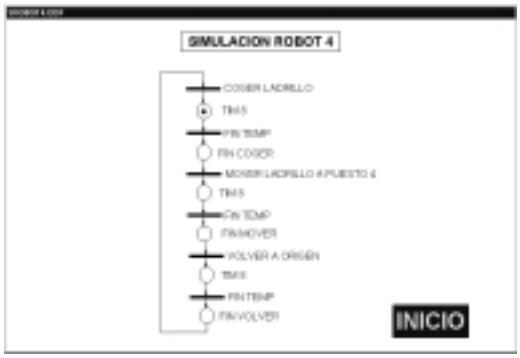

SIMULACION CINTA 3
INICIO

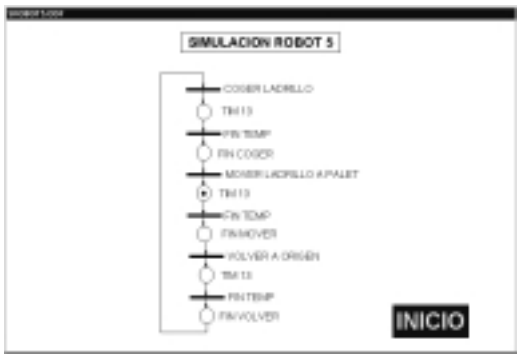

SIMULACION CINTA 4
INICIO


SIMULACION ROBOT 1
INICIO

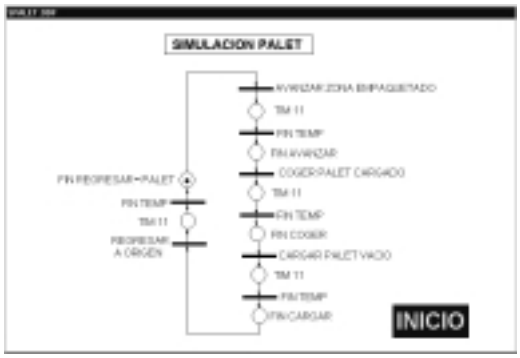

SIMULACION ROBOT 2
INICIO

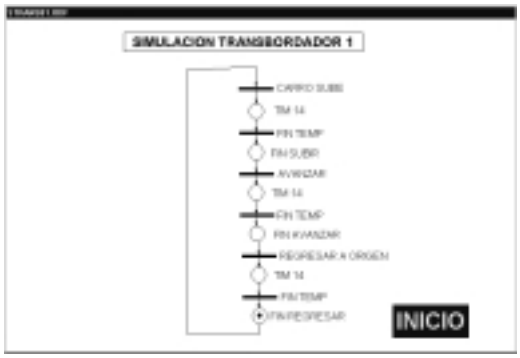

SIMULACION ROBOT 3
INICIO

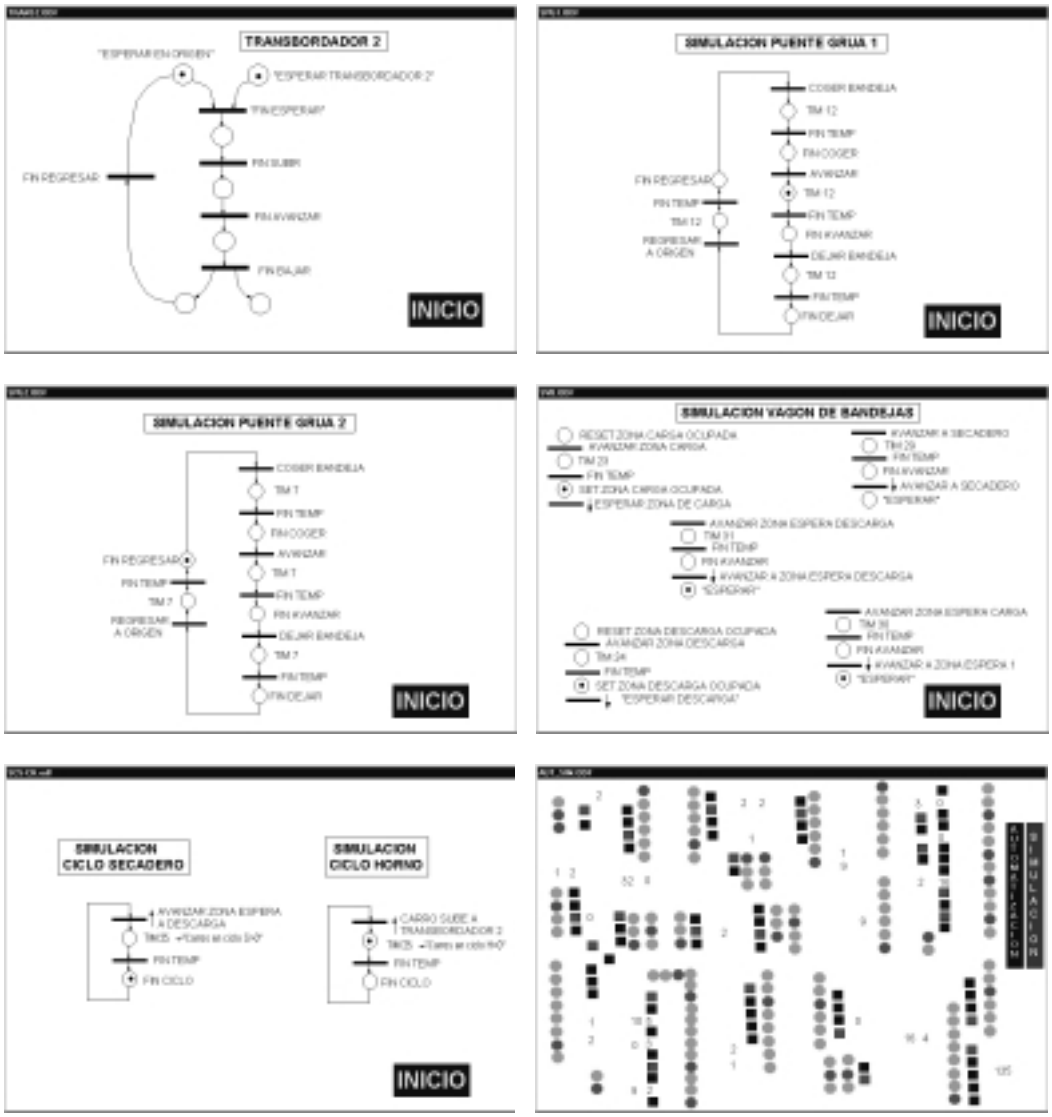

SIMULACION ROBOT 4
INICIO


SIMULACION ROBOT 5
INICIO


SIMULACION VAGON DE LADRILLOS
INICIO


SIMULACION PALET
INICIO


SIMULACION TRANSBORDADOR 1
INICIO



Figuras 42: Pantallas de control del proceso mediante SCADA.

También conviene indicar que es posible la integración de estos sistemas de monitorización con otros de Realidad Virtual, surgiendo entonces lo que se denomina la fábrica virtual. Este nuevo concepto de fábrica virtual tiene más sentido en otro tipo de fábricas en las que las maniobras sean mucho más arriesgadas y un error pueda ocasionar grandes perjuicios, pero se ha considerado interesante exponer su existencia. Algo similar se está realizando a modo de ensayo con la misma planta industrial en otra línea de investigación, más relacionada con la ingeniería gráfica.

De todas formas también pueden integrarse en las aplicaciones SCADA elementos desarrollados en entornos gráficos más avanzados, con lo que la automatización-monitorización-simulación se asemeja mucho a la fábrica virtual [8] (Figura 43).

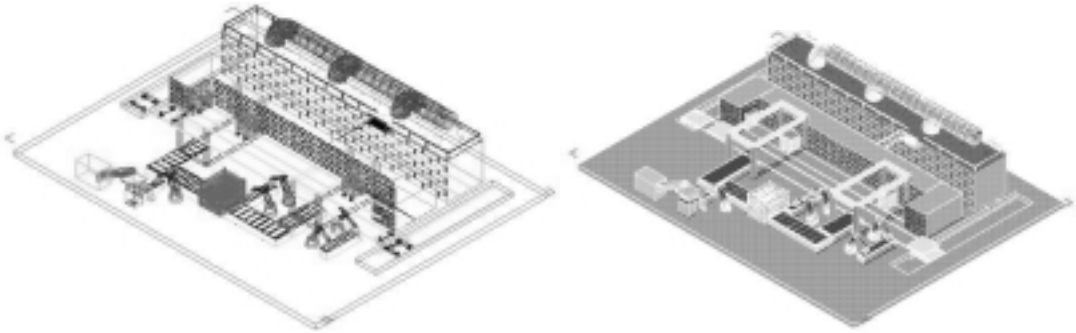


Figura 43: Aplicación de fábrica virtual en la planta de cerámica.

6.7. Simuladores de herramientas gráficas

Aunque sólo sea de pasada también puede ser oportuno indicar que los simuladores de herramientas gráficas que se han comentado en temas anteriores han sido empleados en esta aplicación práctica que culmina este trabajo de investigación. Evidentemente la simulación ha proporcionado los mismos resultados que la realizada con el PLC y la del SCADA. En la figura 44 se muestra la pantalla del editor-simulador de RdP empleado en el que se ha introducido la automatización que hemos desarrollado.

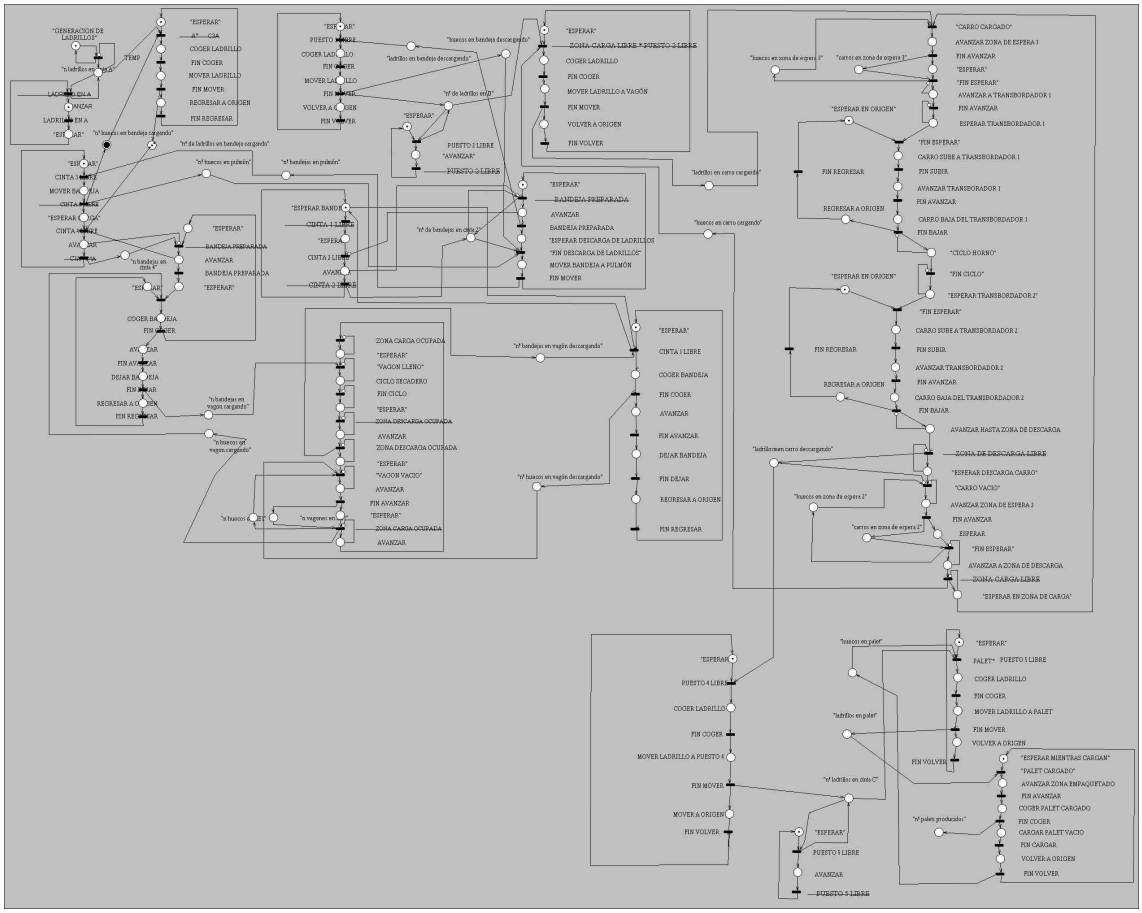


Figura 44: Simulación de la automatización en programa editor-simulador de RdP

6.8. Resultados

Como resultados del presente apartado hay que destacar por un lado la aportación a la metodología de análisis de sistemas en los que el material se va moviendo de unos ciclos a otros, todos ellos interconectados, y en especial cuando se quiera realizar la trazabilidad del producto (saber de cada elemento producido los sitios en los que ha estado y los elementos con los que se ha relacionado). De esta manera es muy fácil detectar las causas de partidas defectuosas, que de otra forma podrían ser muy difíciles de detectar

Por otro lado en la segunda parte de este capítulo se ha intentado resumir la implementación llevada a cabo en una fábrica colaboradora con la línea de investigación para comprobar y matizar los resultados que han ido obteniéndose las labores investigadoras. Por tanto los resultados de este apartado se funden con los de cada uno de los apartados anteriores.

6.9. Bibliografía

1. Burns A., Wellings A.:Real-Time Systems and Programming Languages. Add.-Wesley, California, 1996
2. David, R.: Grafcet: A powerful tool for specification of logic controllers. IEEE Trans on Control Systems Technology (1995) 3 (3) , 253-268
3. Jiménez, E., Miruri, JM., Martínez de Pisón, J.F., Gil, M.: Supervised Real-Time Control with PLCs and SCADA in Ceramic Plant. 6th IFAC Workshop on Algorithms and Architectures for Real-Time Control (2000) 1 , 221-226
4. Morriss, B.: Programmable Logic Controllers. Prentice Hall, New Jersey (1999)
5. N. Bhandari, D.K. Rollins. Superior Semi-Empirical Dynamic Predictive Modeling that Addresses Interactions. IASTED Intelligent Systems and Control ISC'99 (1999)
6. N. Konstas, S. Lloyd, H. Yu, C. Chatwin. Generic Net Modelling Framework for Petri Nets. IASTED Intelligent Systems and Control ISC'99 (1999)
7. W.T. Goh, Z. Zhang. Autonomous Petri-Net for Manufacturing System Modelling in an Agile Manufacturing Environment. IASTED International Confer. Robotics and Applications 1999.
8. Félix Sánz, Emilio Jiménez. Expresión Gráfica en Ingeniería de Sistemas y Automática. JA2001, Barcelona, 2001.

Tema 7

RESULTADOS, APORTACIONES Y PROYECCIÓN.

Este trabajo presenta un análisis y una metodología para la implementación eficiente y robusta, empleando PLCs y sistemas SCADAs, de automatizaciones complejas en plantas industriales con procesos compuestos por sucesivos ciclos con una fuerte interconexión.

Como ya hemos comentado anteriormente las aportaciones están expuestas por separado en cada capítulo, e incluso en algún capítulo en cada sección, cuando conviene destacar ciertos detalles dentro del tema del capítulo. Sin embargo aquí haremos un sucinto repaso a dichas aportaciones para tener una visión global de todas ellas, y para poder profundizar posteriormente en la que nos interese acudiendo al capítulo correspondiente.

Como resultado global se puede considerar que se ha cumplido, con un muy alto grado de satisfacción, con los objetivos planteados, si bien eso supone, lejos de cerrar la investigación, haber abierto la puerta a investigaciones similares en otros tipos de procesos en los que la investigación requiera de otros enfoques o lleve a otros resultados. A ese respecto decir que se eligió como modelo de industria sobre la que realizar el desarrollo de las investigaciones una empresa de cerámica por cuestiones como la experiencia en ese tipo de procesos tanto del director como del doctorando, la importancia relativa de ese tipo de industrias en La Rioja (donde hay dos de las mayores y más modernas del mundo) y en el resto del territorio nacional, y el interés mostrado por la propia empresa, que tiene un Ingeniero Industrial como Gerente, en mejorar en lo posible este tipo de procesos.

El proceso de trabajo para llegar a los resultados se puede resumir como: partiendo de la experiencia en las automatizaciones industriales reales de los procesos de producción, se analizan las posibilidades técnicas y tecnológicas existentes más apropiadas para un tipo de dichos procesos, y se investiga en las posibilidades de mejora de las prestaciones del sistema. A partir de los resultados de dicha investigación inicial se trata de enfocar la investigación en los aspectos concretos que se presentan como susceptibles de ser mejorados, y cuando se llega a resultados satisfactorios se comprueban realmente.

Así, se puede decir que en este trabajo se muestra una metodología para lograr automatizaciones eficientes y robustas, sencillas de realizar pero de diseño eficiente, susceptibles de ser simuladas mediante cualquier programa de simulación o incluso en el propio dispositivo de control del proceso, y finalmente se muestra una aplicación de todo ello.

La distribución en los capítulos de todo eso se puede resumir como:

El segundo capítulo, *Metodología y arquitecturas para la automatización de procesos complejos*, analiza la utilización de las herramientas gráficas para la automatización, la

división del proceso automatizado en modelo del sistema y automatización, e introduce el empleo de paquetes SCADA en las automatizaciones.

El tercer capítulo, *Implementación de las automatizaciones en los dispositivos industriales de control automático*, analiza y propone una metodología de traducción de las herramientas gráficas a los lenguajes de los autómatas programables, dependiendo del tipo y nivel de elementos constitutivos del sistema.

El cuarto capítulo, *Metodología de simulación mediante lenguajes de programación*, analiza y propone la metodología para la traducción desde las herramientas gráficas a los lenguajes de programación en los que pueden ser implementadas para simular el sistema o la automatización. En concreto, dada la potencia y extensión de la aplicación informática Matlab, se resuelve para dicha aplicación.

El quinto capítulo, *Implementación de técnicas avanzadas en la automatización*, propone una metodología de modelar automatizaciones mediante herramientas gráficas que evolucionan y adaptan la automatización, para procesos industriales muy flexibles. También realiza una presentación de métodos de inteligencia artificial que se emplean habitualmente en aplicaciones industriales, y se presenta un pequeño ejemplo.

Y el sexto capítulo, *Aplicación en planta industrial de última generación*, que comienza con un análisis de la metodología para modelizar y automatizar procesos de subsistemas concurrentes con el producto avanzando por ellos, prestando especial interés al seguimiento del material o las piezas de producción a lo largo del proceso, presenta los resultados obtenidos en una planta de tales características al aplicar las metodologías desarrolladas en todo el trabajo.

Con todo esto se da por concluido el trabajo que se presenta, esperando que ciertamente sea útil y constituya una aportación importante al desarrollo industrial, y una referencia para la investigación sobre procesos industriales, así como su desarrollo. Sin embargo esto no cierra el camino en esta línea de investigación, sino que, al contrario, abre todo un abanico de trabajos de investigación, algunos de los cuales incluso ya han comenzado. Los artículos científicos en los que se ha comentado la línea de actuación, han tenido gran éxito, y desde diversos foros se ha solicitado una ampliación de las explicaciones científicas que derivan de este trabajo. Igualmente desde el ámbito industrial, las empresas que firmaron proyectos de investigación como convenio con la Universidad de La Rioja, que fueron realizados por el doctorando de este trabajo y que sirvieron de punto de partida que presentaba una problemática común a varias empresas de la región, han mostrado su interés en renovar los contratos de proyectos de investigación, al comprobar que efectivamente han repercutido en un beneficio para su proceso productivo. Incluso otras empresas de otros sectores han mostrado su interés en desarrollar investigaciones cuyo desarrollo les permita igualmente conseguir automatizar sus procesos industriales de forma más sencilla y más eficiente.

Por tanto a partir de las aportaciones de esta tesis se pretende constituir un nuevo equipo investigador, que inicialmente siga contando con el apoyo del grupo del investigador principal y director de esta tesis, pero capaz de crecer para dar respuesta a esta demanda surgida desde los ámbitos científico e industrial.

Esta última vía de continuidad también está muy avanzada, ya que en los dos últimos congresos en los que se ha presentado esta investigación, a finales de este año 2001, en los que los trabajos presentados han tenido una excelente aceptación (incluso en ambos se solicitó al ponente actuar de director de las sesiones), surgieron conversaciones con grupos de investigación de universidades extranjeras interesadas en participar conjuntamente en la continuación en esta línea, de una manera coordinada entre nuestra universidad y las suyas en un proyecto internacional. Estas conversaciones comenzaron a gestarse en las primeras publicaciones en congresos en las que se trataban aspectos de esta investigación, que a la conclusión de la misma suman una decena, si bien ha sido en las últimas, con la investigación referente a la tesis ya concluida, cuando esa cooperación se ha consolidado e incluso ya se ha comenzado a trabajar en un pequeño proyecto conjunto.

En resumen, como suele ocurrir, el aparente final de un camino no es sino el comienzo de otro, que esperamos sea al menos igual de fructífero y satisfactorio, y que permita cumplir con todas las expectativas que actualmente se presentan sobre esta línea de investigación.