xx (101577.1)

CINVESTAV

Centro de Investigación y de Estudios Avanzados del IPN
Unidad Guadalajara

PROGRAMACION DE PLC A PARTIR DE ESPECIFICACIONES CON RP

TESIS QUE PRESENTA
MARÍA ITALIA JIMÉNEZ OCHOA

PARA OBTENER EL GRADO DE
MAESTRO EN CIENCIAS

EN LA ESPECIALIDAD DE
INGENIERÍA ELÉCTRICA

Guadalajara, Jal. Diciembre de 2001.

PROGRAMACION DE PLC A PARTIR DE ESPECIFICACIONES CON RP

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:

María Italia Jiménez Ochoa

Licenciada en Informática
Instituto Tecnológico de Cd. Guzmán.

Becario de CONACYT, expediente No. 144099

Directores de Tesis:
Dr. Luis Ernesto López Mellado
Dr. Antonio Ramírez Treviño

CINVESTAV del IPN Unidad Guadalajara, Diciembre de 2001.

Dedico esta memoria de tesis a:

Manuel Jiménez y Cristina Ochoa

Salvador Jiménez O.
Martha Jiménez O.
Ruth Jiménez O.
Manuel Jiménez O.
Xóchilt Jiménez O.
Israel Jiménez O.

Jimmy y Manuelito Duarte J.
Carlitos y Liz Jiménez A.

Gracias por su apoyo y ayuda incondicional

Agradecimientos

A Dios por ser un compañero incondicional
y por darme nuevamente la oportunidad de
llegar a una meta más.

Al Dr. Ernesto López y al Dr. Antonio Ramírez por
guiarme en el desarrollo de la tesis con paciencia
y por todos los conocimientos transmitidos.

A Norma por acompañarme como
siempre, en los momentos alegres
y tristes de estos últimos 2 años.

Al Dr. Félix Ramos por darme alientos
a seguir adelante y llegar hasta aquí.

A Iván por ser un amigo excelente
a quien admiro mucho.

A Salvador por ser un modelo a seguir
y por dejarme esta frase: "Imitemos al tiempo"

A Luis por motivarme a terminar
esta meta y por ser alguien de quien
he aprendido mucho durante este
último año.

A Juan José Venegas, quien es un
amigo optimista y quien siempre
conté con su apoyo.

A Bernardo, Carlos, Juan Carlos,
Jorge, Jaime, Omar, Victor y Wicho por
formar un grupo unido y por hacer
agradable la estancia en el CINVESTAV.

Alejandra, María, L. Isidro, Jorge Fausto,
Marco Ramos por su amistad.

A Kena, por ser el primer apoyo que
recibí al entrar al CINVESTAV.

A Leo Reyes por ser una persona
sencilla y grandiosa quien siempre
me dió alientos a seguir.

A Nun por su apoyo y entusiasmo

A todos ellos, muchas gracias...

Índice General

Capítulo 1

Introducción

1.1 Automatización de procesos.

Los Controladores Lógicos Programables (**PLC:** Programmable Logic Controller) son dispositivos electrónicos utilizados en la industria para la automatización de procesos. Su elección para el control de las tareas de los Sistemas de Eventos Discretos (**DES:** Discret Event Systems) es debido a su bajo costo y a su adaptabilidad a casi cualquier proceso de este tipo. En la actualidad, la programación de un **PLC** generalmente se realiza de forma manual, dejando la tarea a la experiencia del programador. Esto trae como consecuencia que el tiempo de programación sea largo y que aumente la posibilidad de introducir errores de programación o bien, comportamientos incorrectos debidos a especificaciones ambiguas.

Un aspecto importante es el mantenimiento del programa residente en el **PLC**. Cuando las especificaciones del funcionamiento del sistema cambian, es necesario reprogramar parcial o totalmente el **PLC** con la consiguiente demora para reinicializar la operación del controlador. Otro problema frecuente, ocasionado por programar de forma empírica, es la localización de un problema ya que llega ser demasiado complicado cuando el programa del **PLC** es complejo.

Estos problemas repercuten en diferentes parámetros que merman el desempeño de la industria. Por ejemplo, cuando se reprograman los **PLCs** se invierte demasiado tiempo y gran esfuerzo humano. La industria no cuenta generalmente con personal capacitado y se traduce esta carencia en una industria ineficiente. Además de las pérdidas económicas que se pueden tener, también está el hecho de que se pierde flexibilidad en los sistemas ya que se introducen largos tiempos de puesta en marcha.

Parte de estos problemas se pueden solucionar utilizando los siguientes estrategias de programación:

- **Especificación del programa a implementar en un lenguaje formal de alto nivel.**

  Permite mejor visualización del sistema si su representación es gráfica.

  El análisis del sistema es con métodos matemáticos, por lo que se pueden obtener resultados óptimos.

  La eliminación de la mayoría de los errores es previa a la implementación del sistema.

- **Traducción automática de este lenguaje de alto nivel a un Diagrama de Escalera (LD: Ladder Diagram).**

  Optimiza el tiempo de la programación del **PLC**.

  Elimina la tarea mecánica de traducción.

  Reduce un alto porcentaje la presencia de fallas al momento de controlar el sistema.

- **Generación de documentación.**

  Proporciona información clara y detallada del programa a implementar.

1.2 Panorama general de la tesis.

El presente trabajo ha sido desarrollado con el fin de implementar las ventajas que ofrecen las estrategias de programación anteriormente mencionadas. Así, para cada una de ellas se ha realizado lo siguiente:

- **Lenguaje formal de alto nivel.** Emplear las **Redes de Petri** (**RdP**) como método formal para el modelado de los **DES**.

- **Traducción automática a Diagrama de Escalera.** Desarrollar una metodología que genere **LD** a partir de las especificaciones en **RdP**

- **Generación de la documentación.** Ofrecer una alternativa para proporcionar información impresa de la distribución y desarrollo del código de **LD** a implementar.

Los puntos anteriores se han desarrollado en un paquete de programación llamado *SpADES*, una herramienta visual que se encarga de la especificación y análisis de los **DES**. Además de estos puntos, *SpADES* cuenta con:

- *Editor de **RdP**:* Permite dibujar una **RdP** ya sea que represente el comportamiento de un sistema o la secuencia que el sistema debe ejecutar (especificación).

- *Generación del controlador:* El sistema *SpADES* genera el modelo en **RdP** del controlador mediante un algoritmo que emplea el esquema de control por regulación.

- *Análisis:* Mediante algoritmos matemáticos, *SpADES* analiza las propiedades que tiene el modelo en **RdP** del controlador como: *vivacidad*, *acotabilidad*, *ciclicidad*, *entre otras*. Así el usuario verifica que el modelo conserve las propiedades requeridas.

- *Implementación del controlador:* Con la metodología previamente mencionada, el sistema *SpADES* genera **LD** de acuerdo al modelo en **RdP** del controlador para ser cargado en un **PLC** comercial.

La razón de que *SpADES* emplee como método las **RdP** es que permite el análisis formal del modelo del controlador. Así, esta metodología garantiza que la salida (**LD**) es correcta. La figura 1.1 muestra el diagrama de bloques del sistema *SpADES*.

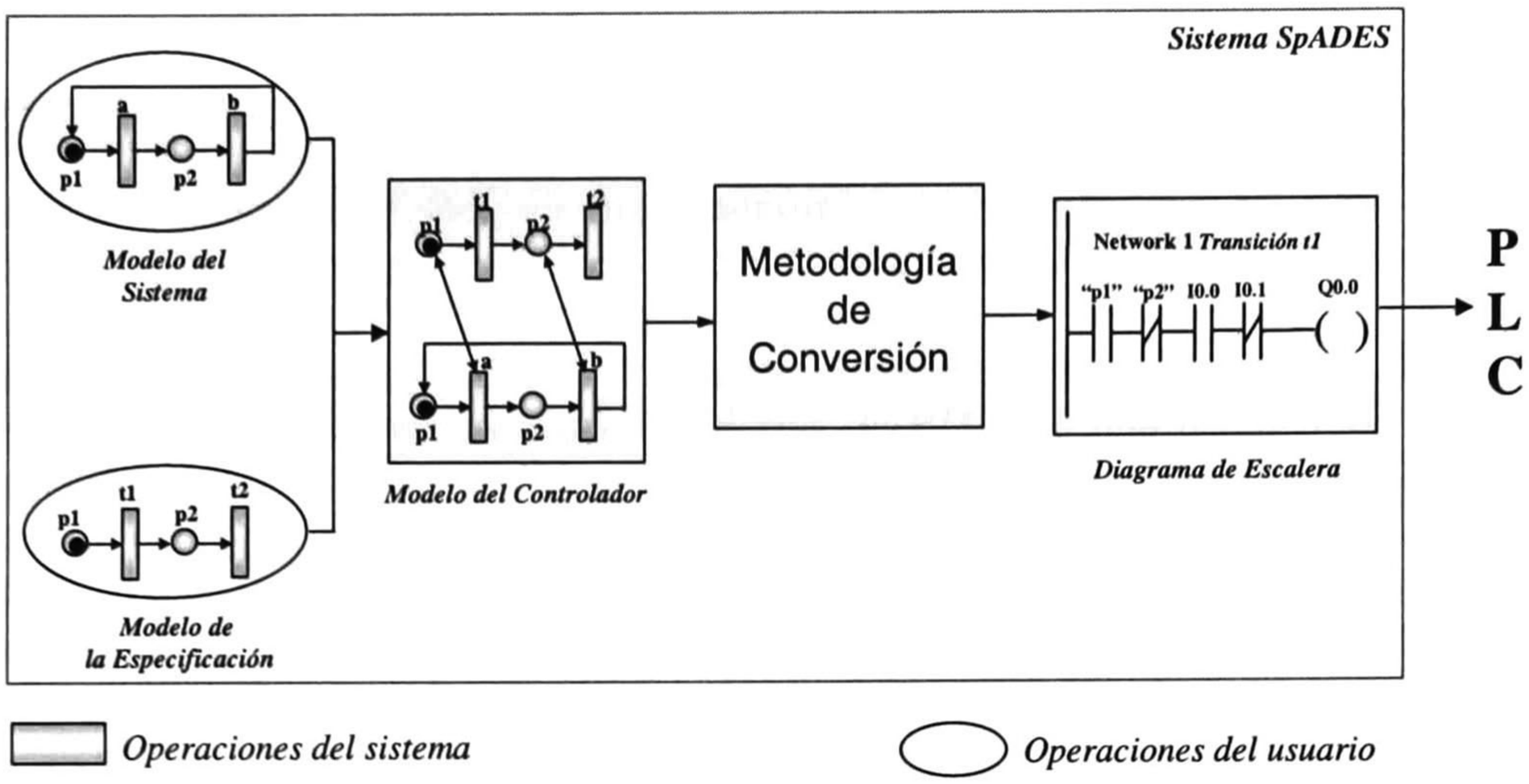


Figura 1.1: Diagrama a bloques del sistema SpADES.

1.3 Revisión de la literatura

Muchas investigaciones han sido realizadas para verificar que en un programa implementado en un **PLC** cumpla con los requerimientos establecidos inicialmente.

Por ejemplo, Mertke y Menzel en [14] muestran el modelado en **RdP** de un sistema de control. Este trabajo consiste de un **Controlador Lógico Programable (PLC)** y un programa que demuestra que el modelo en **RdP** generado cumple con las descripciones de los requerimientos para el sistema. El objetivo es convertir un programa existente escrito en Instruction List (**IL**) en una **RdP** y comprobar que dicho modelo en RdP cumpla con los requerimientos establecidos inicialmente

Los requerimientos pueden ser formulados en un lenguaje de especificación llamado SOSLa. SOSLa está definido por una sintáxis y una semántica formal basada en lógica y permite formular los requerimientos de forma semi-verbal. Por esta razón, este lenguaje no es ambiguo y es posible la generación automática en fórmulas de lógica temporal.

Ahora, para convertir automáticamente el código de un programa en **IL** en **RdP** se utiliza un generador de Redes de Petri. Este produce una **RdP** con transiciones y lugares a salvo. Cada variable y valor del programa en **IL** será representado en una forma binaria (pares de lugares que representan un bit con los valores O y 1). En la generación de un paso, el **IL** será convertido línea por línea en una **RdP**, cada RdP representa exactamente la línea en cuestión. El tamaño de la red es proporcional al número de líneas de código así como a la cantidad y los tipos de variables usadas.

Después de esta conversión (modelo en RdP), se comprueba que los requerimientos funcionales y de seguridad coincidan con los requerimientos establecidos en fórmulas de lógica temporal.

Si el resultado de la comprobación es negativo con respecto a los requerimientos dados, es necesaria una modificación en el código del programa en **IL**.

Otro trabajo relacionado es el de Baresi y Mauri en [2] que presentan una caja de herramientas (tool box) llamada **PLCTools** que fué desarrollado como parte del proyecto Espirit INFORMA. Esta herramienta cuenta con un ambiente amigable para el diseño y análisis

de control de sistemas. Éste está basado en ecuaciones diferenciales para el modelado de la planta, el lenguje de programación Function Block Diagram (**FBD**) del estándar **IEC 1131-3** para el diseño del software de control y **HLTPN** (High Low Timed Petri Nets) para validar el diseño y generar el código a implementar.

PLCTools genera código estándar **ANSI C**. En la generación de código aprovecha tanto el modelo de **FBD** y la **HLTPN** que corresponde al control. El modelo **FBD** da un panorama general del código y la **HLTPN** abarca los aspectos computacionales. **PLCTools** trabaja con una simple tarea: convierte un bloque de **FBD** en una declaración de función en C y usa las subredes del **HLTPN** asociadas a cada bloque para definir el cuerpo de las funciones.

Existen otros trabajos en el campo de los **DES** enfocados al desarrollo de metodologías para convertir de un esquema de alto nivel a un lenguaje de programación de **PLC**.

Uno de ellos es el propuesto por Murat Uzam [23], el cual, muestra una metodología llamada Lógica de Paso de Marcas (**TPL:** Token Passing Logic) para la conversión de Redes de Petri Automatizadas (**APN:** Automated Petri Nets) en Diagrama de Escalera. La característica principal de la técnica **TPL** es que facilita la conversión directa de cualquier **APN** en Control Lógico de Paso de Marcas (**TPLC:** Token Passing Logic Controller). Los **TPLC** son una forma genérica de lógica de control, que puede ser implementada con lenguajes de bajo nivel como *lenguaje máquina, STatment List (STL), Diagrama de Escalera (LD), etc.* o con lenguajes de alto nivel como *C, C++ etc.*

Los lugares de las **APN** son representados por lugares en **TPLC** y las marcas de **APN** son representados por contadores en cada lugar lógico del **TPLC**. En el caso que un lugar del **APN** tenga capacidad de almacenar una marca, en **TPLC** los contadores son reemplazados por banderas. Además, el flujo de las marcas de las Redes de Petri son simuladas por el incremento/decremento de un contador o similarmente activando o desactivando la correspondiente bandera. Los lugares en los cuales se asignan acciones en las **APN** son llamados lugares de acción. La técnica **TPL** considera las siguientes estructuras: *Marcado Inicial, APN sin acción, APN con acción, arco inhibidor, transición AND, transición OR, peso en el arco, conflictos y transición temporizada.* TPL emplea temporizadores, contadores y APN temporizadas para que los **TPLC** muestren una descripción visual del programa de control.

Otro trabajo similar es el de George Frey [7], donde se utilizan las Redes de Petri Interpretadas de Señal (**SIPN:** Signal Interpreted Petri Nets) para la especificación formal de un algoritmo de control. Las **SIPN** tienen propiedades de modelado y análisis de algoritmos de control. Con las SIPN, los controladores lógicos son modelados con lugares (se asignan señales de salida) y transiciones entre los lugares (se asignan funciones booleanas de señal de salida). Por tanto, pueden ser vistas como un subconjunto del lenguaje de programación **Sequential Function Chart (SFC)** del estándard **IEC 1131-3.**

Este trabajo desarrolla un método para la compilación automática de **SIPN** a Lista de Instrucción **(IL Instruction List)** o a **LD**.

El comportamiento dinámico de una **SIPN** está dado por el flujo de marcas a través de la red. Este flujo es realizado por el disparo de transiciones. Las 4 reglas de disparo de las **SIPN** son:

1. Una transición está habilitada si todos sus lugares de entrada están marcados y sus lugares de salida están desmarcados.

2. Una transición se dispara inmediatamente si la transición está habilitada y la condición booleana de disparo se cumple.

3. Todas las transiciones que pueden dispararse y no están en conflicto con otras transiciones se disparan simultáneamente.

4. El proceso de disparo es repetitivo hasta alcanzar un marcado estable (hasta que ninguna transición pueda dispararse).

El método para transformar **SIPN** a **LD** o **IL** está basado por los 2 elementos de la red: *transiciones y lugares.*

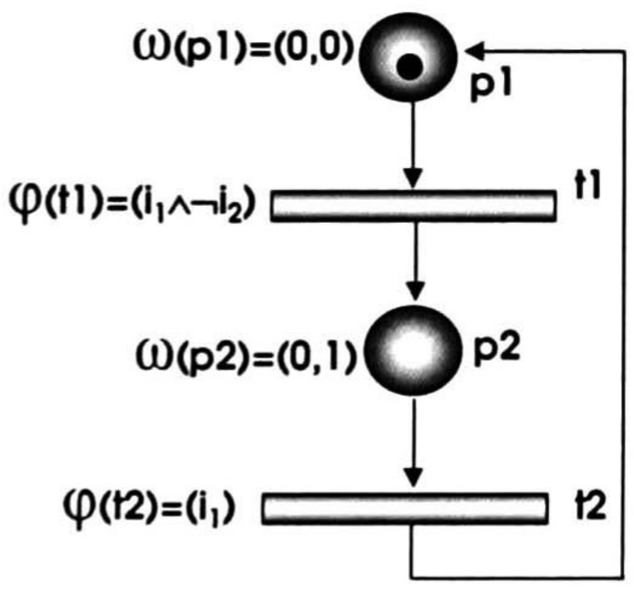
- **Instruction List (IL)**

Transiciones:

Para cada lugar de entrada y salida de la transición utilizan variables booleanas. Además se agrega una función booleana para evaluar la condición de disparo de la transición.

Lugares:

Se utiliza una variable booleana para evaluar si el lugar está marcado y ejecutar la función de salida. La ejecución de la función de salida se realiza activando o desactivando señales correspondientes a la función de salida (ver figura 1.2).



Representación gráfica de una SIPN

```
Ti: LD   Pp1 (* Ti: Si el pre-lugar pi está marcado*)
    AND  pP2     (y el pre-lugar p2 está marcado*)
    .............
    AND  f(E)  (y la condición de disparo se cumple*)
```

Código en IL de una transición

```
Pi: LD   Pp1 (* Pi: Si el lugar pi está marcado*)
    S    Oi                          (*activar O1*)
    .............
    R    Oj                          (* y desactivar Oj*)
    .............
Pj: LD   Pj  (* Pj: si el lugar pj está marcado*)
```

Código en IL de un lugar

Figura 1.2: Representación en IL de una transición y de un lugar de una SIPN.

- **Ladder Diagram (LD)**

Transiciones:

Cada lugar de entrada se representa mediante un Contacto Abierto y cada lugar de salida por un Contacto Cerrado. A cada condición de disparo se le asigna un Contacto Abierto si cumple con verdadero y un Contacto Cerrado si cumple con falso.

Lugares:

Asigna un Contacto Abierto para representar que el lugar está marcado. Para cada elemento de la función de salida se le asigna una Salida Activa si se cumple con verdadero y una Salida Desactiva si cumple con falso.

La característica principal de esta metodología es que cada elemento de la red corresponde a un segmento de código sea del lenguaje **LD** o **IL**. El problema en la generación de código en **PLC** desde **SIPN** es la transformación del comportamiento concurrente e iterativo de la SIPN al comportamiento secuencial de los lenguajes de **PLC**. Por lo que propone diversos algoritmos para poder efectuar dicha conversión (ver figura 1.3).

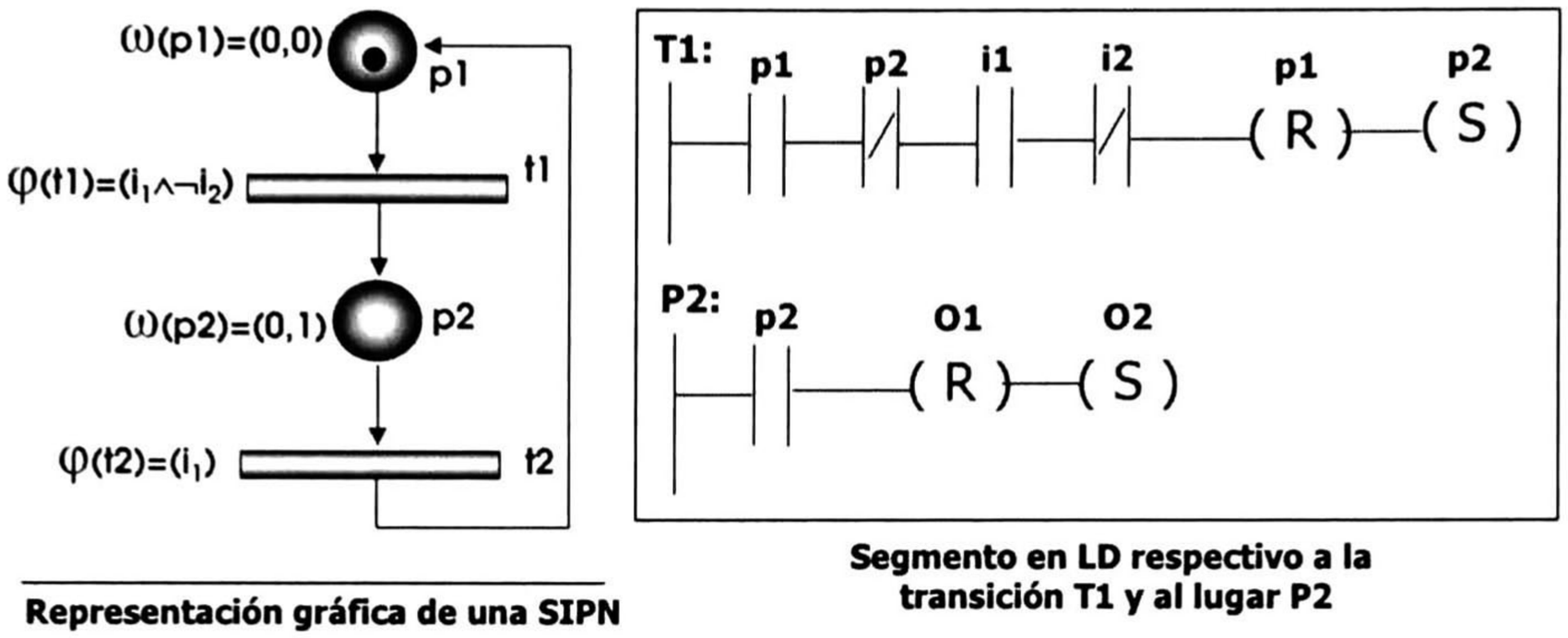


Figura 1.3: Representación en LD de una transición y de un lugar de una SIPN.

El trabajo de tesis que aquí se presenta, propone una metodología encargada de convertir una RdP en un LD. Esta metodología a diferencia de la de George Frey, considera el caso de los sistemas modelados en Redes de Petri No Binarias y Temporizadas para su representación en LD.

Otra ventaja que tiene este trabajo frente al método de Murat Uzam, es que genera directamente LD a partir de un modelo en RdP considerando el caso de la *re-entrancia* de disparo de una transición. Esta metodología está implementada en un sistema llamado *SpADES* con la finalidad de automatizar la generación de LD posterior a su análisis matemático.

1.4 Objetivos.

Los objetivos del presente trabajo son:

- Proponer una metodología para convertir las especificaciones de un sistema modelado en **RdP** al Lenguaje de Programación de **PLC** Diagrama de Escalera.
- Trabajar el lenguaje **LD** de acuerdo al estándar **International Electro-techical Commission: IEC1131-3.**
- Desarrollar un módulo de programación, que forme parte de una herramienta visual llamada ***SpADES.*** Este módulo se programará para implementar la metodología y automatizar la generación de **LD** a partir de especificaciones de una **RdP**.

1.5 Organización de la tesis.

Este trabajo está organizado de la siguiente manera.

- El capítulo 2 presenta los controladores lógicos, su representación en Diagrama de Escalera y en Redes de Petri.
- En el capítulo 3 se propone una metodología para convertir una **RdP** a **LD**.
- El capítulo 4 se enfoca al sistema *SpADES*; una herramienta visual para el modelado y análisis de las **RdP**, dando enfásis al módulo **LDGen**, que es el que se ha desarrollado para la implementación de la metodología descrita en el capítulo 3.
- El capítulo 5 presenta casos de estudio donde modelos en **RdP** son traducidas a **LD** utilizando el sistema *SpADES*.
- El apéndice A enuncia las clases básicas para la edición de una Red de Petri así como la descripción de las funciones miembros de las clases que generan Diagrama de Escalera.

Capítulo 2

Programación de Controladores Lógicos

Resumen. En este capítulo se presentan los diferentes lenguajes de programación de **PLC** incluidos en el estándar IEC1131. Se hace una descripción detallada de los Diagramas de Escalera como lenguaje para **PLC** y se incluye un ejemplo. Finalmente se presentan las Redes de Petri como una alternativa de especificación de programas para **PLC**.

2.1 Control de Sistemas de Eventos Discretos

En los primeros automatismos que se lograron, el sistema era controlado a través de bancos de relevadores. Los relevadores son dispositivos electromecánicos, los cuales constan de un electroimán y de un juego de contactos eléctricos; el electroimán al ser energizado provoca un cambio de estado en los contactos eléctricos, de normalmente abierto (N.A.) a cerrado y de normalmente cerrado (N.C.) a abierto.

Conectados de forma conveniente, los relevadores pueden controlar procesos secuenciales y combinatorios. Sin embargo, entre más complejo se vuelve el proceso más difícil es su diseño y su mantenimiento. Además, el tiempo de vida de un relevador es relativamente corto (comparado con los circuitos electrónicos) y una vez que fallan se pueden esperar fallas constantes por cada relevador existente (una máquina mediana puede contener mas de 100 relevadores). Una limitante más que se tiene en los desarrollos con banco de relevadores es el tiempo de procesamiento de la información. Un relevador tarda aproximadamente 10 ms en conmutar de un estado a otro, por lo que un proceso rápido tendría que retardar su ejecución hasta que se procese la información correspondiente [16].

Otra variante para solucionar la complejidad de los sistemas de control eléctrico fué el árbol de levas. Este dispositivo es un conjunto de levas que accionan interruptores según la secuencia en la que se han dispuesto, las levas se mueven cíclicamente impulsadas por un motor eléctrico de velocidad uniforme. Este arreglo no permite ninguna variación en la secuencia de la máquina, por lo que sólo se usan en procesos secuenciales y no combinatorios.

Aunque este tipo de control reduce considerablemente los costos y la complejidad del diseño, también al paso del tiempo, los interruptores tienden a dañarse y perder su ajuste reduciendo de esta manera la efectividad. Desde el punto de vista del usuario industrial, su única falla es que es difícil su modificación. Si es necesario modificarlos, deben hacerse cambios en las conexiones entre sus dispositivos lógicos o sustituir el dispositivo mismo. Tales cambios a los elementos físicos son indeseables por difíciles y lentos.

Hoy en día se ha popularizado un enfoque fundamentalmente distinto para la construcción de sistemas lógicos industriales. En este nuevo enfoque, la toma de decisiones del sistema se lleva a cabo por instrucciones codificadas, las cuales están almacenadas en un circuito de memoria y ejecutadas por un microprocesador. Si se requiere modificar el sistema de control, basta con cambiar las instrucciones codificadas [24]. Tales cambios se llaman cambios de *programática o software* y se llevan a cabo rápidamente por medio de un teclado. A este nuevo enfoque se refiere a veces como automatización flexible, para diferenciarlo de la automatización dedicada.

Cuando se usa este nuevo enfoque flexible, a la secuencia completa de instrucciones codificadas que controla el comportamiento del sistema se llama *programa.* Por tanto, nos referimos a tales sistemas como sistemas programables. Si todos los componentes de control necesarios son ensamblados y vendidos como una unidad completa, que en la práctica es común, la unidad completa es conocida como *controlador lógico programable.*

2.2 Los Controladores Lógicos Programables.

Un Controlador Lógico Programable (**PLC** Programming Logic Controller) es una computadora industrial usada para monitorear entradas, tomar decisiones en base a la lógica del

programa y controlar salidas para automatizar un proceso o una máquina [22].

El **PLC** se encarga del control del proceso completo ya que es capaz de realizar tareas simultáneamente o en sincronización con otros sistemas, además de ser flexible a condiciones no previstas inicialmente sin que represente un importante costo adicional en su reajuste.

Con el desarrollo tecnológico de la electrónica aplicada en los sistemas de control, se brinda la oportunidad de alcanzar las metas de control de procesos muy complejos. Los **PLCs** se han desarrollado como dispositivos flexibles y adaptables a casi cualquier proceso porque permiten su reprogramación mediante lenguajes fácilmente comprensibles por el personal técnico medio.

Paralelamente, la manera de programar a estos **PLCs** también se ha vuelto más compleja y se ha tenido que normalizar a los diferentes lenguajes usados. En estos momentos, la norma IEC1131 es la más importante y reconoce cuatro lenguajes para programar a un **PLC**. Estos lenguajes de programación están divididos en textuales y gráficos. Los textuales lo comprenden los lenguajes: Instruction List (IL) y Structured Text (ST); los lenguajes gráficos son Ladder Diagram (LD) y Sequential Function Chart (SFC).

Por otro lado, el diseño del controlador se ha realizado en forma empírica, es decir sin el uso de métodos formales. Las desventajas que existen en las diferentes partes de la especificación es que no siempre son claramente establecidos los requerimientos del sistema y no facilita las pruebas de consistencia, completitud y de ambiguedad.

Existe una gran variedad de formalismos para describir o modelar el comportamiento de los sistemas discretos. Podemos destacar las ecuaciones booleanas, las expresiones regulares, los diagramas de tiempo, las tablas de estado, máquinas de estados, álgebras de procesos y las redes de Petri.

Cada formalismo ofrece elementos y convenciones para representar el funcionamiento de un sistema en términos de estados, eventos y relaciones de precedencia entre estados. También cada formalismo permite con diferentes grados claridad y compacidad, la representación de comportamientos tales como secuenciamiento, paralelismo y sincronización entre procesos que evolucionan concurrentemente, además de la toma de decisiones.

2.2.1 Arquitectura de los PLC's.

El *controlador lógico programable* es una computadora de uso específico (industrial); está constituido prácticamente por 3 bloques: **La sección de entrada/salida, el procesador y el dispositivo de programación o terminal** [17].

Sección de Entrada/Salida.

La sección de entrada/salida (sección de E/S) de un **PLC** se encarga del trabajo de la intercomunicación entre los dispositivos industriales y los circuitos electrónicos de baja potencia que almacenan y ejecutan el programa de control.

La sección de E/S contiene módulos de entrada y salida. Supongamos que cada módulo de entrada es una tarjeta de circuito impreso que contiene dieciséis convertidores de señal. Cada una de las diesciséis terminales del módulo recibe una señal de alta potencia de un dispositivo de entrada y la convierte en una señal digital de baja potencia compatible con los circuitos electrónicos del procesador. Ahora, supongamos que cada módulo de salida es una tarjeta de circuito impreso que contiene dieciséis amplificadores de salida. Cada amplificador

de salida recibe del procesador una señal digital de baja potencia y la convierte en una señal de alta potencia capaz de manejar una carga industrial.

El procesador.

El procesador de un **PLC** contiene y ejecuta el programa del usuario. Para poder hacer ese trabajo, el procesador debe almacenar las condiciones de entrada y salida más recientes. El diagrama funcional de la figura 2.1 describe la secuencia de tareas del procesador que a continuación se explican.

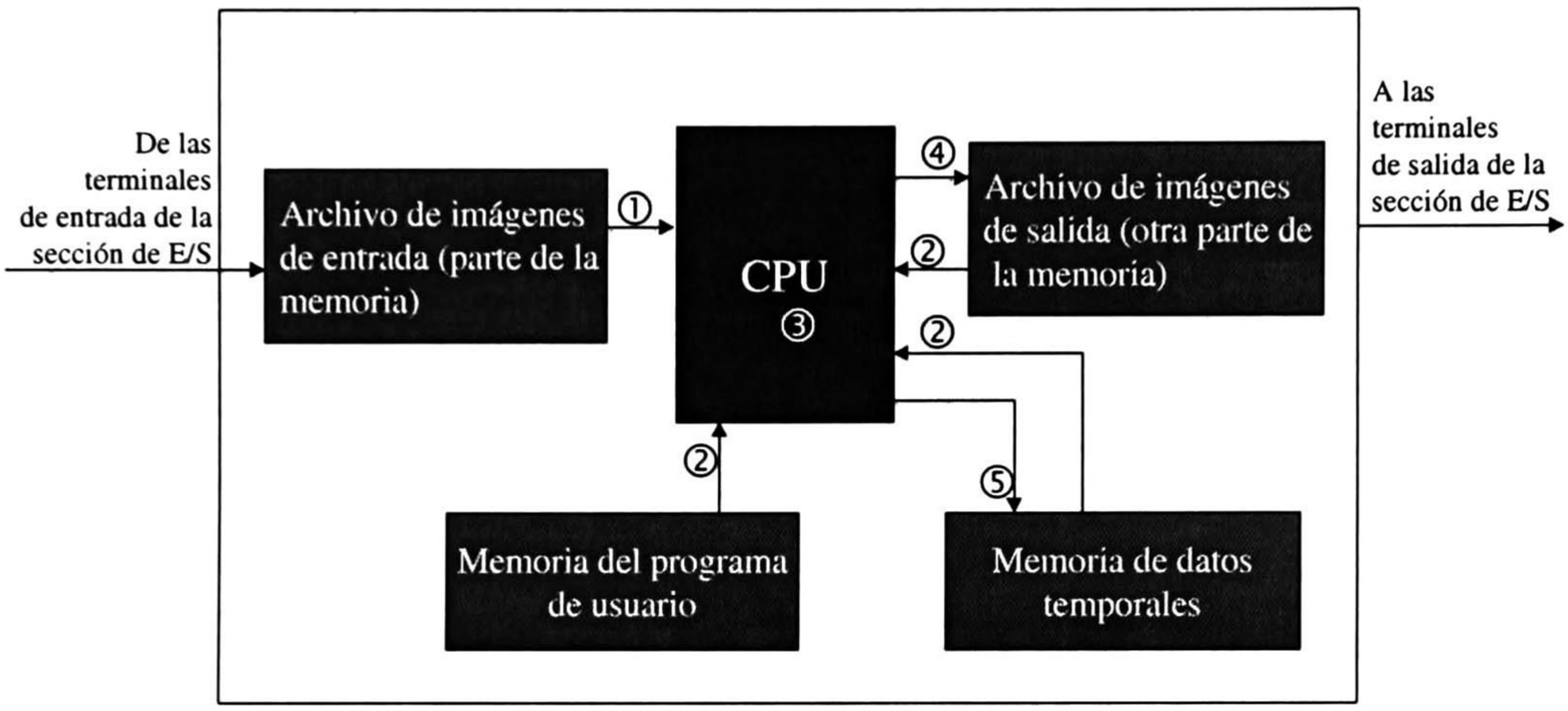


Figura 2.1: Las tareas del procesador son: 1. Obtener información de E/S de los archivos de imágenes y datos numéricos de la memoria de datos variables. 2. Leer y ejecutar las instrucciones de la memoria del programa de control a la CPU. 3. Ejecutar las instrucciones. 4. Toma de decisiones sobre los estados adecuados de las salidas y reflejar que esos estados aparezcan en el archivo de imágenes de salida y 5. el cálculo de los valores de datos variables y su almacenamiento en la memoria de datos temporales.

Archivo de imágenes de entrada. Las condiciones de entrada se almacenan en el *archivo de imágenes de entrada*, que es una parte de la memoria del procesador. Esta memoria es de tipo lectura-escritura (comúnmente conocida como memoria de acceso aleatorio (RAM)). Cada terminal del módulo de entrada de la sección de E/S tiene asignado un lugar determinado dentro del archivo de imágenes de entrada. Este lugar determinado está destinado exclusivamente a la tarea de llevar el registro actual de su terminal de entrada.

El procesador requiere estar alimentado con las actuales condiciones de entrada pues la ejecución del programa de control dependen de esas condiciones.

Archivo de imágenes de salida. Las condiciones de salida se almacenan en el *archivo de imágenes de salida*, que es otra parte de la memoria del procesador. El archivo de imágenes de salida tiene la misma relación con las terminales de salida de la sección de E/S que el archivo de imágenes de entrada tiene con las terminales de entrada. Esa localidad en particular está

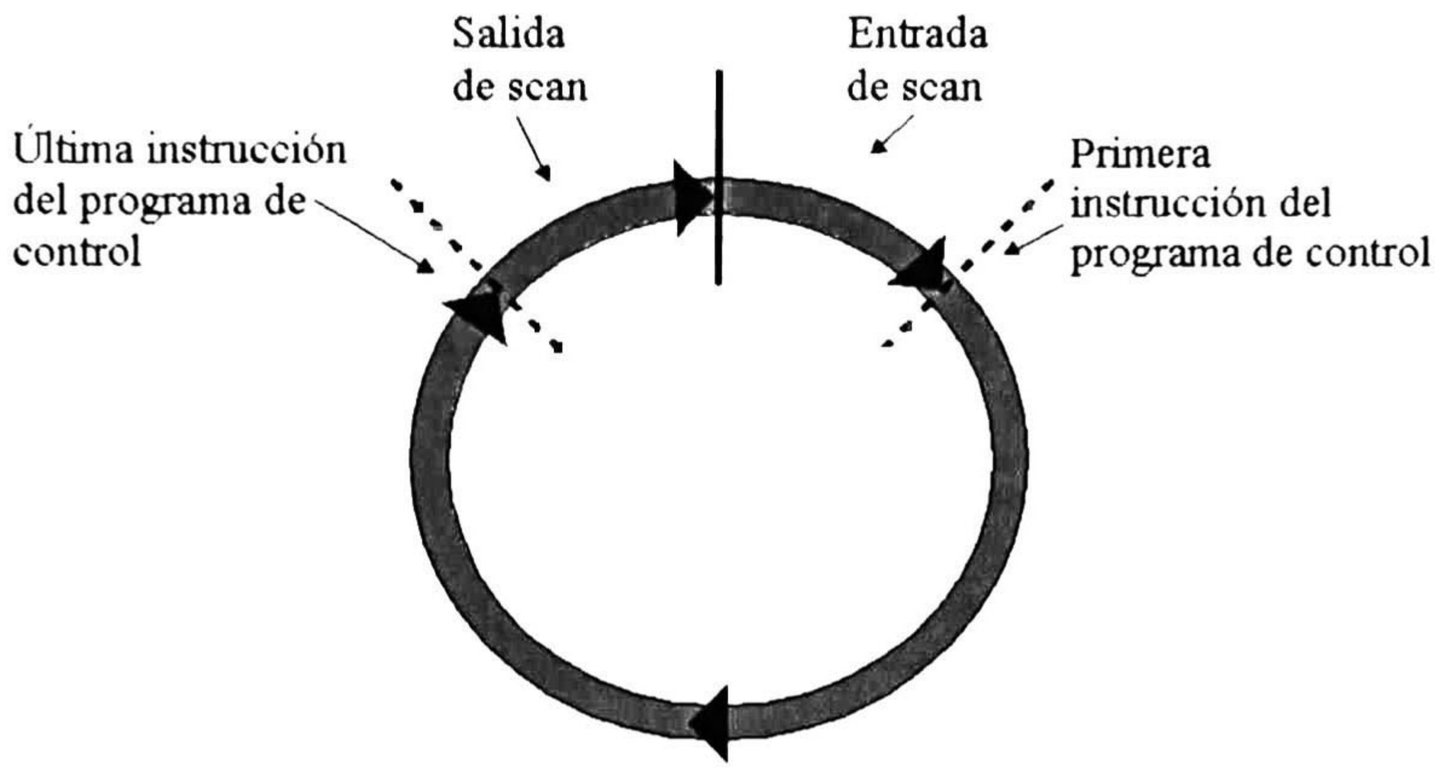


Figura 2.2: El ciclo completo de barrido de un PLC tiene tres partes distintas: barrido de entrada, ejecución del programa y barrido de salida.

dedicada exclusivamente a la tarea de llevar el registro de la última condición de su terminal de salida. Por supuesto, la situación de salida difiere de la situación de entrada en relación con la dirección de flujo de la información. En la situación de salida, el flujo de información es del archivo de imágenes de salida al módulo de salida, mientras que en la situación de entrada, el flujo de información es del módulo de entrada al archivo de imágenes de entrada.

Unidad de procesamiento central. Es la subsección del procesador que se encarga de la ejecución del programa. A medida que la CPU ejecuta el programa del usuario, el archivo de imágenes de salida se está actualizando continuamente. En otras palabras, si la ejecución de una instrucción solicita un cambio en una de las localidades del archivo de imágenes de salida, ese cambio se efectúa de inmediato, antes de que el procesador avance a la siguiente instrucción. Esta actualización inmediata es necesaria ya que las condiciones de salida siempre afectan instrucciones posteriores del programa.

Por tanto, se puede ver que el archivo de imágenes de salida tiene una naturaleza doble: su primera función es la recepción inmediata de información de la CPU, pasándola (inmediatamente) a los módulos de la sección de E/S. Por otra parte, también debe ser capaz de retroalimentar información de salida al CPU, cuando la instrucción del programa de control que la CPU está procesando solicita un elemento de la información de salida.

Memoria del programa de control. Una porción particular de la memoria del procesador se usa para el almacenamiento de las instrucciones del programa de control.

A medida que el usuario ingresa las instrucciones, automáticamente éstas se almacenan en localidades secuenciales en la memoria del programa de control. Esta ubicación secuencial de las instrucciones del programa es autocontrolada por el **PLC**.

Una vez completado el procedimiento de programación, el usuario manualmente conmuta el **PLC** del modo de PROGRAMACION al modo de EJECUCION, lo que hace que la CPU ejecute el programa de principio a fin repetidamente.

Para ejecutar el programa de control, la CPU maneja una red de LD (ver definición en

la sección 2.4) de programa a la vez. A continuación enunciaremos el flujo de ejecución de una red de LD.

1. La CPU, que siempre hace un seguimiento de la localidad precisa en el programa de control, trae la siguiente instrucción secuencial de la memoria del programa de control.

2. La instrucción obtenida por la CPU está destinada a ser una instrucción del tipo examinar. La CPU trae la información requerida de los archivos de imágenes de entrada o de salida para poder evaluar la instrucción.

3. La CPU efectúa una prueba interna combinando la instrucción del paso 1 con la información de E/S del paso 2. Esta prueba determina si la instrucción produce continuidad o discontinuidad lógica.

4. La CPU lee la siguiente instrucción y la ejecuta para determinar si la siguiente información es otra instrucción del tipo examinar o una instrucción energizar-salida. Si es una instrucción del tipo examinar, la CPU nota si existe una condición lógica AND o una OR con la instrucción previa. Si existe una condición lógica AND (en serie en la representación de lógica de la red de LD), entonces ambas instrucciones deben producir continuidad para que la red de LD mantenga la misma continuidad que hasta ahora. Si la siguiente instrucción presenta una condición lógica OR con la anterior (aparecen trayectorias paralelas en la representación de lógica en escalera), entonces basta con que cualquiera de las instrucciones produzca continudad para que la red de LD mantenga la misma continuidad que hasta ahora. Puede ocurrir que la CPU tome su decisión en el momento, en base a las condiciones de verdadero y falso de la red de LD. Por otra parte, puede ocurrir que la CPU no pueda tomar su decisión de verdadero o falso en el momento, pero deba traer la siguiente instrucción del tipo examinar para mayor verificación de continuidad.

5. A la larga, la CPU avanzará por la red de LD hasta el punto en que puede decidir si las condiciones generales de la red de LD son de verdadero o falso. Entonces lee y ejecuta la instrucción energizar-salida de la memoria del programa de control para que pueda conocer la dirección que va a afectar.

6. La CPU se mueve a la subrutina correspodiente de acuerdo a las condiciones presentes y envía la señal apropiada al archivo de imágenes de salida, que luego pasa a la terminal de salida asociada.

Cuando el procesador ha terminado de ejecutar una red de LD pasa a la siguiente localidad secuencial de memoria del programa de control, recogre la siguiente instrucción (la primera de la siguiente red de LD) y repite los pasos 1 al 6. Continúa de la misma manera hasta que cada instrucción haya sido ejecutada. En este punto el programa de control ha sido ejecutado en su totalidad una vez.

Ciclo de barrido completo Mientra el **PLC** esté en el modo de EJECUCION, el procesador ejecutará el programa en forma cíclica. En la figura 2.2 se representa completa la serie repetitiva de eventos. Comenzando por la parte superior del círculo que representa el ciclo de barrido, la primera operación es el barrido de etrada. Durante el barrido de entrada, el

estado actual de cada terminal de entrada se almacena en el archivo de imágenes de entrada, actualizándolo. Como todas las operaciones del **PLC**, el barrido de entrada es bastante rápido. El tiempo transcurrido depende de la cantidad de módulos y terminales de entrada en la sección de E/S, la velocidad de reloj del CPU y otras características técnicas de la CPU. A continuación del barrido de entrada, el procesador ejecuta el programa de control, proceso llamado "barrido de programa" La ejecución consiste en comenzar en la primer red de LD del programa, llevando a cabo la secuencia de ejecución de seis pasos descrita antes; luego, pasar a la siguiente red de LD, llevando a cabo su ejecución y seguir así hasta la última red de LD del programa. El tiempo de ejecución del programa dependerá de la longitud de este programa, la complejidad de las redes de LD y las especificaciones técnicas de la CPU.

Durante toda la ejecución del programa de control, el procesador periódicamente actualiza el archivo de imágenes de salida, como se indicó anteriormente. Sin embargo, las terminales de salida mismas no son actualizadas constantemente. En cambio, el archivo de imágenes de salida completo se transfiere a las terminales de salida durante el barrido de salida que sigue a la ejecución del programa.

Es perfectamente razonable que las terminales de salida sean actualizadas todas a la vez durante el barrido de salida, en lugar de hacerlo en forma individual e inmediata durante la ejecución del programa de control. Esto es porque, en general, los propios dispositivos de carga son lentos en comparación con el ciclo de barrido del **PLC**.

Memoria de datos temporales. En general, un **PLC** estándar tiene las siguientes instrucciones que le dan estas capacidades.

- Puede producir un retardo definido por el usuario en un esquema de control. Esto es, el **PLC** tiene temporizadores internos (instrucciones de temporización) que reproducen las acciones de los temporizadores del programador.
- Puede contar eventos,con los eventos representados como cierre de interrupciones. Esto es, el **PLC** contiene contadores internos, como los contadores ascendentes y descendentes.
- Puede ejecutar operaciones aritméticas y lógicas con los datos residentes en su memoria.
- Puede ejecutar comparaciones numéricas.

Estas capacidades implican que el **PLC** pueda almacenar y trabajar con números. Naturalmente, los números pueden cambiar de un ciclo de barrido al siguiente (ocurren eventos y cuentan, el tiempo transcurre, etc.). Por tanto, el **PLC** debe tener una sección de su memoria reservada para mantener la cuenta de números variables o datos que intervienen en el programa de control.

Hay muchos tipos de datos numéricos que pueden estar presentes en la memoria de datos variables. Seis tipos cuya compresión es importante son:

1. El valor predeteminado de un temporizador o número de ciclos de reloj durante los que el temporizador deberá permanecer activo para dar una señal de interrupción.

2. El valor acumulado de un temporizador o número de ciclos de reloj que han transcurrido desde que el temporizador fue activado.

3. El valor predeterminado de un contador o número al que un contador ascendente debe contar para dar una señal de "conteo completo" Para un contador descendente es el número del que partirá en su cuenta descendente.

4. El valor acumulado de un contador: es la cuenta actual que ha sido registrada por un contador ascendente. Para un contador descendente, es la cuenta actual faltante antes de que el contador llegue a cero.

5. El valor de una variable física en el proceso controlado, que se obtiene midiendo la variable física por medio de un transductor y convirtiendo el voltaje (o corriente) de salida analógico del transductor a un formato digital.

6. El valor de una señal de salida enviada a un controlador en el proceso controlado, que se obtiene mediante un cálculo matemático efectuado por el **PLC**.

Cuando la CPU ejecuta una instrucción para la cual ciertos valores de datos deben conocerse, ese valor del dato es traído de la memoria de datos variables. Cuando la CPU ejecuta una instrucción que produce un resultado numérico, ese resultado se introduce en la memoria de datos variables. Por tanto, la CPU puede *leer de o escribir a* la memoria de datos variables.

Dispositivo de programación La tercera parte esencial de un **PLC** es el dispositivo de programación, que también se llama terminal de programación o solamente programador. Algunos **PLC** están equipados con un dispositivo de programación dedicado construido por la misma compañia que fabrica el **PLC**, pero en muchas instalaciones, el dispositivo de programación es una computadora de escritorio o portátil con una tarjeta de interfaz de comunicación instalada en una ranura de expansión. Un cable de comunicación serial conecta la tarjeta de interfaz con el procesador del **PLC**. Con un software especial instalado en el disco duro de la computadora, las pulsaciones en el teclado de la computadora representan instrucciones del programa de control que son convertidas en el código apropiado por la tarjeta de interfaz. De allí pasan por el cable de comunicación al procesado.

Algunos de los **PLC** más sofisticados tienen capacidad de realizar estadísticas del proceso y de acuerdo al comportamiento hacer auto-ajustes al control para conseguir la máxima eficiencia, a lo que actualmente se le conoce como sistemas inteligentes de auto aprendizaje. Es importante recordar que aunque estos sistemas puedan hacer auto-ajustes, su dependencia con el programador es total para poder realizar su trabajo, esto es debido a que éste es el encargado de establecer los criterios de auto-ajuste, así como las consecuencias y la lógica de todo el proceso, en otras palabras. El controlador solo hará lo que el usuario le indique que haga.

Son estas las razones por las cuales es importante que la comunicación de usuario a **PLC** se realice de la forma más sencilla y clara, para evitar malas interpretaciones por cualquiera de las dos interlocutores. La comunicación con el usuario puede llevarse acabo por diferentes medios, que dependerán de la marca, tipo y modelo del **PLC**.

Independientemente del medio por el cual el usuario dé las instrucciones al **PLC** (programador, teclado, **PLC**, touch-screen, etc.), se respetarán las estructuras para dar la información, es decir los protocolos de comunicación.

El protocolo de comunicación es la forma en que el **PLC** y el usuario se comunicarán, inclusive entre diferentes **PLC**, para garantizar que la comunicación sea correcta. Los protocolos de comunicación pueden ser desde la forma en que se transmite la información por un lazo de corriente o por modulación de pulsos, o simplemente la forma que se denominará cada instrucción a través de un programa determinado.

2.2.2 Programación de un PLC.

Los controladores no tienen la capacidad de interpretación del lenguaje común de los hombres, así que su lenguaje está limitado a la numeración binaria, pero a través de dispositivos electrónicos es posible compilar la información transformádonla de un lenguaje de usuario o viceversa. Sin embargo, estos compiladores también tienen sus limitantes por lo que el usuario debe adaptarse a ciertas condiciones consideradas dentro de los protocolos de comunicación.

- Operador Absoluto.

Una de las condiciones importantes para programar un **PLC** es reconocer específicamente qué entrada, salida, temporizador, contador etc., se desea activar o preguntar por su estado. Para poder identificar cada uno de estos parámetros, se le asigna una dirección específica, esta dirección recibe el nombre de Operador Absoluto.

El operador absoluto es la única forma en que el **PLC** comprenderá a qué parámetro específico del proceso se refiere.

Los operadores absolutos describen las funciones existentes en el **PLC**, por ejemplo, se tienen funciones de señales de entrada (I "Input"), señales de salida (O "Output"), banderas (F "Flag"), contadores (C), temporizadores (T), registros (R), programas (P), etc.

- Operador Simbólico.

En muchas ocasiones la interpretación de un programa de un **PLC** requiere de cierta habilidad y el uso de operadores absolutos dificultan la interpretación de estos programas, es por esto que se ha considerado en el **PLC** el uso de operaciones simbólicas.

Como se muestra en la figura 2.3, el operador simbólico es una función equivalente al operador absoluto pero que sólo el usuario podrá interpretar, para el **PLC** no importa el contenido de este operador inclusive puede existir o no.

Edición del programa

Como es de imaginarse, es muy probable que un programa no funcione perfectamente en su primera corrida. Hay demasiadas posibilidades de errores conceptuales para un novato en el diseño del programa y demasiadas posibilidades de errores tipográficos para un novato en el ingreso del programa, que es casi seguro que el programa requerirá de algún proceso de corrección antes de estar listo para operar. Con esto en mente, los fabricantes de **PLC** ofrecen capacidades de edición con el software de programación.

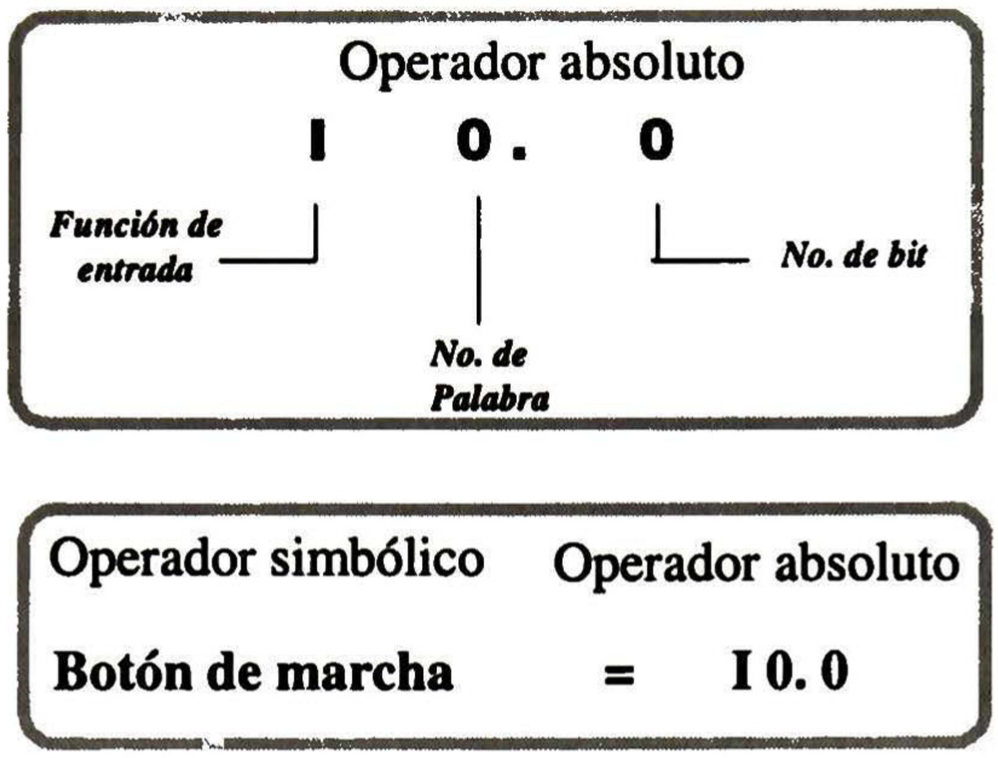


Figura 2.3: Operador absoluto y simbólico de un PLC.

Dada la improbabilidad que un programa funcione de manera satisfactoria la primera vez, los fabricantes de **PLC** proveen un tercer modo de operación del procesador, además de PROGRAMACION y EJECUCION. Este es el modo de PRUEBA, en el que el procesador ejecuta el programa sin energizar las terminales de salida de la sección de E/S. En su lugar, un indicador (LED) para cada terminal de salida se enciende cuando esa terminal debería energizarse si el procesador estuviera operando en el modo de EJECUCION. De esta manera, es posible emular la operación del sistema industrial sin energizar en realidad los dispositivos de carga.

Una vez que el programa está completamente probado, una ejecución en el modo de PRUEBA indicará que todas las salidas operan según lo planeado. Entonces el procesador puede conmutarse al modo de EJECUCION.

2.3 Lenguajes de Programación de PLC (IEC1131)

Un grupo de trabajo fué establecido por el **IEC** (**International Electro-technical Commission**) para buscar el diseño completo de los controladores programables, incluyendo el diseño del hardware, instalación, pruebas, documentación, programación y comunicación [8].

El objetivo de este grupo de trabajo fué establecer un método consistente para la programación de **PLC**'s que fomentaría el desarrollo de software de calidad [17]. El estándar IEC1131 consta de 5 partes, las cuales son:

Parte 1. *Información general.* Establece la definición general de términos usados en el estándar e identifica las características principales para la selección y aplicación de controladores programables y sus componentes periféricos.

Parte 2. *Equipo y requerimientos de prueba.* Contiene los requerimientos mínimos para las características funcionales, condiciones de servicio, características de construcción, seguridad general y pruebas aplicables a los controladores lógicos y sus componentes periféricos

Parte 3. *Lenguajes de Programación.* Especifica la *sintaxis*, que es la combinación permitida de símbolos los cuales pueden ser usados para definir un programa, y especifica la *semántica,* que es el significado y la relación entre operaciones programadas y la combinación de símbolos definidas por la sintaxis de los lenguajes de programación.

Parte 4. *Guía de usuarios.* Contiene una guía para el usuario y un tutorial de informa-

ción y aplicación general de los controladores lógicos y sus componentes periféricos.

Parte 5. *Comunicación.* Define la comunicación entre controladores programables y otros sistemas electrónicos usando la Especificación de Mensajes de Manufactura (MMS) definida en ISO/IEC9506.

Los lenguajes de programación del IEC 1131-3 define 4 diferentes lenguajes para el uso de controladores programables. **Instrucción List** (*IL*), **Structured Text** (*ST*), **Ladder Diagram (LD)**, y **Fuction Block Diagram** (*FBD*) [?]. Estos 4 lenguajes de programación están divididos en textuales y gráficos.

```
          LD    R1
          JMPC  RESET
          LD    PRESS_1
          ST    MAX_PRESS
RESET:    LD    0
          ST    A_X43
```

Figura 2.4: Lenguaje Instruction List (IL).

```
IF SPEED1 > 100.0 THEN
    Flow_Rate := 50.0 + Offset_A1;
ELSE
    Flow_Rate := 100.0;  Steam := ON;
END_IF;
```

Figura 2.5: Lenguaje Structured Text (ST)

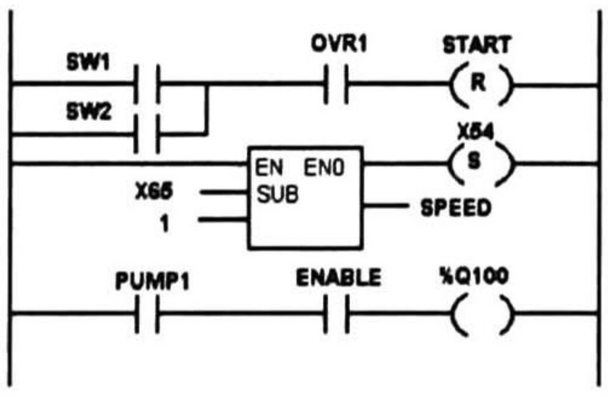


Figura 2.6: Lenguaje Ladder Diagram (LD)

2.3.1 Instruction List (IL)

Está compuesto de una secuencia de instrucciones la cual opera una instrucción por línea. A lo que más se parece es un lenguaje ensamblador. Éste es un método de programación adaptable, pero por la inexperiencia del usuario, puede resultar incómodo y consumiría demasiado tiempo en su aplicación (ver figura 2.4).

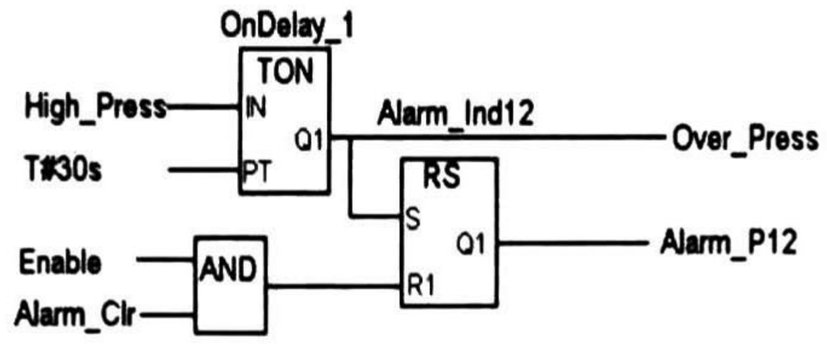


Figura 2.7: Lenguaje Function Block Diagram

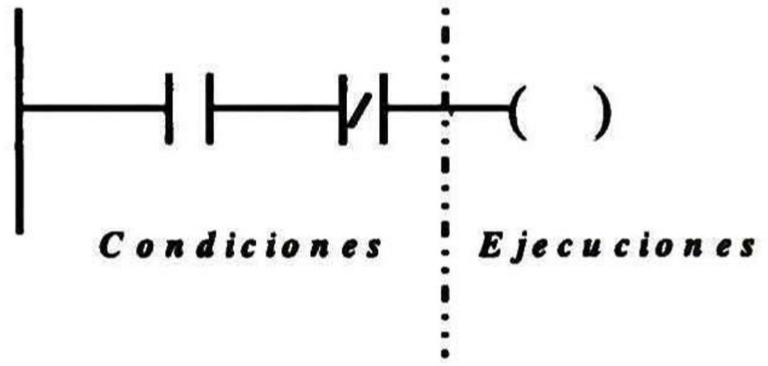


Figura 2.8: Estructura de programación del lenguaje de programación LD

2.3.2 Structured Text (ST)

Es un lenguaje de alto nivel parecido al Pascal. Como se muestra en la figura 2.5, ST programa tareas haciéndolas altamente inteligible al usuario. ST es un excelente lenguaje para tareas complejas tanto matemáticas o algorítmicas; puede ser usado para simplificar en otros casos códigos demasiados extensos de **LD** o FBD.

2.3.3 Ladder Diagram (LD)

Como se ilustra en la figura 2.6, es un método gráfico de programación de **PLC** con sus tradicionales contactos y bobinas. Éste está principalmente orientado hacia aplicaciones de control discreto. Además, el IEC, permite introducir Bloques de Función en el código de **LD**, así lo hace más adecuado las aplicaciones de control de procesos.

2.3.4 Function Block Diagram (FBD)

Permite gráficamente el desarrollo de un programa usando lo básico para hacer bloques que se encuentran en la Librería de Bloque de Función del IEC. Representa señales y flujo de datos a través de bloques de función con el reuso de elementos de software. La figura 2.7 muestra un fragmento del programa en FBD.

2.4 Los Diagramas de Escalera como Lenguaje de Programación

El lenguaje de programación Diagrama de Escalera (**LD**) ha venido siendo uno de los lenguajes gráficos más populares para la programación de **PLC**.

La estructura general de los **LD**s para la programación del **PLC** emula la estructura de un diagrama eléctrico de escalera, con esto, se pretende una fácil interpretación por los técnicos de mantenimiento ya que ellos están bastante familiarizados con este tipo de diagramas.

En estos diagramas, la simbología principal representa lo que eléctricamente serían algunos contactos Normalmente Abierto (**N.A.**) o Normalmente Cerrado (**N.C.**) al igual que bobinas.

Los elementos básicos usuados en el Lenguaje de Programación Diagrama de Escalera bajo el estándar IEC 1131 son:

- **Contactos**. Un contacto es un símbolo que representa un interruptor a través del cual la corriente circula cuando el contacto está cerrado. Los contactos pueden ser *normalmente abiertos o normalmente cerrados.*

- **Bobina.** Una bobina es un símbolo que representa a un relevador que es energizado por la corriente que circula por el relevador.

- **Cajas.** Todas las cajas son símbolos que representan varias funciones que son ejecutadas cuando la corriente activa a la caja. Las funciones típicas representadas por las cajas son: *contadores, temporizadores y funciones matemáticas.*

Las bobinas y las cajas deben de estar conectadas a la línea de potencia derecha.

- **Red de LD.** Es un renglón que están conectados elementos que forma un circuito completo entre la línea de potencia izquierda y la línea de potencia derecha. La línea de potencia izquierda representa el conductor energizado y la línea de potencia derecha representa el retorno del conductor.

En la figura 2.9 se describen los elementos básicos que se mencionaron anteriormente.

Como se muestra en la figura 2.8, la ventaja que ofrecen los **LD** es que la estructura de la programación es muy sencilla y parte de la forma: *condición-ejecución.* La parte izquierda representa todas las condiciones necesarias y suficientes para que una acción se realice.

Tradicionalmente, los **LD**'s son usados para capturar la secuencia de operaciones ejecutadas por el sistema de control. En ellos se especifican los procedimientos de entrada y salida del **PLC** que han de guiar y ejecutar las operaciones repetitivas del sistema.

Por otro lado, actualmente la manera de expresar las especificaciones de un sistema en cualquier Lenguaje de Programación de **PLC** (para nuestro caso en particular **LD**) es manual. Algunas de las desventajas que hay en esta forma de programar son:

- Se invierte demasiado tiempo en programar el **PLC** cuando el sistema es complejo.

- Es difícil el mantenimiento del programa del **PLC** cuando se requieren hacer modificaciones en las especificaciones del sistema.

- Localizar la causa de un problema es una tarea ardua.

- El programa del **PLC** carece generalmente de una buena documentación, lo que hace que el personal que se va involucrando en el sistema no lo conozca a detalle.

Símbolo	*Descripción*
bit ⊣ ⊢	El contacto **Normalmente Abierto** se cierra (OFF) si el bit es igual a 1.
bit ⊣/⊢	El contacto **Normalmente Cerrado** se abre (ON) si el bit es igual a 0.
bit ⊣(S) bit ⊣(R)	Cuando se ejecutan las operaciones **Poner a 1 y Poner a 0**, el valor indicado por el bit se activa o se desactiva respectivamente.
CUTD >CU QU >CD QD R LD PV CV	La operación **Contar adelante/atrás** (CTUD) empieza a contar adelante cuando se produce un flanco positivo en la entrada de contaje adelante (CU). Por el contrario, empieza acontar atrás cuando se produce un flanco positivo en la entrada de contaje atrás (CD). Si el valor actual (CV) es mayor o igual al valor de preselección (PV), se activa el bit de contaje (Cxxx). El contador se inicializa al activarse la entrada de desactivación (R).
GE EN OUT IN1 IN2	La función **Sencuencia Monotonamente decreciente** (GE) compara IN1>=IN2 con el resultado booleano depositado en OUT. Los tipos de datos de entrada y salida pueden variar pero deben ser del mismo tipo.
TP IN Q PT ET	El bloque funcional **Temporizador por impulsos** (TP) se utiliza para generar impulsos de una duración determinada. Cuando el estado de señal de la entrada de habilitación (IN) cambia a "verdadero" durante el impulso especificado en el tiempo prefijado (PT). Cuando el tiempo transcurrido (ET) alcanza el valor del tiempo prefijado, el estado de señal del bit de salida (Q) cambia a "falso".
TON IN Q PT ET	El bloque funcional **Temporizador con retardo al conectar** (TON) temporiza hasta el valor prefijado cuando la entrada de habilitación (IN) cambia a "verdadero". Si el tiempo transcurrido (ET) es mayor o igual al tiempo prefijado (PT), se activará el bit de salida del temporizador (Q). El bit de salida se desactivará cuando la entrada de habilitación cambie a "falso". Cuando se almacena el tiempo prefijado (PT), la temporización se detiene y el temporizador se inhibe.

Figura 2.9: Descripción de los elementos básicos del Lenguaje de Programación Diagrama de Escalera.

- Frecuentemente, la industria recurre a programadores externos ya que muchas veces no cuentan con personal capacitado.
- La programación de los **PLC**s es más tardada ya que el personal externo no conoce bien la industria.
- La corrección de errores se hace posterior a la implementación lo cuál es costoso en aquellos casos que no se cuente con un simulador.
- El programa del **PLC** no es adecuado para el análisis de ejecución del sistema.

Para resolver todos estos inconvenientes que se generan por programar un **PLC** de forma empírica, se propone mediante este trabajo, utilizar un esquema de programación de alto nivel que permita la especificación, análisis y validación formal de un controlador. Este

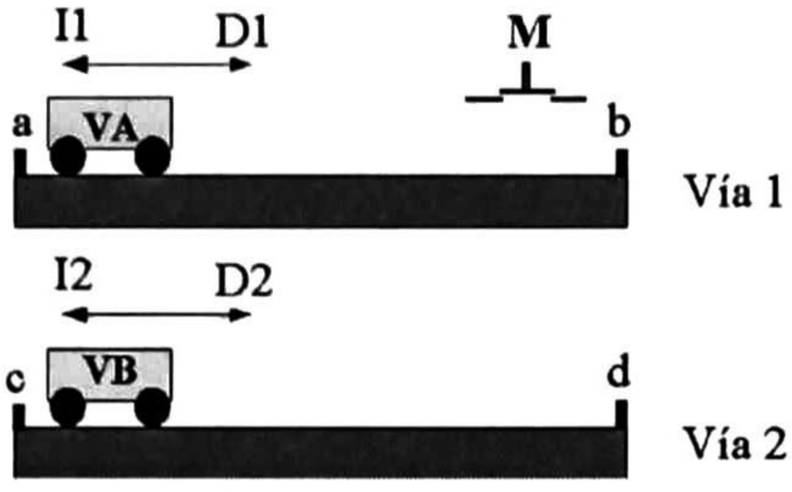


Figura 2.10: Modelo del sistema de los dos vagones.

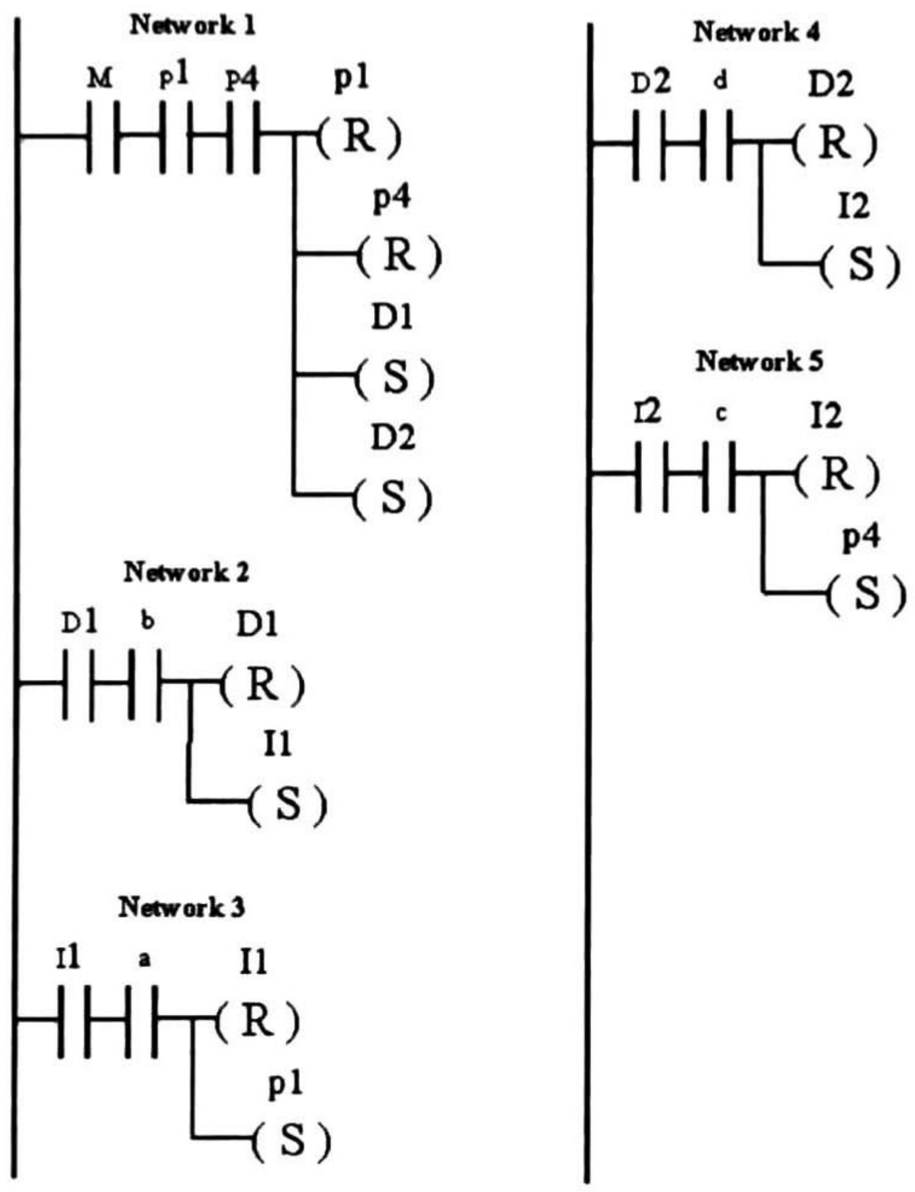


Figura 2.11: Representación en LD del modelo del sistema de los dos vagones.

esquema es expresado en **Redes de Petri Interpretadas Temporizadas (RdPIT)** que en la siguiente sección se explica.

2.4.1 Ejemplo de programación usando LD

El sistema mostrado en la figura 2.10 está compuesto de dos vagones **VA** y **VB** que se desplazan sobre vías independientes por acción de las señales $\mathbf{D_i}$ (movimiento hacia la derecha) e $\mathbf{I_i}$ (movimiento a la izquierda), donde $i = 1, 2$. Las posiciones extremas son detectadas por los sensores **a, b** para la vía 1 y **c, d** para la vía 2. Un botón **M** sirve para iniciar la operación del sistema desde su posición inicial: ambos se encuentran en el extremo izquierdo.

Su comportamiento es el siguiente: Al ser oprimido **M**, ambos vagones parten hacia la derecha; al llegar éstos a su extremo derecho debe regresarse y detenerse al alcanzar su posición inicial. Las velocidades de los vagones pueden ser cualesquiera.

Su representación en **LD** se muestra en la figura 2.11 y la interpretación a cada **red LD** es la siguiente:

1. **Red de LD 1**

Cuando **M** es presionado y los 2 trenes están en estado de reposo (contactos **P1** y **P4** respectivamente) se activan las señales que representan el avance de ambos vagones hacia la derecha (**D1** y **D2**) y se desactiva las señales **P1** y **P4** que son las señales de reposo de los vagones **VA** y **VB** respectivamente.

2 Red de LD 2

Una vez que el vagon **A** llega al extremo derecho y se oprime el detector b (contactos **D1** y **b** respectivamente) se cambia el sentido del vagón haciendo que se desplaze hacia la izquierda (bobinas **D1** e **I1**).

3 Red de LD 3.

Al llegar el vagon **A** al extremo izquierdo y se oprime el detector **a** (contactos **I1** y **a** respectivamente), se desnergiza la vía 1 (**I1**) y se activa el estado de reposo para el tren 1 (bobina **P1**).

4 Red de LD 4

Una vez que el tren **B** llega al extremo derecho y se oprime el detector **d** (contactos **D2** y **d** respectivamente) se cambia el sentido del vagón haciendo que se desplaze hacia la izquierda (bobinas **D2** e **I2**).

5 Red de LD 5

Al llegar el vagon **B** al extremo izquierdo y se oprime el detector C (contactos **I2** y **a** respectivamente), se desnergiza la vía 2 (**I2**) y se activa el estado de reposo para el tren 2 (bobina **P2**).

2.5 Redes de Petri.

2.5.1 Conceptos básicos de Redes de Petri

Una Red de Petri (**RdP**) es un grafo dirigido que contiene dos tipos de nodos: *lugares y transiciones*; éstos están relacionados por arcos orientados que unen siempre nodos de diferente tipo. Gráficamente como se muestra en la figura 2.12, *los lugares* se representan por círculos, *las transiciones* por barras o rectángulos y *los arcos* por flechas [21].

Definición 1 *Una Red de Petri es el par* $N = (G, M_0)$, *donde :*

G es una estructura de RP, y

M_0 la distribucion inicial del marcado.

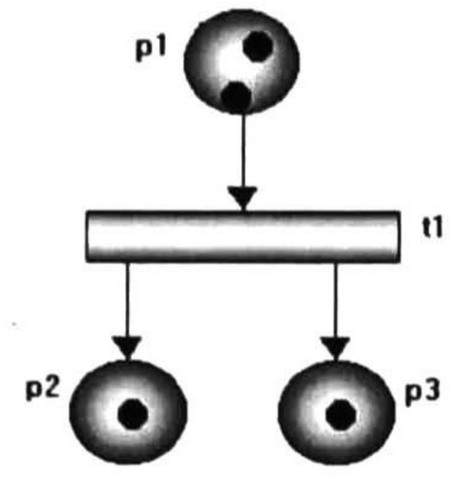


Figura 2.12: Representación gráfica de una Red de Petri

Definición 2 *Una estructura de* ***Red de Petri*** *N es un digrafo bipartido representado por la 4-tupla $G = (P, T, I, O)$ donde:*

- *$P = \{p_1, p_2, .., p_n\}$ es un conjunto de n elementos llamados* ***lugares,***

- *$T = \{t_1, t_2, .., t_m\}$ es el conjunto de m elementos llamados* ***transiciones,***

- *$I : P \times T \longrightarrow \mathbb{Z}^+$ es la* ***función de entrada*** *que representa los arcos que van de los lugares hacia las transiciones, la cual se define de la siguiente forma:*

$$I(p_i, t_j) = \begin{cases} 1 & \text{si el arco de } p_i \text{ a } t_j \text{ existe} \\ 0 & \textit{en caso contrario} \end{cases}$$

- *$O : P \times T \longrightarrow \mathbb{Z}^+$ la* ***función de salida*** *que representa los arcos que van de las transiciones hacia los lugares, definida como sigue:*

$$O(p_i, t_j) = \begin{cases} 1 & \text{si el arco de } t_j \text{ a } p_i \text{ existe} \\ 0 & \text{en caso contrario} \end{cases}$$

donde Z^+ es el conjunto de número enteros no negativos.

Definición 3 *Sean $n = |L|$ el número de lugares en la red y $m = |T|$ el número de transiciones de una RP. La estructura de la RP se puede representar por medio de dos matrices:*

- *Se denomina matriz de entrada o incidencia previa a la matriz:*

$$C^- = [c^-_{i,j}]_{n \times m}$$

donde $c^-_{i,j} = I(p_i, t_j)$.

- *Se denomina matriz de salida o incidencia posterior a la matriz:*

$$C^+ = [c^+_{i,j}]_{n \times m}$$

donde cada elemento de la matriz $c^+_{i,j} = O(p_i, t_j)$.

- *Se denomina matriz de incidencia a la matriz:*

$$C = C^+ - C^-$$

Marcado

Una Red de Petri (**RdP**) representa un comportamiento dinámico con ayuda de marcas; éstas están contenidas dentro de los lugares y son representadas gráficamente por puntos; cuando el número de marcas dentro de un lugar es relativamente grande, se inscribe un entero. La distribución de marcas en un instante dado se denomina marcado y es análogo a la noción de estado estable. Una **RdP** con marcas recibe el nombre de **RdP** marcada. Se define el marcado inicial como el conjunto de lugares marcados al inicio de una ejecución de la **RdP**

- M es *una* ***función de marcado*** *definida como* $M : P \longrightarrow \mathbb{Z}^+$ *es la asignación de un número entero no negativo de elementos llamados marcas a cada lugar de* N*. Una* ***marca*** *se representa por medio de un punto en el interior de un lugar. La distribución de las marcas, o el marcado, representa el estado de la Red de Petri. Por notación,* $M = M(P)$ *será llamado simplemente el* marcado.

- M_0 *es un* ***marcado inicial****, el cual es una asignación arbitraria de marcas en el momento inicial.*

Evolución

El comportamiento dinámico con ayuda de marcas; éstas están contenidas dentro de los lugares y son representadas gráficamente por puntos; dicho comportamiento puede describirse mediante la evolución de las marcas en la red. Aquí, las transiciones juegan un papel importante pues cada una de ellas están encargadas de "mover" marcas desde sus lugares de entrada (cuyos arcos de salida llegan a la transición) hacia sus lugares de salida (aquellos cuyos arcos de entrada provienen de la transición): esto es posible cuando ocurre el disparo de una transición.

a) *Regla de habilitación. Una transición* $t_k \in T$ *de* N *está habilitada si* $\forall p_i \in P$ *de* N, $M(p_i) \geq I(p_i, t_k)$*. Si una transición* t_k *está habilitada, entonces se puede disparar.*

b) *Regla de disparo. Si una transición habilitada* t_j *es disparada en un marcado* M_k *entonces se alcanza el nuevo marcado* M_{k+1}*, eliminando* $I(p_i, t_j)$ *marcas de sus lugares de entrada* p_i *y añadiendo* $O(p_i, t_j)$ *marcas a sus lugares de salida* p_i*.*

Modelado de sistemas

En el modelo de Sistemas de Eventos Discretos con **RdP**, los lugares, las transiciones y las marcas toman un cierto significado acorde al contexto del sistema que se desea modelar.

Los lugares pueden representar recursos, operaciones, estados parciales, etapas de proceso.

Las marcas representan la disponibilidad de recursos, la orden de ejecución de operaciones, información transmitida, etc.

Las transiciones tienen generalmente asociados los eventos tales como el inicio o fin (o ambos) de actividades, comandos o información relevante del entorno.

Una red a la cual se le ha asociado un significado a sus componentes se le llama ***RdP Interpretada.***

Las Redes de Petri Interpretadas (RPI) son una extensión de las redes de Petri que nos permiten representar lenguajes de entrada y salida de SED´s. Un lector interesado puede consultar [15] [13] para una información con más detalle sobre las RPI.

Definición 4 *Una **Red de Petri Interpretada** (RPI) es la 5-tupla* $Q = (N, \Sigma, \Phi, \lambda, \varphi)$ *donde*

- $N = (G, M_0)$ *es una RP,*
- $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_r\}$ *es el alfabeto de entrada de la red, donde* σ_i *es un símbolo de entrada,*
- $\Phi = \{\phi_1, \phi_2, ..., \phi_s\}$ *es el alfabeto de salida de la red, donde* ϕ_i *es un símbolo de salida*
- $\lambda : T \rightarrow \Sigma \cup \{\varepsilon\}$ *es una función de etiquetado de transiciones con las siguientes restricciones:* $\forall t_j, t_k \in T$, $j \neq k$ *si* $I(p_i, t_j) = I(p_i, t_k) \neq 0$ *y ambas* $\lambda(t_j)$, $\lambda(t_k) \neq \varepsilon$, *entonces* $\lambda(t_j) \neq \lambda(t_k)$, *donde* ε *representa un evento interno del sistema,*
- $\varphi : \mathbf{R}(N, M_0) \rightarrow [\mathbb{Z}^+]^q$ *es una función de salida, donde* $\mathbf{R}(N, M_0)$ *es el conjunto de alcanzabilidad definido como en RP. El vector de salida* $\varphi(M_k)$ *de la red es un vector con q elementos*[9].

2.5.2 Propiedades de las Redes de Petri

Las **RdP** tiene ciertas propiedades, que, cuando se interpretan en el contexto del modelado de un sistema de manufactura, permiten identificar la presencia o ausencia de las propiedades funcionales del sistema, tales como *reversibilidad, acotabilidad, conservatividad, vivacidad [6].*

Reversibilidad.

La reversibilidad o ciclicidad está relaciona con el hecho que un modelo admita siempre, a partir de cualquier estado alcanzado, secuencias de disparo que conduzcan al marcado inicial. Esto garantiza que el sistema no alcanzará un estado o un subconjunto de estados en los que permanecerá indefinidamente.

Definición 5 *Una **RdP** es reversible (propia, reinicializable o globalmente cíclica) para un marcado inicial* M_o *ssi* $\forall M \in R(M_o), M_o \in R(M)$

Acotabilidad.

Un parámetro que es importante conocer en una **RdP** es el número máximo de marcas que un lugar puede acumular, o bien si existen lugares en los cuales el marcado crece indefinidamente.

Definición 6 *Un lugar* $p_i \in P$ *de una **RdP** con un marcado inicial* M_o *está k-acotado ssi* $\forall M \in R(M_o), M(p_i) \leq k$. *Si k es el número de marcas más grande de la red se dice que la **RdP** está k-acotada.*

Definición 7 *Una RP es **estructuralmente acotada** si es acotada para cualquier marcado inicial finito.*

Conservatividad.

Definición 8 *Una RP con un marcado inicial M_0 es* ***estrictamente conservativa*** *ssi* $\forall M \in R(M_0)$

$$\sum_{i=1}^{n} M(p_i) = \sum_{i=1}^{n} M_0(p_i)$$

con $n =\mid P \mid$; es decir que $\mid \bullet t_j \mid = \mid t_j \bullet \mid \forall t_j \in T$

donde $\bullet t_j$ es el conjunto de lugares de entrada de la transición t_j y $t_j \bullet$ es el conjunto de lugares de salida de t_j.

Definición 9 *Una RP con un marcado inicial M0 es* ***conservativa*** *respecto a un vector de ponderación $w = [w_1, w_2 ... w_n]$, $w_i \geq 0$, si $\forall M \in R(M_0)$*

$$\sum_{i=1}^{n} w_i M(p_i) = \sum_{i=1}^{n} w_i M_0(p_i)$$

Vivacidad.

En una **RdP** un bloqueo significa que, para un marcado dado, ninguna transición puede ser disparada. Por otro lado, una **RdP** que no se bloquea puede evolucionar solo en una parte de ella. En cualquiera de estos dos casos se dice que la **RdP** es no viva.

Definición 10 *Una* ***transición*** *t es* ***viva*** *para un marcado inicial M_0 en una RP ssi existe una secuencia de disparos σ a partir de cualquier marcado M, alcanzables desde M_0, que comprenda a t :*

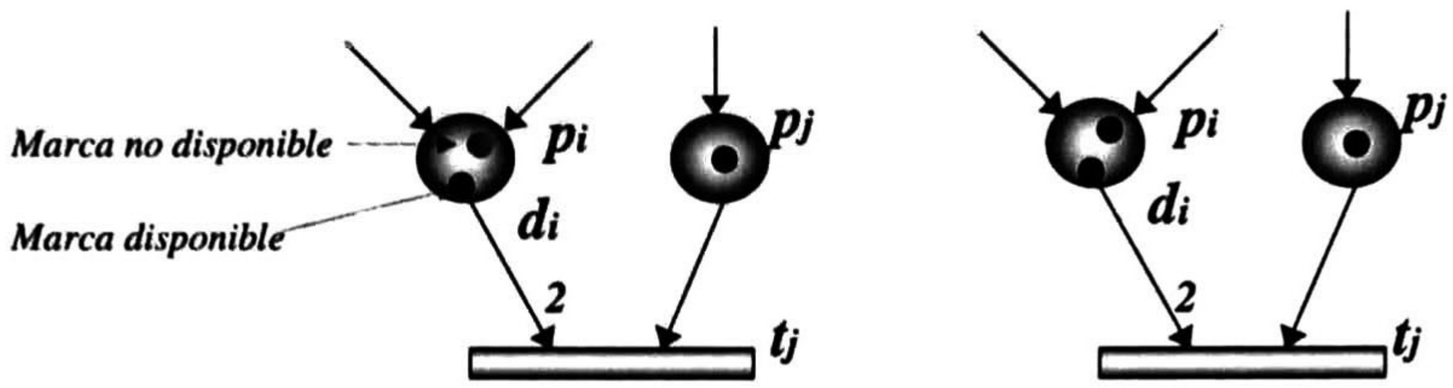
$$\forall M \in \mathcal{R}(N) \exists \sigma : M \xrightarrow{\sigma} M' \mid t \in \sigma$$

2.5.3 Redes de Petri temporizadas

La inclusión del tiempo en una **RdP** puede ser hecha asignando tiempo a las transiciones o a los lugares. Una **RdP** a la cual el tiempo asociado a los lugares se le llama *Redes de Petri con lugares temporizados* ***(RdPTL)*** [18]. En esta convención la duración o retardo afecta a las marcas; cada lugar p_i tiene asociado una duración d_i que es una restricción sobre la disponibilidad de las marcas que lo ocupen: una marca está no disponible antes de transcurrir un tiempo d_i a partir del instante en que llega al lugar p_i; después de transcurrir ese tiempo la marca está disponible. Las marcas disponibles son las únicas capaces de habilitar las transiciones de una **RdPTL**.

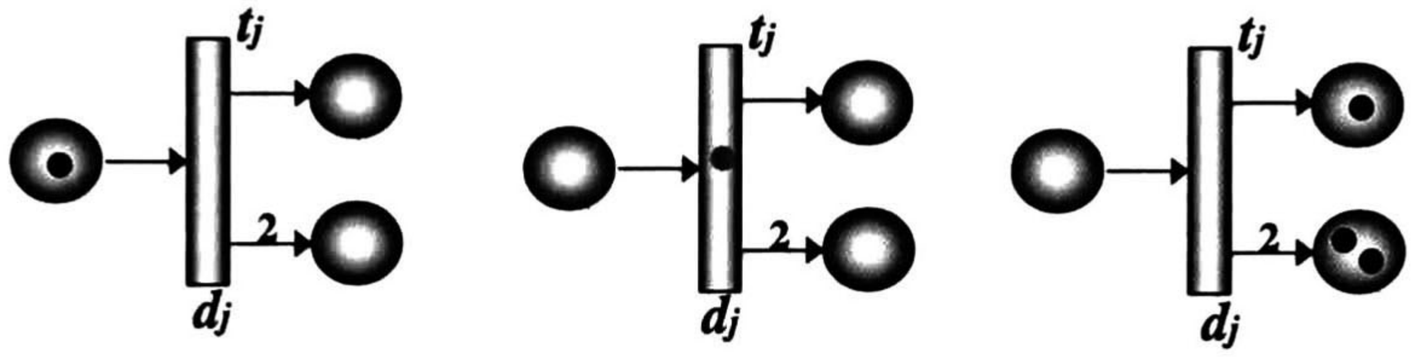
La figura 2.13 ilustra estas nociones; esta convención gráfica no es usada, sólo se dan con propósitos ilustrativos.

Una **RdP** a la cual el tiempo está asociado a las transiciones se le llama *Redes de Petri con transiciones temporizadas* (**RdPTT**). En esta convención el retardo d_j ligado a una transición t_j indica la duración del disparo de ésta. La regla de disparo es como en **RdP** con una ligera variante: al dispararse t_j las marcas son removidas de los lugares de entrada; las que se generan en los lugares de salida aparecen un tiempo d_j después. La figura 2.14 ilustra lo anterior, t_j es dibujada como una caja que "retiene" las marcas durante el tiempo de disparo asociado.



a) t_j no habilitada b) t_j habilitada

Figura 2.13: Representación de la estructura de un lugar temporizado en dos instantes diferentes; las marcas disponibles y no disponibles.



a) t_j habilitada b) Disparo de t_j c) d_j después

Figura 2.14: Representación de la estructura de una transición temporizada antes, durante y después del disparo de la transición t_j.

Definición 11 *Una Red de Petri Temporizada es una* $Q_T = (Q, D_T, D_P)$ *donde*

- Q *es una RPI*
- $D_T : T \longrightarrow \mathbb{R}^+$ *es una función que representa el retardo asociado a la transición. El disparo de* t_j *remueve una marca de cada lugar de entrada de* t_j, *y añade una marca para cada lugar de salida de la transición* t_j $D_T(t_j)$ *unidades de tiempo después..*
- $D_P : \Phi \longrightarrow \mathbb{R}^+$ *es una función que representa la disponibililidad de marcas y la duración de la señal de salida asociada al lugar* p.

En este trabajo, la finalidad de emplear las **RdP** como esquema de alto nivel para modelar un controlador es debido a las siguientes razones:

1. El modelo en **RdP** del controlador de un sistema es más legible por la representación gráfica que éste ofrece.
2. Es una excelente herramienta de modelado para los sistemas de eventos discretos.
3. El análisis del sistema mediante las **RdP** es con un enfoque matemático por lo que los resultados son óptimos y exactos.
4. Permite a través de la evolución de la **RdP** la visualización del comportamiento deseado del sistema.

Con todas estas ventajas que ofrecen las **RdP**, nace la inquietud de utilizarlas como un esquema de alto nivel. Este esquema facilita la generación de **LD** con la metodología que se presenta en el siguiente capítulo.

2.5.4 Ejemplo.

Retomando el enunciado del sistema descrito en la sección anterior, en la figura 2.10 el modelo correspondiente expresado en **RP** es el mostrado en la figura 2.15 asociándole la siguiente interpretación:

Para los lugares:
$p1$: Vagón 1 en reposo.
$p2$: Vagón 1 desplazándose a la derecha (**D1**).
$p3$: Vagón 1 desplazándose a la izquierda (**I1**).
$p4$: Vagón 2 en reposo.
$p5$: Vagón 2 desplazándose a la derecha (**D2**).
$p6$: Vagón 2 desplazándose a la izquierda (**I2**).
Para las transiciones:
$t1$: Se oprime el botón **M**.
$t2$: Detector **b** oprimido
$t3$: Detector **a** oprimido
$t4$: Detector **d** oprimido
$t5$: Detector **c** oprimido

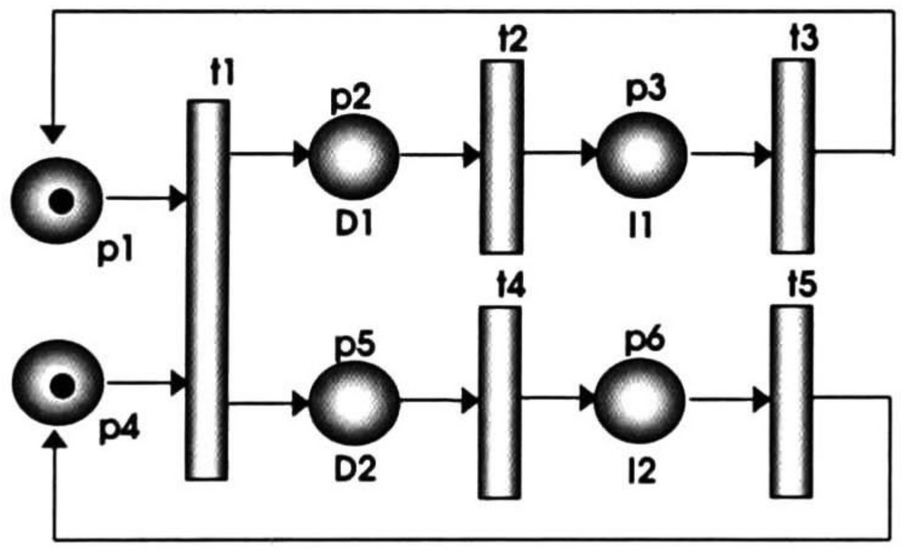


Figura 2.15: Representación en Red de Petri del modelo del sistema de los dos vagones.

Los lugares p_1 y p_4 indican que los vagones están en estado de reposo, lo cual habilitan a la transición t_1. Cuando se oprime el botón M, se dispara t_1 donde quita marcas en los lugares p_1 y p_4 y las deposita en p_2 y p_5 que representan tanto el vago 1 y 2 el desplazamiento hacia la derecha. Suponiendo que llega el vagón 1 a su extrema derecha, se oprime el botón b, la transición t_2 se dispara y ahora cambia el sentido hacia la izquierda que lo indica el disparo de t_2 al quitar la marca del lugar p_2 y es depositada en el lugar p_3. Al llegar a la extrema izquierda, se dispara la transición t_3 para llegar a la condición inicial del vagón: estado de reposo. Esto mismo es el caso para el vagón 2 pero con sus respectivos lugares (p_5 y p_6) y transiciones (t_4 y t_5).

2.6 Conclusiones

Este capítulo presentó qué son los Controladores Lógicos Programables, las ventajas que ofrece el estándar **IEC1131** que está formado por los lenguajes de programación IL, ST, SFC, **LD** y la descripción de cada bloque que conforma un **PLC**.

Se mencionan las razones por las cuales los **LD's** no son adecuados para el análisis del comportamiento que ha sido programado en un **PLC**. Se definen las **RdP**, un esquema de alto nivel que se utilizan para especificar, analizar y validar el comportamiento de un controlador y así emplearlas en la metodología que se detalla en el siguiente capítulo.

Capítulo 3

Conversión de Redes de Petri a Diagramas de Escalera

Resumen. Este capítulo presenta una metodología que convierte un modelo en Red de Petri (**RdP**) a un Diagrama de Escalera (**LD** Ladder Diagram). El principio de la metodología está fundamentada en que cada transición de la **RdP** es equivalente a una **red de LD**. Los elementos básicos de **LD** que se asocian a los lugares de la **RdP** son las bobinas y relevadores; los tiempos asignados a las transiciones y lugares, se representan con temporizadores y para los lugares no binarios de la **RdP** se utilizan contadores y condicionales.

3.1 Introducción

La implementación de un controlador es una de las fases del ciclo de vida con mucha actividad de investigación en los Sistemas de Eventos Discretos. Debido a que se han enfocado más a las fases básicas como la especificación y análisis de un sistema han dejado a un lado las necesidades que hay en la programación de un Controlador Lógico Programable (**PLC**).

Se han desarrollado metodologías en herramientas visuales para verificar que el programa del **PLC** cumpla con los requerimientos iniciales. Algunos de ellos extraen una parte de un programa de algún lenguaje de programación para **PLC** y aplican una serie de métodos para verificar que cumpla con los requerimientos; en caso de no ser así, se tiene que hacer la modificación en el programa de **PLC** de forma empírica. La desventaja que hay con estas metodologías es que los tiempos de reprogramación son largos y recae sobre la experiencia del programador. Este tipo de investigaciones descuidan otros aspectos de la programación de un **PLC** ya que su reprogramación sigue siendo empírica y como consecuencia hay un alto porcentaje de la presencia de errores.

Otro tipo de investigaciones son los que se enfocan al desarrollo de metodologías para convertir un modelo en esquema de alto nivel a un lenguaje de programación para **PLC**, algunos de ellos, emplean los autómatas, álgebras booleanas, las Redes de Petri (**RdP**), entre otras, como herramientas de modelado de un sistema. La ventaja que hay en este tipo de investigaciones es que el programador del **PLC** solo se preocupa por modelar el comportamiento del sistema en el esquema de alto nivel y antes de su implementación obtiene los resultados del análisis matemático que se realiza sobre dicho modelo.

Este capítulo, presenta una metodología para generar **LD** a partir de las especificaciones en Redes de Petri. Esta metodología emplea Redes de Petri Interpretadas Temporizadas (**RdPIT**) que son usadas como herramienta formal para modelar el comportamiento del controlador y la interacción con el sistema a controlar porque capturan concurrencia, exclusión mutua, sincronización, etc.

3.2 Metodología para convertir una Red de Petri a un Diagrama de Escalera

Esta metodología permite obtener **LD** a partir de un modelo expresado en Redes de Petri Interpretadas Temporizadas (**RdPIT**).

Se emplean las Redes de Petri Temporizadas (**RdPT**) porque los tiempos de retardo asociados a los lugares y a las transiciones pueden representar por ejemplo, la duración de un proceso, la disponibilidad de recursos hasta después de un determinado tiempo o bien el retardo de una información transmitida, entre otros. Estos tiempos pueden son representados en **LD** con temporizadores.

Para las Redes de Petri Interpretadas (**RdPI**) cada símbolo asociado a un lugar o una transición tiene una interpretación del sistema físico, por ejemplo, que un vagón al llegar al extremo derecho de la vía tiene que cambiar de sentido para su regreso, durante su recorrido por un segmento de vía mantener energizado tal segmento, entre otros casos. Así, cada señal de entrada del sistema se representa en **LD** mediante un contacto.

La metodología que a continuación se describe, está dividida en tres partes; a) Principio de la metodología, b) Conversión de Redes de Petri Temporizadas, y c) Conversión de Redes

de Petri No Binarias.

Los elementos e instrucciones en **LD** usados son de la norma **IEC 1131**, los cuales su descripción y funcionamiento se detallaron en el capítulo 2.

3.2.1 Principio de la metodología.

La característica principal de esta metodología es que la traducción se hace modularmente, considerando todas las subredes formadas por una transición y sus lugares de entrada y de salida en una red de **LD**. Una red de **LD** es un renglón que está compuesto básicamente por 2 partes: *Las condiciones y las ejecuciones.* El comportamiento estructural de una **RdP** puede ser representado en tales partes, es decir, la analogía que hay entre las *reglas de habilitación y de disparo* de una transición con las *condiciones y ejecuciones* de una red de **LD**.

La representación de una transición en una red de **LD** se logra asociando cada objeto de entrada y salida de la transición con elementos básicos de **LD**. Así, a cada lugar de entrada a la transición le corresponde un contacto normalmente abierto y una bobina inactiva. Los lugares de salida de la transición se representa con bobinas inactivas. Si la transición tiene asociado una señal externa, ésta se representa mediante un contacto normalmente abierto.

La evolución de la **RdP** consiste en el disparo de la transiciones, donde sólo puede dispararse aquella transición donde sus lugares de entrada están marcados y la condición de disparo de la transición ocurre. Cuando la transición se dispara, se remueven las marcas de los lugares de entrada; en **LD** se desactivan las señales asociadas a los lugares de entrada; y depósita una marca en los lugares de salida de la transición, en **LD** se activa la señal correspondiente a cada lugar de salida. Ver figura 3.1. Esta estrategia es sistematizada mediante el siguiente algoritmo.

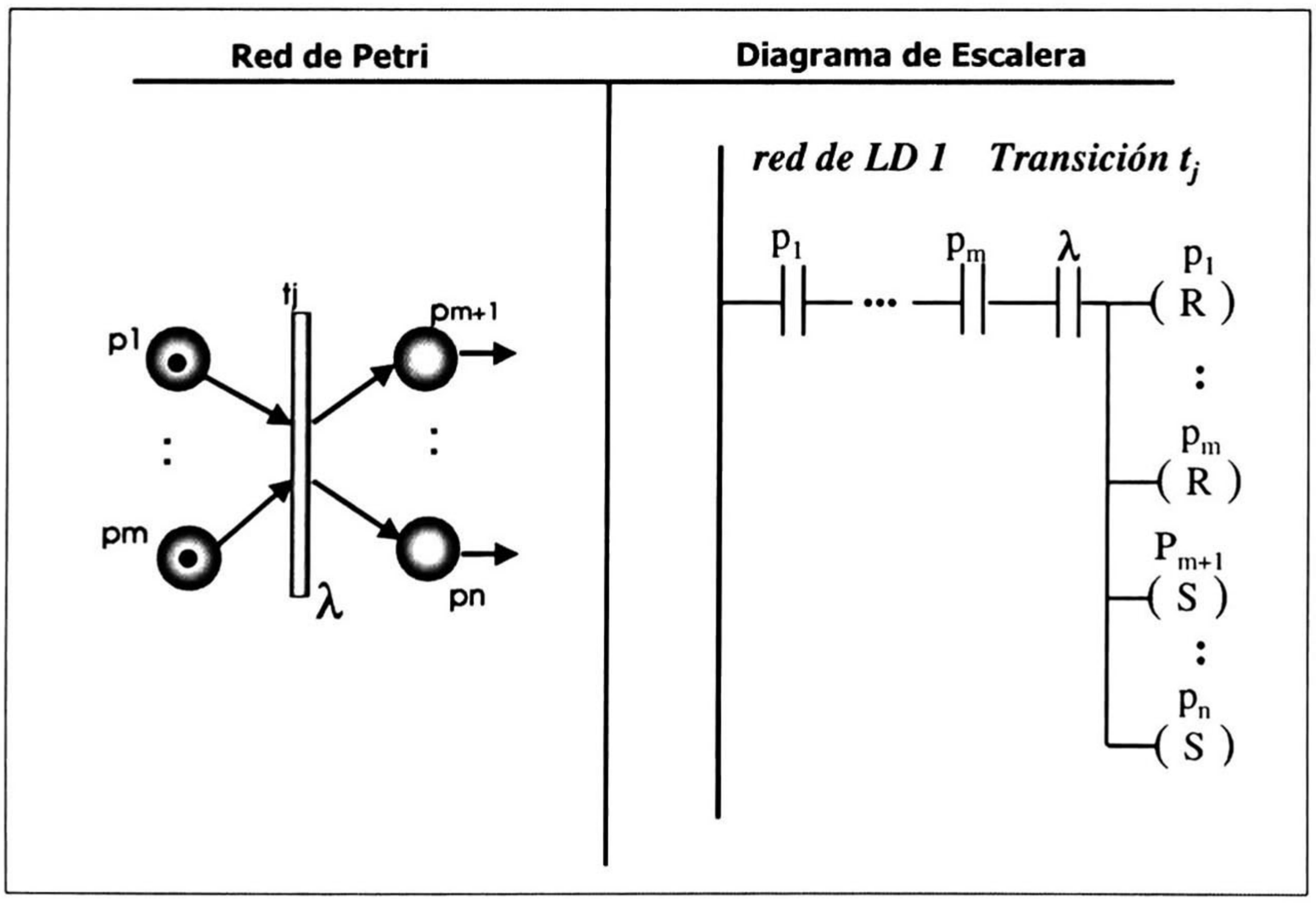


Figura 3.1: Representación en Diagrama de Escalera de una transición.

Algoritmo 12 ***Transición_ LD(t_j,p_i, p_j;LD(t_j))***. *Representación en* ***LD*** *de una Transición*

entradas: t_j= Transición de la Red de Petri.
p_i= Lugar de entrada de la transición t_j.
p_j= Lugar de salida de la transición t_j.
salidas: **LD**(t_j) red de **LD** de la transición t_j.
1. Nueva red de **LD** asociada a t_j
2. $\forall p_i \in \bullet t_j$ **HACER**
   (a) Añadir en serie un contacto normalmente abierto asociado a la señal de p_i.
3. Si $\lambda(t_j) \neq \varepsilon$ **ENTONCES**
   (a) Añadir contacto normalmente abierto asociado a la señal de entrada $\lambda(t_j)$
4. $\forall p_i \in \bullet t_j$ **HACER**
   (a) Añadir bobina inactiva asociado a la señal del lugar p_i
5. $\forall p_j \in t_j \bullet$ **HACER**
   (a) Añadir bobina activa asociada a la señal del lugar p_j.

3.2.2 Marcado Inicial.

El marcado inicial de un modelo en **RdP** se asigna mediante una red de LD simple, la cual consiste de:

- Un contacto normalmente abierto asociado a la señal externa de inicialización.
- Un conjunto de bobinas activas correspondientes a los lugares marcados inicialmente.
- Un conjunto de bobinas inactivas correspondientes al resto de los lugares no marcados.

La figura 3.2 muestra la representación en **LD** del marcado inicial.

Algoritmo 13 ***MarcadoInicial_ LD(p_m;LD(M_o))***. *Representación en* ***LD*** *del marcado Inicial*

entradas: p_m= Lugar de la Red de Petri
salidas: **LD**(M_o) red de **LD** del marcado inicial de la Red de Petri.
1. Nueva red de **LD**
2. Agregar contacto normalmente abierto asociado a la señal de inicio de la ejecución del sistema
3. $\forall\ p_m \in P$ **HACER**
   (a) Si $M(p_m) = 1$**ENTONCES**
      i. Añadir bobina activa asociada a la señal del lugar p_m
   sino
      i. Añadir bobina inactiva asociada a la señal del lugar p_m.

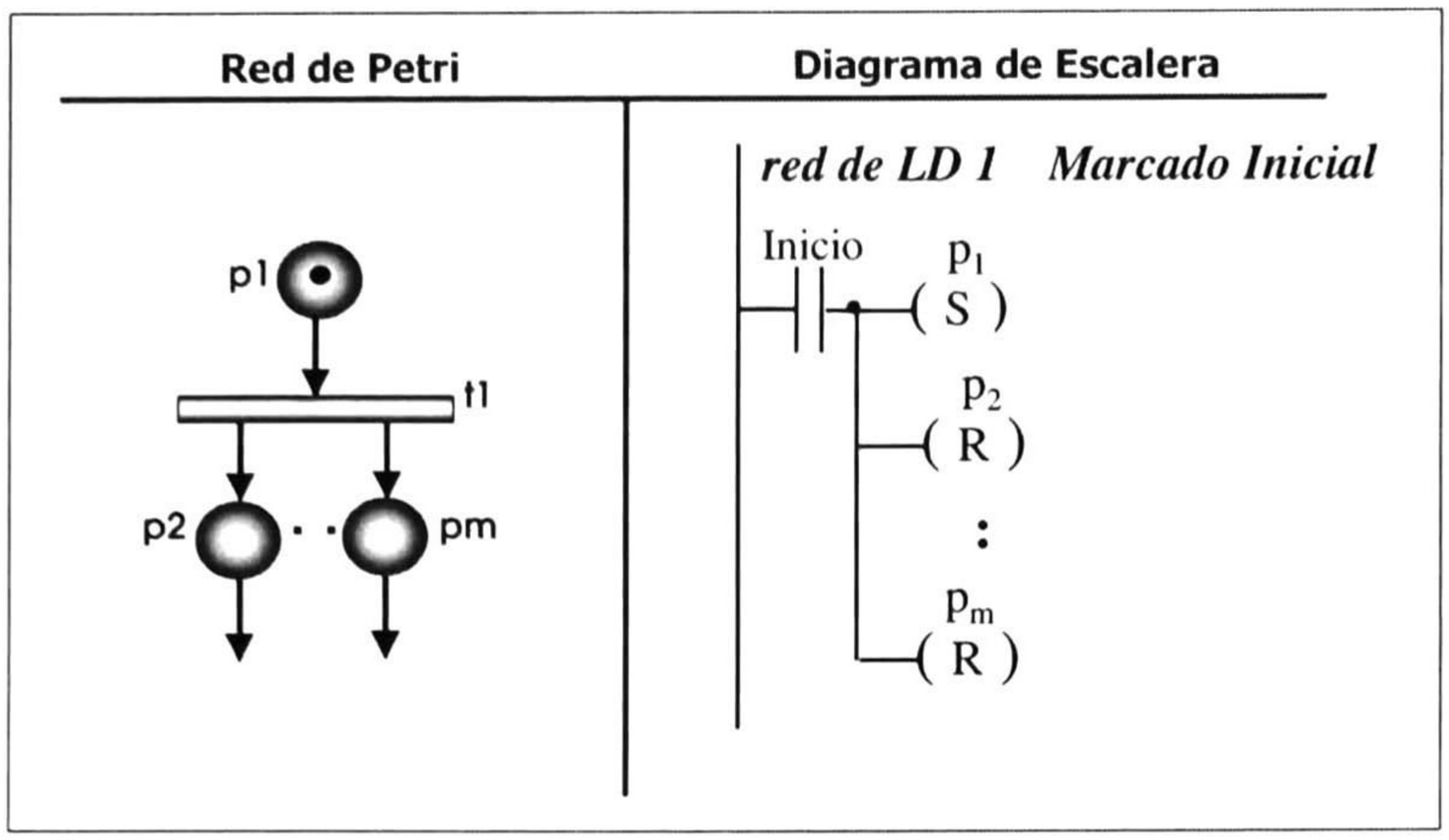


Figura 3.2: Representación en Diagrama de Escalera del Marcado Inicial de una **RdP**.

3.3 Tratamiento de nodos temporizados

Esta sección trata la representación en **LD** de las Redes de Petri Temporizadas (**RdPT**). Las **RdPT** son aquellas **RdP** a las que se les ha asignado un tiempo tanto a los lugares como a las transiciones. Cada uno de ellos tiene una interpretación diferente en la evolución de la **RdP**. Por tanto, el tiempo se asocia con un elemento de **LD** llamado temporizador. Los dos tipos de temporizadores que se emplean tienen un funcionamiento característico que se adecúa al comportamiento tanto de un lugar temporizado como de una transición temporizada.

Un lugar temporizado genera una señal durante un determinado tiempo; a éste se le asocia un Temporizador por Impulsos (TP) ya que el temporizador genera una salida mientras transcurre el tiempo asignado al lugar. Una transición temporizada retarda la acción asociada a ella; el tipo de temporizador que se utiliza un Temporizador con Retardo al Conectar (TON) ya genera una salida hasta que finaliza su cuenta.

Como veremos en esta sección, tanto los lugares como las transiciones temporizadas generan dos redes de **LD**.

3.3.1 Lugar Temporizado.

Los elementos básicos de **LD** que se emplean para representar un lugar temporizado son: un temporizador tipo **TP**, contactos normalmente abiertos y normalmente cerrados asociados al temporizador y una Salida Constante para generar la señal asociada al lugar temporizado p_i.

Retomando la definición de un lugar temporizado, la duración d_i o retardo de p_i afecta a las marcas, donde p_i tiene una restricción sobre la disponibilidad de las marcas.

El disparo de una transición habilitada t_j se realiza como en las **RdP**; en el instante del disparo, la marca que se deposita en p_i está no disponible antes de transcurrir un tiempo d_i. Después de transcurrir este tiempo, la marca está disponible y es capaz de habilitar la transición de salida t_k.

En **LD** se representa de la siguiente manera: La transición t_j se traduce como en la sección 3.2.1 con una ligera variante: En las ejecuciones de la red de LD, debe existir el

temporizador TP.

El estado no disponible de la marca en p_i se representa en una segunda red de **LD**. Esta red de **LD** tiene solo un contacto normalmente abierto asociado a la variable booleana de la salida Q del temporizador. Mientras se mantiene activo el temporizador (genera una salida) se produce la señal asociada del lugar p_i mediante la Salida Constante.

El estado disponible de la marca en p_i se representa con un contacto normalmente cerrado, indica que el temporizador ha finalizado su cuenta. Como se ilustra en la figura 3.3, este contacto normalmente cerrado debe estar en las condiciones de la red de **LD** de la transición t_k.

Algoritmo 14 ***LugarTemporizado_ LD***$(t_j, p_i, p_j, Dp(p_j)$;***LD***(t_j) ***).*** *Representación en* ***LD*** *de un lugar temporizado.*

entradas: t_j = Transición de la Red de Petri.
p_i= Lugar de entrada de la transición t_j.
p_j= Lugar de salida de la transición t_j.
$Dp(p_j)$= Tiempo asociado a p_j
salidas: **LD**(t_j) red de **LD** de la transición con lugares temporizados de entrada y salida de la Red de Petri.

1. Nueva red de **LD**
2. $\forall\ p_i \in \bullet t_j$ HACER
   (a) Añadir contacto normalmente abierto asociado a la señal de p_i.
   (b) Si $D_p(p_i) > 0$, ENTONCES
      i.Añadir contacto normalmente cerrado asociado al temporizador del lugar p_i
3. $\forall p_i \in \bullet t_j$ HACER
   (a) Añadir bobina inactiva asociado a la señal del lugar p_i
4. $\forall p_j \in t_i \bullet$ HACER
   (a) Agregar bobina activa asociada a la señal del lugar p_j
   (b) Si $D_p(p_j) > 0$ ENTONCES
      i. Añadir Temporizador TP asociado a p_j, con las siguientes propiedades:
         - Tiempo transcurrido = 0
           Tiempo predeterminado = $D_p(p_j)$
      -.Salida Q=variable Tp
5. $\forall p_j \in t_i \bullet$ HACER
   (a) Si $D_p(p_i) > 0$ ENTONCES
      i. Nueva red de escalera
      ii. Añadir contacto normalmente abierto asociado a la variable Tp_j
      iii. Añadir salida constante

3.3.2 Transición Temporizada.

El tiempo asociado a una transición se repesenta en **LD** con un temporizador tipo TON. La representación en **LD** del resto de los objetos asociados a la transición temporizada t_j son los mismos como se describieron en la sección 3.2.1. Una ligera variante que hay es que son representados en dos redes de **LD**. Estas dos redes de **LD** son el resultado de la dinámica del disparo de t_j.

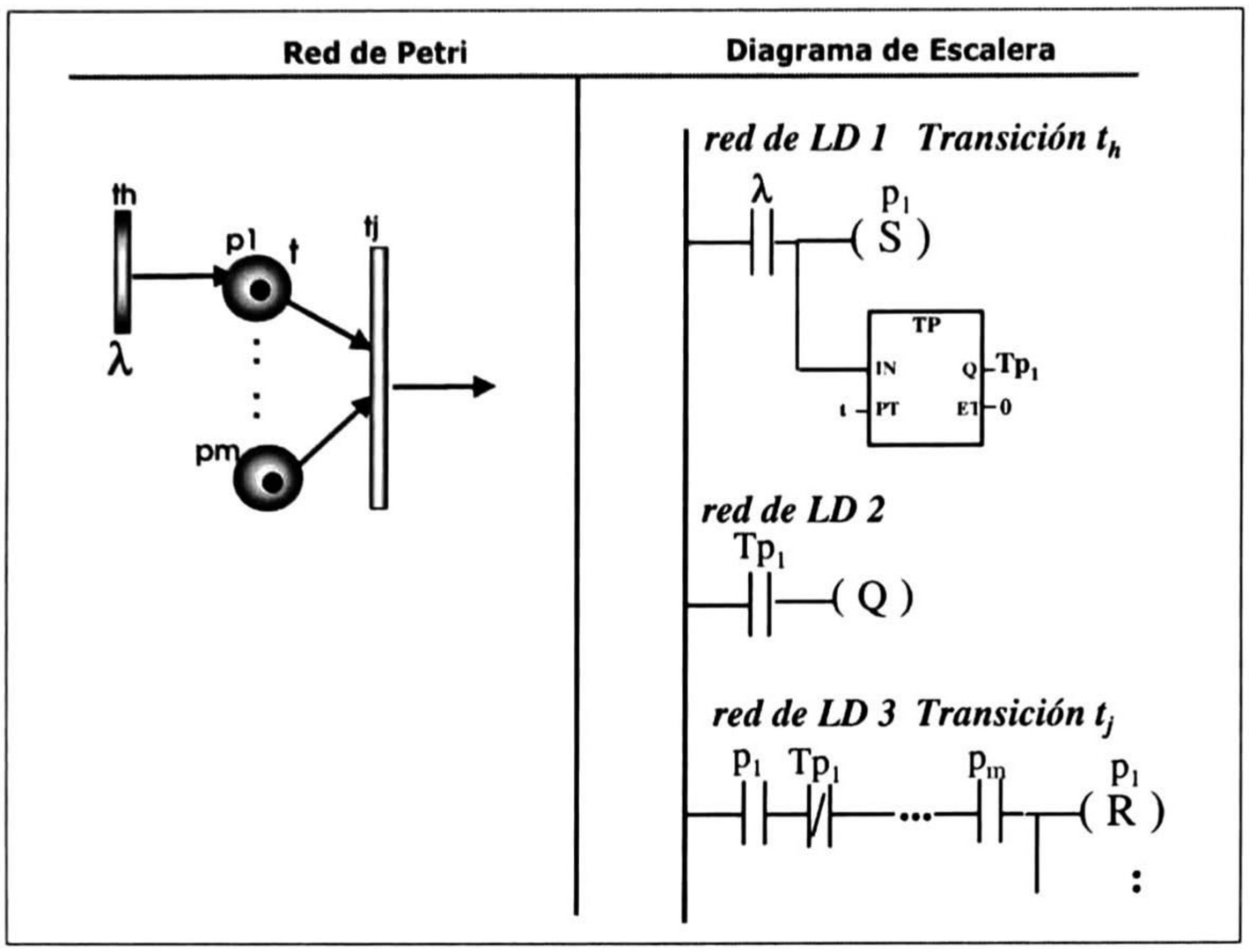


Figura 3.3: Representación en Diagrama de Escalera de un Lugar Temporizado

Recordemos que en RdPTT, al dispararse t_j, las marcas son removidas de los lugares de entrada y se mantienen "retenidas" durante el tiempo de disparo d_j asociado; las marcas que se generan en los lugares de salida aparecen un tiempo d_j después.

En la primera red de **LD**, se representa la desactivación de las señales asociadas a los lugares de entrada y la activación del temporizador TON.

En la segunda red de **LD** se activan las señales asociadas a los lugares de salida. El único contacto normalmente abierto de esta red de **LD** está asociada a una variable booleana de la salida Q del temporizador. Esta variable se activa cuando el temporizador se ha desactivado (ver figura 3.4).

El problema que existe en esta representación es cuando el lugar de entrada p_i de t_j es no binario; cuando el $M(p_i) > 1$ y t_j se dispara una vez, la transición puede dispararse de nuevo antes de concluir la ejecución del disparo anterior. A este comportamiento se le denomina *re-entrancia* de disparo de una transición.

El esquema de **LD** correspondiente descrito arriba no funciona correctamente ya que el temporizador TON se reinicializa al segundo disparo.

La solución que permite traducir correctamente los casos de transiciones reentrantes implica aplicar un método para efectuar una transformación estructural en la **RdPTT**.

En la teoría de las **RdPT**, hay dos métodos de equivalencia entre las **RdTT** y las Redes de Petri con Lugares Temporizados (**RdPLT**) [13]. Con estos métodos, es posible transformar una **RdPLT** a una **RdPTT** y viceversa. Aprovechando estas equivalencias, tomamos la convención de convertir una **RdPTT** a una **RdPLT** ya que facilitan su representación en **LD**.

Una transición temporizada es equivalente a dos transiciones y un lugar temporizado; la duración del disparo de t_j es representado por el tiempo en que las marcas permanece no disponible en p_{tj} (ver figura **??**).

A continuación se presenta el algoritmo para el modelado de la **RdPTT** a una **RdPLT**.

Algoritmo 15 ***RdPTT_ RdPLT***$(\mathbf{t}_j, \mathbf{d}_j; \mathbf{t_j}, \mathbf{p}_{tj}, \mathbf{t}_j\prime)$. *Conversión de una RdTT a una* ***RdPLT****.*

entradas: t_j= Transición temporizada de la **RdP**
d_j= Tiempo asociado a la transición t_j
salidas: t_j = Transición no temporizado.
p_{tj}= Lugar temporizado
t'_j= Transición no temporizada.

1. Agregar lugar p_{tj} con los siguientes atributos:
   (a) $d(p_{tj}) = d_j$
   (b) $\bullet p_{tj} = t_j$
   (c)$M(p_{tj}) = 0$
   (d) $K(p_{tj}) = 1$
2 Agregar transición $t_j\prime$ con los siguientes atributos:
   (a) $d(t_j\prime) = 0$
   (b) $\bullet t'_j = p_{tj}$
   (c) $t_j\prime\bullet = t_j\bullet$
3 $d(t_j) = 0$
4 $t_j\bullet = t_j \bullet \cap t'_j\bullet$

La **RdPLT** que se obtiene por medio del algoritmo anterior puede ser representada en **LD** con el algoritmo para generar **LD** de un lugar temporizado.

Por ejemplo, la **RdPLT** de la figura 3.6 es el resultado de aplicar la transformación **RdPTT-RdPLT**. De acuerdo al algoritmo 14, su representación en **LD** es el de la figura 3.6.

3.4 Conversión de Redes de Petri no binarias.

Hasta aquí se mostró la traducción a **LD** de las Redes de Petri Interpretadas Temporizadas (**RdPIT**) con lugares binarios. Una extensión de ellas, son las Redes de Petri no binarias. Las Redes de Petri no binarias son aquellas Redes de Petri que tienen lugares que pueden almacenar más de una marca. En esta metodología, los lugares no binarios se transforman en contadores ascendente/descendente (CTUD); se debe a que un lugar no binario, a parte de verificar que esté marcado, es necesario llevar la cuenta de las marcas que tiene.

Un punto importante que es necesario resaltar en el elemento contador CTUD, es la capacidad de incremento/decremento de su cuenta. De acuerdo al estándar IEC1131, dicha capacidad es de una sola unidad. En este trabajo se maneja la suposición de que permite el incremento/decremento de más de una unidad. Esto, porque esperamos que en un tiempo corto, el estándar IEC1131 modifique la estructura y funcionamiento de este elemento. Con ello se evitará que quede obsoleta esta parte de la metodología que trata con los lugares no binarios.

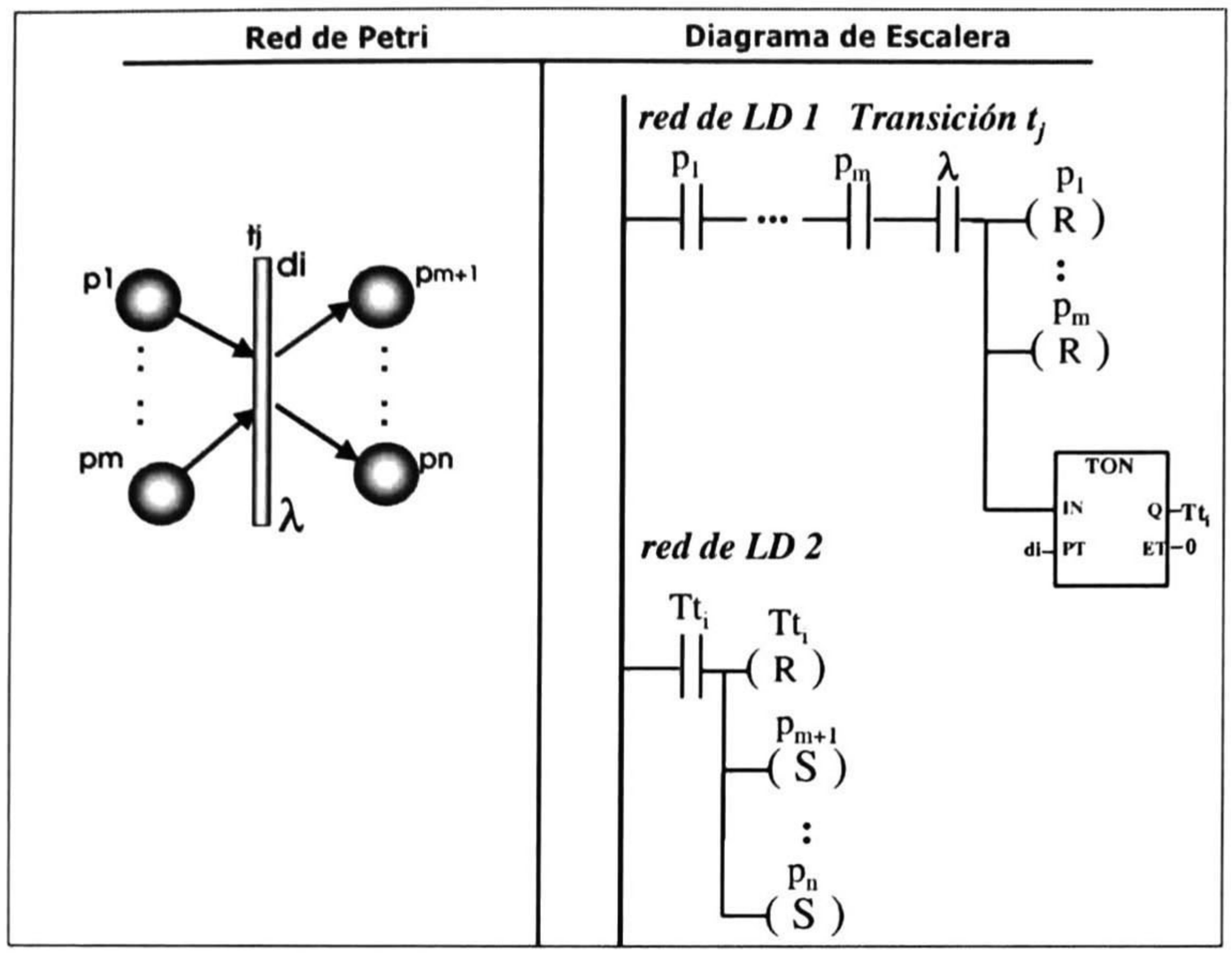


Figura 3.4: Representación en Diagrama de Escalera de una Transición Temporizada

3.4.1 Marcado Inicial no binario

El marcado inicial de una **RdP** con lugares no binarios se representa en **LD** de una forma muy similar de la sección 3.2.2. La diferencia lo hacen aquellos lugares donde se permite almacenar más de una marca. A esos lugares, en **LD** se les asocia un contador CTUD con la cuenta inicial igual al número de marcas que tenga el lugar. Aquellos lugares no binarios que estén desmarcados, también se les asocia un contador pero con valor inicial a 0. Esta inicialización con 0 se realiza con el fin de indicar que hay un lugar no binario desmarcado inicialmente en la **RdP**. Ver fig. 3.7

Algoritmo 16 ***MarcadoInicialNoBinario_LD***$(p_m, k_{um}; LD(M_o))$. *Representación en* ***LD*** *del marcado inicial de una Red de Petri no binaria.*

entradas: p_m= Lugar de la Red de Petri
k_{um}= Cota máxima del lugar p_m
salidas: **LD**(M_o) red de **LD** del marcado inicial de la Red de Petri no binaria.
1. Nueva red de **LD**
2. Añadir contacto normalmente abierto asociado al inicio de la ejecución del sistema
3. $\forall p_m \in P$ HACER
   (a) Si $M(p_m) > 0$ ENTONCES
      i. Añadir bobina activa asociada a la señal del lugar p_m
   sino
      ii. Añadir bobina inactiva asociada a la señal del lugar p_m
   (b) Si $k_{um} > 1$ ENTONCES
      i. Añadir contador tipo CTUD asociado al lugar p_m con las siguientes propiedades:
         - entrada de contaje adelante (CU)
         - valor actual = $M(p_m)$
         - valor predeterminado = k_{um}.

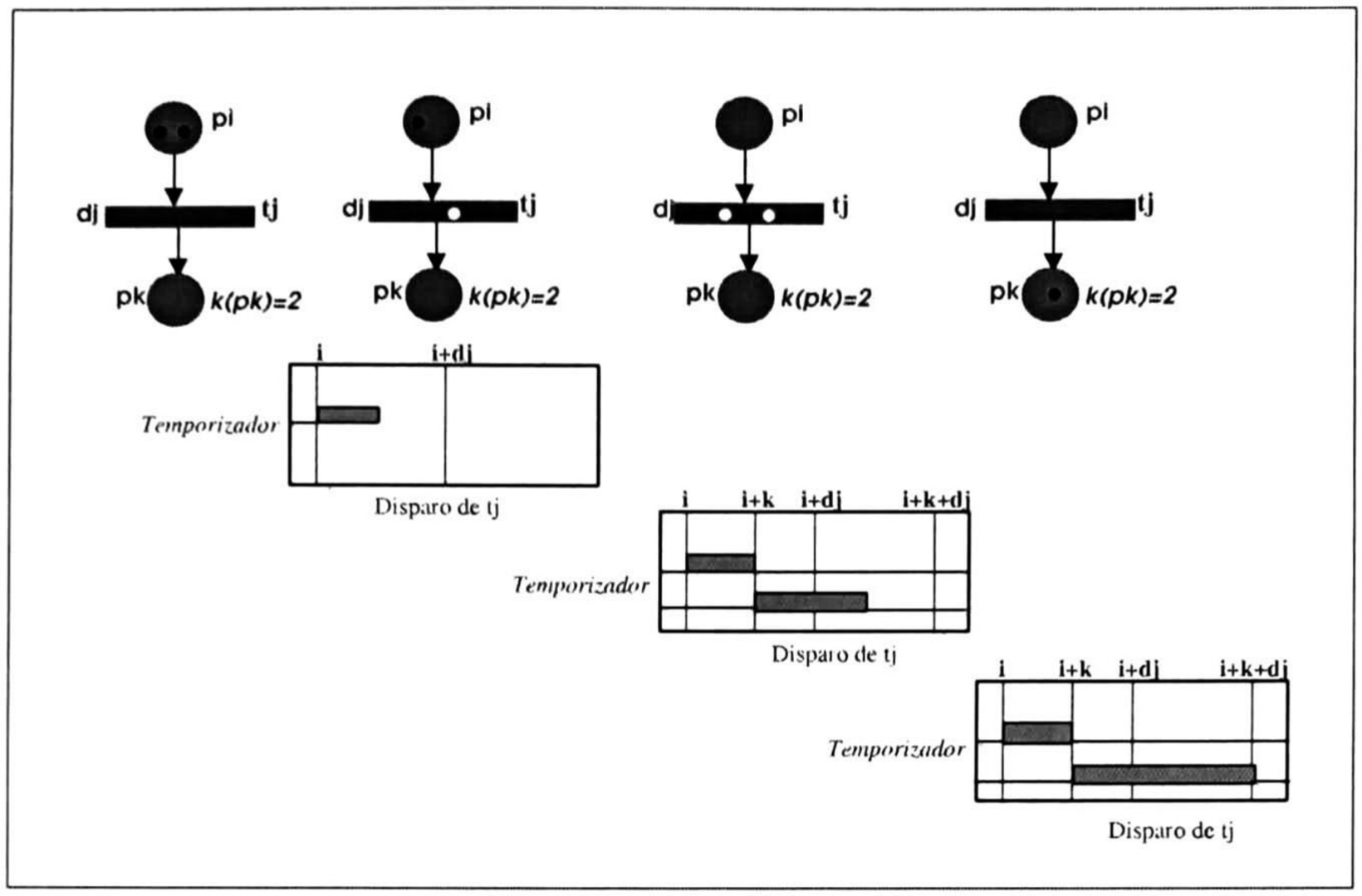


Figura 3.5: Diagramas de tiempo del temporizador asociado a la transición temporizada t_j.

3.4.2 Transición con lugares de entrada/salida no binarios

La representación en **LD** de una transición con lugares no binarios de entrada y salida es una extensión de la sección 3.2.1. A los lugares no binarios se les asocia un contador tipo ascendente/descendente (CTUD).

El flujo de las marcas en una **RdP** es simulada con el decremento e incremento del contador. El límite inferior y superior del contador estará determinado por las cotas mínimas y máximas del lugar no binario.

En las RdP no binarias, la transición t_j se habilita cuando sus lugares de entrada p_i cumple con la condición $M(p_i) \geq w(p_i, t_j)$, donde $M(p_i)$ es el marcado del lugar p_i y $w(p_i, t_j)$ es el peso del arco del lugar p_i a la transición t_j. Así, en las condiciones de la red de **LD** de la transición t_j debe existir condicionales GE (mayor ó igual a) para cada lugar de entrada p_i. Cada condicional evalúa si el valor actual del contador CTUD asociado al lugar p_i es mayor ó igual a $w(p_i, t_j)$. Por lo tanto, al disparar la transición t_j, quita $w(p_i, t_j)$ marcas a cada lugar p_i y depósita $w1(p_j, t_j)$ unidades en los lugares no binarios p_j de salida, donde $w1(p_j, t_j)$ es el peso del arco que va de la transición t_j al lugar lugar p_j

Por otro lado, en **LD**, cada contador incrementa $w_1(p_j, t_j)$ unidades y activan la señal asociada al lugar p_j cuando t_j es disparada.

Como se muestra en la figura 3.8, en la segunda red de **LD**, tiene un único contacto normalmente abierto asociado a una variable booleana del contador. Esta variable se activa mediante una de las salidas de contador (QD) cuando el valor es menor o igual a 0. En **RdP**, esto indica que el lugar ya no tiene marcas.

Algoritmo 17 ***TransiciónNoBinaria_LD***$(t_j, p_i, w(p_i, t_j), w_1(p_j, t_j), k_{ui}, k_{di}; \mathbf{LD}(t_j))$. *Representación en* ***LD*** *de una transición con lugares no binarios de entrada y salida.*

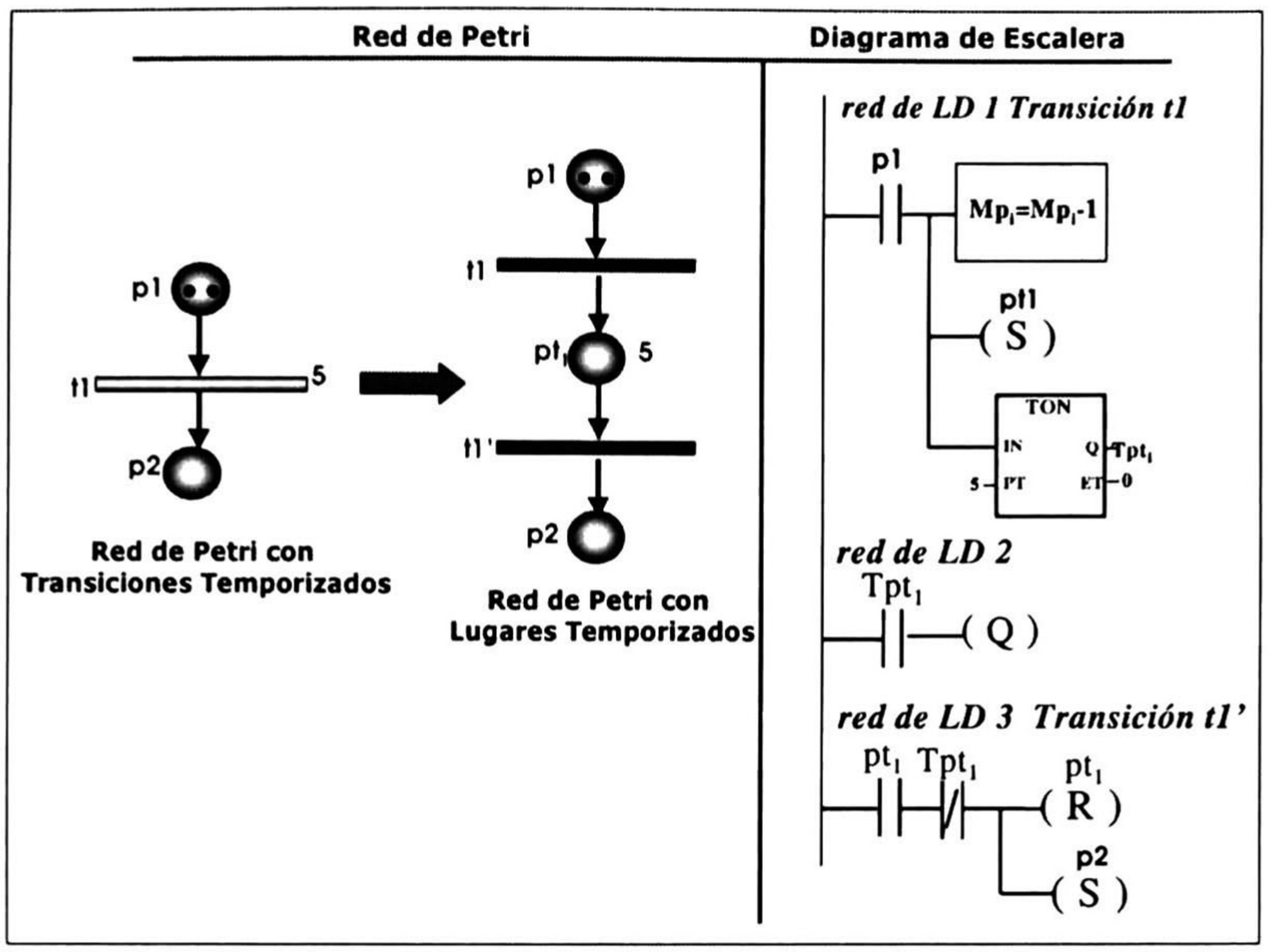


Figura 3.6: Representación en Diagrama de Escalera de la equivalencia de una RdPTT a una RdPLT.

entradas: t_j= transición de la Red de Petri no binaria.
p_i= Lugar no binario de entrada de la transición t_j.
p_j= Lugar no binario de salida de la transición t_j.
$w(p_i, t_j)$ = Peso del arco (p_i, t_j)
$w_1(p_j, t_j)$ = Peso del arco (t_j, p_j)
k_{ui} = Cota máxima del lugar p_i.
k_{di} = Cota mínima del lugar p_i
salidas: **LD**(t_j) red de **LD** de la transición t_j.

1. Nueva red de **LD**
2. $\forall p_i \in \bullet t_j$ HACER
   (a) Añadir condicional GEcon las siguientes características:
      i. Entrada IN1=contador CTUD asociado a p_i
      ii. Entrada IN2=$w(p_i, t_j)$
3. Si $\lambda(t_j) \neq \varepsilon$ ENTONCES
   (a) Añadir contacto normalmente abierto asociado a la señal de entrada $\lambda(t_j)$
4. $\forall p_i \in \bullet t_j$,HACER
   (a) Si $k_{ui} > 1$ ENTONCES
      i. Añadir contador tipo CTUD asociado al lugar p_i con las siguientes propiedades:
         entrada de contaje atrás (CD)
         - valor actual = $w(p_i, t_j)$
         valor predeterminado = k_{di}.
         - salida QU=variable booleana Cp_i
5. $\forall\ p_j \in t_j \bullet$ HACER
   (a) Añadir Salida Activa asociada a la señal del lugar p_j
   (b) Si $k_{uj} > 1$ ENTONCES

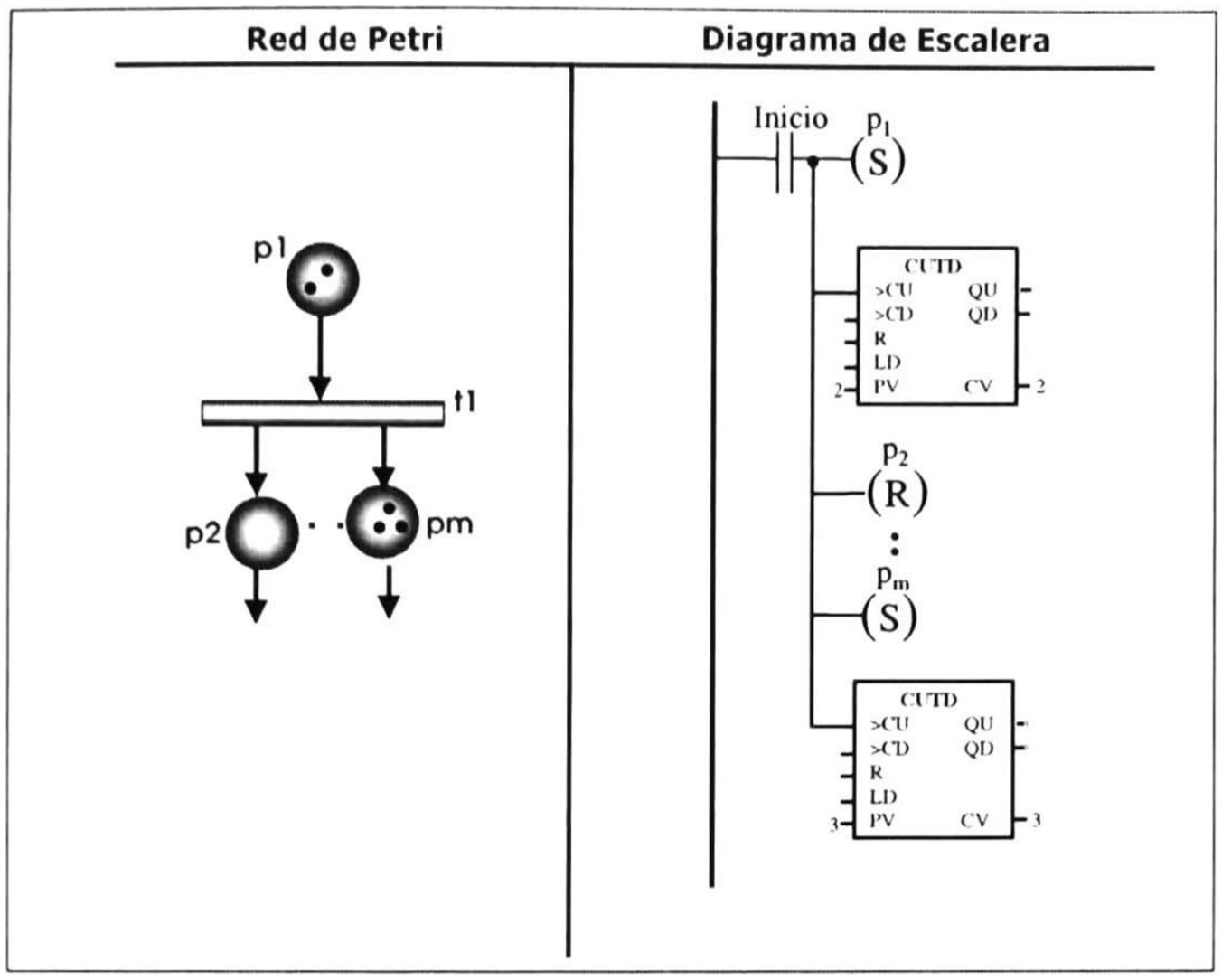


Figura 3.7: Representación en Diagrama de Escalera del Marcado Inicial de una Red de Petri No Binaria

i. Añadir contador tipo CTUD asociado al lugar p_j con las siguientes propiedades:
   - entrada de contaje adelante (CU)
   - valor actual $= w_1(p_j, t_j)$,
   - valor predeterminado $= k_{uj}$

6. $\forall p_i \in \bullet t_j$,HACER
   (a) Si $k_{ui} > 0$
      i. Nueva red de **LD**
      ii. Añadir contacto normalmente cerrado asociado a la variable Cp_i
      iii. Añadir bobina inactiva asociada a la señal del lugar p_i

3.4.3 Lugar Temporizado no binario.

Cuando una marca llega a un lugar binario temporizado p_j, entra al estado no disponible, permanece en p_j un tiempo d_i en espera. Cuando una marca no disponible termina su tiempo de espera, entonces entra al estado disponible; a este instante se le refererirá como "el tiempo de disponibilidad".

En el caso de un lugar no binario temporizado p_i, cada marca debe ser considerada individualmente con su propio "tiempo de disponibilidad", por lo que el lugar p_i se considera como una cola tipo FIFO (First Input, First Output), primero en entrar, primero en salir. Esta individualidad debe ser considerada en **LD** de tal forma que se pueda representar dicho "tiempo de disponibilidad", es decir, a cada marca le corresponderá un temporizador tipo TP.

Hacer posible la simulación del tiempo de cada marca de p_i, requiere de colocar los temporizadores de forma que se activen secuencialmente. Cada temporizador se ubicará en una red de **LD** y deberá activarse siempre y cuando los anteriores estén activos.

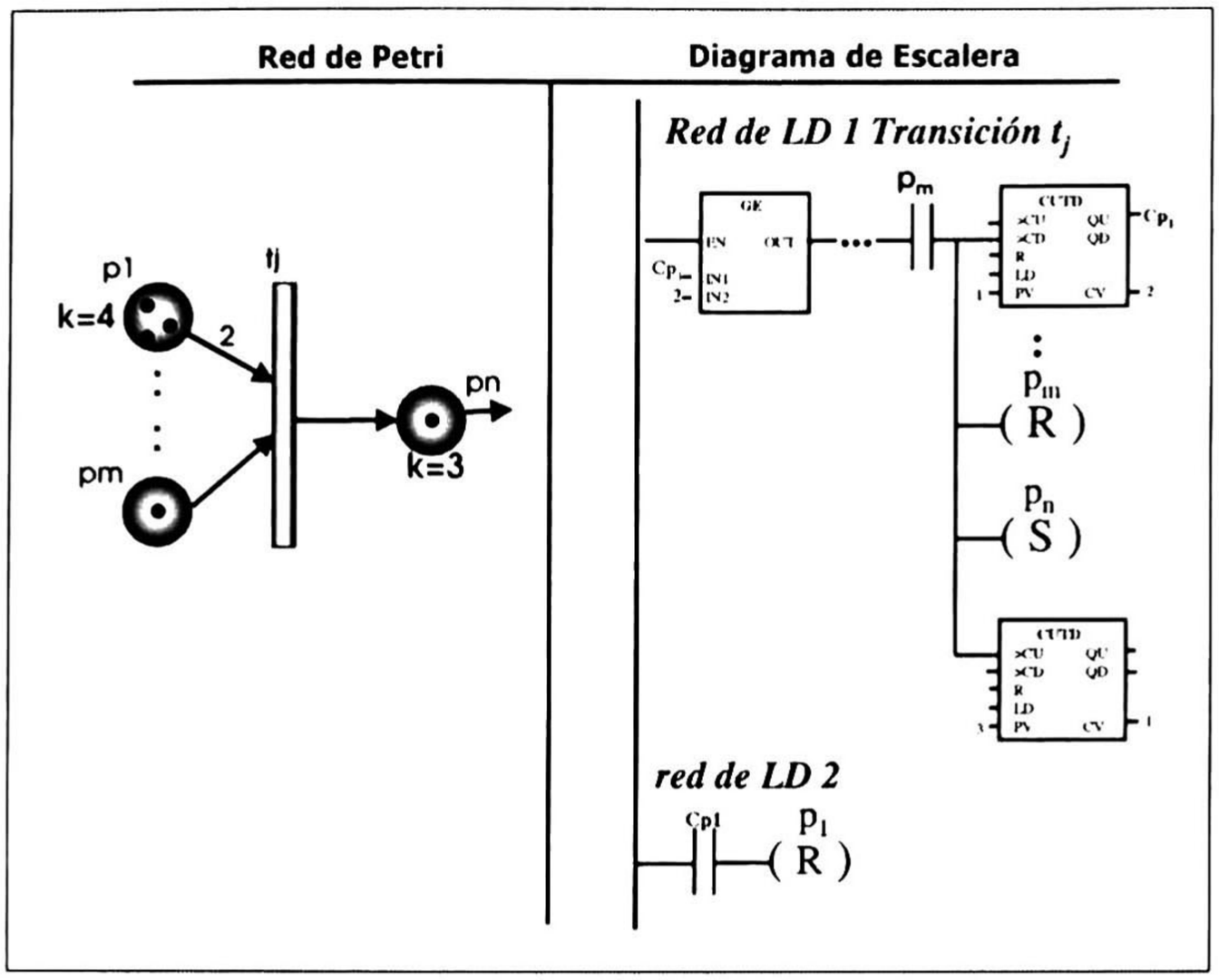


Figura 3.8: Representación en Diagrama de Escalera de una transición con lugares no binarios de entrada y salida.

De acuerdo a lo anterior, el lugar p_i de la figura 3.9 genera 4 redes de **LD** donde cada una tiene un temporizador. Al disparar t_j, se depositan dos marcas en p_i en el instante k y los dos temporizadores correspondientes se desactivarán en el instante $k + d_i$. Como las dos primeras redes de **LD** generan la misma señal en el instante k, entonces convendrá representarlas en una sola red de **LD**. Así, en el instante k, se depositan dos marcas en el lugar p_i y solo un temporizador se desactivará en el instante $k + d_i$ (ver figura 3.10).

La fusión de las redes de **LD** está determinado por el peso del arco de la transición t_j al lugar p_i $(w(p_i, t_j))$.

A continuación enunciaremos el algoritmo para obtener el número óptimo de redes de **LD** que representan a un lugar no binario temporizado de salida de t_j.

Algoritmo 18 ***#RLD_LNBT***$(k_i, w_i; Total\_RLD)$

entradas: k_i = Cota máxima del lugar p_i.
w_i = Peso del arco de la transición tj al lugar p_i.
salidas: $Total\_RLD$ = Número de redes de **LD** que representan a un lugar no binario temporizado.

1. $x = 0; y = 0;$
2. $x = k_i$ div w_i; y=k_i mod w_i.
3. $Total\_RLD = x$.
4. Si $y>0$
   $Total\_RLD = Total\_RLD+1$

Si el lugar No Binario Temporizado p_i se representa con n redes de **LD**, cada una de éstas deben estar compuestas de elementos de **LD** como se indica a continuación.

- En la red de **LD** i donde $i = 1...n$, debe haber $i - 1$ contactos Normalmente Abierto asociados a las variables booleanas de la salida Q de los temporizadores precedentes, un contacto normalmente cerrado a la variable booleana del contador i y los elementos de **LD** para repesentar a los lugares de entrada de t_j como en la sección 3.4.2.

- En las ejecuciones de la red de **LD** i debe haber una bobina inactiva (ed_i). Esta bobina es una bandera auxiliar para detectar que la(s) marca(s) han entrado en estado no disponible. También, deben estar los elementos de **LD** para representar a un lugar no binario como en la sección 3.3.1.

La representación del estado *disponible* de las marcas del lugar *pi* se muestra en la figura 3.11; se hace de la siguiente manera: En otra red de **LD**, las condiciones está compuesto de dos contactos Normalmente Cerrado. El primero indica que el temporizador está desactivado y el segundo señala que las marcas estuvieron en estado no disponible. En las ejecuciones, se cambia el estado de la bandera auxiliar y se incrementa el contador CTUD asociado al lugar p_k.

El orden que se coloca cada red de **LD** i es de n hasta 1 para controlar que los temporizadores de las n redes de **LD** no se activen en un mismo *ciclo de barrido del programa* (ver descripción del barrido del programa en la sección 2.2.1).

El resto de lugares no temporizados de salida de t_j se representan en una red de **LD** adicional como en la sección 3.4.2.

Algoritmo 19 ***LugarTemporizadoNoBinario_LD***$(t_j, p_i, p_j, Dp(p_i), k_{ui}, k_{di}; \mathbf{LD}(t_j))$**.** *Representación en* ***LD*** *de un lugar temporizado no binario.*

entradas: t_j = Transición de la Red de Petri.
p_i= Lugar de entrada de la transición t_j.
p_j= Lugar de salida de la transición t_j.
$Dp(p_j)$= Tiempo asociado a p_j
k_{ui} = Cota máxima del lugar p_i.
k_{di} = Cota mínima del lugar p_i
salidas: $\mathbf{LD}(t_j)$ red de **LD** de la transición con lugares no binarios temporizados de entrada y salida de la Red de Petri.

1.$\forall p_j \in t_j\bullet$ HACER
  (a) Si (k_{uj}>1) y ($d(p_j) > 0$) ENTONCES
    i. x=$\#\mathbf{RLD\_LNBT}(k(\ p_i), w(p_i, t_j))$
    ii.Para i=x to 1 HACER
      (a) Nueva Red
      (b) $\forall\ p_i \in \bullet t_j$ HACER
        i. Si k_{ui}>1 ENTONCES
          (a). Añadir condicional GE con las siguientes características:
            Entrada IN1=contador CTUD asociado a p_i
            Entrada IN2=$w(p_i, t_j)$

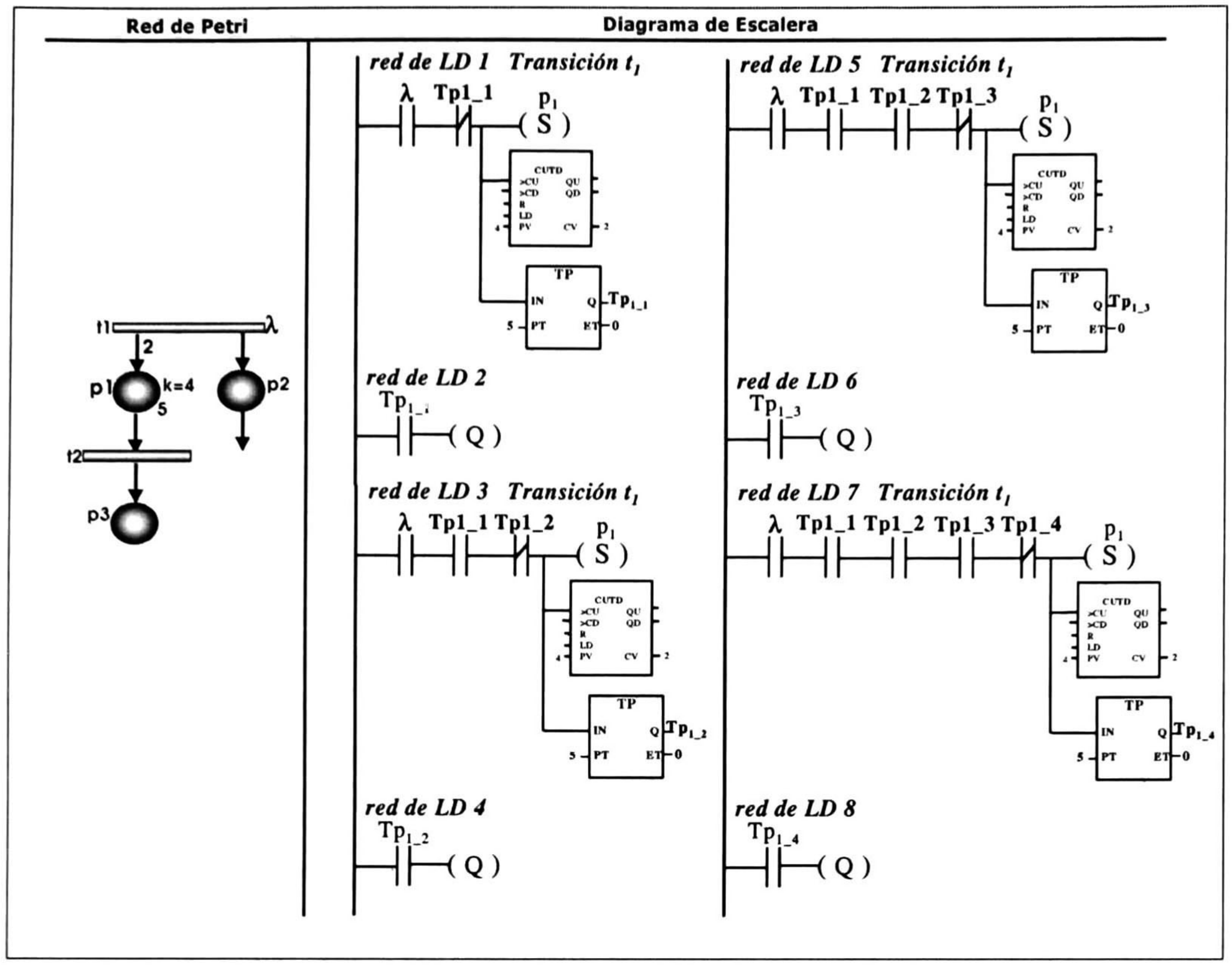


Figura 3.9: Representación en Diagrama de Escalera del lugar p_i. Genera 4 redes de LD; cada temporizador representa una marca de p_i.

sino

(a) Añadir contacto normalmente abierto asociado a la señal de p_i.

(c) Si $\lambda(t_j) \neq \varepsilon$, ENTONCES

i. Añadir contacto normalmente abierto asociado a la señal $\lambda(t_j)$

(d) Para t=1 hasta $i-1$ HACER

i. Agregar contacto normalmente abierto asociado a Tp_{jt}

(e) Agregar contacto normalmente cerrado asociado a Tp_{ji}

(f) Agregar bobina inactiva ed_i

(g) Agregar Temporizador asociado a Tp_j con los siguientes atributos:

Tiempo transcurrido = 0

Tiempo predeterminado = $D_p(p_j)$

-.Salida Q=variable Tp_{ji}

(h) Nueva Red de **LD**

(i) Agregar contacto normalmente abierto asociado a Tp_{ji}

(j) Agregar Salida Constante asociado a la señal de p_j.

(k) Nueva Red de LD

(l) Añadir Conctacto Normalmente Cerrado asociado a Tp_{ji}

(m) Agregar contacto normalmente cerrado asociado a ed_i

(n) Añadir bobina activa END_z

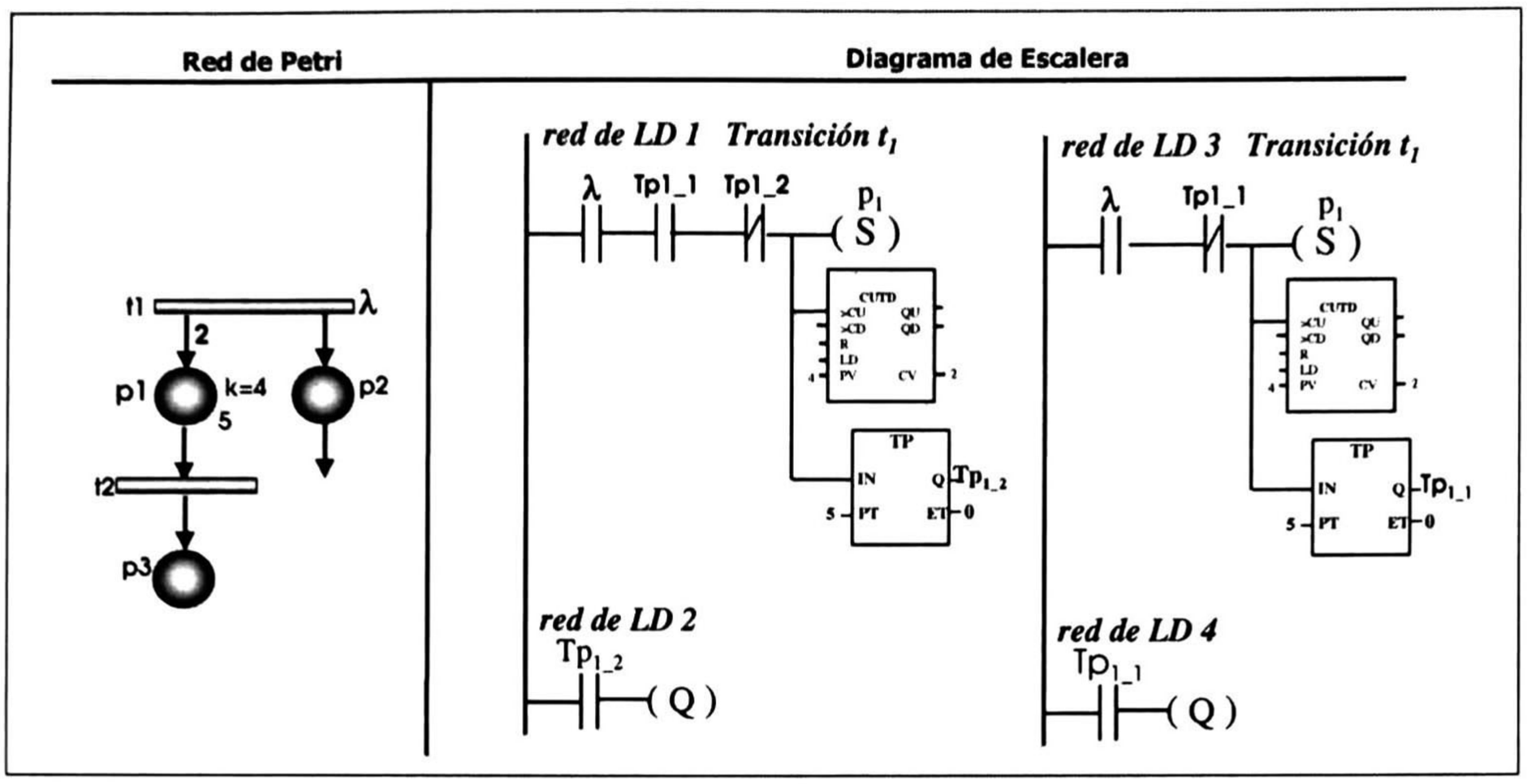


Figura 3.10: Cada red de LD representa a dos marcas del lugar p_1.

(o) Agregar contador CTUD asociado a p_j con las siguientes propiedades:
- entrada de contaje adelante (CU)
  valor actual $= w_1(p_j, t_j)$,
- valor predeterminado $= k_{uj}$.

2. Seguir pasos del algoritmo 17.

3.5 Traducción del Controlador de seguimiento en Diagrama de Escalera.

En esta sección presentamos la representación en Diagrama de Escalera del controlador de un sistema. El diseño del controlador se supone que ha sido diseñado utilizando la metodología del esquema de control por regulación. Esta técnica usa retroalimentación de estados evitando la típica secuencia de retroalimentación del control supervisor; para más detalles del algoritmo, consultar [20].

Si el controlador se representa mediante una **RdP**, pueden aplicarse los algoritmos de las secciones anteriores. Ahora, si el controlador se especifica mediante la función H; cada uno de los paramétros de la función tiene una representación en **LD**. Formalmente la Función **H** está definida como:

$$H : \mathbf{R}(S_f, M_0) \times \mathbf{R}(R_m, \breve{M}_0) \rightarrow \Sigma$$
$$u_l^n := \mathbf{H}(M_l, \breve{M}_n)$$

donde:

$R(S_f, M_0) \times R(R_m, \breve{M}_0)$ *es el conjunto de marcados alcanzados a partir del marcado inicial del modelo de la especificación y del sistema y*

Σ *es el alfabeto de entrada de la* ***RdP***

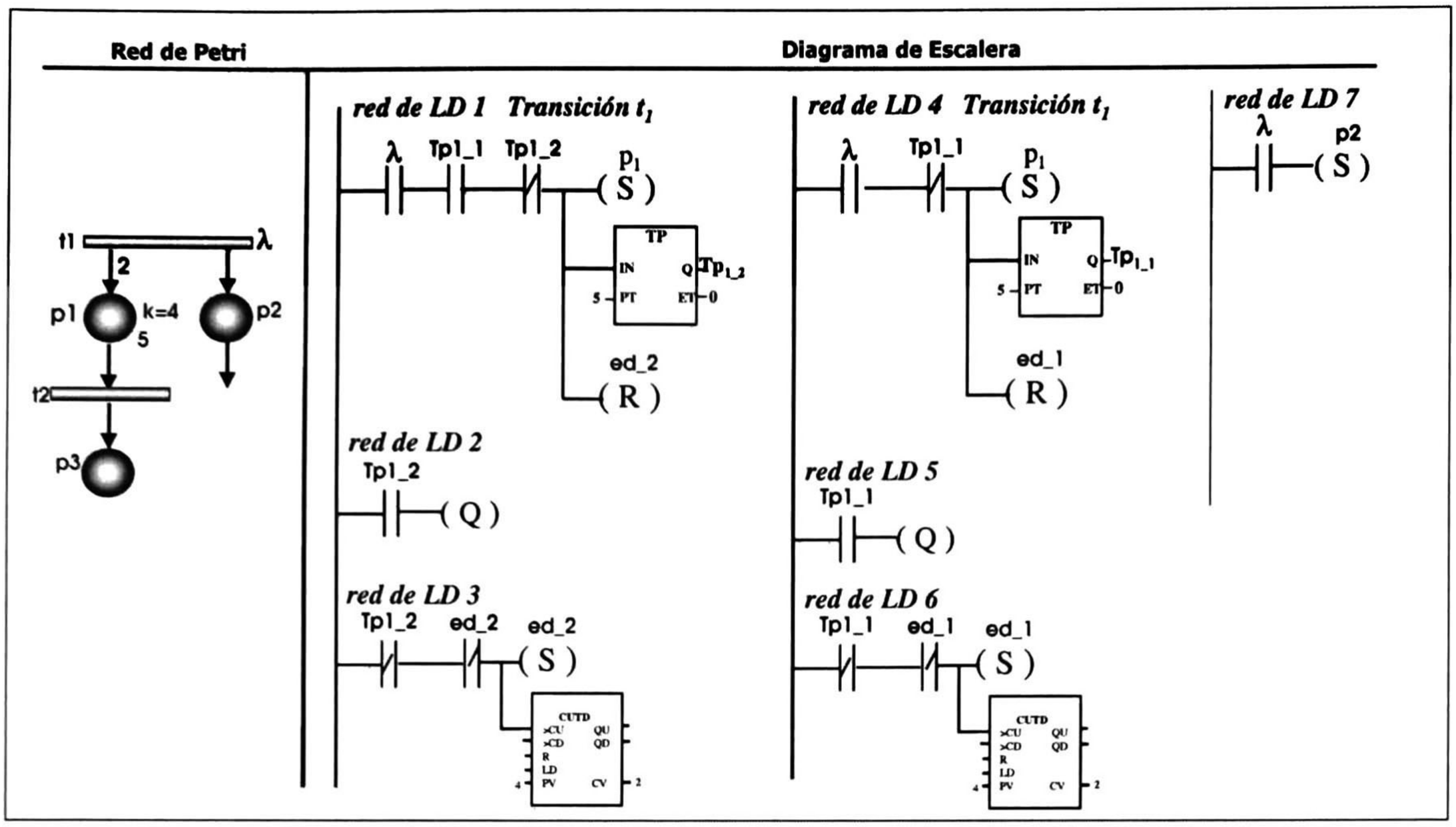


Figura 3.11: Representación en Diagrama de Escalera de un Lugar Temporizado No Binario.

Para cada marcado alcanzado $\mathbf{R}(S_f, M_0) \times \mathbf{R}(R_m, \breve{M}_0)$ se activa un símbolo de entrada σ_i. Dicho marcado se representa con una red de **LD**. Para cada elemento binario de $\mathbf{R}(S_f, M_0) \times \mathbf{R}(R_m, \breve{M}_0)$ debe haber un contacto normalmente abierto o un contacto normalmente cerrado; el símbolo σ_i se representa con una Salida Constante.

Para ejemplificar la representación en **LD** del controlador, tomaremos la figura 3.12. Cada marcado de la **RdP** está compuesto de dos conjuntos: Marcado de la especificación y el Marcado del sistema. En cada contacto que representa el marcado de la especificación, se emplea el operador simbólico mientras que los contactos correspondientes al marcado del sistema se utiliza el operador absoluto de la señal de entrada. Así, su representación en **LD** le corresponde una red de **LD** para cada marcado alcanzado del controlador.

Algoritmo 20 *Representación en LD de la función* **H**.

entradas: $\mathbf{H}(M_l, \breve{M}_n)$: Función H.
salidas: $\mathbf{LD}(\mathbf{H}(M_l, \breve{M}_n))$: red de **LD** de la función **H**.
1.$\forall$ *función* $H(M_l, \breve{M}_n)$ del controlador del sistemaHACER
(a). Nueva Red **LD**
(b) $\forall\ p_i \in P_s$ donde P_s es el conjunto de lugares del modelo del sistema HACER
i. Si $M_a(p_i) = 1$, donde $M_a(p_i)$ es el marcado del lugar p_i ENTONCES
(a) Añadir contacto normalmente abierto asociado al lugar p_i
sino
(a) Añadir contacto normalmente cerrado asociado al lugar p_i
(c) $\forall p_j \in P_e$ donde P_e es el conjunto de lugares del modelo de la Especificación

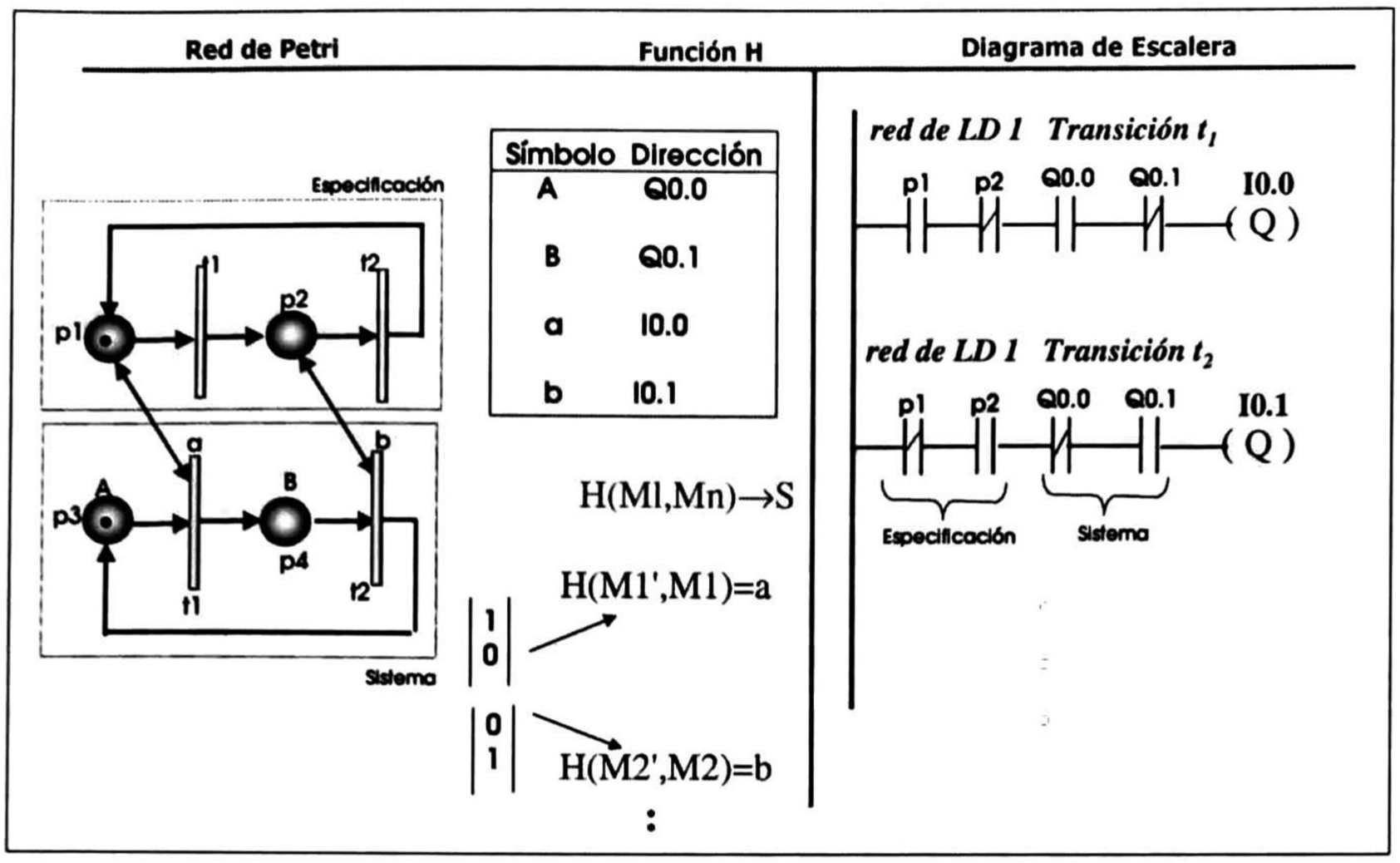


Figura 3.12: Representación en Diagrama de Escalera de la función H. La función H emplea el esquema de control de regulación.

HACER
i. Si $M_a(p_j) = 1$ ENTONCES
(a) Añadir contacto normalmente abierto asociado al lugar p_j
sino
(a) Añadir contactos normalmente cerrado asociado al lugar p_j.
(d) Añadir Salida Constante asociado al símbolo de entrada σ_i

3.6 Ejemplo:

La figura 3.13 muestra un esquema que representa una vía circular donde transitan trenes del metro. La vía ha sido dividida en 6 segmentos (V_i) de igual longitud, donde $i = 1..6$ y dos trenes A y B que circulan en el mismo sentido. El comportamiento de los trenes consiste en que no deben ocupar segmentos contiguos por razones de seguridad.

La **RdP** mostrada en la figura 3.14 tiene la siguiente interpretación.

- TA_i: El tren A se encuentra viajando por el segmento i, donde $i = 1..6$
- TAi: El tren A se encuentra en "el fin del segmento i", donde $i = 1..6$
- TBi: El tren B se encuentra en el tramo i, donde $i = 1..6$
- Vi : Permiso para pasar al tramo i, donde $i = 1..6$.
- $\lambda(tj)$ = Señal externa de las transición tj. Indica la señal del sensor i= 1..6

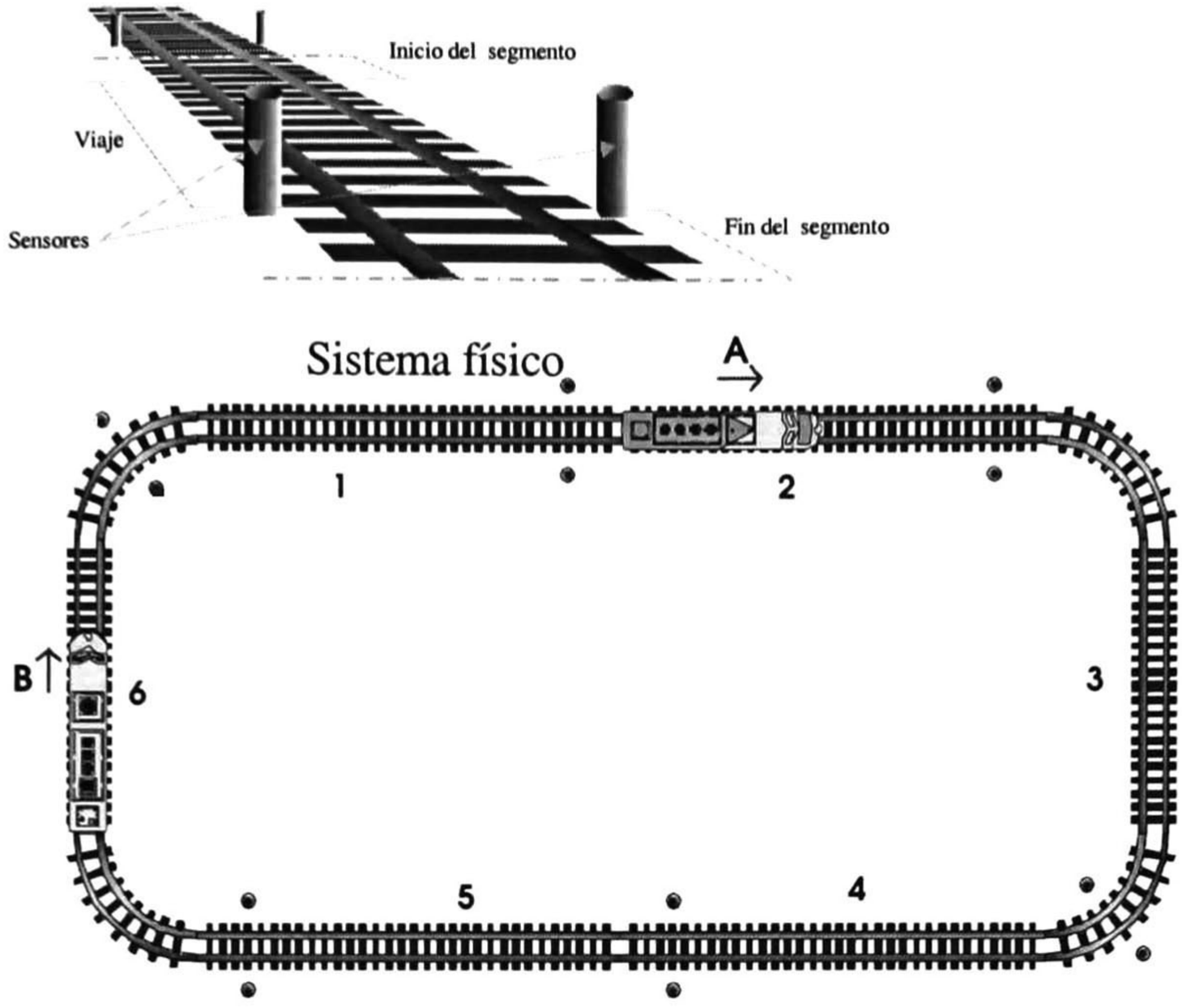


Figura 3.13: Modelo del sistema de los trenes en vía circular de 6 segmentos.

- d(TAi)= Es el tiempo asignado al lugar TAi. Representa el retardo en salir del segmento.

Aplicando la metodología para generar **LD** del modelo de la RP presentado para este ejemplo, obtenemos el **LD** que se muestran en las figuras 3.15 y 3.16. Su interpretación para cada red **LD** es la siguiente:

1. **Red LD 1**. Representa el marcado inicial de la RP. Se aplicó el algoritmo del caso *marcado inicial* de las Redes de Petri Ordinarias.

2. **Red LD 2**. Representa a la transición $t1$. El tren A comienza a viajar por el segmento 1. Al pasar por los sensores, tiene un tiempo específico de 5 unidades de tiempo para recorrer el fin del segmento. Se aplicó el algoritmo del principio de la metodología y la extensión del lugar binario temporizado.

3. **Red LD 3**. Continúa energizado el segmento 1.

4. **Red LD 4**. Representa a la transición $t2$. Si el tren A tiene permiso para pasar al segmento 2, desnergiza el segmento 1, otorga el permiso para que se ocupe el segmento 6 y energiza el segmento 2. Se aplicó el algoritmo del lugar no binario temporizado.

5. **Red LD 5 -7** .Representa el recorrido del tren A en el segmento 2.

6. **Red LD 8 -10** .Representa el recorrido del tren A en el segmento 3.

7. **Red LD 11-13** .Representa el recorrido del tren A en el segmento 4.

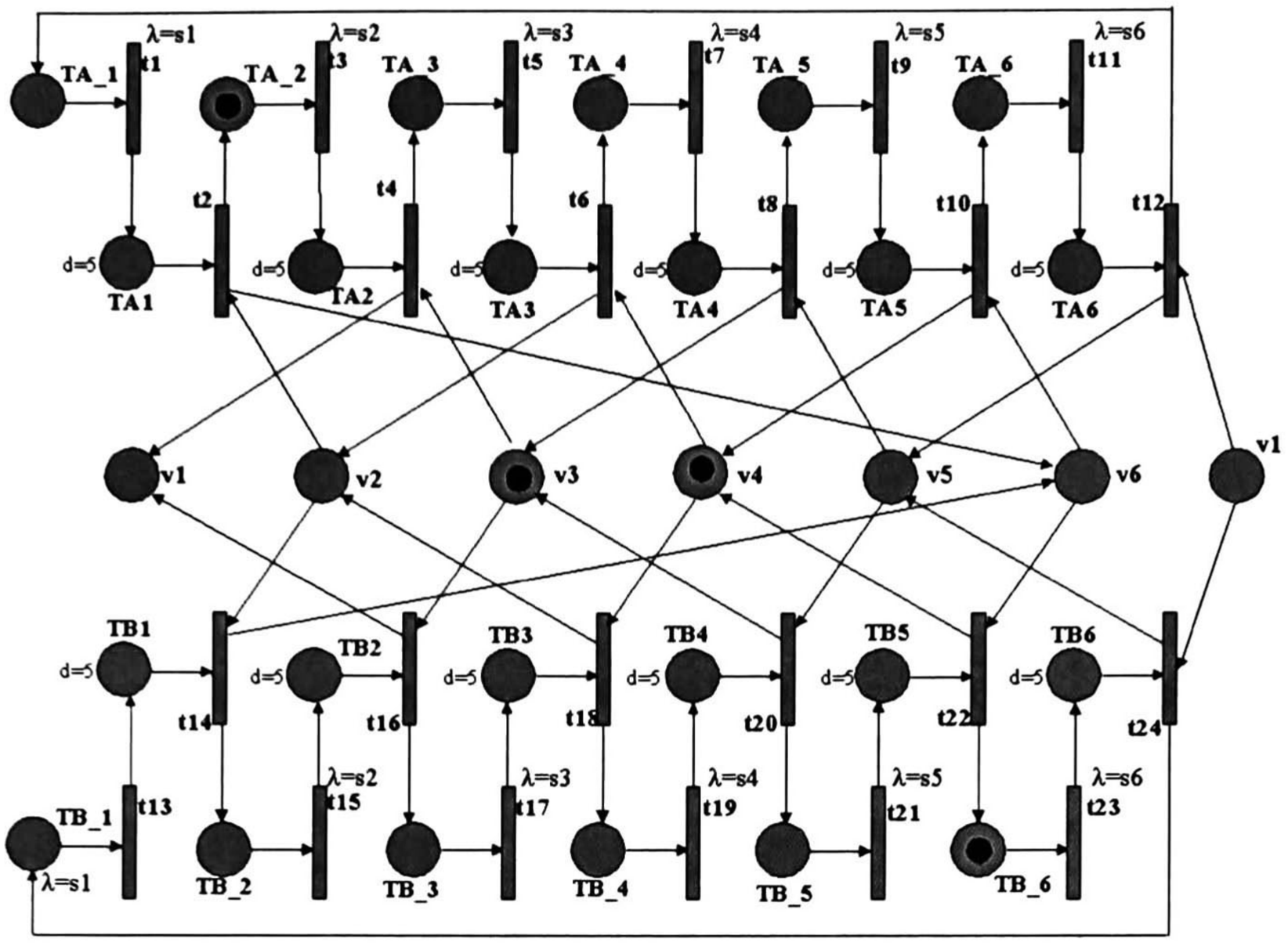


Figura 3.14: Representación en Red de Petri del modelo del sistema de los trenes de la vía circular.

8. **Red LD 14-16** .Representa el recorrido del tren A en el segmento 5.
9. **Red LD 17-19** .Representa el recorrido del tren A en el segmento 6.
10. **Red LD 20 -22** .Representa el recorrido del tren B en el segmento 1.
11. **Red LD 23 -25** .Representa el recorrido del tren B en el segmento 2.
12. **Red LD 26-28** .Representa el recorrido del tren B en el segmento 3.
13. **Red LD 29-31** .Representa el recorrido del tren B en el segmento 4.
14. **Red LD 32-34** .Representa el recorrido del tren B en el segmento 5.
15. **Red LD 35-37** .Representa el recorrido del tren B en el segmento 6.

La siguiente tabla de símbolos contiene las direcciones de los operadores simbólicos del **LD**. Las direcciones enunciadas corresponden a las vias y sensores del **PLC CPU Siemens 214**.

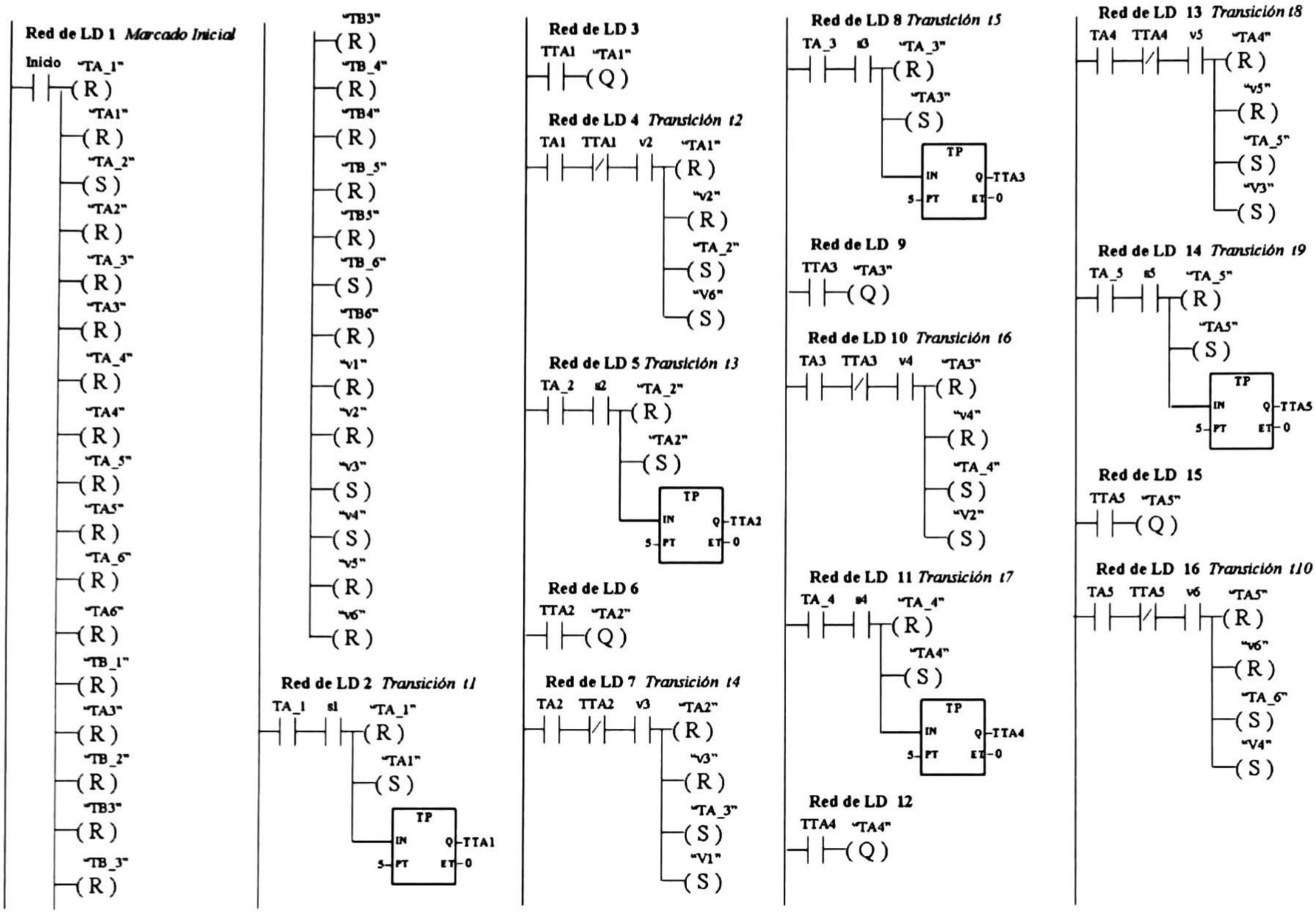


Figura 3.15: Representación en Diagrama de Escalera del modelo en Red de Petri del sistema de los trenes. Parte I.

Símbolo	*Dirección*	*Comentario*
"*TA_1*"	Q0.0	Segmento de vía 1
"*TA1*"	Q0.0	Segmento de vía 1
"*TA_2*"	Q0.1	Segmento de vía 2
"*TA2*"	Q0.1	Segmento de vía 2
"*TA_3*"	Q0.2	Segmento de vía 3
"*TA3*"	Q0.2	Segmento de vía 3
"*TA_4*"	Q0.3	Segmento de vía 4
"*TA4*"	Q0.3	Segmento de vía 4
"*TA_5*"	Q0.4	Segmento de vía 5
"*TA5*"	Q0.4	Segmento de vía 5
"*TA_6*"	Q0.5	Segmento de vía 6
"*TA6*"	Q0.5	Segmento de vía 6
"*TB_1*"	Q0.0	Segmento de vía 1
"*TB1*"	Q0.0	Segmento de vía 1
"*TB_2*"	Q0.1	Segmento de vía 2
"*TB2*"	Q0.1	Segmento de vía 2
"*TB_3*"	Q0.2	Segmento de vía 3
"*TB3*"	Q0.2	Segmento de vía 3
"*TB_4*"	Q0.3	Segmento de vía 4
"*TB4*"	Q0.3	Segmento de vía 4
"*TB_5*"	Q0.4	Segmento de vía 5
"*TB5*"	Q0.4	Segmento de vía 5
"*TB_6*"	Q0.5	Segmento de vía 6
"*TB6*"	Q0.5	Segmento de vía 6
"*S6*"	I2.0	Sensor de la vía 6
"*S1*"	I2.1	Sensor de la vía 1
"*S2*"	I2.2	Sensor de la vía 2
"*S3*"	I2.3	Sensor de la vía 3
"*S4*"	I2.4	Sensor de la vía 4
"*S5*"	I2.5	Sensor de la vía 5
"*Inicio*"	I0.0	Señal de inicio de ejecución del sistema

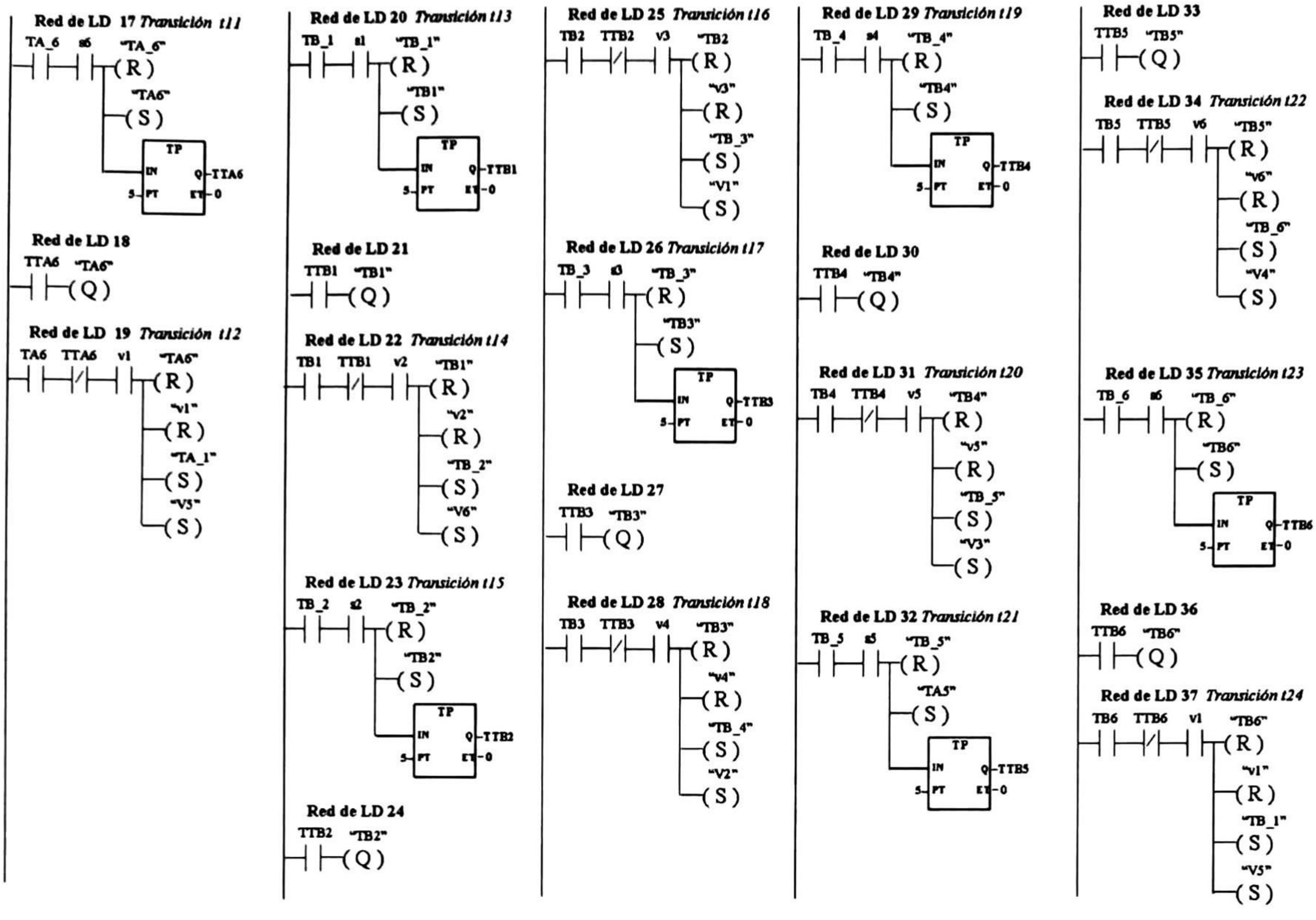


Figura 3.16: Representación en Diagrama de Escalera del modelo del sistema de los trenes. Parte II.

3.7 Conclusiones.

La metodología que se presentó en éste capítulo para convertir una **RdPIT** a **LD** se describió diviéndola en los casos de las Redes de Petri Temporizadas y las Redes de Petri No Binarias.

La conversión de una **RdP** a **LD** es compacta ya que la cantidad de elementos que se generan están polinomialmente acotado por el número de lugares y transiciones de la **RdP** que se ha de convertir a **LD**.

Se propone la conversión de una Red de Petri con Transiciones Temporizadas a una Red de Petri con Lugares Temporizados para evitar la reentrancia en **LD** del disparo de una transición temporizada.

Se describe un algoritmo para obtener el número de redes de **LD** para representar un lugar no binario temporizado.

Además de que las **RdP** pueden modelar los Sistemas de Eventos Discretos, se puede aplicar la teoría de control supervisor y el controlador resultante puede ser implementado en un **PLC** comercial.

La finalidad de esta metodología es, implementarla como una herramienta la cual, un ingeniero pueda diseñar y analizar estrategias de control usando **RdPIT** y así convertirla a **LD** para usar un **PLC** como controlador.

Capítulo 4

Generación automática de Diagramas de Escalera

Resumen. Este capítulo presenta de forma general las tareas de cada módulo que integra el sistema *SpADES*. Se describe en detalle el funcionamiento del módulo *LDGen* así como su interfaz gráfica para generar Diagrama de Escalera.

Actualmente existen sistemas que se encargan de la edición y manipulación de las **RdPI**, algunos de estos sistemas cuentan con un buen editor de **RdP** pero carecen de otros servicios que normalmente los usuarios necesitan realizar de manera automática. Por ello se ha creado *SpADES*, que ofrece como ventaja conjuntar las operaciones de las **RdP** tales como la edición, simulación, emulación, análisis y traducción a Diagrama de Escalera (**LD** Ladder Diagram).

4.1 El Sistema *SpADES*

El sistema *SpADES* (Specification and Analysis of Discret Event Systems) es una herramienta visual para el modelado y análisis de los Sistemas de Eventos Discretos (**DES**) usando las Redes de Petri Interpretadas Temporizadas (RdPIT), en su mayor parte, *SpADES* ha sido desarrollado por Claudia Pulido [4]. El sistema lo integran los módulos de *Edición de RdPIT, Simulación de RdPIT, Emulación de RdPIT, Análisis de RdPIT y la Conversión de RdPIT a **LD***, todos ellos desarrollados en el Lenguaje de Programación Orientado a Objetos C++ Builder bajo el ambiente Windows ^{TM}

A continuación se describe de una forma muy general los módulos mencionados.

- **Edición de RdPIT**. El sistema permite al usuario la creación y modificación de una **RdPI** para posteriormente poder trabajar con ellas en cualquiera de las otras funciones del sistema.

- **Simulación de RdPIT**. La simulación se encarga de mostrar gráficamente al usuario la evolución del marcado de una **RdPI** editada anteriormente.

- **Emulación de RdPIT**. Es el procedimiento del sistema en que se visualiza la evolución de la **RdPI** enviando señales a un dispositivo externo al sistema mediante el puerto serial RS232 para manipularlo.

- **Análisis de RdPIT**. Se divide en los siguientes algoritmos:

Algoritmos de Diseño. Controladores, Observadores e Identificadores.
Algortimos de análisis. Los cuales lo integran el cálculo de: P_Semiflujos, T_Semiflujos, Trampas, Cerrojos, Lugares Implicitos, Distancia Sincrónica, Acotabilidad, Vivacidad, Ciclicidad, Clases de **RdPI**, Observabilidad, Controlabilidad, Evento Detectabilidad, Grafo de alcanzabilidad, Componentes Fuertemente Conexas y Homespace.
Algoritmos para el análisis de Desempeño. Tiempo de Ciclo, Marcado Inicial Mínimo y Scheduling.

- **Conversión de RdPIT a LD**. Este es el módulo donde se genera un **LD** de acuerdo a las especificaciones de una **RdPI**, obtenida en el módulo de "Edición de **RdPI**"[11].

Dejando un poco el sistema *SpADES* de forma general, nos enfocaremos a detallar los módulos que concierne a esta tesis: Módulo de Edición de Redes de Petri y el Módulo *LDGen*.

4.2 Módulo de Edición de Redes de Petri.

SpADES permite la edición de RdP, cuenta con las opciones necesarias para dibujar lugares, transiciones y arcos. Como se visualiza en la figura 4.1, para dibujarlos sólo hay que seleccionar un botón de la barra de herramientas de edición y enseguida pulsar el botón izquierdo del ratón en la ventana de edición.

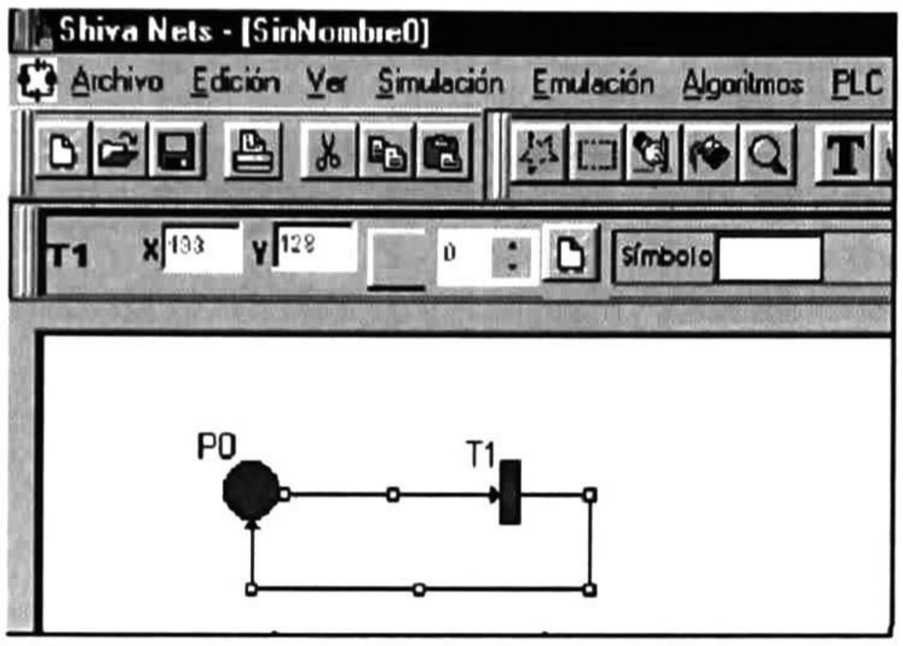


Figura 4.1: Interfaz del módulo de Edición de Redes de Petri.

Una vez editada la red, se permite modificar las propiedades que se muestran en la barra de información de los lugares; por ejemplo, el marcado, asignar una descripción, cambiar el símbolo de entrada, etc. Todos estos cambios se logran pulsando el botón izquierdo del ratón y acceder a la casilla correspondiente de la barra de información de un lugar.

Las transiciones también poseen propiedades, las cuales pueden ser alteradas mediante su barra de información. Para acceder a esta barra se procede igual que con los lugares.

Los arcos, al igual que los lugares y las transiciones pueden ser modificados mediante su barra de herramientas, donde permite cambiar el tipo de arco (bidireccional, cambiar el sentido o inhibidor) y el peso del arco.

Cabe señal que también cuenta con las operaciones de manejos de archivos e impresión. Una caractarística más que ofrece, es que permite la importación de archivos bajo el formato del Petri Maker, una herramienta para la edición y análisis de las RdP.

La figura 4.2, muestra el diagrama de flujo para generar **LD**. Como se puede ver, el primer paso y el dato de entrada a la metodología del capítulo 3 es una RdP; así, una vez editada la red de Petri se procede a generar **LD** mediante la elección de la opción **LD** de la barra de herramientas de la interfaz principal de *SpADES*.

4.3 Módulo *LDGen*: Conversión de Red de Petri a Diagrama de Escalera.

LDGen es el módulo que automatiza la metodología presentada en el capítulo 3, para nuestro caso en particular se realizaron pruebas utilizando el **PLC Siemens CPU 214.** Como se visualiza en la figura 4.3, el módulo *LDGen* es mostrado en una sola interfaz del usuario[12], en donde se integran básicamente los siguientes componentes.

- *Barra de Menú:* Permite ejecutar funciones utilizando el ratón ó combinación de teclas.

Figura 4.2: Esquema de la secuencia para generar LD.

- *Barra de Herramientas:* Permite acceder fácilmente con el ratón a las funciones del módulo *LDGen*.

- *Area de Trabajo:* Es el área donde contiene la tabla de símbolos y la vista del programa correspodiente al editor de **LD**.

- *Tabla de Símbolos:* Es una tabla que muestra las asignaciones de señales a los lugares y a las transiciones hechas en el momento de editar la correspondiente **RdP** (es decir, los operadores simbólicos utilizados para representar los Contactos Abiertos y Cerrados). Mediante esta tabla, el usuario puede asignar y/o modificar, las señales asociadas a cada uno de los operadores simbólicos así como su descripción que representan a los lugares del modelo en **RdP**

- *Editor de **LD**:* Es el área principal del módulo *LDGen*, donde se despliega cada objeto en **LD** que representa a la **RdP** que se editó en la interfaz de "Edición de **RdP**".

4.3.1 Funciones comunes del módulo *LDGen*.

El módulo *LDGen* está integrado básicamente de las siguientes funciones:

- **Edición:** Guardar, abrir e imprimir
- **Generación de LD.**

A continuación se describen brevemente cada una ellas.

- **Guardar.** Es la opción que se proporciona al usuario del sistema *SpADES* para que almacene en disco el **LD** mostrado en el área de trabajo.

- **Abrir.** Es la función que permite al usuario el acceso de **LD** de un archivo en disco para mostrarlo en el área principal del módulo *LDGen*.

- **Imprimir.** Genera un reporte impreso tanto del **LD** como toda la información relacionada a cada una de los elementos de **LD** y de los datos de la tabla de símbolos.

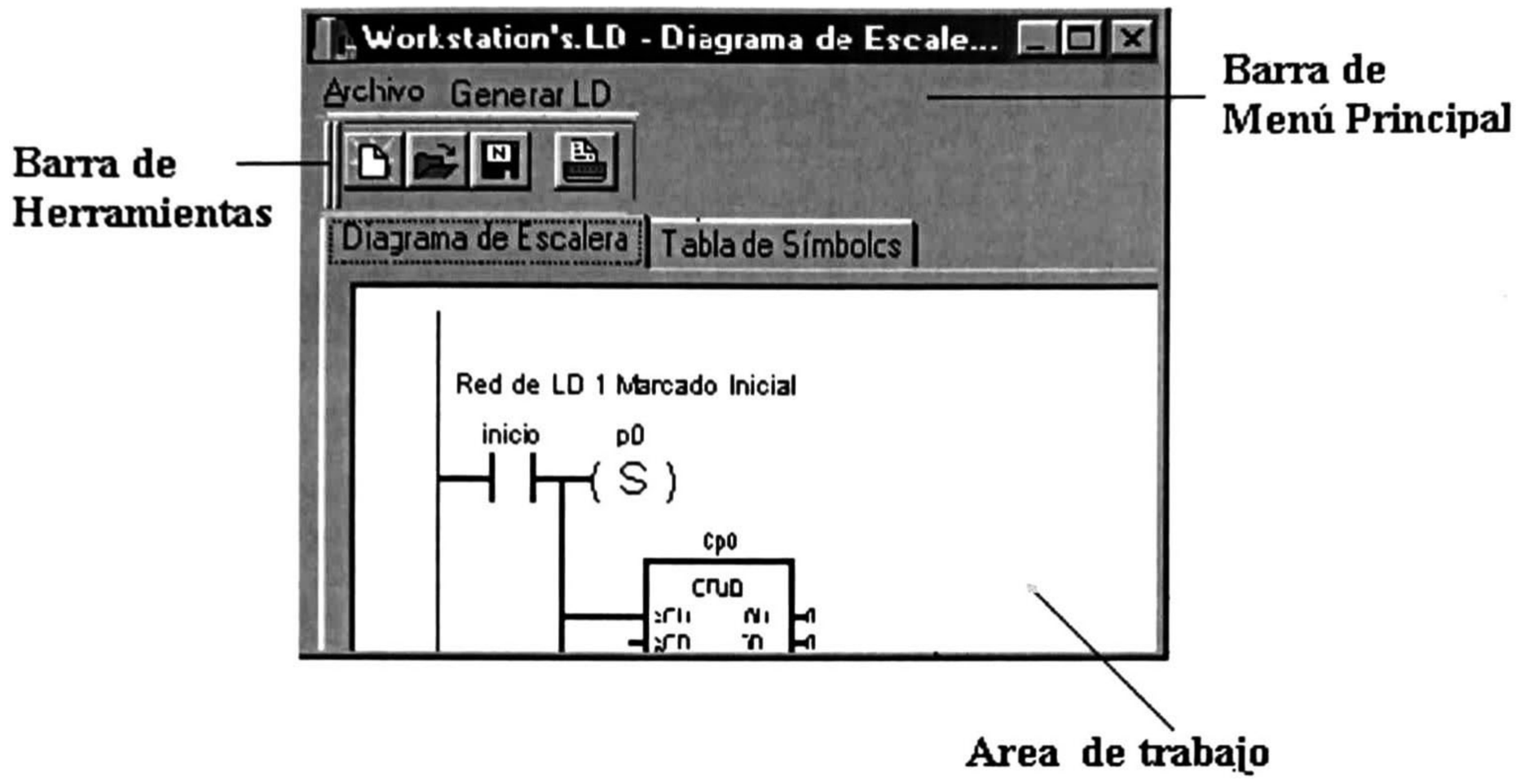


Figura 4.3: Interfaz de Usuario del módulo **LDGen**

- **Generar LD**. Es la opción donde se automatiza la traducción a **LD** de acuerdo a la **RdP** de la ventana activa del módulo Edición de **RdP**.

En la tabla de símbolos, se habilitan aquellos campos que el usuario puede modificar, como la señal asociada (dirección) y su descripción. De acuerdo al número de lugares y transiciones que componen a la **RdP** es el número de contactos y condiciones externas que se muestran en la tabla de símbolos.

4.4 Procedimiento para convertir una Red de Petri a Diagrama de Escalera.

Para convertir un modelo de **RdP** a **LD** es necesario efectuar una serie de pasos, los cuales se describen a continuación

Paso 1 **Editar una RdP.** En la interfaz del módulo de Edición de **RdP** el usuario edita una **RdP**. Puede auxiliarse de los íconos *lugar, transición y arcos* que se ubican en la barra de herramientas de edición. Para asignar atributos a una transición o a un lugar, como el tiempo, marcas, símbolo externo, señal de entrada/salida, descripción, etc., se pulsa Click en el botón izquierdo del ratón sobre el objeto que se desea asignar tales atributos.

Para obtener **LD** del modelo en **RdP** de un controlador, lo que tiene que hacer es lo siguiente:

1.1 Construir un modelo en **RdP** de la especificación como del comportamiento del sistema.

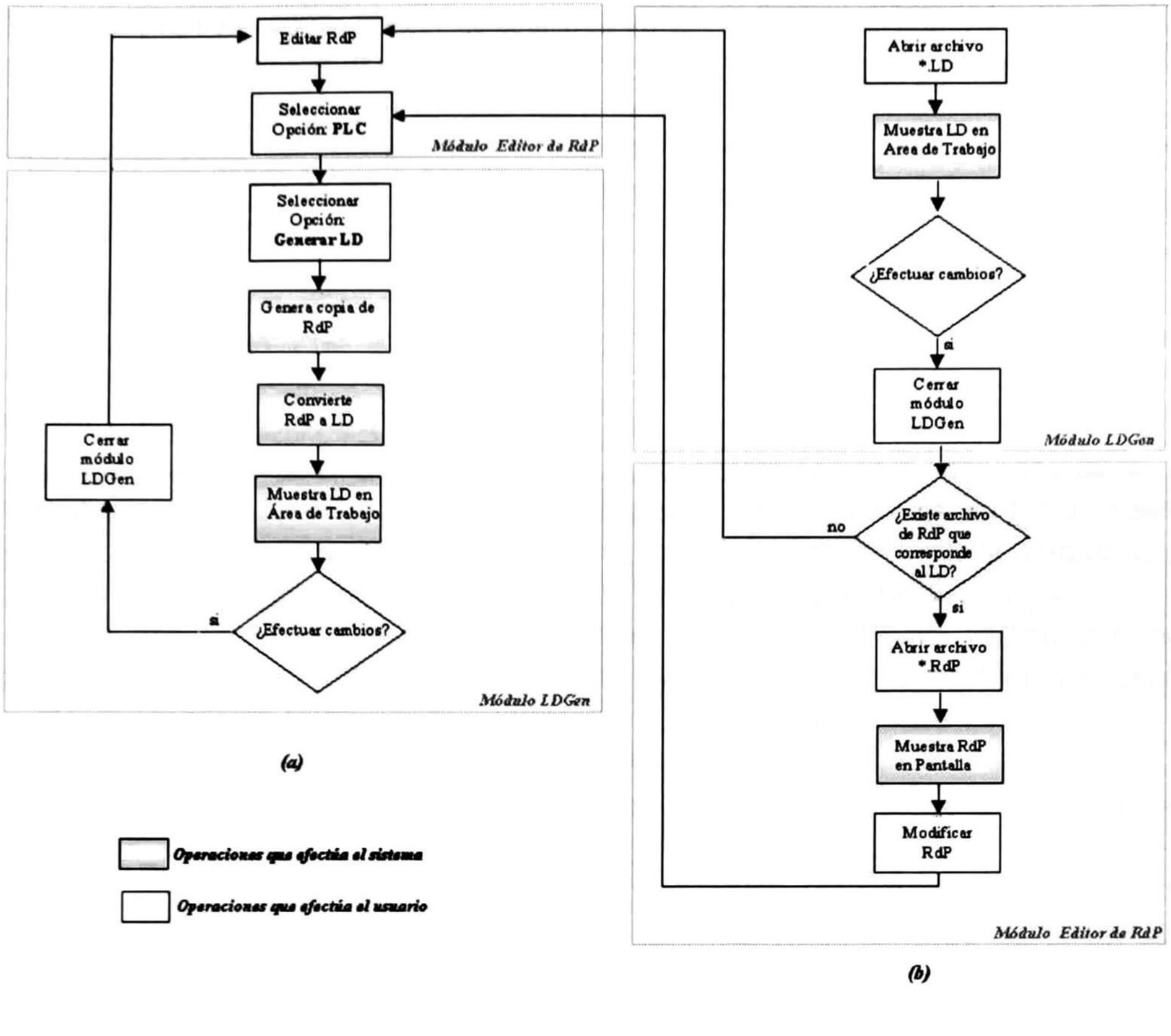


Figura 4.4: Procedimiento general para obtener Diagrama de Escalera.

1.2 Seleccionar la opción *Algoritmo* y elegir la subopción: *Controlador*.

1.3 En una tercera interfaz se muestra el modelo en **RdP** del controlador.

1.4 Ejecutar paso 2 y 3.

Paso 2 **Selecionar opción: PLC**. Esta opción se ubicada en la barra del menú principal del sistema *SpADES*. Ésta presenta la interfaz del módulo *LDGen*.

Paso 3 **Seleccionar opción: Generar LD**. Es la opción del menú principal del módulo *LDGen* que invoca todos los métodos necesarios para presentar el **LD** correspondiente a la **RdP** de la ventana activa.

El diagrama de flujo de la figura 4.4-a, muestra el procedimiento para obtener **LD** de una **RdP.** Existe un punto importante que no se había mencionado; el acceso denegado al usuario para modificar el **LD**. Todo cambio en los requerimientos iniciales o de funcionalidad del sistema, solo podrá realizarse en el modelo de la RdP, así, se verán reflejados en el **LD** al momento de volver a generar **LD**.

Se remarca que al generar **LD** de un modelo en **RdP**, se maneja tal conversión de una forma independiente del modelo en **RdP**. En otras palabras, en la función de Abrir un archivo

con extensión **ld,** el **LD** que se abre y se muestra en el Area de Trabajo no tiene vínculo con el archivo correspondiente a la **RdP** (si es que el usuario guardó en disco el modelo de **RdP**) ni genera una **RdP** equivalente al **LD**. El usuario si desea hacer modificaciones tendrá que tener el modelo de **RdP** en la interfaz de la edición de una **RdP,** ya sea que lo haya vuelto a editar o lo haya abierto de un archivo y así generar de nuevo un **LD** que contendrá las modificaciones deseadas, ver figura 4.4-b.

4.5 Desarrollo de LDGen

La programación de *LDGen* utilizó FUSION como método de desarrollo. FUSION es un método orientado a objetos llamado de segunda generación, el cual cubre las fases de desarrollo tanto del análisis como diseño e implementación. Cada una de las fases preserva la estructura en objetos del sistema y permite un seguimiento directo desde la definición de los requerimientos hasta la implementación en algún lenguaje de programación [5].

El desarrollo del sistema *SpADES*, en particular el módulo *LDGen*, utilizó este método de programación Orientado a Objetos porque en cada fase trata un problema diferente, y tiene sus propios modelos conceptuales y notaciones para tales modelos.

De acuerdo a FUSION, el software fué obtenido a través de 3 frases:

Análisis: El análisis produce la especificación de *qué* es lo que debe hacer el módulo **LDGen.**

Diseño: El diseño define *cómo* hacer la conversión de una RdP a Diagrama de Escalera.

Implementación. La implementación codifica el diseño en el lenguaje de programación **C++Builder**.

FUSION no cuenta con la fase de la captura de los requerimientos, por tal motivo se utilizó el método de Especificación de Requerimientos (**SRS:** System Requeriment Specification) del estándar IEEE 830.

Las etapas del desarrollo del módulo **LDGen** son las siguientes:.

4.5.1 Requerimientos.

De acuerdo el estándar IEEE 830, la especificación de los requerimientos tiene la finalidad de proporcionar un panorama general y objetivos del módulo *LDGen* describiendo el contexto de ejecución del mismo para dar una introducción del entendimiento del mismo. Las partes que comprenden el **SRS** son:

Introducción

- *Propósito:* El módulo **LDGen** tiene como propósito implementar una metodología para automatizar la conversión de una Red de Petri a Diagrama de Escalera.
- *Ámbito.* Este sistema sólo implementará funciones básicas para generar Diagrama de Escalera a patir de una Red de Petri usando el estándar definido por el **IEC 1131-3**. Para el módulo *LDGen* se utilizarán las siguientes herramientas:

1. Compilador de C++ Builder

2. Paradigm Plus Enterprise Edition: Herramienta para la implementación del modelado de los requerimientos, análisis y diseño de sistemas.

Las aplicaciones y beneficios que tendrá la implementación de este sistema serán:

1. Interfaz de usuario amigables.
2. Transparencia en el manejo de la información desde el punto de vista del usuario cotidiano.
3. Control de archivos (abrir, guardar, imprimir y modificar la tabla de símbolos).

Descripción General.

Este producto requiere el uso de la POO por esta razón, se programará en C++ Builder versión 4 ya que este lenguaje proporciona todas las ventajas de los lenguajes orientados a objetos.

- *Interfaces del Software.* La máquina en la cual se ejecutará el módulo **LD**Gen será una PC con requerimientos básicos: Procesador Pentium I o superior y 16 Mbytes de RAM o superior. La máquina que ejecute el sistema debe tener Windows 95 o versiones más recientes ya que éste proporciona un mejor manejo de gráficos.

- *Funciones del producto:* En términos generales estos son los servicios que el módulo **LD**Gen debe realizar:

1. Generar **LD**.
2. Abrir **LD**
3. Guardar **LD**
4. Imprimir **LD**
5. Modificar la tabla de símbolos.

- *Representación del flujo de información:* El módulo *LDGen* se encargará de generar **LD** de una Red de Petri. El usuario deberá editar la RdP en la interfaz del módulo de Edición de RdP y **LD**Gen presentará en pantalla el equivalente en **LD**.

- *Características del usuario.* Los usuarios del módulo **LD**Gen deben tener conocimientos sobre el modelado de sistemas en Redes de Petri y el lenguaje de programación para **PLC**: **LD**.

- *Limitaciones Generales.*

1. La velocidad de la conversión de la RdP a **LD** está limitada por el número de lugares y transiciones así como los atributos de dichos nodos ya que son los que determinan el número de elementos de **LD** que los representan.

2. No permite la modificación del **LD** presentado en la pantalla ya que los cambios se efectúan sobre el modelo de RdP equivalente.

3. La implementación directa en el PLC CPU 214 de Siemens del **LD** generado no se contempla ya que no se cuenta con el protocolo de comunicación entre el CPU y el Software de programación.

Estudios de Viabildad del Sistema

- *Viabilidad Económica.* En este caso, tratándose de un trabajo de tesis el financiamiento proviene del CONACYT (beca) y CINVESTAV.

- *Viabilidad Técnica.* Los recursos técnicos que se utilizarán en el desarrollo del sistema; tales como libros, artículos, manuales, software, equipo de cómputo, etc., son proporcionados por el CINVESTAV-IPN Unidad Guadalajara.

Especificaciones de requerimientos funcionales del módulo *LDGen*

- *Generar LD.*

Esta función permite obtener **LD** de la Red de Petri de la ventana activa de la interfaz del editor de Redes de Petri.

Entradas.

El dato de entrada que debe ingresar el usuario para generar **LD** es tener una RdP editada. Posteriormente seleccionar la opción PLC.

Proceso.

El módulo **LD**Gen debe verificar los siguientes aspectos:

La existencia de por lo menos un lugar y una transición.

El tiempo asignado a una transición para modificar la estructura de la RdP.

El tiempo correspondiente a un lugar no binario para invocar el método que determina el número de redes de **LD** que lo han de representar.

Efetuar una copia de la RdP.

Salidas

Una vez procesada la entrada, genera un **LD** que es mostrado en pantalla. Se presenta los datos correspondientes a los contactos normalmente abierto y cerrados de la tabla de símbolos.

- *Abrir archivo de Diagrama de Escalera.*

Entradas

El usuario tiene que introduce o seleccionar el nombre y ruta del archivo que contiene **LD**.

Proceso

Verifica que el archivo especificado tenga la extensión **LD** y que el nombre y ruta sean válidos.

Salida

El **LD** del archivo que se abrió es presentado en pantalla junto con la información correspondiente en la tabla de símbolos.

- *Guardar **LD** en archivo.*

Entradas

El usuario deberá tener en pantalla un **LD** y especificar el nombre y ruta del archivo en que se ha de guardar.

Proceso

El módulo **LDGen** verifica que la ruta especificada sea válida , que el nombre del archivo no esté duplicado y que la etiqueta sea correcta.

Salidas

Genera un archivo con **LD** y la información desglosada de la tabla de símbolos.

- *Imprimir **LD***

Entradas

El usuario debe especificar la impresora a utilizar y deberá tener en pantalla **LD**.

Proceso

Valida que la impresora seleccionada se encuentre activa.

Salida

Genera la impresión de **LD** de la pantalla así como la información de la tabla de símbolos.

- *Modificar Tabla de símbolos*

Entradas

El usuario debe elegir la celda a modificar e introducir los datos a sustituir.

Proceso

El módulo **LDGen** válida los datos que introduce el usuario en aquellos que tenga acceso.

Salidas.

Actualiza la tabla de símbolos en base a los cambios efectuados.

4.5.2 Análisis.

El método FUSION tiene como primera fase el análisis. La meta es, dado los requerimientos, capturarlos de forma consistente, completa y no ambigua. FUSION produce dos modelos: *modelo de objetos* y el *modelo del interfaz.* Ambos modelos describen: las clases de objetos existentes en el sistema, las relaciones entre estas clases, las operaciones que pueden ser realizadas en el sistema y la secuencia permisible de estas operaciones.

En este sección, solo se presenta el modelo de objetos, ya que define la estructura estática de la información del sistema; éste es mostrado en la figura 4.5.

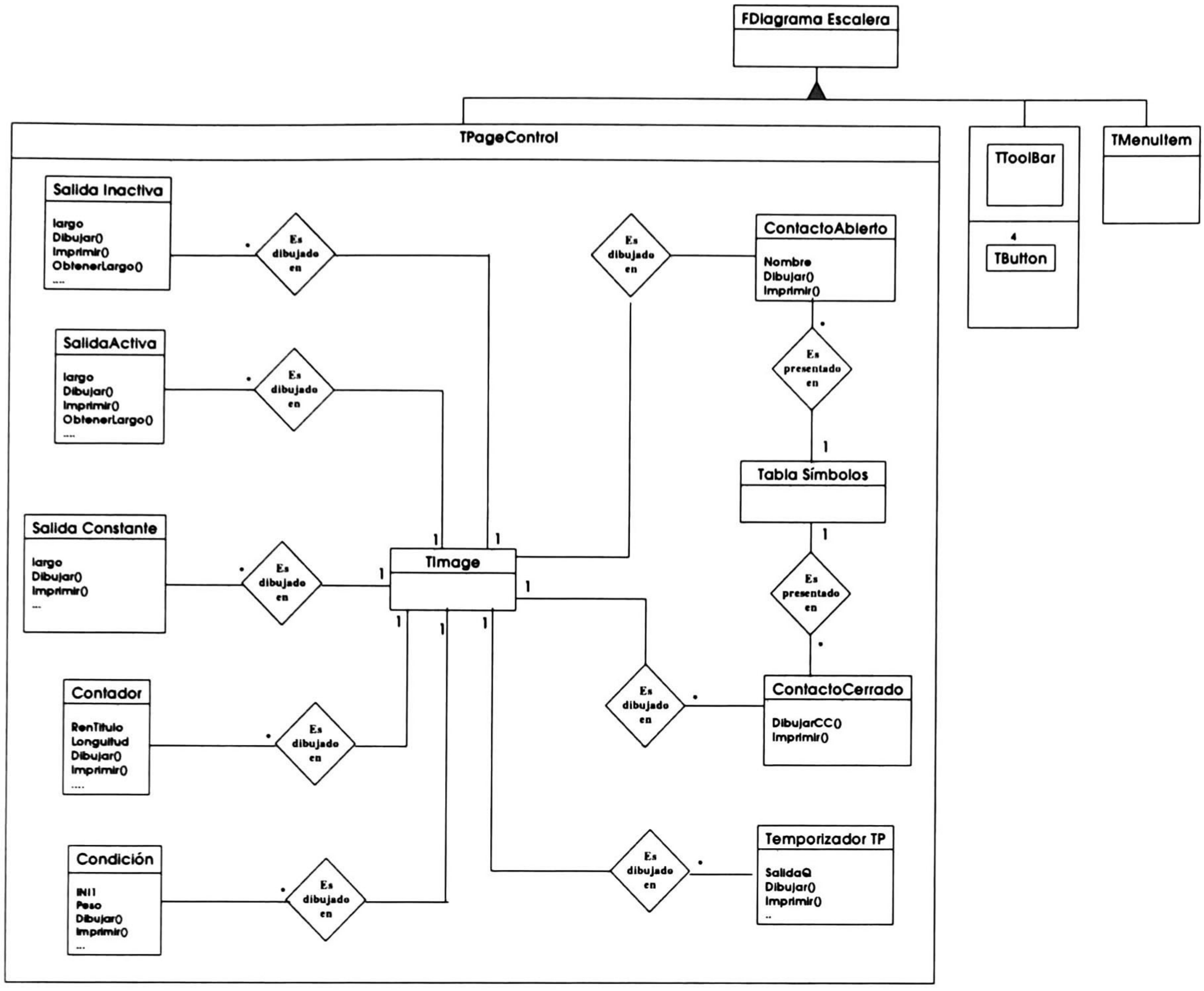


Figura 4.5: Diagrama de Objetos del módulo LDGen.

4.5.3 Diseño.

La estructura del software se introduce para satisfacer las definiciones abstractas producidas por el análisis. Durante la etapa de diseño, los modelos que se obtienen muestran cómo las operaciones del sistema son implementadas, cómo las clases se refieren una a otra, cómo se relacionan mediante la herencia atributos y operaciones de las clases. Dichos modelos son:

- *Grafos de interacción.* Describen cómo los objetos interactúan en tiempo de ejecución.
- *Grafos de visibilidad.* Describen las rutas de comunicación entre objetos.
- *Diagramas de clases.* Proveen una especificación de las fases, atributos y métodos.
- *Grafos de herencia-* Describen las estructuras de herencia entre clases/subclases.

Las clases TMenuItem, TToolBar y TButton son proporcionados por C++ Builder y son gestionados por éste. Por tal motivo, forman parte de la aplicación FDiagramaEscalera. La aplicación FDiagramaEscalera a su vez hereda propiedades y atributos al área de trabajo

TPageControl en el cuál se han de dibujar los elementos de **LD** que representen a un RdP. La herencia y atributo de las clases diseñadas para la conversión de una RdP a **LD** se presenta mediante el diagrama de clases de la figura 4.6.

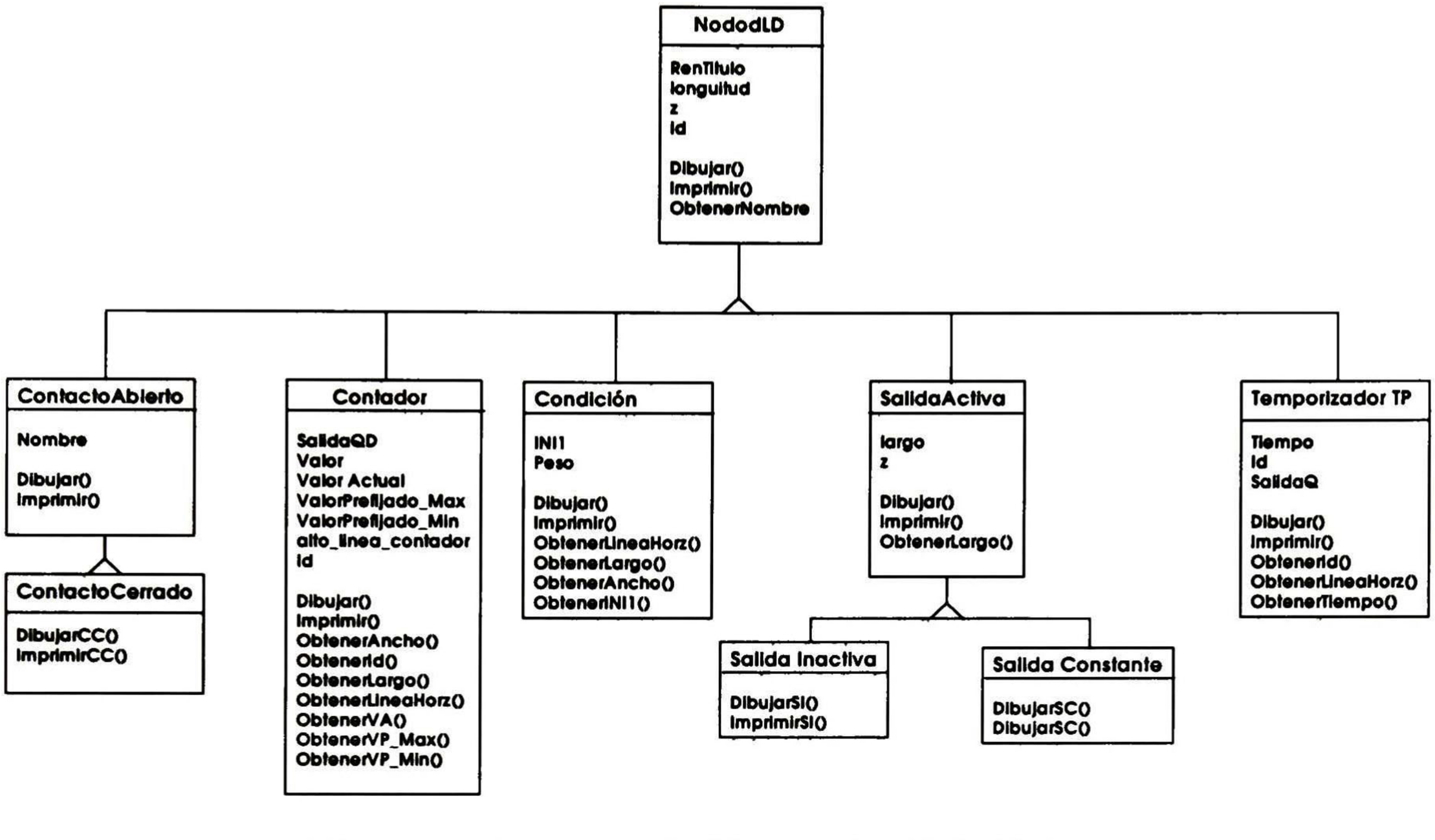


Figura 4.6: Diagrama de Clases del módulo LDGen.

4.5.4 Implementación.

El módulo *LDGen* como el resto de los módulos del sistema *SpADES* han sido desarrollados en un lenguaje de Programación Orientado a Objetos C++Builder con la finalidad de dar más facilidad y legibilidad en su uso. Entre otras ventajas que esto ofrece, cada uno de los elementos de **LD** son tratados con clases con el fin de que cada objeto que se cree, se les asigne información de forma independiente. Cada uno de los métodos que integra el módulo *LDGen* así como el módulo edición de RdP se encuentran descritos en el ápendice A.

4.6 Conclusiones.

Este capítulo describió el funcionamiento del sistema *SpADES* y los módulos que lo integran. El módulo *LDGen* es el módulo que se desarrolló para la implementación de la metodología presentada en el capítulo 3 con la finalidad de automatizar la generación de **LD** a partir de la especificación de un sistema modelado en **RdP**

Se mencionan las fases del proceso de desarrollo del módulo *LDGen* de acuerdo al método de Programación Orientado a Objetos: FUSION. Se utiliza el estándar IEEE 830 para los requerimientos del sistema ya que FUSION no contempla dicha fase.

Capítulo 5

Operación de LDGen

Resumen. En este capítulo se ilustra la utilización del módulo LDGen a través de 2 ejemplos del área de sistemas de manufactura. Para cada sistema se presenta el modelo del controlador, el diagrama de escalera generado y algunos utilitarios del sistema *SpADES*.

5.1 Ejemplo 1. Sistema de Estaciones de Trabajo.

Considere el esquema de la figura 5.1. Éste representa a un sistema de manufactura que está compuesto de *3 estaciones de trabajo:* W_1, W_2, W_3, *un robot* R_1 *y 2 áreas de almacenamiento:* S_e *y* S_s. Las estaciones de trabajo, W_1 y W_2 pueden procesar una pieza mientras que la estación de trabajo W_3 puede procesar b piezas. El comportamiento específico de éste sistema es el siguiente:

Una pieza debe ser tomada del área de almacenamiento S_e y se deposita en W_1 por el robot R_1, luego, cuando la tarea de W_1 ha finalizado, la pieza es automáticamente depositada en W_2. Una vez que W_2 haya finalizado la tarea, el robot R_1 toma la pieza y la deposita en W_3. Finalmente, cuando la pieza ha tomado su forma final, éste es automáticamente descargada de W_3 y es almacenada en S_s. Una vez que el robot comienza con el proceso de cargar en cualquier estación de trabajo, éste no puede ser interrumpido hasta que finalice la descarga.

La figura 5.2 muestra un modelo de RdPI para este sistema de manufactura, donde:

Condición: (Lugar p_0*):* Las partes procesadas en el área de almacenamiento están disponibles a W_3.

Operación 1 (Lugar p_1*):* El robot carga una parte del ára de almacenamiento; la estación de trabajo W_1 procesa la parte; el robot descarga en W_1 y la parte procesada es automáticamente cargada en la estación de trabajo W_2.

Operación 2 (Lugar p_2*):* La estación W_2 procesa la parte la que es depositada.

Operación 3 (Lugar p_3*):* El robot carga una parte a W_3; W_3 procesa la parte; el robot descarga enW_3 y finalmente deja las piezas en el área de almacenamiento.

La figura 5.3 muestra una parte del código en LD en la interfaz del módulo LDGen del sistema SpADES. La representación en Diagrama de Escalera del modelo en Red de Petri se visualiza en la figura 5.4.

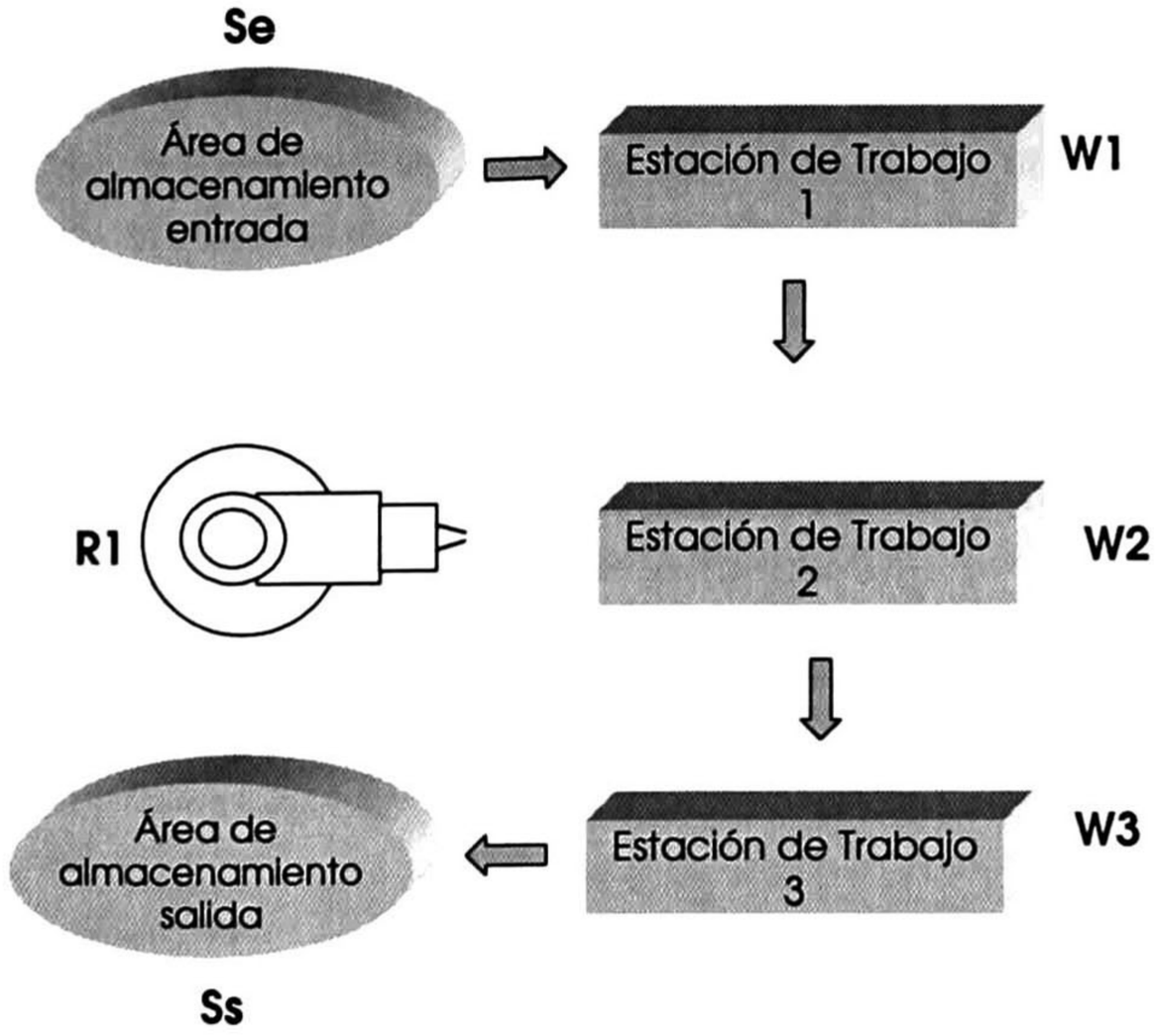


Figura 5.1: Modelo del Sistema de Estaciones de Trabajo.

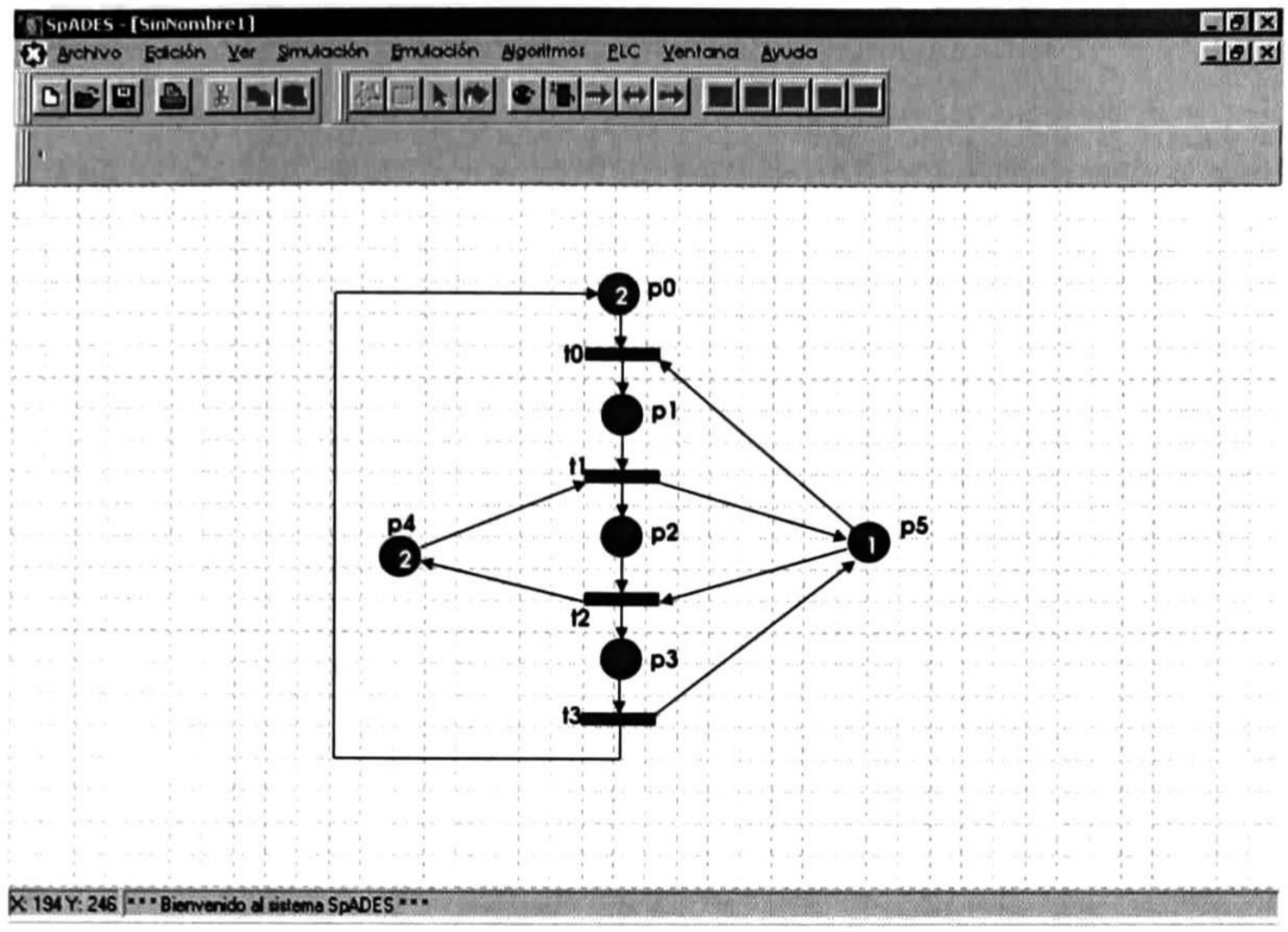


Figura 5.2: Representación en Redes de Petri del modelo del sistema de las Estaciones de Trabajo.

5.2 Ejemplo 2. Generador de Vapor.

El sistema de Generación de Vapor presentado en la figura 5.5 está integrado de los siguientes elementos: *válvula de aire, válvula de combustible y la bujía de encendido.* Este sistema tiene el siguiente comportamiento:

Para comenzar la generación de vapor la válvula de aire se abre después de que transcurren 5 unidades de tiempo, la bujía comienza a dispararse durante 2 unidades de tiempo. Después de la primera unidad de tiempo que la bujía es disparada, la válvula de combustible es abierta introduciendo el combustible a la cámara. El combustible estará quemándose y el vapor se estará generando. Durante el proceso de apagar el sistema, la válvula de combustible es la primera que se cierra; 5 unidades de tiempo después, la válvula de aire es cerrada y el generador de vapor estará en estado ocioso o de paro.

La figura 5.6 muestra el esquema del tiempo que describe este proceso. En la figura 5.7 se puede ver el modelo de RdPIT en la interfaz del editor de Redes de Petri. La figura 5.8 muetra una parte del código en DL en la interfaza del módulo LDGEN del sistema SPaDES. La representación en LD completa se presenta en la figura 5.9.

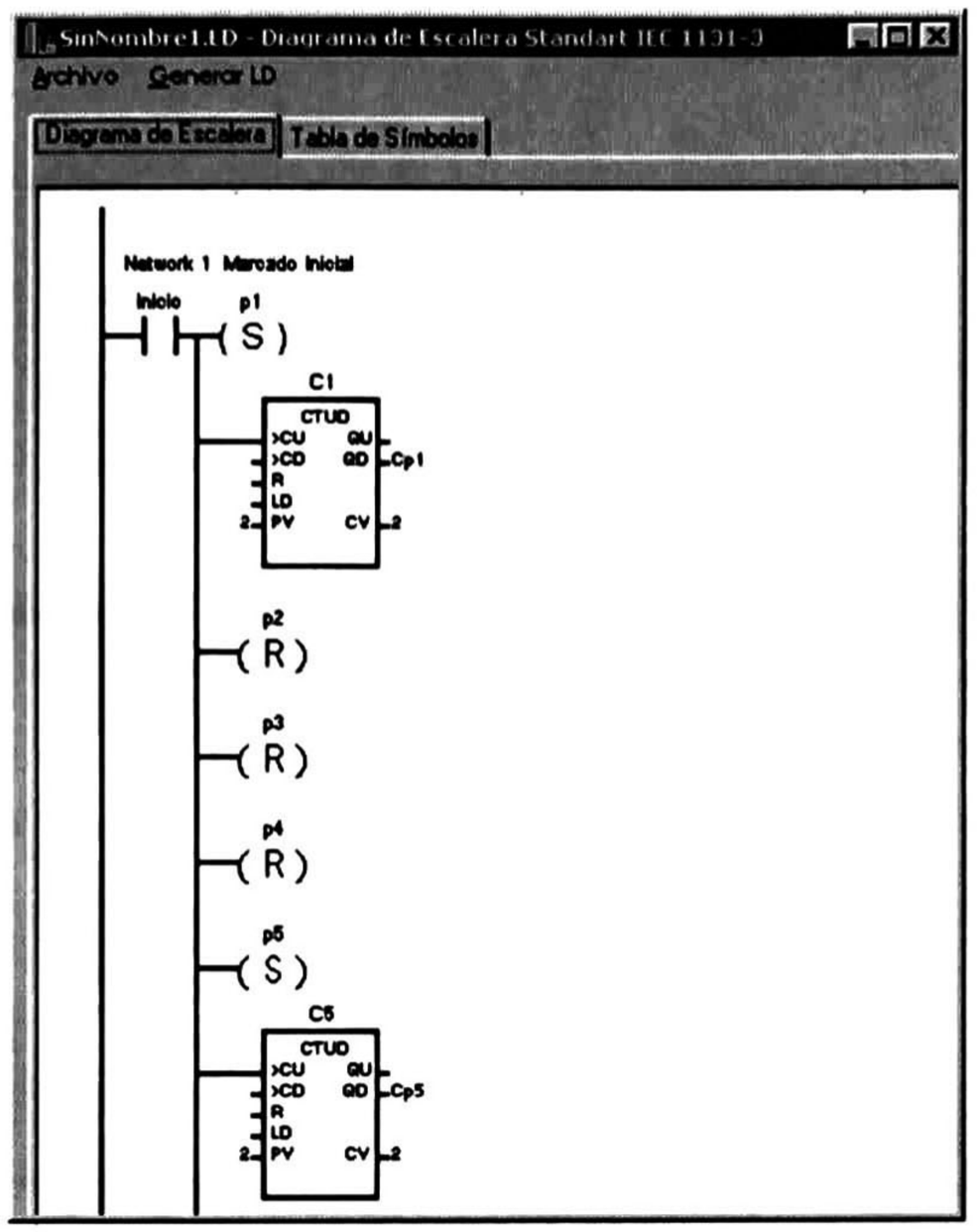


Figura 5.3: Representación en Diagrama de Escalera del marcado inicial.

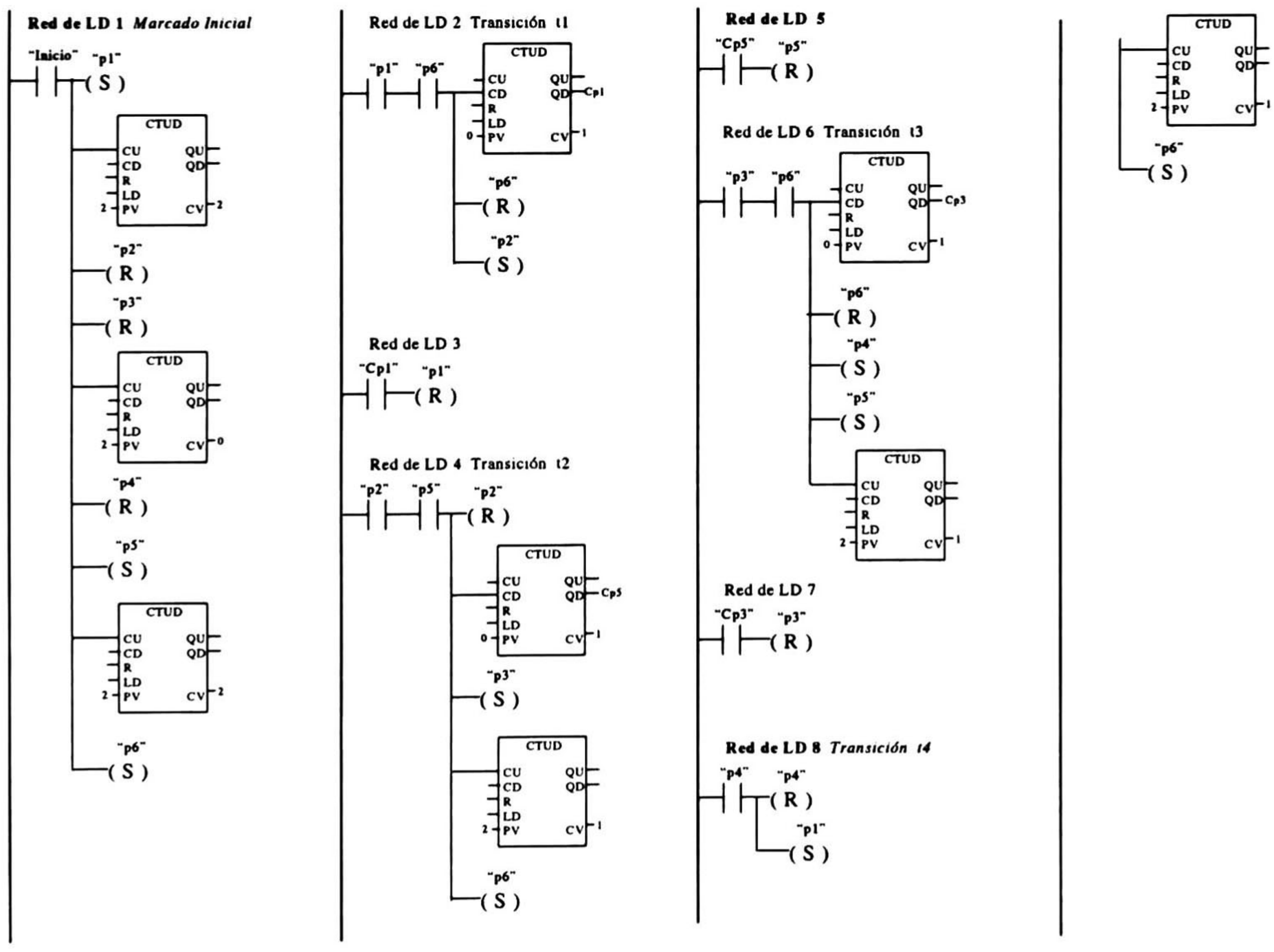


Figura 5.4: Diagrama de Escalera del modelo en Red de Petri del sistema de las estaciones de trabajo.

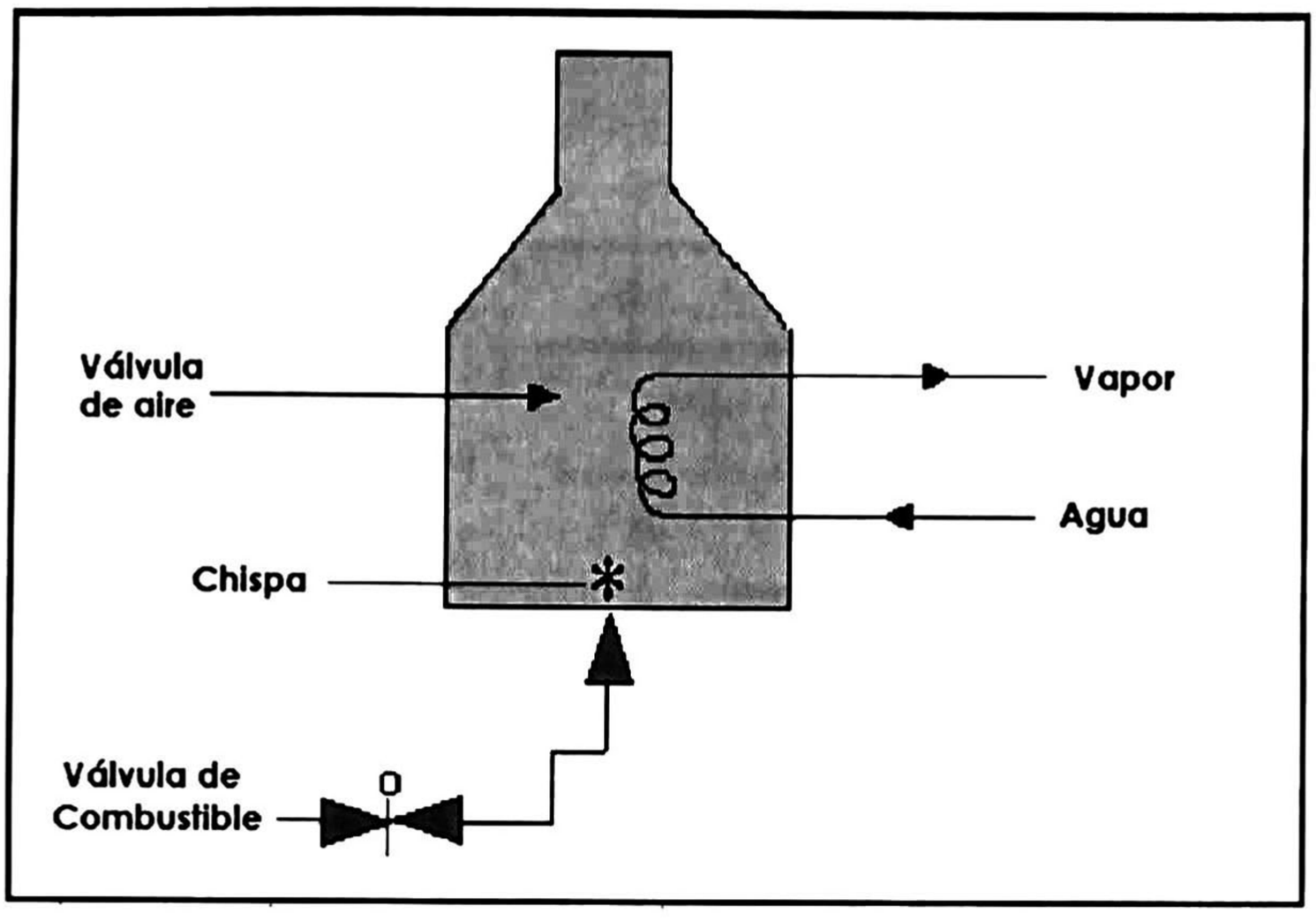


Figura 5.5: Modelo del sistema del Generador de Vapor.

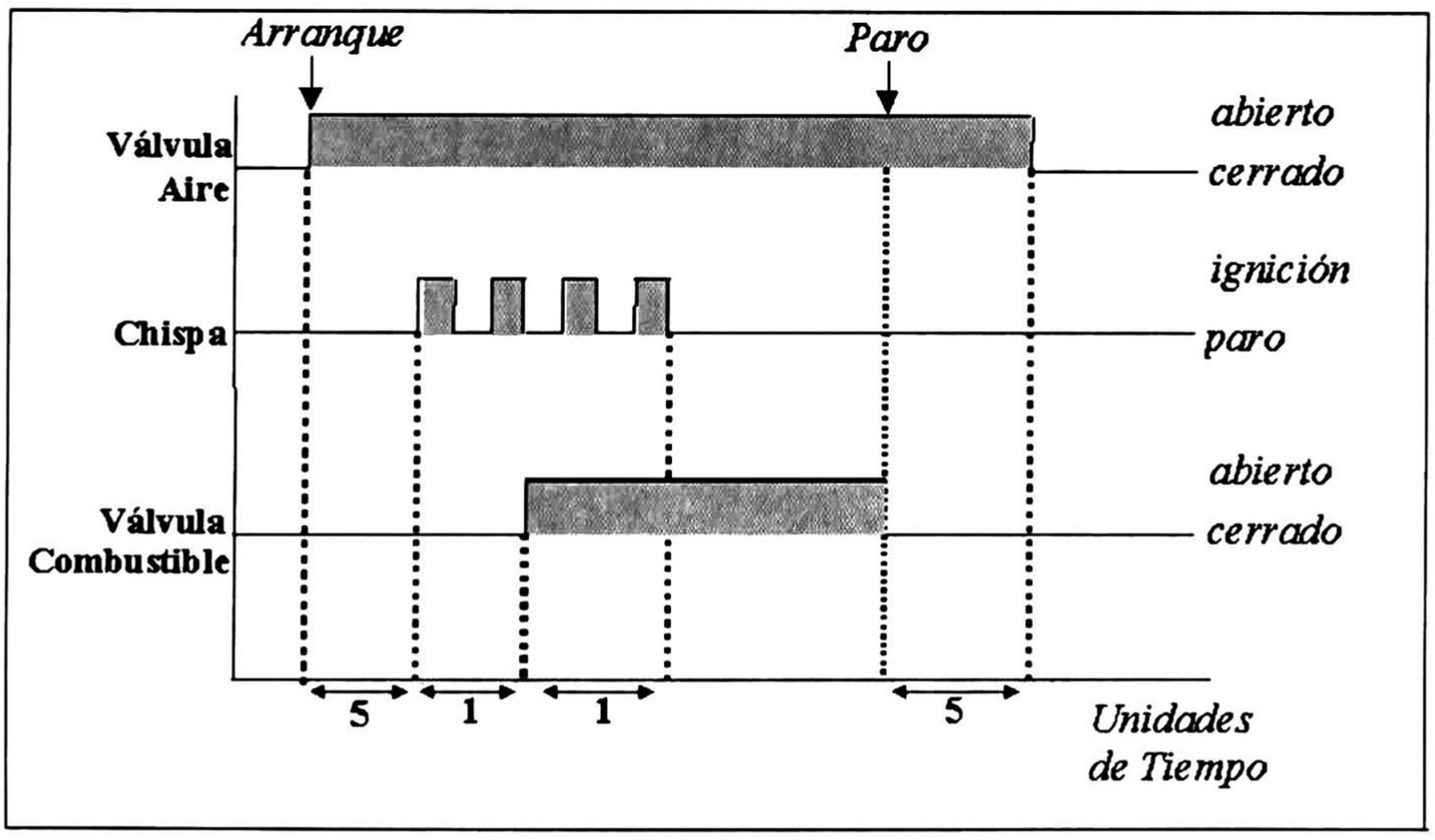


Figura 5.6: Esquema de tiempo del arranque y paro del sistema del generador de vapor.

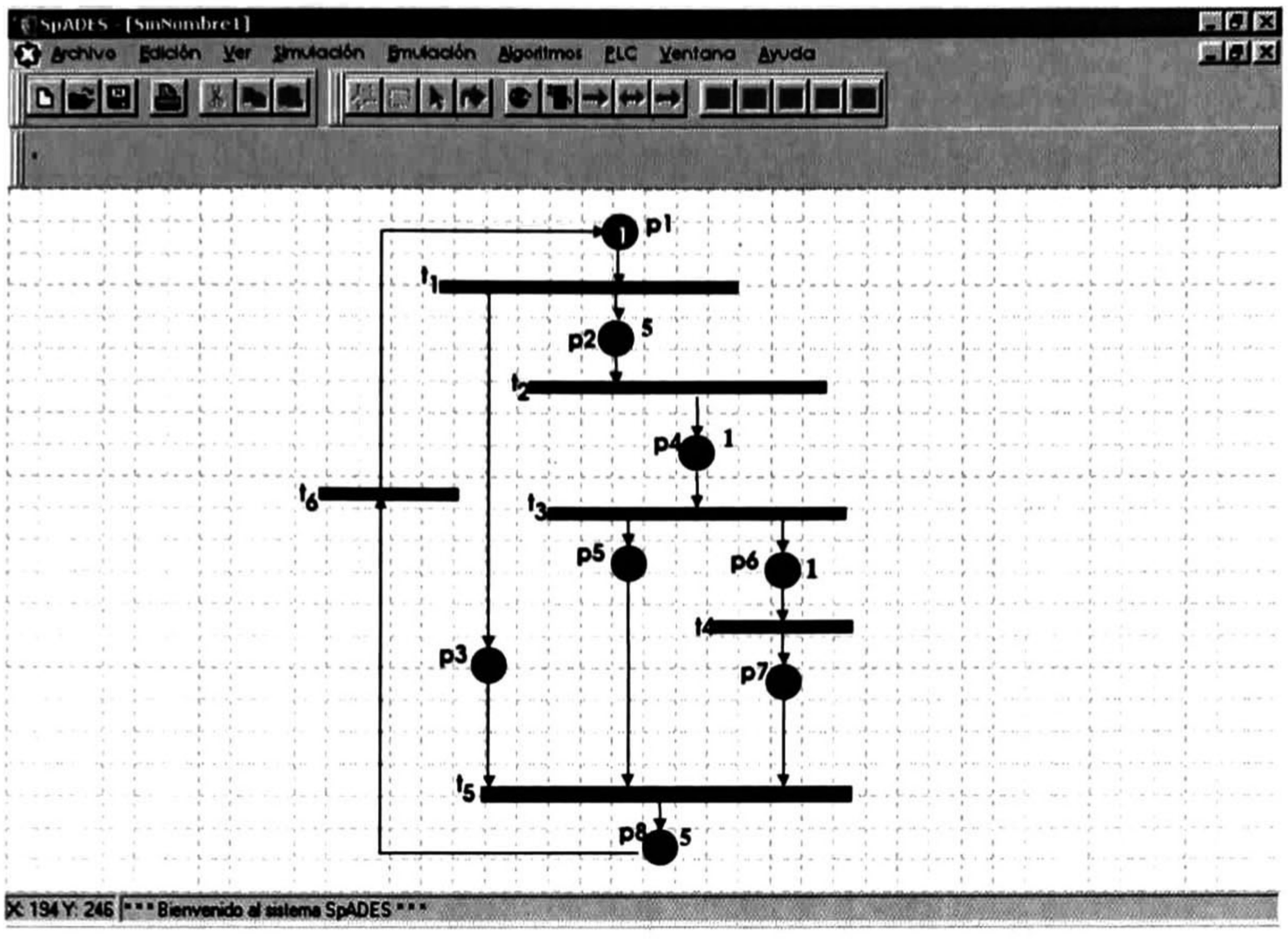


Figura 5.7: Representación en Redes de Petri del modelo del sistema del generador de vapor.

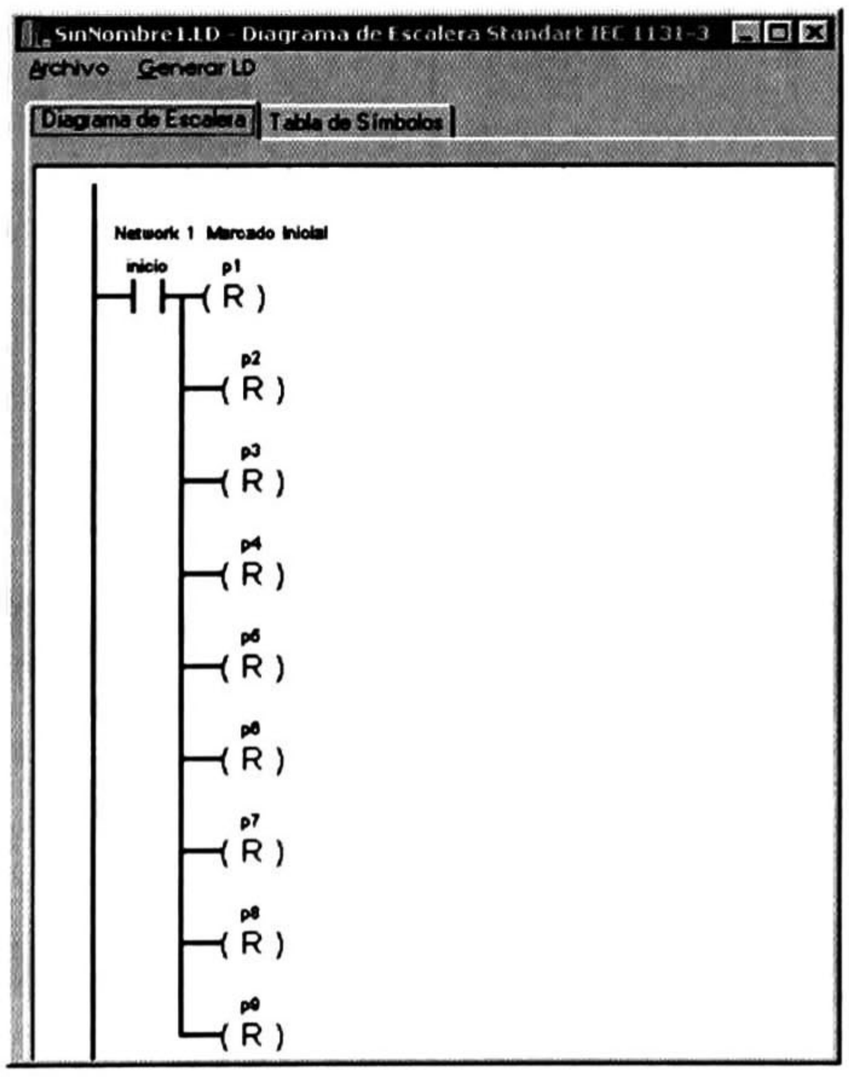


Figura 5.8: Representación en Diagrama de Escalera del modelo del sistema del generador de vapor. Interfaz del módulo LDGen.

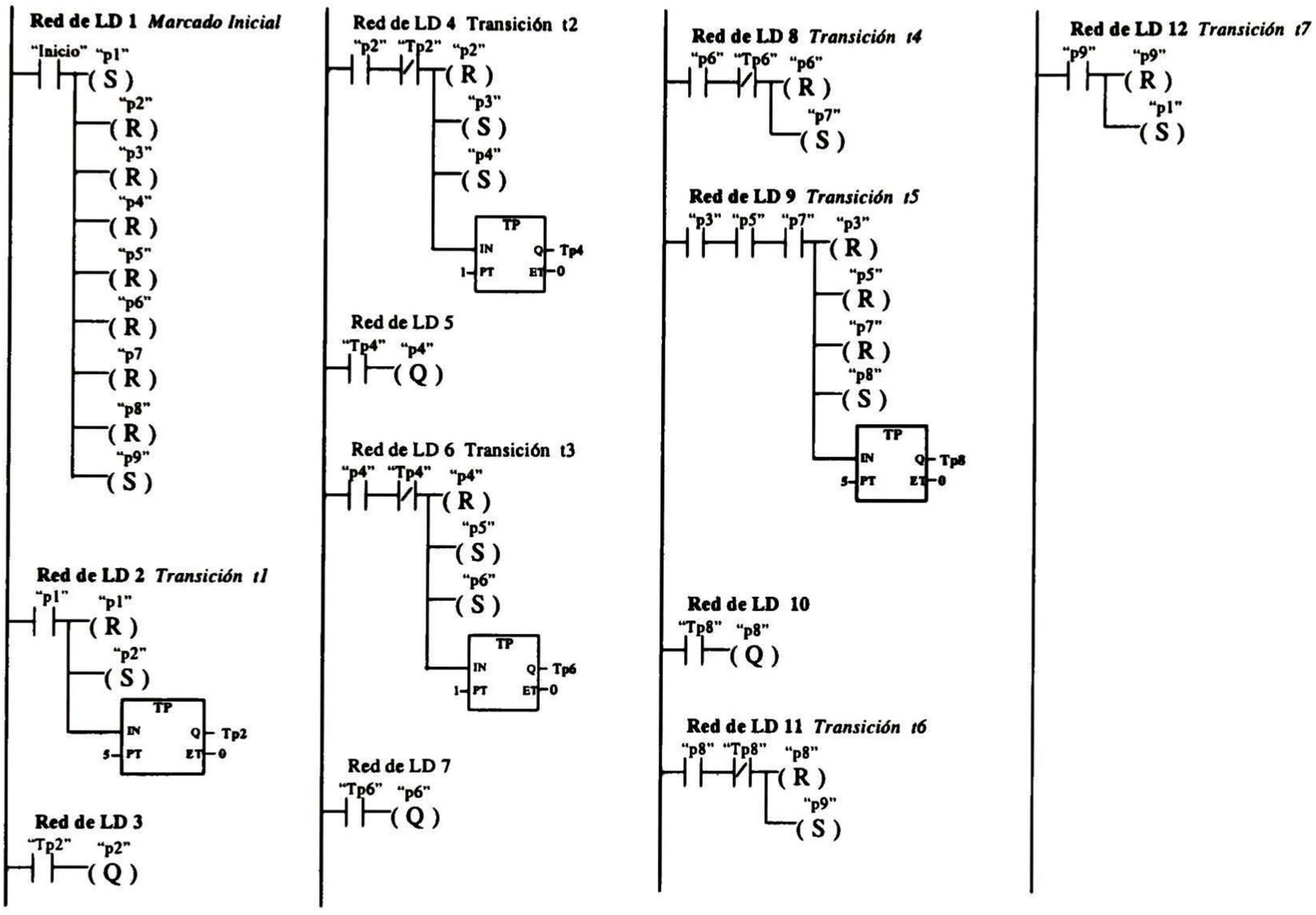


Figura 5.9: Diagrama de Escalera del modelo en Red de Petri del Sistema del generador de vapor.

Capítulo 6

Conclusiones

En el presente trabajo se abordó el problema de la programación alto nivel de controladores lógicos programables. Se presentó una metodología de conversión de modelos expresados con Redes de Petri generalizadas temporizadas a diagramas de escalera y se desarrolló un ambiente visual que automatiza la conversión basado en la metodología propuesta.

La conversión de un modelo en RP se realiza analizando su estructura de la red para cada transición. En función de las características de los lugares que relaciona se aplica un algoritmo que da lugar a una o varias redes de LD que involucran, además de contactos y bobinas, contadores y temporizadores; en un caso el algoritmo asociado optimiza número de redes de LD que representan a un lugar no binario temporizado. En general, los algoritmos de conversión se ejecutan en tiempo polinomial. La metodología de conversión de RP a LD aquí presentada trata con RP más complejas que los métodos reportados en la literatura.

La implementación de la metodología de conversión en el módulo LDGen automatiza el proceso de escritura de LD reduciendo el tiempo de desarrollo de controladores y el riesgo de introducir errores de diseño. LDGen y el editor gráfico constituyen una valiosa herramienta que hace eficiente el ciclo de desarrollo de programas para PLC y complementa el sistema de desarrollo SpADES. Una extensión inmediata al presente trabajo es la realización de un módulo que permita cargar directamente al PLC el código derivado del diagrama de escalera que representa el programa de control. Dada la carencia de información sobre el protocolo de comunicación entre el CPU y el software de programación del PLC Siemens 214, no fue posible desarrollar este módulo en el marco de esta tesis.

Por otro lado, en la metodología de conversión solo se optimizó la representación en LD de lugares temporizados no binarios; el estudio de optimalidad podría extenderse a otros casos con el fin de minimizar el número de elementos de un LD generado.

Apéndice A

Detalles de implementación de LDGen

Los entornos de desarrollo visual proporcionan una serie de ventajas como la simplificación del diseño de la interfaz así como la facilidad de uso por parte del usuario. Por ello, el sistema SpADES ha sido desarrollado en el lenguaje de programación Orientado a Objetos C++ Builder para la edición de Redes de Petri y la generación automática de Diagrama de Escalera.

A.1 Editor de Redes de Petri

A continuación se enuncian las clases relacionadas a las Redes de Petri. Estas clases han sido creadas para facilitar la estructura de la edición de una Red de Petri. La descripción de cada función miembro de las clases se detallan en el manual técnico del sistema *SpADES*.

```
class Nodo : public TShape
{
public:
        Nodo(TForm * TF);
        __fastcall Nodo::~Nodo();
        void ModificarDescripcion(AnsiString descripcion);
        AnsiString ObtenerDescripcion(void);
        void ModificarColor(TColor color);
        TColor ObtenerColor(void);
        void ModificarTiempo(float tiempo);
        float ObtenerTiempo(void);
        void ModificarSensor(AnsiString sensor);
        AnsiString ObtenerSensor(void);
        void ModificarSimbolo(AnsiString simbolo);
        AnsiString ObtenerSimbolo(void);
        void ModificarMedible(bool medible);
        bool ObtenerMedible(void);
        bool ObtenerTemporizado(void);
        void ModificarTemporizado(bool tempo);
        void AgregarArcoEntrada(int nArco);
        void EliminarArcoEntrada(int i);
        int ObtenerArcoEntrada(int i);
```

```
void ModificarArcoEntrada(int i, int nArco);
int TotalArcosEntrada(void);
void AgregarArcoSalida(int nArco);
void EliminarArcoSalida(int i);
int ObtenerArcoSalida(int i);
void ModificarArcoSalida(int i, int nArco);
int TotalArcosSalida(void);
void ModificarCoordenadas(int x, int y);
int TotalNodosEntrada(void);
void AgregarNodoEntrada(int nNodo);
int ObtenerNodoEntrada(int i);
void EliminarNodoEntrada(int i);
void ModificarNodoEntrada(int i, int nNodo);
int TotalNodosSalida(void);
void AgregarNodoSalida(int nNodo);
int ObtenerNodoSalida(int i);
void EliminarNodoSalida(int i);
void ModificarNodoSalida(int i, int nNodo);
bool Sel;
TForm *Forma;
TLabel *ETiempo;
TLabel *Titulo;
int cx, cy;
 private:
      TColor clOriginal;
      bool Temporizado;
      float Tiempo;
      AnsiString Descripcion;
      bool Medible;
      AnsiString Sensor, Simbolo;
      VArcosEntrada AE;
      VArcosSalida AS;
      VNodosEntrada NE;
      VNodosSalida NS;
};
class Arco
 public:
      int Ide;
      int Peso; TLabel *E_Peso;
      TList *LineaS;
      TList *Puntos;
      Arco(int identificador, int peso, bool bidi, bool inhibi);
      Arco::~Arco(void);
      void ModificarIdentidicador(int ide);
      void ModificarPeso(int peso);
      int ObtenerPeso();
```

```
        void ModificarOrigen(int NumOr, bool Orig);
        int ObtenerOrigen(void);
        void ModificarDestino(int NumDs, bool Dest);
        int ObtenerDestino(void);
        void ModificarSentido(unsigned int sentido);
        unsigned int ObtenerSentido(void);
        void ModificaCoorIniciles(void);
        PPunto ObtenerCoorIniciles(void);
        void ModificaCoorFinales(void);
        PPunto ObtenerCoorFinales(void);
        void ModificarTipo(bool B, bool I);
        unsigned int ObtenerTipo(void);
        bool VerificarNodos(int O, int D);
        void Sencillo_A_Bidireccional(void);
        void ReDibujar(void);
        void __fastcall Arco::OnArcoClick(TObject *Sender, TMouseButton Button, TShift-
State Shift, int X, int Y);
        void __fastcall Dibuja(TList *P, TForm *Forma, bool n);
        bool Or;
        bool Ds;
    private:
        int nOr, nDs;
        unsigned int Tipo;
        int x1, x2, x3, x4, y1, y2, y3, y4 ;
        float m;
        TList *Linea;
        int d1, d2, d;
        float D;
        int xp1, yp1, xp2, yp2;
        float CalculaPendiente(int ax, int ay, int bx, int by);
        void __fastcall DibujaPunto(int x, int y, TForm *Forma);
        void __fastcall DibujaLinea(int x1, int y1, int x2, int y2, TForm *Forma);
        void __fastcall DibujaInicio(int x1, int y1, TForm *Forma);
        void __fastcall DibujaFlecha(int xi, int yi, int xf, int yf, float d, TForm *Forma);
        void __fastcall InicioLugar(void);
        int DistaciaTran(void);
        void __fastcall FinalLugar(void);
    };
    class Lugar : public Nodo
    {
    public:
        Lugar(TForm *Forma, int I, int x, int y, int marcas, int tiempo, AnsiString desc);
// Constructor
        __fastcall Lugar::~Lugar();
        void ModificarMarcas(int marcas);
        int ObtenerMarcas();
```

```
        void ModificarCotaMaxima(int cota);
        int ObtenerCotaMaxima();
        void ModificarCotaMinima(int cota);
        int ObtenerCotaMinima();
        void ModificarPosicion(int X, int Y);
        void Dibuja(TForm *Forma, int Ide, int X, int Y, int marcas, int tiempo);
        bool ArcoInhibidor;
        TLabel *Marcado;
        float radio;
    private:
        int Marcas;
        int CotaMax, CotaMin;
        void Dibuja(int Ide, int X, int Y, int marcas, int tiempo);
};
class Transicion : public Nodo
{
public:
        Transicion(TForm *Forma, int Ide, int x, int y, int tiempo, AnsiString desc);
        void ModificarControlable(bool Controlable);
        bool ObtenerControlable();
        void ModificarProbabilistica(bool prob);
        bool ObtenerProbabilistica();
        void ModificarProbabilidad(float prob);
        float ObtenerProbabilidad();
        void ModificarAngulo(float Angulo);
        float ObtenerAngulo();
        void ModificarActuador(AnsiString Actuador);
        AnsiString ObtenerActuador();
    private:
        bool Controlable;
        AnsiString Actuador;
        bool Probabilistica;
        float Probabilidad;
        float Angulo;
        void Dibuja(int Ide, int X, int Y, int tiempo);
};
```

A.2 Módulo LDGen.

En esta sección se describen las clases que representan los elementos de LD. Cada uno de ellos son los Contactos Normalmente Abierto, Contacto Normalmente Cerrado, Bobinas Inactivas, Bobinas Activas, Contadores Ascendente/Descendente, y Condicionales.

CLASE: NodoLD
Función NodoLD
descripción: El método NodoLD es el constructor de la clase.
entradas : nombre. Variable de tipo cadena que representa el nombre del elemento de LD.
salidas: Ninguna.
Función: dibujar
descripción : Método privado en el cuál dibuja la línea horizontal de todo objeto de LD así como el nombre
que le corresponde en la forma Editor PLC.
entradas : colI, colF, ren: Variable tipo entero que indican la posición a dibujar el elemento LD
tipo: Qué elemento de LD es el que se le ha de imprimir el nombre
ancho: Es el ancho del elemento de LD sobre el cuál se centra su nombre.
salidas : Ninguna
Función : ImprimirTitulo
descripción: Imprime el nombre del elemento LD.
entradas : colI, colF, ren: coordenadas sobre la cual se ha de imprimir el elemento.
ancho: Medida de lo ancho del elemento de LD.
alto_linea: Alto de la línea a imprimir en la hoja
salidas : Ninguna
Función : ObtenerNombre
descripción : Método que proporciona el nombre del elemento de LD.
entradas : Ninguna
salidas : Nombre del elemento de LD.
Atributos
RenTitulo, longitud,z,Id

CLASE Condicion
Función : Condicion
descripción : Constructor de la clase Condición.
entradas :col,ren : Coordenadas donde incia a dibujarse el condición
cad: Es la cadena con la que se ha de comparar el valor de 0.
name: Nombre de la condición.
numero: Es el número del lugar el cuál reprensenta el contador y la condición
salidas : Objeto tipo Condición
Función Dibujar
descripción : Método que dibuja el objeto Condición.
entradas : col, ren: Coordenada donde incia a dibujarse el condición
salidas :Columna hasta donde abarca el objeto condición.
Función : Imprimir

descripción : Imprimir en papel el objeto condición
entradas : col: Coordenada x
ren: Coordenada y
alto_linea: Ancho del tipo de letra.
salidas : Columna derecha del objeto tipo Condición.
Función ObtenerLineaHorz
descripción : Calcular la posición sobre la cuál le corresponde la entrada de la condición.
entradas :Ninguna
salidas : Renglón sobre el cuál se sitúa la entrada de la condición.
Función : ObtenerLargo
descripción : Calcula el largo del objeto Contador.
entradas : Ninguna
salidas : Largo del contador
Función : ObtenerAncho
descripción : Calcula el ancho del contador
entradas : Ninguna.
salidas : Ancho del contador
Función: ObtenerINI1
descripción : Devuelve el nombre del contador sobre el cuál se evalúa.
entradas : Ninguna
salidas : Nombre del contador.
Atributos:
INI1,Peso

CLASE ContactoAbierto
Función : ContactoAbierto
descripción : Constructor de la clase ContactoAbierto.
entradas : nombre: Nombre del ContactoAbierto
salidas : Objeto tipo ContactoAbierto
Función : Dibujar
descripción : Método privado en el cuál dibuja un contacto normalmente abierto en la forma Editor PLC.
entradas : col,ren : Coordenadas donde incia a dibujarse el contacto abierto
nombre: Nombre del Contacto.
salidas : Ninguna.
Función : Imprimir
descripción : Imprime en papel el Contacto.
entradas : col: Coordenada x
ren: Coordenada y
alto_linea: Ancho del tipo de letra.
salidas : Columna hasta donde abarca el Contacto Abierto.
Atributos:
Nombre

Clase ContactoCerrado
Función : ContactoCerrado

descripción : Constructor de la clase.
entradas : Nombre del Contacto Cerrado
salidas : Objeto tipo Contacto Cerrado
Función : DibujarCC
descripción : Método privado en el cuál dibuja la línea transversal del Contacto Cerrado.
entradas : col,ren : Coordenadas donde incia a dibujarse el contacto cerrado
salidas : Ninguna.
Función : ImprimirCC
descripción : Imprime en papel la línea transversal.
entradas : col: Coordenada x.
ren: Coordenada y.
salida : Columna que abarca el contacto cerrado.
Atributos:
Ninguno.

CLASE Contador
Función : Contador
descripción : Constructor de la clase Contador.
entradas : valoractual: Valor que ha de incrementar el contador.
valorprefijado_max: Valor máximo que el contador ha de incrementar.
valorprefijado_min: Valor mínimo que el contador ha de decrementar.
id: Identificador del contador
salidas : Objeto tipo Contador.
Función : Dibujar
descripción : Método privado en el cuál dibuja un contador en la forma Editor PLC.
entradas: col,ren : Coordenadas donde incia a dibujarse el contador.
inc: Es el "sentido" del contador, incrementa o decrementa.
minicial: Si el contador va en la red de LD del marcado inicial.
salidas : Columna derecha del contador.
Función : ObtenerAncho
descripción : Devuelve el ancho que abarca el contador.
entradas : Ninguna
salidas : Ancho del contador.
Función : ObtenerId
descripción: Devuelve el identificar del contador
entradas : Ninguna
salidas : Identificador del contador.
Función : ObtenerLargo
descripción : Calcula el largo del contador.
entradas : Ninguna.
salidas : Largo del contador.
Función : ObtenerLineaHorz
descripción : Calcula el renglón donde corresponde la entrada al contador.
entradas : Ninguna
salidas : Renglón de la línea horizontal.
Función : ObtenerVA

descripción :Proporciona el valor de la variable VA.
entradas : Ninguna
salidas : VA: Valor Actual
Función : ObtenerVP_Max
descripción : Devuelve el valor de la variable VP_Max.
entradas : Ninguna
salidas : VP_Max: Vapor predeterminado máximo.
Función : ObtenerVP_Min
descripción : Devuelve el valor de la variable VP_Min.
entradas : Ninguna
salidas : VP_Min: Valor predeterminado mínimo.
Función : Imprimir
descripción :Imprimir en papel el objeto contador.
entradas : col y ren: Coordenadas para imprimir el objeto contador.
inc: Si el contador incrementa o decrementa
minicial: Si el contador es de la red de LD del marcado inicial
alto_linea: Alto de la línea por el tipo de letra.
salidas :Ninguna.
Atributos:
SalidaQD,valor,ValorActual,ValorPrefijado_Max, ValorPrefijado_Min,alto_linea_contador,Id

CLASE SalidaActiva
Función : SalidaActiva
descripción : Constructor de la clase SalidaActiva
entradas : nombre del elemento de LD salida activa
salidas : Objeto tipot Salida Activa
Función : Dibujar
descripción : Método privado en el cuál dibuja una bobina activa (salida) en la forma Editor PLC.
entradas : col,ren : Coordenadas donde incia a dibujarse la bobina activa
salidas : Ninguna.
Función : Imprimir
descripción : Imprime en pantalla el objeto Salida Activa
entradas : col y ren: Coordenadas para imprimir el objeto contador.
alto_linea: Alto de la línea por el tipo de letra.
salidas : Coordenada derecha de la salida activa.
Función : ObtenerLargo
descripción : Devolver lo largo de la salida Activa.
entradas : Ninguna
salidas : Largo de la Salida Activa
Función : ObtenerAncho
descripción : Devuelve el ancho de la salia activa.
entradas : Ninguna
salidas : Ancho de la salida activa.
Atributos
largo,z.

CLASE SalidaInactiva:
Función : SalidaDesactiva.
descripción : Constructor de la clase SalidaInactiva.
entradas : Nombre del objeto.
salidas : Objeto tipo SalidaInactiva.
Función : DibujarSI
descripción : Método privado en el cuál dibuja el símbolo que representa ("R") la salida inactiva.
entradas : col,ren : Coordenadas donde incia a dibujarse la salida inactiva
salidas : Ninguna.
Función : ImprimirCC
descripción : Imprime en papel el símbolo que representa la salida inactiva.
entradas : col: Coordenada x.
ren: Coordenada y.
alto_linea: Alto de la línea por el tipo de letra.
salida : Columna que abarca la salida inactiva.
Atributos
largo,z.
CLASE SalidaConstante
Función : SalidaTemporizada
descripción : Constructor de la clase Salida Constante
entradas : Nombre del elemento SalidaConstante
salidas : Objeto tipo Salida Constante.
Función : DibujarSC
descripción : Método privado en el cuál dibuja el símbolo que representa (" ") la salida constante.
entradas : col,ren : Coordenadas donde incia a dibujarse la salida constante
salidas : Ninguna.
Función : ImprimirSC
descripción : Imprime en papel el símbolo que representa la salida constante
entradas : col: Coordenada x.
ren: Coordenada y.
alto_linea: Alto de la línea por el tipo de letra.
salida : Columna que abarca la salida constante.
Atributos
largo,z.

class TemporizadorTP
Función : TemporizadorTP
descripción : Constructor de la clase Temporizador
entradas : col,ren : Coordenadas donde incia a dibujarse la bobina activa
tiempo: Son las unidades de tiempo que
salidas : Objeto tipo TemporizadorTP
Función : int Dibujar
descripción : Método privado en el cuál dibuja un temporizador en la forma Editor PLC.
entradas : col,ren : Coordenadas donde incia a dibujarse el temporizador

salidas : Ninguna
Función : ObtenerId
descripción : Devuelve el identificar del temporizador
entradas : Ninguna
salidas : Identificador del temporizador.
Función : ObtenerLineaHorz
descripción : Devolve el renglón sobre el cual se dibujará la entrada del temporizador.
entradas : Ninguna
salidas : Renglón de la línea horizontal.
Función : ObtenerTiempo
descripción :Devuelve el valor del tiempo del temporizador.
entradas : Ninguna
salidas : Tiempo del temporizador.
Función : Imprimir
descripción : Imprime en hoja el objeto temporizador.
entradas : col, ren: coordenadas sobre la cual se ha de imprimir el temporizador.
alto_linea: Alto de la línea a imprimir en la hoja.
salidas : Renglón hasta donde abarcó lo largo del temporizador.
Atributos:
Tiempo, Id

El archivo DiagramaEscalera además de estar enfocado a la generación de Diagrama de Escalera está encargado de las tareas de edición de la Tabla de Símbolos, abrir y guardar un Diagrama de Escalera, guardar los cambios efecutados en la Tabla de Símbolos e imprimir el Diagrama de Escalera presentada en la inferfaz del módulo **LDGen**. A continuación se presenta la declaración de los atributos y métodos del archivo DiagramaEscalera.

```
typedef vector<int, allocator<int> > Vca;
typedef vector<int, allocator<int> > Vcc;
typedef vector<int, allocator<int> > Vsa;
typedef vector<int, allocator<int> > Vsd;
typedef vector<int, allocator<int> > Vst;
typedef vector<SalidaActiva *, allocator<SalidaActiva *> > VISActiva;
typedef vector<SalidaDesactiva*, allocator<SalidaDesactiva *> > VISDesactiva;
typedef vector<SalidaActiva *, allocator<SalidaActiva *> > VSActiva;
typedef vector<SalidaDesactiva*, allocator<SalidaDesactiva *> > VSDesactiva;
typedef vector<SalidaTemporizada*, allocator<SalidaTemporizada *> > VSTemporiza-
da;
typedef vector<Contador*, allocator<Contador *> > VContador;
typedef vector<ContactoAbierto*, allocator<ContactoAbierto *> > VCAbierto;
typedef vector<ContactoAbierto*, allocator<ContactoAbierto *> > VSexterna;
typedef vector<ContactoCerrado*, allocator<ContactoCerrado *> > VCCerrado;
typedef vector<TemporizadorLugar*, allocator<TemporizadorLugar *> > VTLugar;
typedef vector<TemporizadorTran*, allocator<TemporizadorTran *> > VTTran;
typedef vector<Condicion*, allocator<Condicion *> > VCondicion;
struct Aux_Base
```

```
{
        int pVISA,pVISD,pVSA,pVSD,pVCA,pVCC,pVC,pVCO,pVCOSA,pVCOCA,pVTL;
        char Senal[10],Descripcion[250];
};
typedef vector<Aux_Base, allocator<Aux_Base> > Aux_VEbase;
struct Aux_Etransicion
{
        int pVSExterna,pVTT,pVCA, pVSD, numIden;
        bool tiempoTran,tiempoLugar,contador;
};
typedef vector<Aux_Etransicion, allocator<Aux_Etransicion> > Aux_VEtransicion;
struct Aux_MInicial
{
        int pVCA;
        char Senal[10],Descripcion[250];
        Aux_VEbase VEBase;
};
struct EContactos
{
        char Nombre[10];
};
struct EContador
{
        int Id,VA,VP_Max,VP_Min;
};
struct ECondicion
{
        char Nombre[10], INI1[10];
        int lugar;
};
struct ETemporizador
{
        float tiempo;
        int Id;
};
struct EGeneral
{
        int tot_tran;
        int tot_lugar;
};
struct ELugarNoBinarioTemporizado
{
        int Lugar, LugaresPrevios,Cota;
};
typedef vector<ELugarNoBinarioTemporizado, allocator<ELugarNoBinarioTemporizado>
> VLugarNoBinarioTemporizado;
```

```
class TFDiagramaEscalera : public TForm
{
private:
        int col, ren, x,y,incRen,colaux,tot_lugar, tot_transicion;
        int alto_linea, largo_hoja, ancho_hoja,inicio,fin,Numero_Hoja; //variables para
la impresión
        bool banContadorLugar,banTiempoTran,banTiempoLugar,componente_creado,ld_creado;
        bool CambioRenglon, cambios_tabla;
        AnsiString NombreArchivo;
        Matriz *mincidencia;
        Matriz *mentrada;
        Matriz *msalida;
        RdP *N;
        Lugar *l;
        Transicion *t;
        Arco *a;
        VCAbierto VCA;
        VCCerrado VCC;
        VISActiva VISA;
        VISDesactiva VISD;
        VSActiva VSA;
        VSDesactiva VSD;
        VSTemporizada VST;
        VContador VC;
        VCondicion VCO;
        VTLugar VTL;
        VSexterna VSExterna;
        VTTran VTT;
        Vca VpCA;
        Vcc VpCC;
        Vsa VpSA;
        Vsd VpSD;
        Vst VpST;
        Condicion *CO;
        ContactoAbierto *CA;
        ContactoCerrado *CC;
        Contador *C;
        SalidaActiva *SA;
        SalidaDesactiva *SD;
        SalidaTemporizada *ST;
        TemporizadorTran *TT;
        TemporizadorLugar *TL;
        bool AbrirArchivo();
        void Cambio_Hoja();
        void Cambio_Renglon(int num_contactos, int y);
        void CrearRdP(RdP *n);
```

```
void ConvertirLugarNBT_LNB(int x, int tot_tran);
void ConvertirRdP_Estructura();
void Encabezado();
void Encabezado_TablaSimbolos();
void Estructura_Inicio();
void Estructura_Transicion(int y);
bool Guardar(bool guardar);
void GuardarArchivo();
void ImprimirArchivo();
void Imprimir_MarcadoInicial();
void Imprimir_Transiciones();
void Imprimir_VariablesGlobales();
void LlenarTabla();
void MarcadoInicial();
void NombreArchivo_Pantalla();
void PiePagina();
void Transiciones();
int TotalLugarPrevios();
void ObtenerMatrices();
LD* ObtenerLD();
void VaciarVLNBT();
void VerificaLugarNoBinarioTemporizado();
void Verificar_AnchoSalidas(int y);
    Aux_MInicial EMInicial;
    Aux_VEtransicion VETransicion;
    Aux_Base ABase;
    Aux_Etransicion AETransicion;
    EContactos ES;
    EContador EC;
    ECondicion ECO;
    ETemporizador ET;
    EGeneral EInformacion;
    ELugarNoBinarioTemporizado ELNBT;
    VLugarNoBinarioTemporizado VLNBT;
public:
    RdP* ObtenerRdP();
    LD *ld;
    void CrearComponente();
   virtual __fastcall TFDiagramaEscalera(TComponent* AOwner);
    TPaintBox *PaintBoxPLC;
};
```

Los métodos que a continación se presentan, invocan las funciones correspondientes para generar Diagrama de Escalera. El método GenerarLD crea el objeto tipo LD y se encarga de crear la estructura de una Red de Petri

```
void __fastcall TFDiagramaEscalera::GenearLD(TObject *Sender)
{
     N=new RdP();
      CrearRdP(Child->ObtenerRdP());
      ld=new LD();
      CA=new ContactoAbierto("inicio");
      ld->VCA.insert(ld->VCA.end(),CA);
      ld->DiagramaEscalera.EMInicial.pVCA=ld->VCA.size()-1;
      ConvertirRdP_Estructura();
      delete N;
}
```

El siguiente método duplica la estructura de la Red de Petri a generar LD con el fin de que, si en la Red de Petri hay una transición temporizada sea transformada en un lugar binario para no alterar la Red de Petri original.

```
void TFDiagramaEscalera::CrearRdP(RdP *n)
{
      int x,y,z;
      Matriz *mi;
      Matriz *me;
      Matriz *ms;
for (x=0; x<n->ObtenerNumeroLugares(); x++)
{
      l=new Lugar(FDiagramaEscalera,n->Vl[x]->Tag+1,n->Vl[x]->Left,
     n->Vl[x]->Top,n->Vl[x]->ObtenerMarcas(),n->Vl[x]->ObtenerTiempo(),"");
      l->ModificarCotaMaxima(n->Vl[x]->ObtenerCotaMaxima());
      l->ModificarCotaMinima(n->Vl[x]->ObtenerCotaMinima());
      l->ModificarSensor(n->Vl[x]->ObtenerSensor());
      l->ModificarDescripcion(n->Vl[x]->ObtenerDescripcion());
      l->Visible=false;
      N->AgregarLugar(l,true);
}
for (x=0; x<n->ObtenerNumeroTransiciones(); x++)
{
      t=new Transicion(FDiagramaEscalera,n->Vt[x]->Tag+1,n->Vt[x]->Left, n->Vt[x]-
>Top,n->Vt[x]->ObtenerTiempo(),"");
          t->ModificarSimbolo(n->Vt[x]->ObtenerSimbolo());
          t->ModificarSensor(n->Vt[x]->ObtenerSensor());
          t->Visible=false;
          N->AgregarTran(t,true);
    }
          mi=Child->N->ObtenerMIncidencia();
          me=Child->N->ObtenerMEntrada();
          ms=Child->N->ObtenerMSalida();
          ObtenerMatrices();
```

```
for(y=0; y<mincidencia->TotalRenglon(); y++)
  for(x=0; x<mincidencia->TotalColumna(); x++)
      {
          z=mi->ObtenerValor(x,y);
          mincidencia->ModificarValor(x,y,z);
          z=ms->ObtenerValor(x,y);
          msalida->ModificarValor(x,y,z);
          z=me->ObtenerValor(x,y);
          mentrada->ModificarValor(x,y,z);
      }
}
```

El método ConvertirRdP_Estructura invoca los métodos Estrucura_Inicio() y Estructura_Transición (y) para crear la estructura. de Diagrama de Escalera.

```
void TFDiagramaEscalera::ConvertirRdP_Estructura()
{
      Estructura_Inicio();
      for(y=0; y<N->ObtenerNumeroTransiciones(); y++)
          Estructura_Transicion(y);
}
```

Estructura_Inicio es el método que se encarga de representar con un elemento de LD cada lugar de la Red de Petri.

```
void TFDiagramaEscalera::Estructura_Inicio()

{
      int x,y;
      unsigned int cont=0;
for(x=0; x<N->ObtenerNumeroLugares(); x++)
{
      cont=0;
      for(y=0; y<mentrada->TotalRenglon();y++)
      if (mentrada->Vector[y][x]==1) // arco inhibidor
      {
          Arco *A=Child->N->BuscarArcoComplemento(x,y,1);
      if ((A->ObtenerTipo()==0) || (A->ObtenerTipo()==1))
          cont=1;
      if (A->ObtenerTipo()==2)
          cont=4;
      break;
}
for(y=0; y<msalida->TotalRenglon();y++)
      if (msalida->Vector[y][x]==1)
      {
          cont=cont+2;
          break;
```

```
        }
    ld->Inicio(N->Vl[x]->Tag,N->Vl[x]->ObtenerMarcas(), N->Vl[x]->ObtenerCotaMaxima(),
    N->Vl[x]->ObtenerCotaMinima(), N->Vl[x]->ObtenerTiempo(),cont, N->Vl[x]->ObtenerSens

    N->Vl[x]->ObtenerDescripcion());
    ld->AgregarBase();
}
}
```

El método Estructura_Transición es la función miembro de la clase DiagramaEscalera que invoca los métodos del objeto tipo LD para representar cada transición en una red de LD.

```
void TFDiagramaEscalera::Estructura_Transicion(int y)
{
int x;
bool salida;
banContadorLugar=false; banTiempoTran=false; banTiempoLugar=false;
for(x=0; x<N->ObtenerNumeroLugares(); x++)
{
    salida=true;
    if (mentrada->Vector[y][x]==1)
    {
    if (msalida->Vector[y][x]==1) salida=false;
    else salida=true;
    Arco *A=Child->N->BuscarArcoComplemento(x,y,1);
    if (((A->ObtenerTipo()==0) || (A->ObtenerTipo()==1)) && (A->ObtenerPeso()==1))
        ld->AgregarTransicion_CondicionesAbiertas(x,salida);
    else
        if(A->ObtenerPeso()>1) ld->AgregarTransicion_Condicional(x,A->ObtenerPeso());
    if (A->ObtenerTipo()==2) ld->AgregarTransicion_CondicionesCerradas(x);
    if ((N->Vl[x]->ObtenerMarcas()>1) || (N->Vl[x]->ObtenerCotaMaxima()>1))

}
else
    if ((N->Vl[x]->ObtenerTiempo()>0) && (mentrada->Vector[y][x]==0)) banTiem-
poLugar=true;
    }
    AnsiString cad=N->Vt[y]->ObtenerSimbolo();
    if (N->Vt[y]->ObtenerSimbolo()!="") ld->AgregarTransicion_CondicionExterna(N->Vt[y]-
>ObtenerSimbolo());
    if (N->Vt[y]->ObtenerTiempo()>0)
    {
        ld->AgregarTransicion_Tiempo(N->Vt[y]->ObtenerTiempo(),y+1);
        ld->AgregarTransicion(0,y+1);
        banTiempoTran=true;
```

```
        ld->AgregarTransicion_SDTemporizador(y+1);
}
for(x=0; x<N->ObtenerNumeroLugares(); x++)
        if ((msalida->Vector[y][x]==1) && (mentrada->Vector[y][x]==0)) ld->AgregarTransicion_Sali
if(banTiempoTran==true) ld->AgregarTransicion(2,y+1);
else ld->AgregarTransicion(0,y+1);
if (banContadorLugar==true)
        for(x=0; x<N->ObtenerNumeroLugares(); x++)
                if ((mentrada->Vector[y][x]==1) && (msalida->Vector[y][x]==0))
                if ((N->Vl[x]->ObtenerMarcas()>1) || (N->Vl[x]->ObtenerCotaMaxima()>1))
                {
                        ld->AgregarTransicion_CondicionesAbiertas(x,false);
                        ld->AgregarTransicion(1,y+1);
                }
if (banTiempoLugar==true)
for(x=0; x<N->ObtenerNumeroLugares(); x++)
{
        if ((msalida->Vector[y][x]==1) && (mentrada->Vector[y][x]==0))
        if (N->Vl[x]->ObtenerTiempo())
        {
                ld->AgregarTransicion_TiempoLugar(N->Vl[x]->Tag,N->Vl[x]->ObtenerTiempo());
                ld->AgregarTransicion(3,y+1);
        }
        }
        }
```

La clases donde se encuentra cada elemento de LD correspondiente a cada nodo de la Red de Petri es llamada *LD*. La clase LD maneja una estructura donde sus componentes son:

```
        typedef vector<int, allocator<int> > Vca;
        typedef vector<int, allocator<int> > Vcc;
        typedef vector<int, allocator<int> > Vsa;
        typedef vector<int, allocator<int> > Vsd;
        typedef vector<int, allocator<int> > Vst;
        typedef vector<int, allocator<int> > Vco;// Vector Condición
        typedef vector<SalidaActiva *, allocator<SalidaActiva *> > VISActiva;
        typedef vector<SalidaDesactiva*, allocator<SalidaDesactiva *> > VISDesactiva;
        typedef vector<SalidaActiva *, allocator<SalidaActiva *> > VSActiva;
        typedef vector<SalidaDesactiva*, allocator<SalidaDesactiva *> > VSDesactiva;
        typedef vector<SalidaTemporizada*, allocator<SalidaTemporizada *> > VSTem-
porizada;
        typedef vector<Contador*, allocator<Contador *> > VContador;
        typedef vector<ContactoAbierto*, allocator<ContactoAbierto *> > VCAbierto;
        typedef vector<ContactoAbierto*, allocator<ContactoAbierto *> > VSexterna;
        typedef vector<ContactoCerrado*, allocator<ContactoCerrado *> > VCCerrado;
```

```
        typedef vector<TemporizadorLugar*, allocator<TemporizadorLugar *> > VTLu-
gar;
        typedef vector<TemporizadorTran*, allocator<TemporizadorTran *> > VTTran;
        typedef vector<Condicion*, allocator<Condicion *> > VCondicion;
    struct Base
    {
        int pVISA,pVISD,pVSA,pVSD,/*pVST,*/pVCA,pVCC,pVC,pVCO,pVCOCA,pVTL;
        AnsiString Senal,Descripcion;
        int LugaresPrevios,Cota;
    };
        typedef vector<Base, allocator<Base> > VEbase;
struct Etransicion
{
        Vca VpCA;
        Vcc VpCC;
        Vsa VpSA;
        Vsd VpSD;
        Vst VpST;
        Vco VpCO;
        int pVSExterna,pVTT,pVCA, numIden,pVSD;
        bool tiempoLugar,tiempoTran,contador;
};
        typedef vector<Etransicion, allocator<Etransicion> > VEtransicion;
struct MInicial
{
        int pVCA;
        AnsiString Senal,Descripcion;
        VEbase VEBase;
};
struct diagramaescalera
{
        MInicial EMInicial;
        VEtransicion VETransicion;
};
class LD
{
public:
        diagramaescalera DiagramaEscalera;
        Etransicion ETransicion;
        Base EBase;
        VCAbierto VCA;
        VCCerrado VCC;
        VISActiva VISA;
        VISDesactiva VISD;
        VSActiva VSA;
        VSDesactiva VSD;
```

```
        VSTemporizada VST;
        VContador VC;
        VCondicion VCO;
        VTLugar VTL;
        VSexterna VSExterna;
        VTTran VTT;
        LD();
        ~LD();
        void Inicio(int x, int marcas, int CotaMax, int CotaMin, float tiempo,int entra-
da,AnsiString sensor, AnsiString
    descripcion);
void AgregarTransicion_CondicionesAbiertas(int x, bool salida);
void AgregarTransicion_CondicionesCerradas(int x);
void AgregarTransicion_CondicionExterna(AnsiString sexterna);
void AgregarTransicion_Condicional(int x, int peso);
void AgregarTransicion_SalidaActiva(int x);
void AgregarTransicion_SDTemporizador(int y);
void AgregarTransicion_Tiempo(float tiempo, int tran);
void AgregarTransicion_TiempoLugar(int x, float tiempo);
void AgregarBase();
void AgregarTransicion(int ban, int y);
void LimpiarVectores_Transicion();
void LimpiarVectores_Base();
 private:
        ContactoAbierto *CA;
        ContactoCerrado *CC;
        SalidaActiva *SA;
        SalidaDesactiva *SD;
        SalidaTemporizada *ST;
        Contador *C;
        Condicion *CO;
        TemporizadorLugar *TL;
        TemporizadorTran *TT;
        void Limpiar();
};
```

El diagrama de clases de la edición de una RdP es mostrado en la tesis [4]. Para ver a detalle el desarrollo del resto de cada uno de los métodos y funciones de LDGen y del módulo de edición de RdP, así como los otros módulos que integran el sistema *SpADES*, puede consultar el *manual técnico del sistema SpADES*.

Bibliografía

[1] R. David, H. Alla, *Petri Nets and Grafcet,* Prentice Hall, 1992.

[2] L. Baresi, M. Mauri, A. Monti and M. Pezze. "PLCTools: Design, Formal Validation, and Code Generation for Programmable Controllers".Proc. of the *IEEE Systems, Man and Cybernetics*, pp. 2437-2442, October 2000.

[3] G. Canet, S. Couffin, J.-J. Lesage, A. Petit and Ph. Schnoebelen. "Towards the automatic verification of PLC programs written in Instruction List" Proc. of the *IEEE Systems, Man and Cybernetics*, pp. 2449-2454, October 2000.

[4] Claudia Gabriela Pulido. "Desarrollo de un entorno para el análisis, diseño y especificación de Sistemas de Eventos Discretos" Tesis Maestría en desarrollo, Cinvestav Unidad Guadalajara, México. 2001.

[5] Derk Colemman. "Object-Oriented Development, the FUSION method" Prentice Hall. 1994.

[6] Jorg Desel y Javier Esparza. "Free Choice Petri Nets" Cambridge University Press, 1995.

[7] G. Frey, "Automatic implementation of Petri Net based control algortihms on PLC" Proc. of the *IEEE American Control Conference ACC 2000*, pp. , June 2000.

[8] Estándard Internacional $CEI\ IEC1131$ Parte 3.

[9] L. Isidro Aguirre. "Observadores asintóticos para sistemas de Eventos Discretos modelados con Redes de Petri" Tesis Maestria, Cinvestav Unidad Guadalajara. México. 1999.

[10] G.B. Lee, and J.S. Lee, "The state equation of Petri net for the **LD** program" Proc. of the *IEEE Systems, Man and Cybernetics*, pp. 3051-3056, October 2000.

[11] Italia Jiménez, Ernesto López, Antonio Ramírez, Claudia Pulido. "Programming PLC from Petri Nets Specificactions" *IEEE International Symposium on Robotics and Automation, ISRA'2000*, pp. 614-618, Noviembre 2000.

[12] Italia Jiménez, Ernesto López, Antonio Ramírez. "Synthesis of Ladder Diagrams from Petri Nets Controller Models". *IEEE International Symposium Inteligent Control, ISIC' 2001*, pp. 225-230, Septiembre 2001.

[13] E. López Mellado. "Introducción a las Redes de Petri", Apuntes de la Universidad Autonoma de Nuevo León. Octubre 1997.

[14] T. Merkte and T. Menzel. "Methods and tools to the verification of safety-related control software" Proc. of the f the *IEEE Systems, Man and Cybernetics*, pp. 2455-2457, October 2000.

[15] Peterson. "Petri Net Theory and the modeling of Systems" Prentice-Hall. 1981.

[16] G. Michel. "Programmable Logic Controllers. Architecture and Application" John Wiley & Sons Ltd. England 1990.

[17] Phil Melore. "Your Personal PLC Tutor Site" Vease la página **www.PLCOPEN.com**

[18] C. Rammoorthy and G. HO. "Performance evaluation of asynchronous concurrent systems using Petri nets" *IEEE Transactions on Software Engineering*, 6(5):440-449, September 1980.

[19] A. Ramírez-Serrano, S.C. Zhu, S.K.H. Chaan and B. Benhabib. "A Hybrid PC/PLC Architecture for Manufacturing System Control-Implementation" Proc. of the *IEEE Systems, Man and Cybernetics*, pp. 1697-1702, October 2000.

[20] A. Santoyo, I. Jiménez-Ochoa, A. Ramírez-Treviño. "A complete cycle for controller design in Discrete Event Systems" Proc. *IEEE Systems, Man and Cybernetcis Conference.* pp. 2688-2693, Octubre 2001.

[21] M. Silva. "Las Redes de Petri: en la automática y la informática" Editorial A.C. Madrird, España 1985.

[22] Timothy J. Maloney. "Electrónica Industrial Moderna" Editorial Prentice Hall-Hispanoamericana.México, 1997.

[23] M. Uzam. "Petri-Net based supervisory control of discrete event systems and their ladder logic diagram implementations" Tesis doctoral. Universidad de Salford. 1998.

[24] M.C. Zhou, and K. Venkatesh, "Modeling, Simulation and Control of Flexible Manufacturing Systems, A Petri Net Approach" Ed. World Scientific Pu. Singapore, 1999.

ANIVERSARIO
Cinvestav

Centro de Investigación y de Estudios Avanzados del IPN

Unidad Guadalajara

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó la tesis: PROGRAMACIÓN DE PLC A PARTIR DE ESPECIFICACIONES CON RP del(a) C. María Italia JIMÉNEZ OCHOA el día 17 de Diciembre de 2001 .

Dr. Jose Luis Leyva Montiel
Investigador Cinvestav 3B
CINVESTAV GDL
Guadalajara

Dr. Luis Ernesto López Mellado
Investigador Cinvestav 3A
CINVESTAV GDL
Guadalajara

Dr. Arturo del Sagrado Corazón Sánchez Carmona
Investigador Cinvestav 3B
CINVESTAV GDL
Guadalajara

Dr. Félix Francisco Ramos Corchado
Investigador Cinvestav 2A
CINVESTAV GDL
Guadalajara

Dr. Gustavo Adolfo Sánchez Galindo
Gerente Técnico
Aplicaciones Industriales Avanzadas, S.A. de C.V.
Monterrey, Nuevo León