APLICACIÓN BASADA EN REDES DE PETRI PARA LA PROGRAMACIÓN AUTOMÁTICA DE PLCs EN PROCESOS BATCH

Jorge Eliecer Calvo Giraldo
Mitchel Esteban Collazos Agredo

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Electrónica, Instrumentación y Control
Ingeniería en Automática Industrial
Popayán
2015

APLICACIÓN BASADA EN REDES DE PETRI PARA LA PROGRAMACIÓN AUTOMÁTICA DE PLCs EN PROCESOS BATCH

Jorge Eliecer Calvo Giraldo
Mitchel Esteban Collazos Agredo

Trabajo de Grado Presentado como Requisito para Optar al Título de Ingeniero en Automática Industrial

Director:
PhD. **Carlos Alberto Gaviria López**

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Electrónica, Instrumentación y Control
Ingeniería en Automática Industrial
Popayán
2015

Nota de Aceptación

_____

_____

_____

Director _____

PhD. Carlos Alberto Gaviria López

Jurado _____

Jurado _____

Fecha de sustentación: Popayán, 2015

CONTENIDO

Lista de tablas

Lista de figuras

RESUMEN

El trabajo presentado a continuación desarrolla las siguientes temáticas:

- El uso de una metodología formal basada en el estándar ISA 88 para el modelado de sistemas de manufactura tipo batch. Con la obtención de un modelo de equipos y de un modelo de control de procedimientos, se describe un procedimiento para el diseño de las especificaciones de control basado en redes de Petri. El procedimiento de diseño de la lógica de control abarca los siguientes puntos:
    - Configuración de las distintas secuencias de funcionamiento de la lógica de control en condiciones normales y anormales según la definición dada en el estándar de ISA 88 para la configuración de estados de fase en los procesos batch.
    - Uso de modelos en redes de Petri para representar el comportamiento en eventos discretos de la planta en lazo abierto y en lazo cerrado (comportamiento deseado).
    - Generación de la lógica de control para PLC basado en dos lenguajes de programación definidos en el estándar IEC 61131-3: Diagramas en lógica escalera o Ladder (LLD), y Cartas de funciones secuenciales o Sequential function charts (SFC).
- La presentación de una aplicación software desarrollada en Java que permitirá al usuario:
    - Configuración del modelo de estados de fase propuesto por ISA-88.
    - Implementar redes de Petri construidas a partir de los editores gráficos PIPE/CRP o Snoopy.
    - Realizar el proceso de traducción de manera automática del comportamiento deseado de la planta para implementación en PLC.

INTRODUCCION

Los mercados globales están evolucionando a sistemas más dinámicos y ágiles que sean reactivos a los cambios del entorno, esto conlleva a la búsqueda de mejoras en los sistemas de manufactura tradicionales, los cuales están estructurados de forma centralizada y jerárquica, carentes de reactividad, flexibilidad, robustez y reconfigurabilidad. Actualmente, los factores que están liderando la evolución de los sistemas de control de procesos por lotes o procesos batch, están impulsados por un mercado altamente competitivo, y una creciente necesidad de aumentar el intercambio de información electrónica, donde es predominante el uso de PLCs para la implementación de algoritmos de control.

Para facilitar el modelado de este tipo de procesos, se ha desarrollado el estándar ISA-88, que tiene la filosofía de separar la descripción de los equipos, de las operaciones a realizar con los equipos. En las herramientas software comerciales que soportan el diseño, simulación y ejecución de control de procesos por lotes ajustado al estándar ISA-88, la programación final de los PLCs queda abierta a la destreza y subjetividad del programador.

Para la programación de los PLCs, existen métodos formales para definir algoritmos de control, entre ellos, la teoría de autómatas finitos, la lógica temporal, el lenguaje Z [1], etc. Sin embargo, estas técnicas no son utilizadas ni aceptadas por los fabricantes de PLCs para la programación de sus productos [2]. Los fabricantes han estandarizado sus lenguajes en la norma IEC 61131-3, pero dicha norma sólo describe los lenguajes, no las metodologías de diseño. Se ha adoptado el Grafcet como aproximación en la norma IEC 60848, pero ésta carece de partes esenciales en el diseño de algoritmos para controladores, tal como el levantamiento de requerimientos, los criterios de aceptación y el proceso de validación del algoritmo [3]. Por tanto, existe un vacío en la cadena de metodologías y formalismos matemáticos para diseñar los algoritmos de control que son programados en PLCs, y aplicados a sistemas reales de manufactura industrial.

Un sistema de manufactura industrial puede ser tratado como un Sistema a Eventos Discretos (SED), donde las actividades que transforman materiales en productos finales varían en función a la ocurrencia de un conjunto discreto de eventos [4]. Por lo tanto, las redes de Petri pueden usarse para el modelado formal de este tipo de procesos, en el que el diseño de las especificaciones de control de estos sistemas puede ser realizado con base a la teoría de control supervisor para SEDs.

Una de las ventajas del uso de las redes de Petri, es que los modelos generados pueden utilizarse tanto para el análisis de las propiedades de comportamiento, como para la evaluación de desempeño, gracias a su representación formal y gráfica que permite formular el estado de recursos, condiciones, y nodos de transición entre eventos [5]. Esta

herramienta ha sido utilizada en otros trabajos cómo apoyo a la filosofía de los modelos planteados en ISA-88.

A medida que los sistemas industriales se hacen más grandes y complejos, es indispensable el uso de herramientas software de apoyo que soporten el diseño de algoritmos de control (previo a la implementación de posibles soluciones), para que el usuario pueda modelar, simular y validar comportamientos.

De acuerdo a lo anterior, este proyecto plantea la construcción de una herramienta software que facilite la programación de la lógica del equipo a partir de modelos de redes de Petri, tomando como punto de partida los modelos de la parte 1 del estándar ISA-88.

1. Capítulo I: Bases para el diseño del software de traducción de red de Petri a lenguaje para PLCs

El trabajo realizado en esta tesis hace uso de conceptos a cerca de los procesos batch, el estándar ISA-88, modelado con redes de Petri, síntesis de supervisores para redes de Petri, y traducción de modelos de red de Petri a código estándar para PLCs. Este capítulo tiene como objetivo, ubicar la aplicación software desarrollada en este trabajo en el contexto del modelado según ISA-88 y su relación con los conceptos acerca del formalismo de redes de Petri que se han empleado para su diseño.

1.1. Sistemas de producción Batch

Un proceso batch, es un proceso en el cual su salida se da en cantidades finitas de material denominadas lotes, por medio de un conjunto de actividades de transformación ordenadas, denominado *receta*. Este tipo de procesos tiene un principio y un final, de esta forma en el caso en el que la cantidad solicitada de material de salida sea mayor que la del lote, el proceso de producción se debe realizar repetidas veces hasta completarlo.

La configuración de los equipos de la planta para un proceso batch, su estructura, puede ser diferente a la estructura dispuesta en la receta, dado que en ocasiones es posible ejecutar varios pasos de la receta en un mismo equipo de forma generalmente consecutiva. Además, la configuración del equipamiento puede cambiar cada vez que un producto diferente sea elaborado. Esto resulta conveniente, dado que brinda flexibilidad para fabricar diferentes productos en la misma planta de procesamiento [6]. Por otro lado las plantas de procesamiento batch permiten ser adaptadas ante cambios en la producción y de esta forma abarcar un extenso rango de condiciones de funcionamiento dentro de la misma configuración de los equipos en planta.

Por lo general, los procesos batch son usados para manufacturar un gran número de productos [4], siendo esta una de sus características principales: capacidad de adaptarse con rapidez y facilidad, lo cual justifica en gran medida su adquisición debido a que se realiza una sola vez. Esto en términos económicos resulta fundamental, ya que las empresas no están exentas de caer en las incertidumbres que deparan los futuros escenarios de trabajo, y bajo qué condiciones estarán regidos. Por tanto la adaptabilidad y la rápida respuesta ante la demanda, son factores que proporcionan gran versatilidad a las plantas que operan bajo procesos batch.

Controlar un proceso batch es bastante diferente de lo que resultaría hacerlo en un proceso continuo. Un proceso continuo usualmente opera en un cierto punto y el sistema de control tratará siempre de mantener la operación en ese punto. Un proceso batch involucra partes continuas y secuenciales [7]. Por ejemplo, en la figura 1.1 se muestra cuando un tanque es llenado, para un determinado nivel el flujo del tanque es continuo. Cuando se obtiene el nivel deseado, el flujo es apagado y la siguiente etapa de control puede tomar lugar. De esta forma se han ejecutado operaciones secuenciales, como lo son la apertura de la válvula, la lectura entregada por el sensor, que en este caso corresponde a la medida del nivel en el tanque y la última parte de esta secuencia sería el cierre de la válvula.

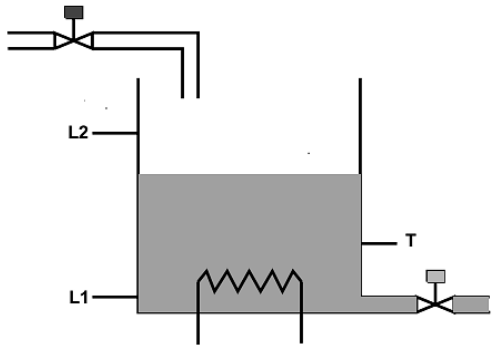


Figura 1.1 Tanque simple. ***Fuente:*** [7]

1.2. Sistema administrador de procesos batch

Un sistema administrador de procesos batch realiza la supervisión de la ejecución de la receta de producción [8]. En términos generales, un administrador batch permite construir y editar las recetas de producción, de modo que tengan como consecuencia, un impacto en las acciones de control ejecutadas sobre el proceso, partiendo de una estrategia de producción definida en el nivel de gestión de la empresa [9].

En la figura 1.2 se muestran las fases de un proceso batch, creadas en un sistema de control de equipos una vez que ha sido enlazado con el sistema administrador de procesos batch. A

partir de los anterior se busca que la lógica para cada uno de los estados (*ver ítem 1*.5) de las fases que conforman los procesos como el de la figura en mención, pueda ser diseñada en red de Petri y traducida posteriormente a lenguaje para PLC.

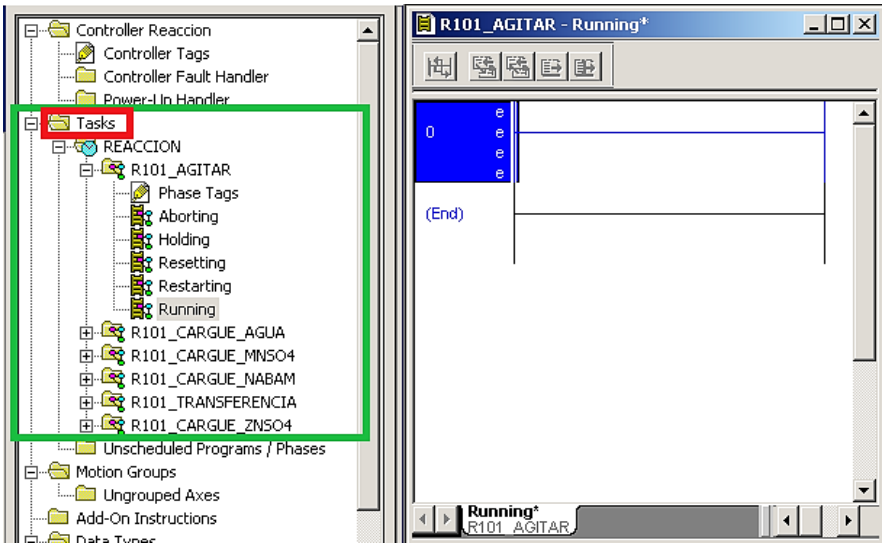


Figura 1.2 Estados para la fase Agitar de un proceso de reacción. *Fuente: RsLogix5000*

Para este proyecto se requiere que el diseño de la lógica de control en redes de Petri cumpla con los requerimientos planteados en el estándar ISA-88, es decir, se deberá tener en cuenta el concepto de modularidad para tener una vista del proceso divido en partes que se puedan manipular fácilmente como se observa en la figura 1.2.

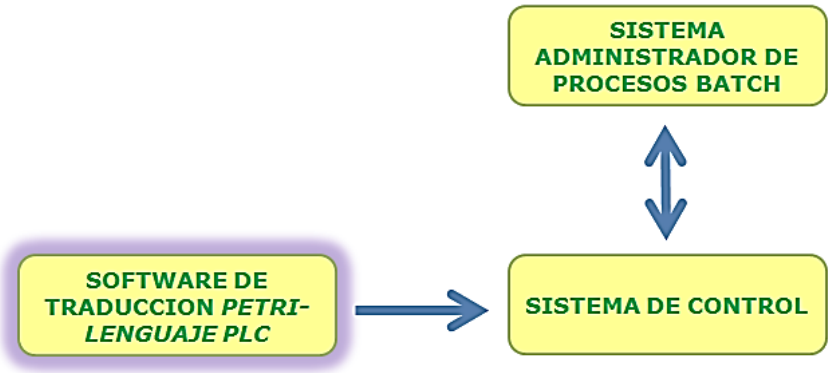


Figura 1.3 Interacción entre el software de traducción de RdP a lenguaje PLC, Sistema administrador batch y el sistema de control. *Fuente: Propia*

Después de haber cumplido con este requerimiento, y una vez el software propuesto traduzca la lógica de control de red de Petri (RdP) a lenguaje para PLC, se deberá contar

con un sistema que administre dicha lógica, el cual se conoce como sistema administrador de procesos batch, y así poder validar que la lógica efectivamente cumple con las especificaciones de control del proceso. Esta idea se ilustra en la figura 1.3.

Para este trabajo el sistema administrador de procesos batch a utilizar será FactoryTalk Batch, dado que es con el cual se cuenta en el programa de Ingeniería en automática industrial de la Universidad del Cauca. Para conocer a cerca del uso de esta herramienta consultar en [10].

1.3. El estándar ISA-88

En lo que se refiere a la automatización de los procesos batch, la sociedad ISA (International Society of Automation) estableció un criterio común para este tipo de procesos, el cual fue denominado S88. Este estándar se divide en 5 partes:

1. ANSI/ISA88.01 – 1995. Parte 1: Definen los modelos y terminología que describen los sistemas de control tipo Batch.
2. ANSI/ISA88.00.02 – 2001. Parte 2: Define las estructuras de los datos que se manejan en los sistemas Batch, así como las directrices para los lenguajes que se vayan a utilizar para la descripción de los modelos. En general, se definen guías de intercambio de información, con el objetivo de hacer posible las comunicaciones en una o entre varias implementaciones de control de procesos.
3. ANSI/ISA88.00.03 – 2003. Parte 3: Definen los modelos y representaciones generales y específicas de las recetas.
4. ANSI/ISA88.04 – 2006. Parte 4: Define detalladamente la arquitectura de los históricos de producción. Las implementaciones basadas en esta parte del estándar permitirán la recuperación y análisis de los reportes de producción.
5. ANSI/ISA88.05. Parte 5. Se encuentran en desarrollo. Definirá la implementación de los modelos y terminología para los equipos modulares de control.

Para el desarrollo de este proyecto, se hará uso de la primera parte del estándar, la cual será aplicada a un proceso batch caso de estudio.

1.4. Estándar ISA-88 Parte 1: Definición de modelos y terminología

El objetivo de esta primera parte del estándar se fundamenta en el establecimiento de los modelos del proceso productivo, partiendo de una proposición básica que establece la separación entre el conocimiento que se tiene del producto (*Récipes*) y la capacidad de los equipos en planta. Es decir, establecer el desarrollo de las recetas sin un conocimiento detallado de los sistemas de control como se muestra en la figura 1.4. Los modelos de la primera parte corresponden al Modelo Físico, el Modelo de control de Procedimientos y el

Modelo de Proceso, en base a los cuales se hace la configuración del sistema administrador batch, el cual se enlaza con el sistema de control de equipos, para así coordinar las tareas (*fases*) de producción, las cuales para este proyecto serán diseñadas a partir de redes de Petri y traducidas a lenguaje para PLC desde el software desarrollado.

El uso de los tres modelos antes mencionados, es un paso necesario para obtener con mayor claridad los elementos más básicos y elementales del proceso, los cuales se definen como: *capacidades o fases de proceso*. Este concepto se deriva del modelo de control de procedimiento y se profundiza con mayor claridad en el capítulo 2.

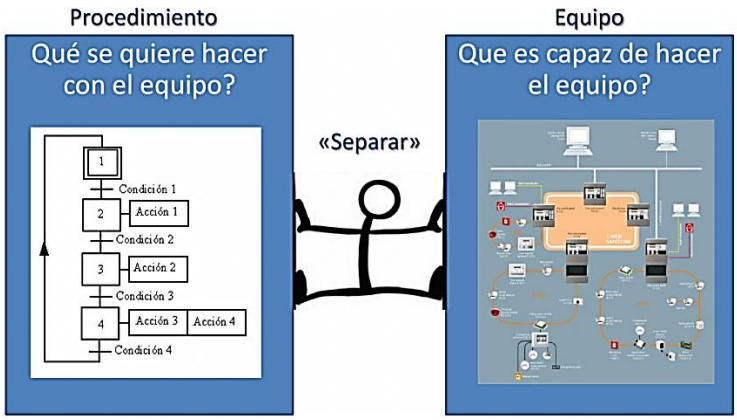


Figura 1.4 Recetas y Equipos. *Fuente:* [11]

A continuación se describen brevemente los modelos de la parte 1 de ISA–88:

1.4.1. Modelo Físico

La implementación del Modelo Físico en un proceso productivo implica la agrupación de todos los recursos físicos de la empresa de forma jerarquizada de acuerdo a la complejidad de las operaciones que cada uno de estos activos pueda realizar en las etapas de dicho proceso. El estándar ISA-88 define la jerarquía de los activos físicos en relación a siete niveles como lo muestra la figura 1.5, de los cuales los tres niveles superiores (Empresa, Sitio y Área) no están dentro del alcance del estándar ISA-88, siendo que son los encargados de soportar los procesos de negocio. En consecuencia la jerarquización de los activos físicos de la empresa se realiza en referencia a Células, Unidades, Módulos de equipo y módulos de control.

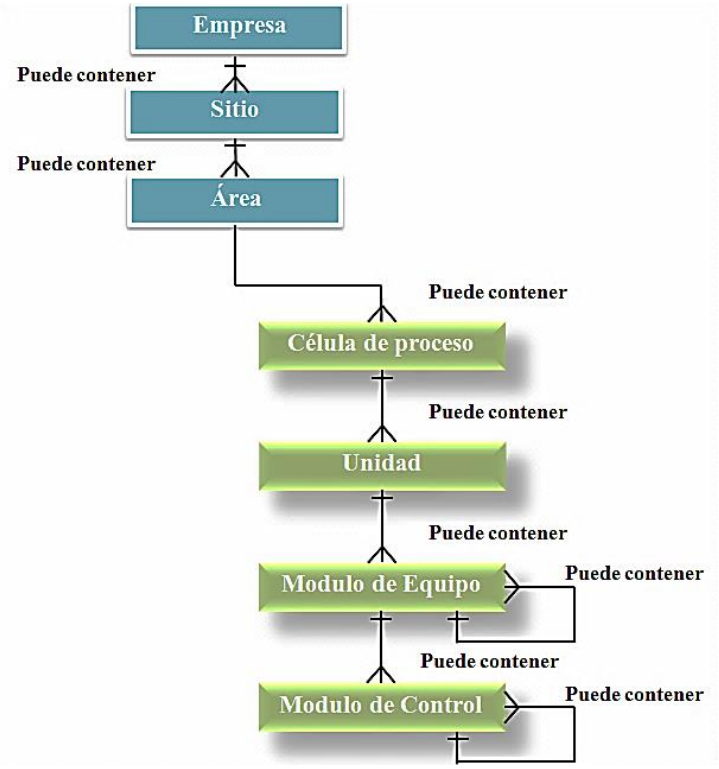


Figura 1.5 Estructura jerárquica del modelo Físico. ***Fuente:*** [12]

1.4.2. Modelo de control de procedimientos

La ejecución de todo proceso productivo consta de una serie de etapas que son efectuadas por los equipos requeridos para llevar a cabo dicho proceso. Estas etapas de acciones de ejecución orientadas al proceso para la obtención de un producto, pueden ser especificadas a partir del modelo de control de procedimiento, el cual define la estructura y el comportamiento dinámico del sistema a través de una secuencia ordenada.

La composición del modelo de control de procedimientos se define a partir elementos procedimentales dispuestos en una estructura jerárquica que agrupa las acciones dependiendo del grado de complejidad, como se muestra en la figura 1.6, siendo [12]:

- Procedimientos: Correspondiente al nivel de mayor jerarquía, definiendo la mayor acción de proceso para la obtención de un Batch. Este nivel se define a partir de un conjunto ordenado de procedimientos de unidad.
- Procedimientos de unidad: Consta de un conjunto ordenado de operaciones.
- Operación: Es la formación de un conjunto ordenado de fases, logrando una secuencia de procesamiento, llevando el material a cambios de estado.

Fases: Es el elemento más básico del Modelo de Control de Procedimiento Orientado a la ejecución de acciones definidas como control básico.

Figura 1.6 Estructura jerárquica del Modelo de Control de Procedimiento. ***Fuente:*** [12]

1.4.3. Modelo de proceso

El modelo de proceso se obtiene de relacionar el modelo de control de procedimiento y el modelo físico. La jerarquía del modelo se establece a partir de cuatro etapas que son: Proceso, Etapa de proceso, Operación de proceso y Acción de proceso como se muestra en la figura 1.7.

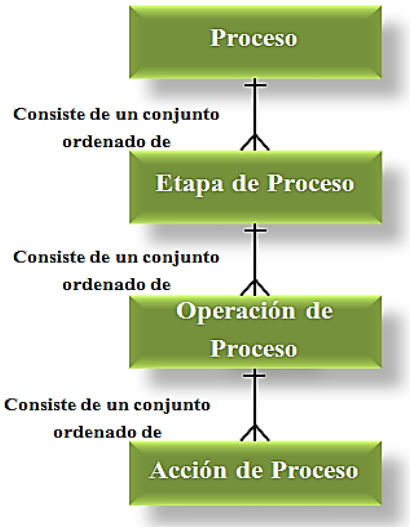


Figura 1.7 Estructura jerárquica del Modelo de Proceso. ***Fuente*****:** [12]

1.5. Estados

Para el diseño de los procesos es importante identificar claramente los estados por los que pueden pasar los elementos que lo conforman. En este sentido, y habiendo hecho mención de la modularidad que brinda el uso del estándar ISA-88, el software desarrollado permite el diseño de estados para elementos de proceso en base a la siguiente definición:

Los elementos procedimentales (*Cada uno de los niveles jerárquicos del modelo de control de procedimiento*) y las entidades de equipo (*Estas entidades se forman de la combinación del control de equipo y los equipos físicos, de esta combinación resultan cuatro entidades de equipo, conformando así el modelo físico*) [7] pueden tener varios estados. Un estado es la condición en la que se encuentra un elemento procedimental o entidad de equipo, en relación con los cambios que influyen en su condición. En el caso de un equipo, por ejemplo una válvula, el estado puede ser "porcentaje de apertura", y en el caso de un elemento procedimental, podría ser "running" o "holding" [12],[13]. La cantidad de estados y comandos, y sus nombres varía para entidades de equipo y para elementos procedimentales

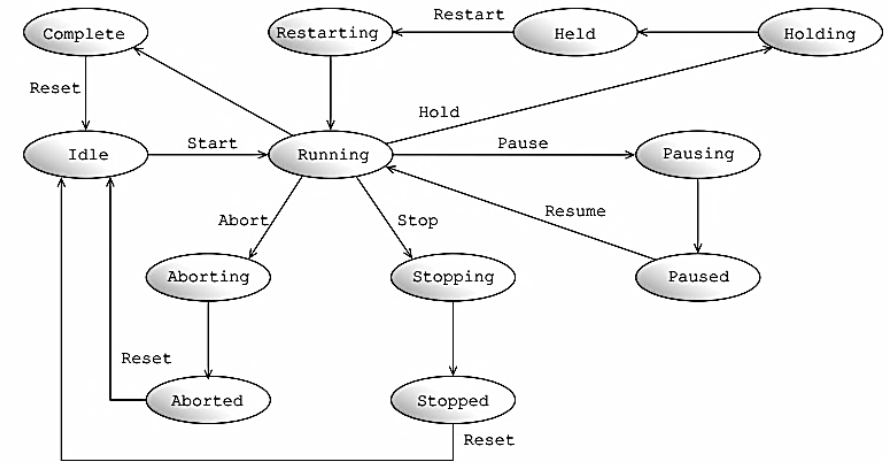


Figura 1.8 Diagrama de estados para un elemento procedimental descritos en ISA-88. *Fuente*: [12]

En la figura 1.8 se muestra un diagrama típico para elementos procedimentales cuyos estados son: *idle, running, complete, pausing, paused, holding, held, restarting, stopping, stopped, y aborted*. Para las entidades de equipo algunos estados podrían ser: *on, off, closed, open, failed, travelling, tripped y available*. Ejemplos de comandos aplicables a elementos procedimentales son *start, hold, pause, stop, y abort* [13].

1.6. Redes de Petri y la teoría de control supervisorio

Como característica de los procesos batch, se estableció que estos consisten de una secuencia de etapas que deben ser realizadas en un orden definido. En general existirán etapas que operan de manera discontinua y otras en forma semicontinua. La ejecución del proceso está regida por tareas programadas, definiéndose para cada una de ellas, eventos de inicio y finalización, y el orden de su ejecución. Precisamente la descripción más sencilla de un proceso batch consiste en una "receta" que describe las tareas a realizar, y especifica los tiempos de procesamiento o las condiciones que dan por terminada cada tarea. Cada inicio o finalización de una tarea está claramente asociada a un momento en el tiempo y el concepto de evento surge naturalmente [14].

Los procesos batch son esencialmente discontinuos [4], por ende las dinámicas predominantes se pueden determinar como de *Sistemas Dinámicos de Eventos Discretos* (DES), los cuales están limitados por conjuntos finitos de estados y eventos, en tiempo igualmente finito.

Los eventos representan la ocurrencia de hechos en el entorno del sistema que hacen que se produzcan cambios en la condición de sus elementos. Los eventos se producen en instantes particulares de tiempo, ver figura 1.9. Los estados por su parte, representan la condición de un elemento en relación a los eventos que influyeron en su condición, y pueden representarse mediante un conjunto discreto de valores. En la figura 1.8, los estados se denotan con la letra *x,* y se representan en el eje de las abscisas. La ocurrencia de un evento produce un cambio de estado del sistema.

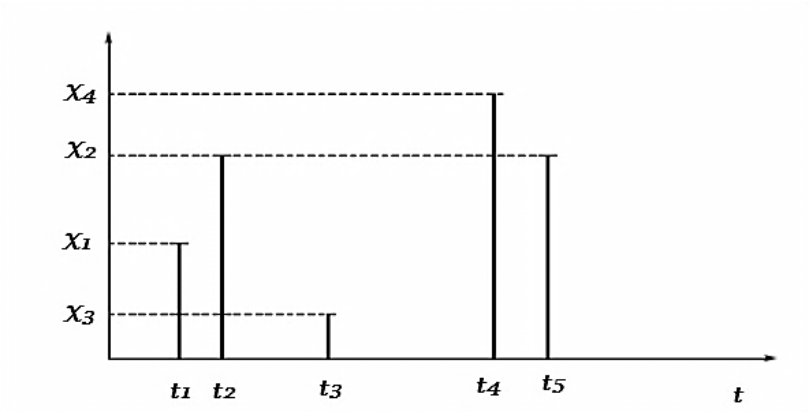


Figura 1.9 Representación de una trayectoria de eventos. *Fuente:* [15]

El comportamiento de una planta o proceso desde los sistemas discretos, se abstrae como una red de eventos relacionados que exhiben una dinámica no controlada. Este comportamiento exige la existencia de una entidad que lo controle y plantea la necesidad de

abordar el estudio de los sistemas de eventos discretos desde la teoría de control. El comportamiento simbólico derivado a partir de la generación de eventos, posibilita el uso de la teoría de lenguajes para su estudio [16].

Para el establecimiento del comportamiento del sistema, existen dos bases muy utilizadas, que han dado grandes resultados en el modelado de procesos: los Autómatas de estado finito (FSA) y las redes de Petri (PN). Ambos lenguajes representan el sistema a partir de secuencias estado-transición. De igual forma proveen la capacidad de evaluar el comportamiento de cada uno de los modelos SED, y su interacción, mediante técnicas de análisis que determinan el desempeño del modelo global del sistema.

En este trabajo se hará uso de las redes de Petri ya que éstas han demostrado ser más útiles para el manejo de la complejidad ya que permiten representaciones más compactas y la descripción natural de fenómenos tales como la concurrencia o la sincronización, entre otros [16].

1.6.1. Redes de Petri

Una red de Petri es un par $(\boldsymbol{\delta}, \boldsymbol{\mu})$ compuesto por un grafo dirigido bipartito $\boldsymbol{\delta} = (\boldsymbol{E}, \boldsymbol{V})$ y una marca inicial $\boldsymbol{\mu}$. [17]. El conjunto de nodos $\boldsymbol{V}$ está dividido en dos subconjuntos disjuntos $\boldsymbol{P}$ y $\boldsymbol{T}$. Los elementos $\boldsymbol{P}$ reciben el nombre de lugares, y son dibujados como círculos mientras que los elementos $\boldsymbol{T}$ reciben el nombre de transiciones y son dibujados como barras o rectángulos. Los lugares serán representados por: $\boldsymbol{P_i}, \boldsymbol{i} = \mathbf{1}, \dots, |\boldsymbol{P}|$ y las transiciones por: $\boldsymbol{T_j}, \boldsymbol{j} = \mathbf{1}, \dots, |\boldsymbol{T}|$. Los arcos $\boldsymbol{E}$ van de los lugares a las transiciones o bien de las transiciones a los lugares, ver figura 1.10, a cada arco se le asigna un peso el cual está dado por un numero entero, cuando este no se indica se asume que el peso es 1.

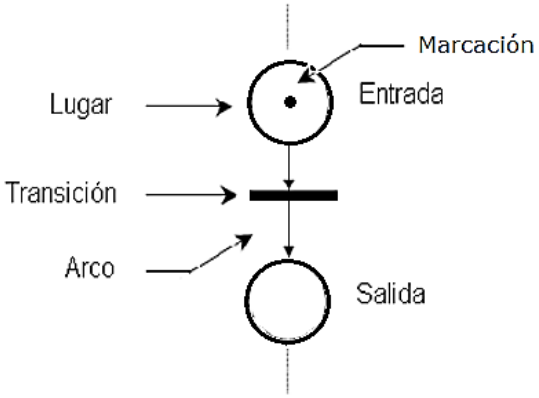


Figura 1.10 Elementos de una Red de Petri. ***Fuente:*** [18]

A los lugares se les es asignado un número entero no negativo de marcas, las cuales se representan como puntos negros dentro de cada lugar $\boldsymbol{P}$. Esta asignación se conoce como vector columna de marcación $\boldsymbol{\mu}$. La marca inicial asigna un valor $\boldsymbol{\mu_0}$ a cada lugar $\boldsymbol{P_i}$.

Un lugar $\boldsymbol{P_i}$ es anterior a una transición $\boldsymbol{T_j}$, si hay un arco que va de $\boldsymbol{P_i}$ a $\boldsymbol{T_j}$. Análogamente, se dirá que un lugar $\boldsymbol{P_i}$ es posterior a la transición $\boldsymbol{T_j}$ si hay un arco que va de $\boldsymbol{T_j}$ a $\boldsymbol{P_i}$ [18].

Los lugares representan los estados del sistema modelado, y los arcos representan la dirección de la evolución de la red. Las transiciones por su parte, están asociadas a los eventos del sistema, y su activación dependerá de que cada lugar anterior posea el número de marcas mayores o iguales al peso del arco, en este caso se dirá que la transición está habilitada. El disparo de la transición provoca el traspaso de marcas del lugar anterior al posterior igual al peso del arco.

Dado que no es el interés en este trabajo abarcar los amplios conceptos a cerca de las propiedades de las redes de Petri, el lector interesado en profundizar sobre esta técnica de modelado puede consultar en [18].

Para el cumplimiento de las especificaciones de control del modelo de RDP es importante considerar conceptos que al ser implementados permitan modificar el comportamiento del modelo del proceso que exhibe un comportamiento no deseado. El modelo de la planta sin ningún control, el cual se denomina de lazo abierto, puede cambiar de estado sin que haya alguna condición dentro de la secuencia que determine una evolución segura para alcanzar un conjunto de especificaciones dado. En este sentido en [19] se plantea el uso de una teoría general para la síntesis de controladores, llamados supervisores, en sistemas de eventos discretos. El supervisor es un autómata que restringe el comportamiento del sistema mediante la habilitación y deshabilitación de eventos, afectando la secuencia actual de eventos y la trayectoria de los estados. Uno de los métodos más usados para el diseño de supervisores, es el método de invariantes de lugar propuesto por Moody y Antsaklis el cual se analiza con detalle en [20, 21].

1.7. Software para la edición de redes de Petri

Para el diseño de la lógica de control en redes de Petri, se cuenta con la herramienta PIPE la cual es de código libre y cuenta con opciones que permiten realizar el análisis y simulación de la RdP obtenida. De igual forma se cuenta con la herramienta de modelado llamada Snoopy, que permite implementar arcos de activación, y arcos de lectura, con lo cual se logra simplificar el modelo en RdP. Estas dos herramientas se enlazan con el software propuesto cada vez que se requiera diseñar la lógica de un proceso.

Dado que la mayoría de los procesos exhiben un comportamiento no deseado en lazo abierto, se hace necesaria la implementación de supervisores como se mencionó anteriormente. Para esto se puede hacer uso de la herramienta CRP [22], diseñada para la implementación supervisores en base a la metodología de invariantes de lugar, debido a que desde ella se pueden editar los archivos con los modelos de RdP generados desde el software PIPE.

1.8. Implementación de la lógica en redes de Petri en un controlador lógico programable

Para la implementación de los modelos de red de Petri en controladores lógicos programables (PLC) es muy común utilizar gráficos de función secuencial (SFC), sin embargo el estándar IEC 61131-3 propone cinco lenguajes de programación incluido SFC.

La Comisión Electrotécnica Internacional (IEC) ha desarrollado una serie de especificaciones para controladores programables. Estas especificaciones están destinadas a promover la unificación internacional de equipos y lenguajes de programación para uso en la industria de controladores [23].

La norma IEC 61131 está dividida en cinco partes:

- Parte 1: Información general
- Parte 2: Prueba de equipos y requisitos
- Parte 3: Lenguajes de programación
- Parte 4: Pautas para el usuario
- Parte 5: Especificación del servicio de mensajes

La parte 3 de la norma define los cinco lenguajes de programación (*Lista de instrucciones, Texto estructurado, Diagrama de lógica de escalera, Diagrama de función secuencial y diagramas de bloques de funciones*) como opcionales. En este trabajo se hará uso de lenguajes de programación de lógica en escalera como traducción de la lógica diseñada en redes de Petri.

Para la implementación de la lógica traducida de RdP a lenguaje para PLC, se hará uso del *sistema de control* RsLogix5000 [23] dado que es con el cual se cuenta en el programa de Ingeniería en Automática Industrial de la universidad del Cauca. Este software es compatible con las opciones de lenguajes de texto estructurado, diagrama de lógica de escalera, diagrama de funciones secuenciales, y diagrama de bloques de funciones. El software RsLogix5000 también utiliza un sistema que le permite exportar e importar archivos en formato ASCII basado en lenguaje de texto estructurado, de igual forma, RsLogix5000 puede ejecutarse dando cumplimiento a los modelos de ISA-88, para lo cual

hace uso de una interfaz que maneja el secuenciamiento de la lógica de estados de fase (PLI) basada en el diagrama de la figura 1.8.

El proceso para diseñar la lógica en RdP desde el software propuesto requiere la lectura de un archivo con formato L5K exportado desde el sistema de control RsLogix5000, el cual contendrá las fases del proceso, pero sin su lógica de control. Esto a su vez requiere previamente haber configurado la plataforma de administración de procesos batch FactoryTalk Batch. Finalmente, una vez se tenga la lógica diseñada en RdP, el software propuesto ejecuta su algoritmo de traducción, sobrescribiendo la lógica en el archivo L5K antes mencionado, finalizando con la importación de dicho archivo desde el sistema RsLogix5000. Esta idea se ilustra en la figura 1.11.

Teniendo ya una visión general de lo que puede hacer el software propuesto, es importante considerar el vacío que existe en cuanto al cómo se debe programar la lógica de fase a partir del modelo de la figura 1.8, para la cual no existe un procedimiento definido, quedando finalmente sujeta a la experiencia del programador.

En base a esta necesidad, en el siguiente capítulo se abordan los conceptos más relevantes relacionados a la lógica de fase, a partir de los cuales se propone un procedimiento que define el **cómo** implementar dicha lógica.

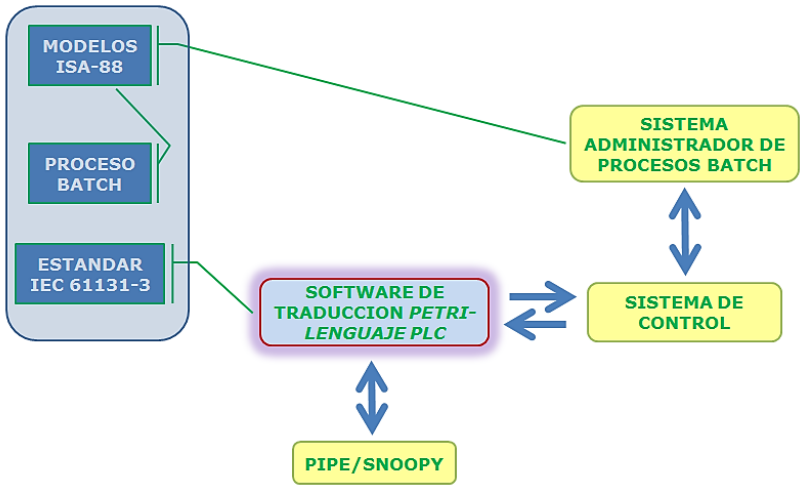


Figura 1.11 Diagrama de interacción con software y conceptos de la herramienta de traducción de RdP a lenguaje para PLCs. *Fuente: Propia*

2. Capítulo II: Especificación para el modelado y diseño de fases de equipo con redes de Petri

La definición del control del equipo es quizá la parte que más demanda complejidad a la hora de una especificación funcional. Aunque el estándar ISA-88 brinda un conjunto de pautas y normativas para un desarrollo rápido de las recetas de lotes reduciendo el esfuerzo de ingeniería, ahorrando tiempo y aportando flexibilidad, los detalles de su programación quedan ocultos. De acuerdo a esto, los objetivos de este capítulo están enfocados en definir y analizar los requerimientos de forma genérica de los componentes fundamentales requeridos para la implementación del control del equipo, y así proponer un procedimiento para un diseño detallado que saque provecho del formalismo de red de Petri y la teoría de control supervisorio.

2.1. Control de equipos según ISA-88

El estándar ISA 88 define el modelo de receta de control que se deriva originalmente de la receta maestra, la cual ha sido complementada y/o modificada con la planificación, funcionamiento e información de los equipos (una receta de control puede ser vista como una instanciación de una receta maestra), pero este modelo en si no contiene la información suficiente para operar una celda de proceso [24]. En este sentido se hace necesario relacionarlo en algún nivel con el control de equipos de proceso, es decir asociar las acciones de control con los objetos de equipamiento instanciados. La receta de control, y el control de equipos, pueden tener procedimientos, procedimientos de unidad, operaciones y fases. La figura 2.1 muestra que el control de los equipos de proceso no hace parte del récipe.

Al contar con modelos separados, se evita el tener que colocar los procedimientos, procedimientos de unidad, operaciones y las múltiples fases dentro del control del equipo, que implicaría la programación de todos estos elementos procedimentales en un PLC, lo cual representaría una gran dificultad en el momento de desarrollar la lógica requerida, y además de esto, tener en cuenta la coordinación o el manejo del orden de ejecución. Por tanto, ISA 88 sugiere que el sistema administrador batch y el sistema de control de equipos manejen los elementos de la receta de control y los elementos de control de equipo respectivamente, de manera separada, debiendo existir una relación en base a los modelos en algún punto de la jerarquía.

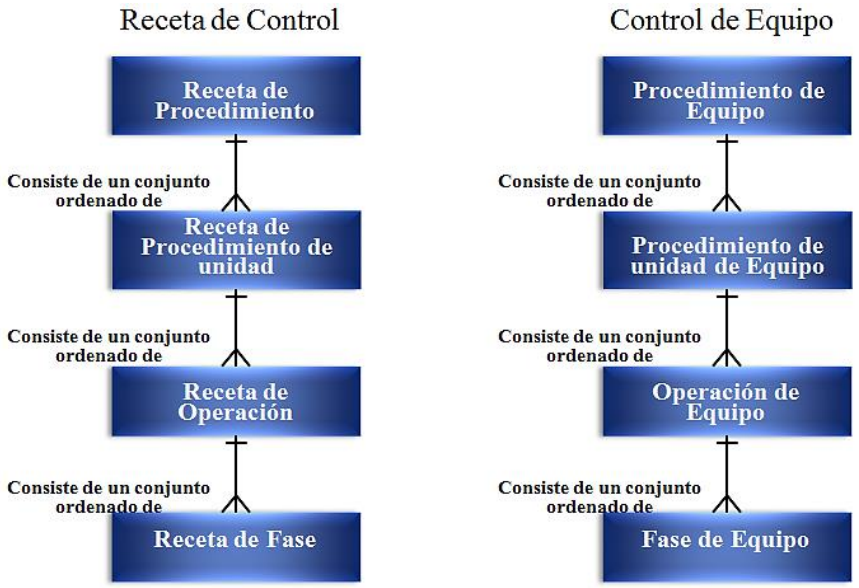


Figura 2.1 Procedimientos de Receta de control y Control de Equipo. ***Fuente:*** [12]

En cuanto a esto el estándar no dictamina el nivel en el que deba ocurrir un enlace de los modelos, de modo que un procedimiento de control de receta y una fase de equipo siempre deben existir, como se ve en la figura 2.2.

La mayor flexibilidad que se puede obtener en el momento de enlazar los modelos, se obtiene al nivel de fase, ver figura 2.3. Esto permite tener un mayor control de las capacidades que fueron diseñadas en el modelo de Control de equipo [25]. En el modelo de Receta de control al nivel de fase, se hace una referencia a la ***Fase de equipo*** en el modelo de Control de equipo, el cual se ejecuta en un sistema de control para realizar una función durante la ejecución de un batch, siendo este el nivel de enlace más común. Por ejemplo, una receta de fase puede referenciar a una fase de equipo llamada *Agitar*, la cual puede ser implementada en cualquier número de unidades.

La descripción de las fases de equipo tiene un nivel de importancia muy alto debido a que describen cómo deberá desempeñarse el equipo físicamente.

Las fases de equipo de una unidad, son el nivel más bajo del equipo que es visible en una receta. Mientras la receta de control únicamente específica bajo qué condiciones será ejecutada la fase, las fases de equipo se encargan de ejecutar las acciones que se requieren, y podrían ser ejecutadas de manera diferente en equipos diferentes.

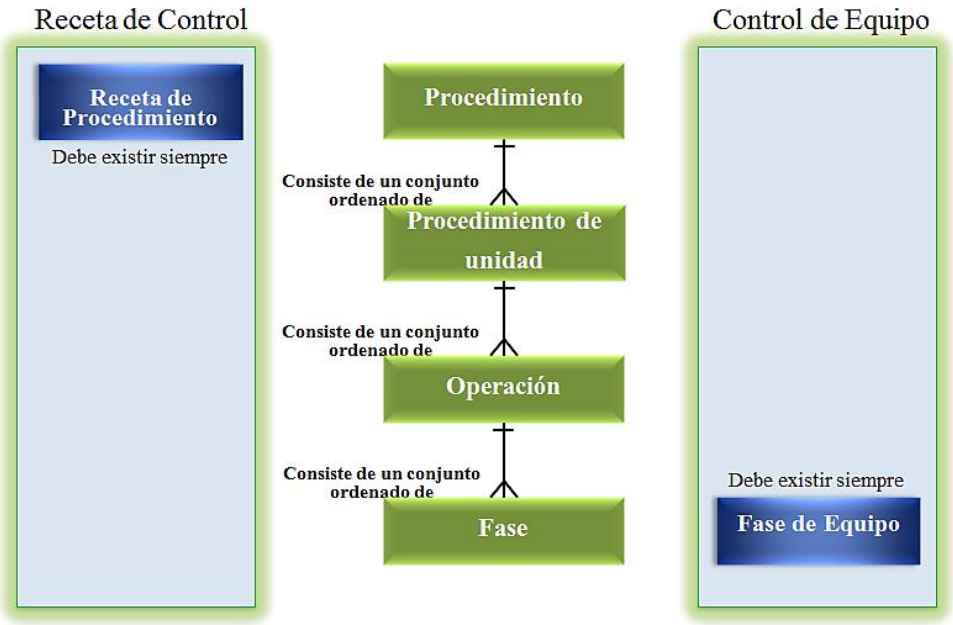


Figura 2.2 Modelo de control de procedimiento enlazado a los modelos de Receta de control y Control de equipo. *Fuente:* [12]

La fase de equipo está a cargo siempre del control del equipo. La relación entre el modelo de Receta de control y Control de equipo se basa en que el primero puede solamente "solicitar" el equipo para ejecutar la fase. El equipo decide si es seguro iniciar la acción, realizar el control asociado con la acción, y decide cuando la acción está completa. El equipo está siempre a cargo y tiene la responsabilidad de finalización segura [26]. Las fases de equipo generalmente son independientes de cualquier producto, y definen lo que es capaz de hacer el equipo.

Dentro de las fases de equipo, es importante considerar los siguientes puntos:

- Lógica de fase.
- Modos y estados.
- Manejo de excepciones.

- Asignación y arbitraje.
- Sincronización entre unidades.
- Colección de datos

Para el desarrollo de este trabajo se abordarán los tres primeros puntos.

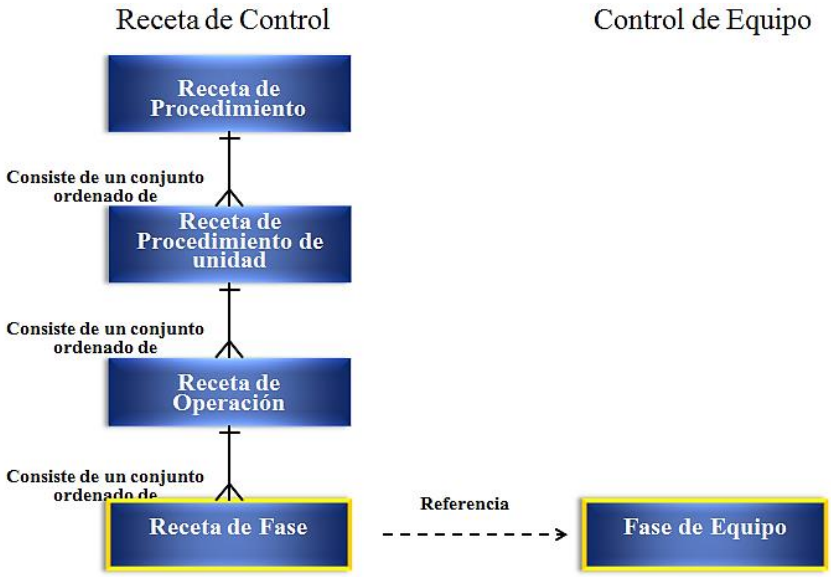


Figura 2.3 Receta de control y Control de equipo enlazados al nivel de fase. ***Fuente:*** [12]

2.2. Diseño de la lógica fase

Una fase es una secuencia de etapas y transiciones que causan una o más acciones orientadas a un proceso. Cada conjunto de acciones en una etapa, conforman las instrucciones necesarias para que los equipos físicos puedan operar, tal como iniciar una bomba o abrir una válvula, etc. El conjunto lógico de pasos que configuran una fase son específicos para sus equipos, y no forman parte de la receta [27].

Como se mencionó anteriormente, debe existir un enlace para vincular la receta de control, con el control del equipo, siendo al nivel de fase el más común. Generalmente las fases de equipo son ejecutadas en un PLC o en un DCS. Enlazar los modelos al nivel de fase, significa que el sistema administrador batch contiene procedimientos, procedimientos de unidad, operaciones, y fases dentro del Control de receta, y el sistema de Control de equipo contiene solamente la lógica de fase [28], ver figura 2.4.

La lógica de fase se compone de etapas de control que a su vez están compuestas de acciones de control. Para el control directo de los equipos, que permite ejecutar tareas

orientadas al proceso, la lógica de la fase escrita y definida por el usuario debería haber sido implementada como un conjunto de secuencias bien definidas, las cuales deberían incluir la lógica del funcionamiento normal de la fase, así como la lógica para una posible eventualidad. Esta lógica también debería contemplar la detección de fallos.

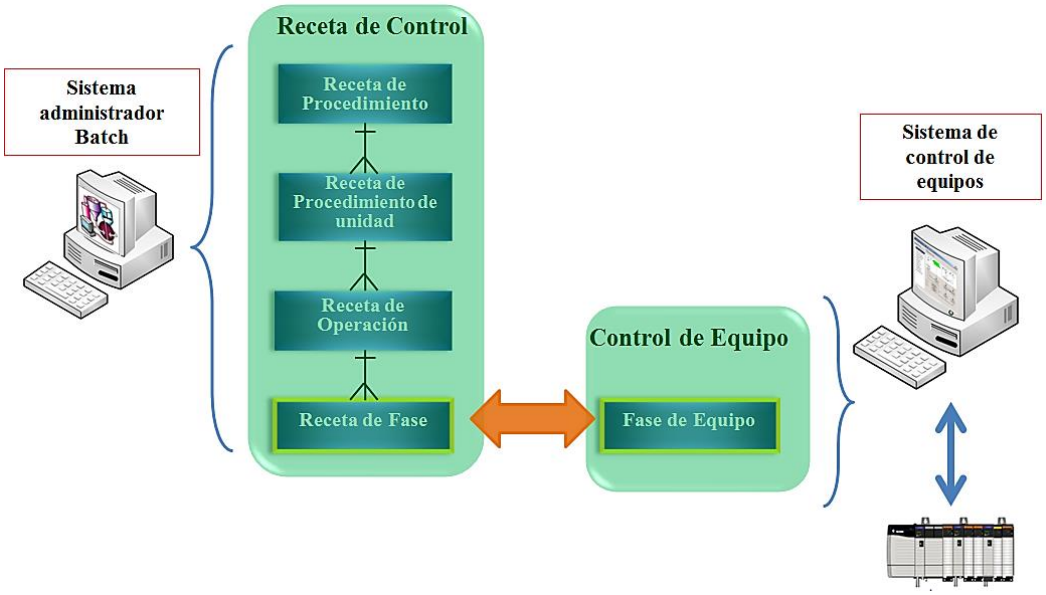


Figura 2.4 Enlace. ***Fuente:***[27]

Algunos trabajos en [29, 30], describen y hacen aproximaciones en cuanto a la estructura que define una fase, pero estas definiciones hacen referencia únicamente a las etapas requeridas para su ejecución y no al diseño detallado, el cual debe incluir los pasos para definir cada uno de los estados por los que puede llegar a pasar una fase, así como cualquier condición de excepción. Lo más importante antes de diseñar la lógica de fase, es identificar la unidad en la cual las fases controlan los equipos que la conforman. Por ejemplo, una unidad de mezcla, ver figura 2.5, está conformada por varias fases que operan los equipos y módulos de control asociados con dicha unidad, estas fases se determinan en función de las capacidades de proceso que se pueden ejecutar en la unidad, es decir, agitación, control de temperatura, adición de materias primas, descarga de producto, etc. de tal forma que para realizar el control de temperatura por medio de la chaqueta de enfriamiento, se deberá crear una fase que la controle.

El diseño de las fases debe enfocarse en la consecución de modelos genéricos para su uso en múltiples recetas de control. De esta manera, para la obtención de productos diferentes debido a cambios en la producción, es menos probable que la lógica de fase requiera cambios sustanciales, proveyendo facilidad para su documentación e implementación, y así poder lograr un diseño flexible y modular de fases que puedan correr en múltiples unidades.

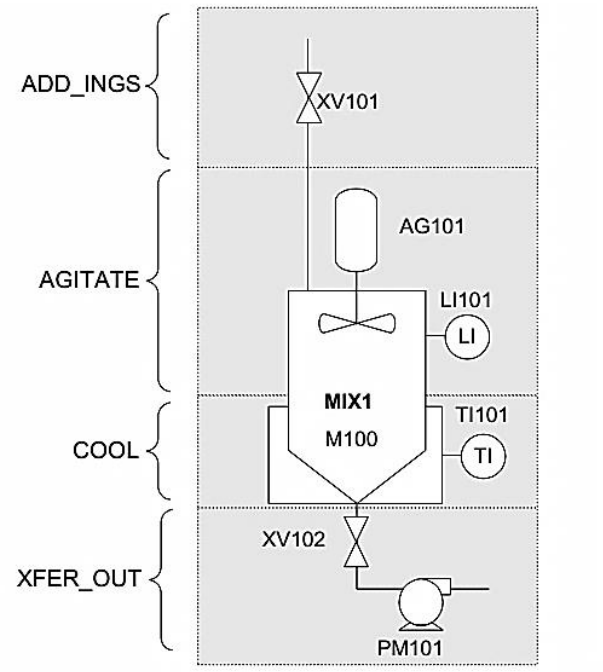


Figura 2.5 Fases que conforman la unidad de agitación. ***Fuente:*** [30]

Una vez se han identificado correctamente todas las capacidades de proceso las cuales se asocian directamente a las fases en el modelo de control de procedimiento, se debe definir el **cómo** ejecutar dichas capacidades siguiendo un orden lógico, ya que es en este punto donde se establecen las acciones o comportamientos que se requieren por parte de los equipos para el correcto funcionamiento del proceso definido por el conjunto de fases. En este sentido es importante considerar las etapas individuales que conforman una fase.

Como en todo proceso de modelado es imprescindible antes de iniciar, contar con la información suficiente para solucionar el problema correctamente. Crear las mejores soluciones requiere seguir un proceso detallado para analizar los requerimientos del proyecto, es decir, definir qué es lo que se supone debe hacer el sistema de producción, y desarrollar un diseño que cumpla con esos requerimientos.

Equipos que forman parte del proceso pueden ser [31]: **(a)** sensores, los cuales se encargan de transmitir la información a los equipos de control del estado en el que se encuentra el proceso, tales como sensores de presión, temperatura, flujo, nivel, etc. **(b)** equipos de proceso como lo son tanques, reactores, intercambiadores. **(c)** actuadores, los cuales ejecutan las acciones de control y como consecuencia de ello provocan un cambio de estado en el proceso, tales como motores y válvulas. Del mismo modo es importante tener en

cuenta todas las señales de entradas y salidas del controlador como parte de las especificaciones.

Para el modelado de los actuadores es importante tener en cuenta sus tipos, los cuales se diferencian entre ellos por el comportamiento de las entradas y las salidas. En [20] se hace esta descripción:

Tipo 1. Una entrada y una salida. El controlador envía un comando de salida y el actuador responde confirmando que la ha recibido a través de una entrada. Un ejemplo típico de este tipo de equipos son los motores.

Tipo 2. Dos entradas y una salida. Este tipo de actuadores tiene dos posiciones, cada entrada se activa para indicar que el actuador se encuentra en alguna de las dos posiciones. El controlador da un comando de salida para ir de una posición a otra y lo quita para regresar al estado inicial. Un ejemplo de estos actuadores son las válvulas con un solenoide de activación y reposición por resorte.

Tipo 3. Dos entradas y dos salidas. Son muy similares al anterior, con la diferencia que para ir de una posición a otra se activa una de las salidas y para regresar se activa la otra salida. Un ejemplo de estos tipos de actuadores son las válvulas solenoides.

Tipo 4. Una entrada y dos salidas. Estos actuadores tienen dos modos de operación, cada modo es activado por una de las salidas. La confirmación de operación de ambos modos se recibe por la misma entrada. Un ejemplo de estos actuadores son los motores reversibles.

Además de fijar el comportamiento que exhibirán los equipos, se debe tener en cuenta una filosofía de operación, es decir especificar la forma en que deberán interactuar con otros equipos.

2.2.1. Propuesta de un método para modelado con RdP de la lógica de fase

Como se mencionó anteriormente, la lógica de fase consta de una serie de etapas que esta debe seguir para cumplir con una tarea orientada al proceso. Para esto el diseño será propuesto en base al diagrama de estados de la figura 1.8. En este sentido se propone el siguiente esquema el cual corresponde a la lógica del estado *Running*:

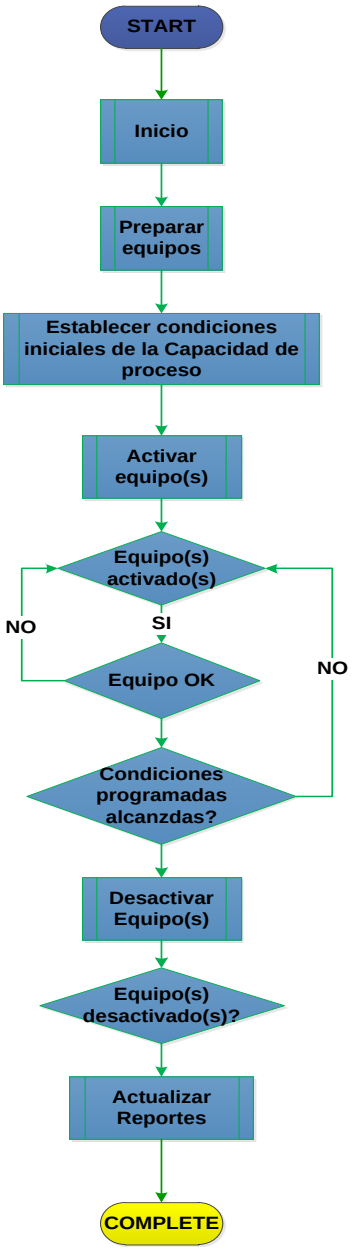


Figura 2.6 Esquema de diseño para la lógica del estado *Running*. *Fuente: Propia*

En la figura 2.6 se plantea un procedimiento que define de manera sencilla una fase. Amanera de ejemplo de aplicación considere una fase ficticia de ***ADICIONAR_AGUA***, a partir de las propiedades de modelado de las redes de Petri y la teoría de control supervisorio.

La fase para adicionar agua estará conformada por una válvula de dos vías que dirige el fluido hacia dos tanques, de los cuales podrá ser llenado solo uno a la vez. Cada tanque contará con dos sensores para indicar si el tanque está lleno o está vacío.

La primera etapa del esquema propuesto, corresponde a un comando de inicio el cual puede ser representado en una RdP simplemente como un lugar de entrada. Las dos etapas posteriores, **"*Preparar Equipos*" y "*Establecer condiciones iniciales de la capacidad de proceso*"**, corresponden al establecimiento del modelo de la planta en RdP, es decir a los tipos de actuadores que serán controlados por la fase, y la definición de las condiciones que limitarán el comportamiento de los equipos respectivamente. En este trabajo se propone el uso de la teoría de control supervisor con RdP para el diseño de esta última etapa.

- **Preparar Equipos (*Modelo de la planta de lazo abierto en RdP*)**

El modelo de la válvula se implementa como un actuador tipo 2 indicando cuál es el tanque que se está llenando, y para cada par de sensores en los tanques se hace la equivalencia a dos lugares y dos transiciones, tal como se muestra en la figura 2.7.

- **Establecer condiciones iniciales de la capacidad de proceso (*Modelo de lazo cerrado en RdP usando control supervisor*)**

El supervisor debe ser diseñado bajo las siguientes reglas:

Las reglas 1 - 4 se definirán como reglas de condición acción [20]:

1. Si el nivel del tanque 1 está en alto la válvula no debe adicionar agua para ese tanque:

$$u1 \rightarrow q4$$

2. Si el nivel del tanque 2 está en alto la válvula no debe adicionar agua para ese tanque:

$$u6 \rightarrow q7$$

3. Si la válvula está llenando el tanque 1 y regresa a su estado P3 antes de que el tanque 1 alcance su estado P0, el tanque no puede ser llenado.

$$u2 \rightarrow q0$$

4. Si la válvula está llenando el tanque 2 y regresa a su estado P3 antes de que el tanque 2 alcance su estado P5, el tanque no puede ser llenado.

$$u4 \rightarrow q6$$

5. Cuando cualquiera de los tanques haya sido llenado, el proceso habrá terminado para dicho tanque, con lo cual no podrá activarse ningún evento. Para el caso del tanque uno:

$$v1 + v5 - v4 \leq 0$$

La red de Petri agregando los lugares y arcos generados por las restricciones implementadas en el software **CRP** se puede observar en la figura 2.8.

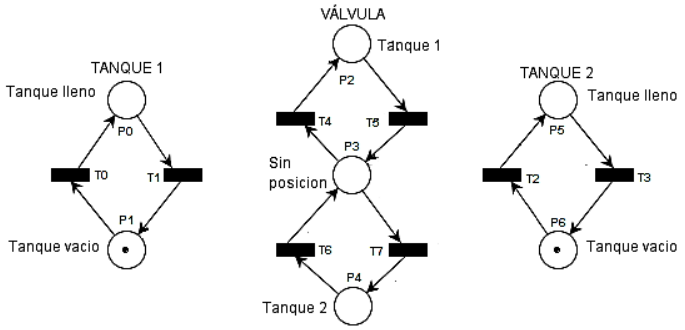


Figura 2.7 Modelo de la planta en lazo abierto. ***Fuente: Propia***

Continuando con el esquema de la figura 2.6, la etapa siguiente corresponde a la activación de los equipos, como se puede observar en la figura 2.9, donde la válvula esta activada y puede tomar posición para llenar cualquiera de los dos tanques inmediatamente después de haber sido disparada la transición T10, habilitada por el lugar de inicio P9.

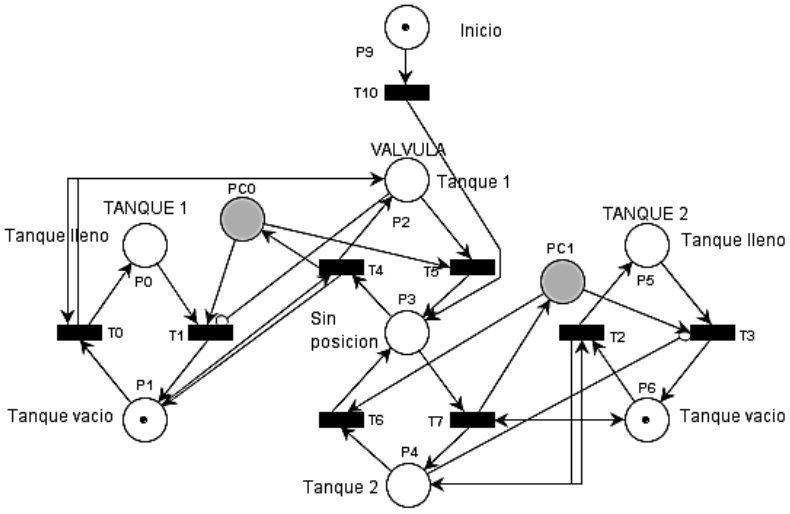


Figura 2.8 Modelo de la planta en lazo cerrado. ***Fuente: Propia***

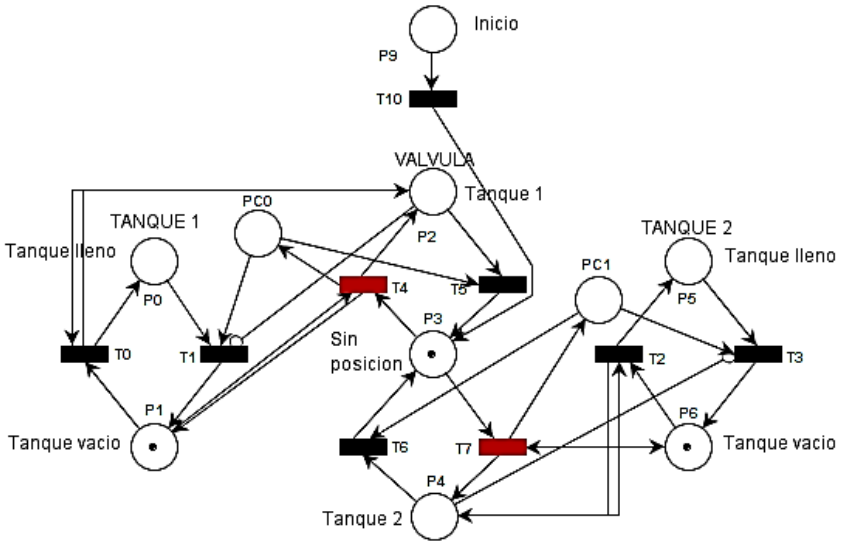


Figura 2.9 Transiciones T4 y T7 activadas del modelo de la válvula. ***Fuente: Propia***

Las etapas de decisión ***"Equipos activados"*** y ***"Equipos Ok"***, pueden ser comprobadas inicialmente en la figura 2.9, donde se pudo observar que la válvula efectivamente pudo ser activada. Luego en la figura 2.10 se observa que al ser disparada la transición T4 del modelo de la válvula, se activa inmediatamente la transición T0 del modelo del tanque 1. Lo mismo puede ocurrir si hubiera sido disparada la transición T7 del modelo de la válvula, con lo cual se habría activado la transición T2, comprobando de esta manera las dos primeras etapas de decisión mencionadas inicialmente.

Cuando se dispara la transición T0, la marca avanza hacia el lugar P0 indicando que el tanque 1 ha sido llenado, con lo cual la válvula debe regresar a su posición intermedia disparando la transición T5 y colocando una marca en el lugar P3, ver figura 2.11 Con esto se puede comprobar que efectivamente una vez el tanque 1 ha sido llenado, la válvula queda deshabilitada para ese mismo tanque. Dado que el modelo es simétrico lo mismo puede ocurrir para el modelo del tanque 2, verificando de esta forma las reglas mencionadas anteriormente, comprobando de manera afirmativa la etapa ***"Condiciones programadas alcanzadas?"***.

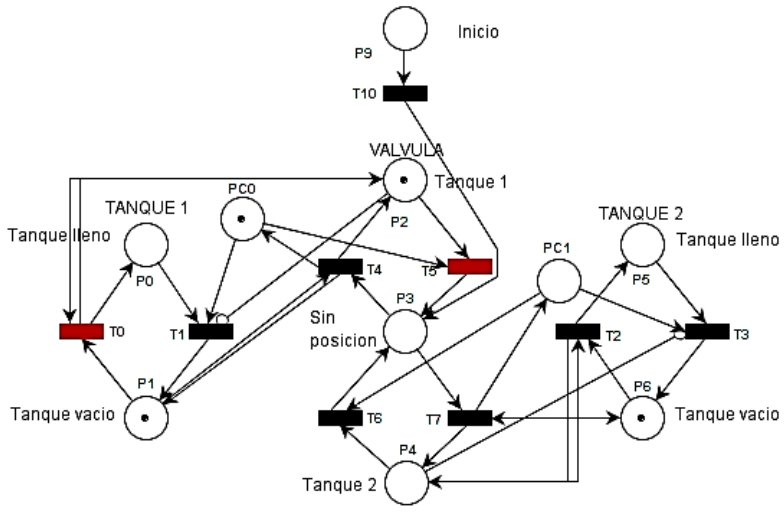


Figura 2.10 Transición T0 activada después de haber sido disparada la transición T4. ***Fuente: Propia***

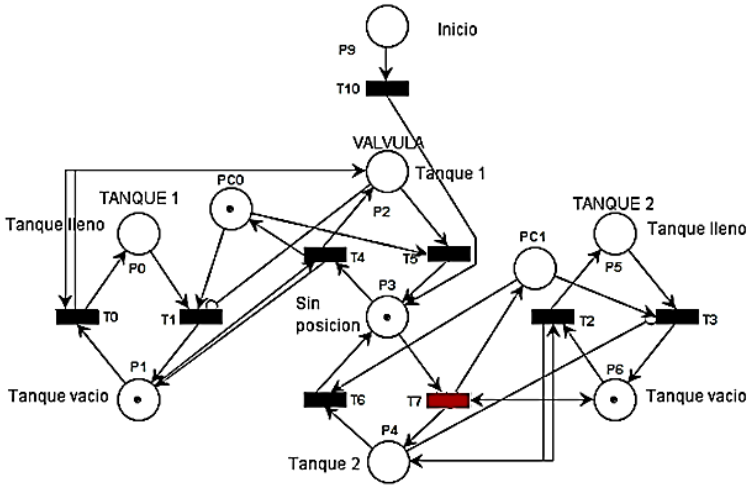


Figura 2.11 Válvula habilitada para el tanque 2. ***Fuente: Propia***

Finalmente, cuando los equipos hayan cumplido con su objetivo deberán ser desactivados, para luego actualizar los reportes completando así las tres últimas etapas del esquema de la figura 2.6 para la lógica de la fase, adicionando para esto los lugares y las transiciones P12, P13, P14 y T9, T11, y T12 respectivamente, ver figura 2.12. Una observación importante a cerca de este modelo es que la RdP no es acotada y además presenta un bloqueo. Esto es debido a que al traducir la lógica a lenguaje ladder, una vez se ejecute, y en una etapa posterior requiera nuevamente ser ejecutada, el reinicio se da de forma automática, lo cual

es administrado por la plataforma de administración del proceso. Es decir que los modelos de RdP propuestos serán diseñados para que se bloqueen una vez hayan cumplido con su objetivo de proceso, no requiriendo una lógica adicional para su reinicio.

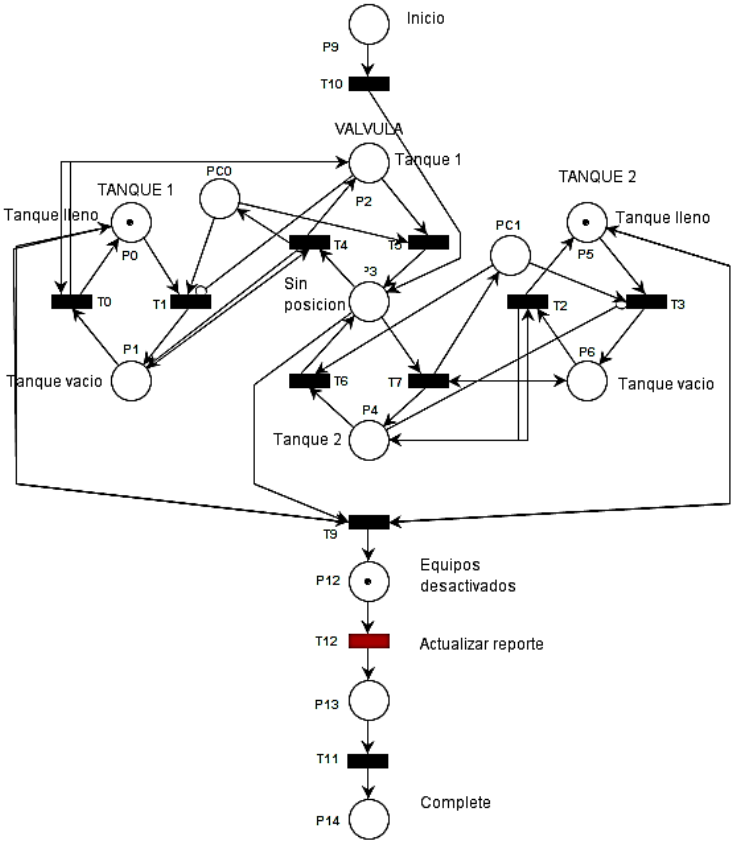


Figura 2.12 Lógica *Running* para la fase *ADICIONAR_AGUA. Fuente: Propia*

La anterior descripción define cómo debe operar una fase en condiciones normales sin abordar ninguna condición de mal funcionamiento. Se podría considerar lo que puede ocurrir si la válvula falla en el momento en el que debe adicionar agua. De acuerdo a esta situación y en base al estándar ISA-88 un comando HOLD debería ocurrir, comenzando la ejecución de una sección separada de código (La lógica *Holding*). El análisis de estas situaciones será abordado en el siguiente ítem.

2.2.2. Comandos y estados

La figura 1.8 muestra el diagrama de transición de estados que pueden ser soportados en la lógica de fase de equipo y la ruta que se debe seguir para alcanzar cualquiera de ellos. Para cada estado transitorio*: Running, Holding, Restarting, Stopping,* and *Aborting.* La lógica de fase deberá dividirse en cinco secciones diferentes. Para el control de la transición de

estados [30] para la fase, el sistema de control RsLogix5000 cuenta con una interfaz estandar PLI (Phase Logic Interface) basada en el diagrama de la figura 1.8, esto de forma gráfica se ubica entre el administrador batch y la fase de equipo, ver figura 2.13.

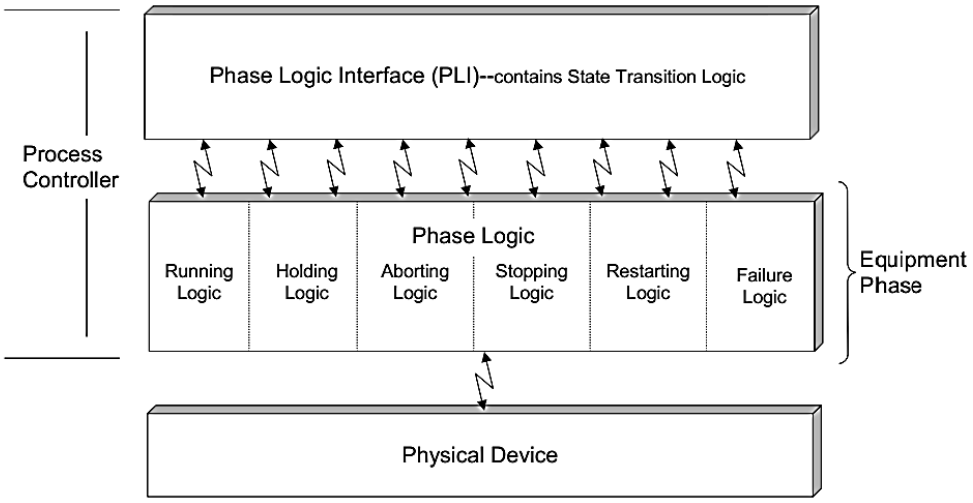


Figura 2.13 Interfaz de la lógica de fase. *Fuente:* [30]

En el ítem anterior, únicamente se habló de la lógica de funcionamiento normal o lógica de estado *Running* la cual se determinó a partir del modelo de la figura 2.6. La lógica de *Running* es utilizada únicamente cuando la producción del batch transcurre bajo condiciones normales, pero esto no ocurrirá siempre.

Las transiciones entre los estados de la figura 1.8 pueden ser causados solamente debido a los siguientes tres tipos de eventos [29]:

2.2.2.1. Comandos

La lógica de fase espera para responder a un comando legal dependiendo del estado en el que se encuentre, por ejemplo cuando la lógica de fase está en el estado *Running*, y un comando Hold es emitido, esto causará que la lógica de fase pase del estado Running al estado *Holding*. Los comandos definidos para cada lógica de fase son [32]:

- Start: Causa una transición del estado idle al running.
- Stop: Comanda la lógica de procedimiento para parar la ejecución y realizar una detención normal de las operaciones.
- Hold: Causa una transición del estado running a holding. Usualmente usada por un operador para pausar el procedimiento lógico o cuando se presenta un evento que interrumpe la operación de la fase.
- Restart: Causa una transición del estado held a restarting.

- Abort: Causa una transición de running, holding, held, restarting, stopping a aborting. Comanda el procedimiento lógico para detener la ejecución y realizar la detención anormal de las operaciones.
- Reset: Comanda el procedimiento lógico para reiniciar y preparar una nueva ejecución. Este comando es usualmente emitido por el sistema de ejecución de la receta y se envía automáticamente después de que el sistema haya determinado si el elemento procedimental ha sido completado.

2.2.2.2. Terminación de la lógica de fase de equipo

Los cambios de estado pueden ocurrir como resultado de la terminación de uno de los siguientes estados: Running, *Holding, Restarting, Stopping,* y *Aborting.*

En la secuencia cada estado espera pasar a otro estado. Los estados finales para la terminación de la secuencia son: *Completed, Held, Aborted, y Stopped.*

2.2.2.3. Respuesta a fallos

Responder ante un fallo puede causar un cambio de estado. Por ejemplo si una fase para adicionar agua está en el estado *Running* y la motobomba utilizada en esta fase no responde, se deberá emitir un comando *held* y la fase pasara al estado holding, con el fin de detener la fase y garantizar la seguridad del proceso e integridad de los equipos.

2.2.2.4. Estados activos

2.2.2.4.1. Running

El running es la secuencia de operación normal de la fase (*ver figura 2.6*). En la figura 2.14, se muestra el cambio de estado desde el estado Running a partir de tres eventos.

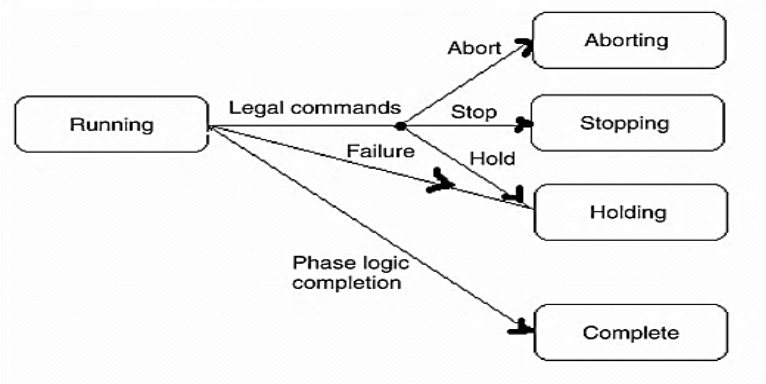


Figura 2.14 Cambios de estado desde Running. *Fuente:* [29]

2.2.2.4.2. Holding

El cambio hacia este estado genera una suspensión temporal de todas las operaciones de la fase, la cual debe realizar la lógica necesaria para llevar a los equipos a un estado seguro, que en este caso sería el estado *held.* En el caso de un mal funcionamiento de un equipo, se deberán detener las operaciones de la fase a la cual está asociado el equipo mientras se corrige la falla. En este sentido un procedimiento para la lógica *Holding* es propuesto en la figura 2.15. Después de la corrección se podrá emitir un comando *Restart* para ejecutar la lógica del estado *Restarting* y volver al estado Running.

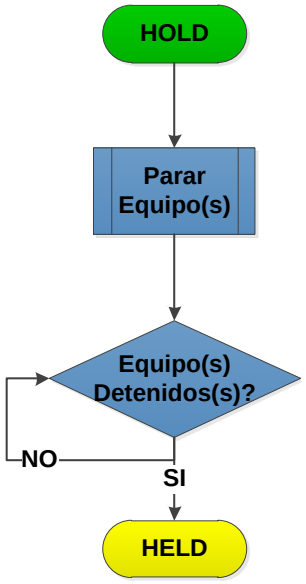
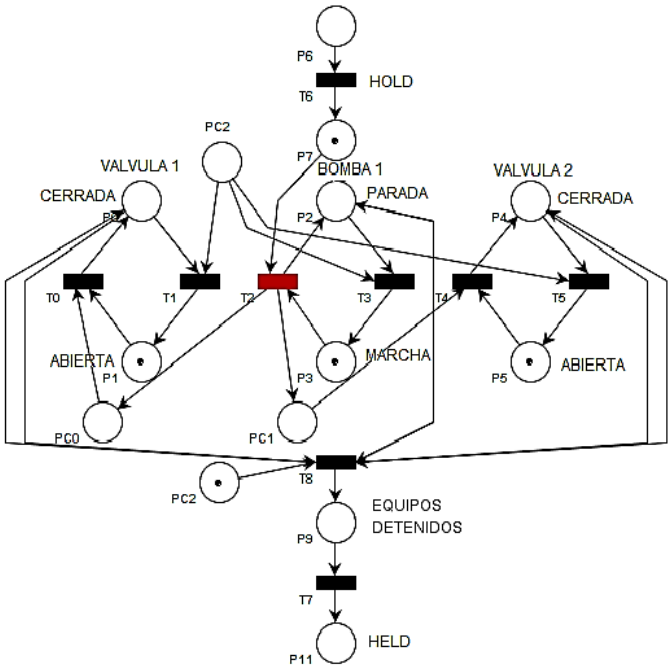


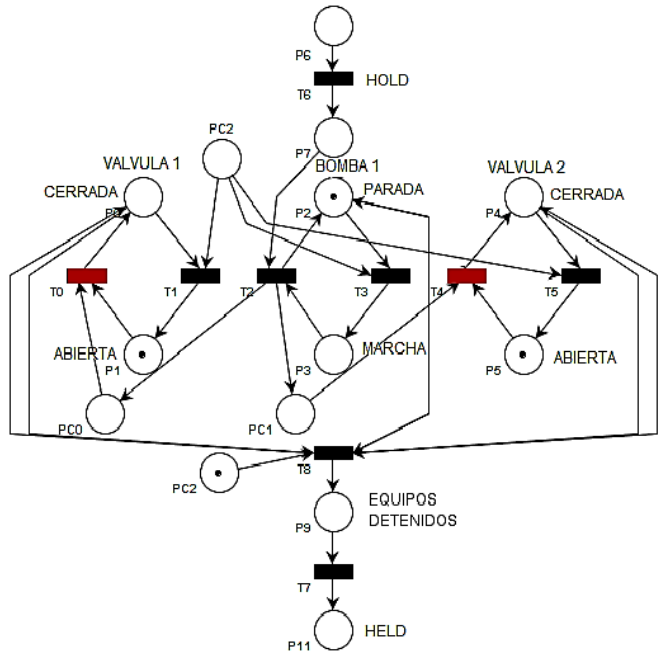
Figura 2.15 . Lógica de estado Holding. ***Fuente: Propia***

Si se considera la fase ***ADICIONAR_MATERIASPRIMAS*** que controla una motobomba y dos válvulas, y un comando *Hold* es emitido dado que no se registró ningún flujo debido a una falla en la motobomba, entonces la lógica *Holding* comienza su ejecución.

Esto se ilustra en la figura 2.16, donde se observa que la lógica *Holding* hace parar la bomba e inmediatamente después cierra las válvulas correspondiendo esto a la etapa ***"Parar equipo(s)"***, y la etapa ***"Equipo(s) Detenido(s)"*** se asegura mediante las restricciones: $v2 - v1 \leq 0$ y $v4 - v1 \leq 0$, de lo cual se obtiene dos lugares de control: PC0 y PC1.



(a)



(b)

Figura 2.16 Lógica Holding para la fase *ADICIONAR_MATERIASPRIMAS*. (a) Transición habilitada para detener motobomba. (b) Transiciones habilitadas para la detención de las válvulas. *Fuente: Propia*

2.2.2.4.3. Stopping

Este estado logra terminar la fase antes de la transición normal al estado *complete*. Por ejemplo, si un operador decide transferir 150 galones de un ingrediente a un mezclador en vez de los 200 galones especificados en la lógica *Running*, el operador deberá emitir un comando *stop* para terminar la fase antes de que se transfieran los 200 galones [30].

La figura 2.17 muestra la lógica que corresponde al estado *Stopping*, la cual es similar a la lógica *Holding* con la diferencia de que una nueva etapa es considerada para reporte de datos. Lo cual es de gran importancia para saber la cantidad de material consumido o simplemente como datos esperados por fases posteriores. Teniendo esto en cuenta, en la figura 2.19 se muestra el resultado de aplicar el esquema de la figura 2.17 a la fase ***ADICIONAR_MATERIASPRIMAS***.

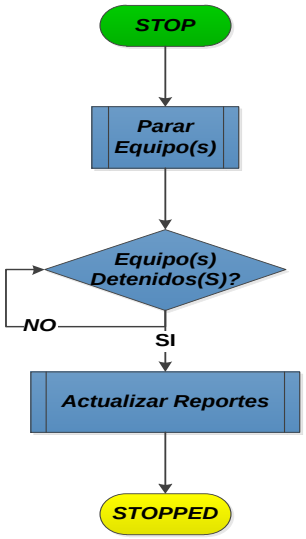


Figura 2.17 Lógica de estado Stopping. *Fuente: Propia*

2.2.2.4.4. Aborting

Este estado se relaciona a una detención anormal de la fase debida a una condición insegura como puede ser el derrame de alguna sustancia, en este caso un comando *Abort* deberá ser emitido, ver figura 2.18. La ejecución de esta lógica, supone una detención mucho más rápida en comparación con la lógica Stopping, por tanto las condiciones que aseguran una detención más "limpia" son limitadas. En este sentido, en la figura 2.20 se diseña el estado Aborting para la fase ***ADICIONAR_MATERIASPRIMAS*** donde se observa que los equipos pueden ser detenidos sin aplicar ninguna restricción previa.

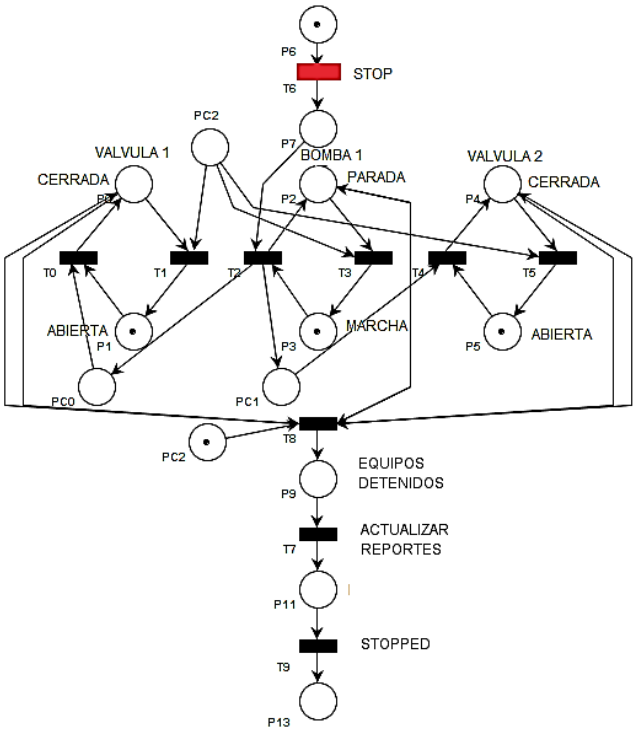


Figura 2.18 Lógica Stopping para la fase *ADICIONAR_MATERIASPRIMAS. Fuente: Propia*

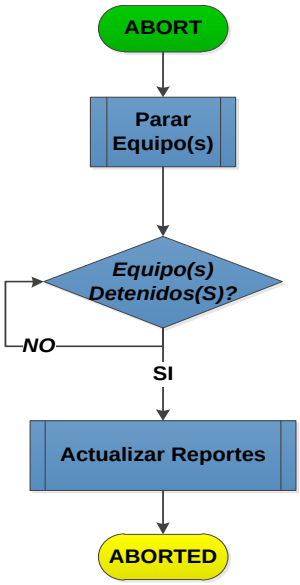


Figura 2.19 Lógica de estado Aborting. *Fuente: Propia*

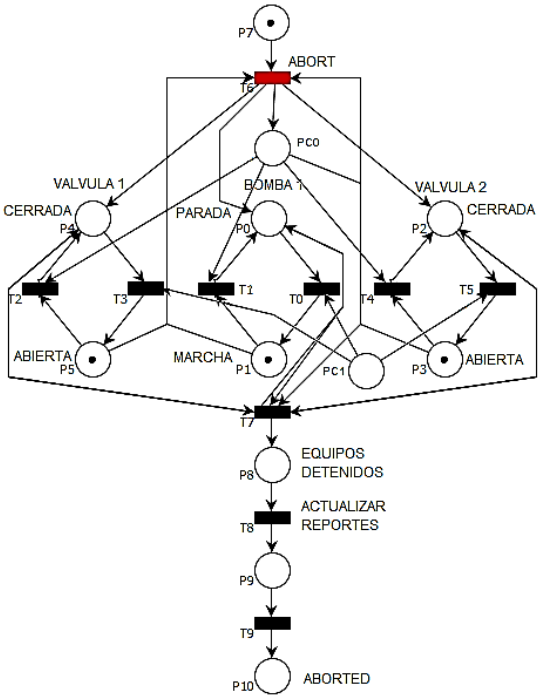


Figura 2.20 Lógica Aborting para la fase *ADICIONAR_MATERIASPRIMAS*. *Fuente: Propia*

2.2.2.4.5. Restarting

Se ejecuta la lógica para ir del estado *held* al estado *Running*. Si no se requieren acciones, entonces se puede pasar inmediatamente al estado *Running*, ver figura 2.21. En la figura 2.22 se muestra la lógica *Restarting* para la fase de ejemplo: para esto se supondrá que la bomba asociada a la fase falla. En caso de no requerir una lógica asociada a este estado, se pasará inmediatamente al estado *Running* y se iniciara el proceso donde había quedado

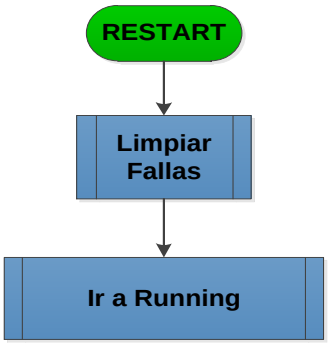


Figura 2.21 Lógica de estado Restarting. *Fuente: Propia*

cuando entro a Holding, es decir, como no se reinició el algoritmo al no hacer uso del estado *Restarting*, se deberá contar con una lógica adicional para que referencie cada una de las etapas y así el administrador del proceso batch sepa donde comenzar la ejecución, esto se hará automáticamente en el momento de la traducción por lo que el usuario no deberá hacerlo. Esto en código ladder sería implementado con la función EQU estableciendo un comando StepIndex.

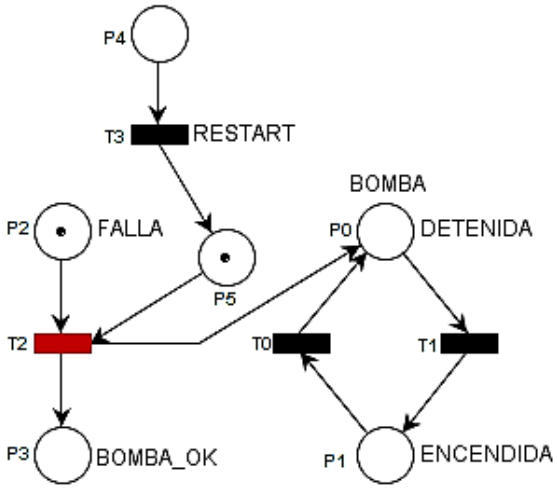


Figura 2.22 Logica Restarting para la fase *ADICIONAR_MATERIASPRIMAS.*
Fuente: Propia

2.2.3. Manejo de excepciones

El estándar ISA-88 define una excepción como un evento que ocurre fuera del comportamiento deseado del sistema, el cual puede afectar el estado de una entidad de equipo o elemento procedimental.

Las excepciones no tienen que estar relacionadas a fallas como lo puede ser un motor atascado, una válvula atascada, etc. Así un tanque de quedarse sin ingredientes en medio de un batch podría ser considerado una excepción. La mayoría de las excepciones son manejadas comandando a *Hold* el elemento procedimental, de esta forma se da tiempo al operador para evaluar la situación y tomar las acciones correctivas [25]. En el caso de obtener productos en mal estado un comando *Stop* es emitido para detener el proceso. Para la fase de ejemplo "***ADICIONAR MATERIASPRIMAS***" si se presenta el problema de falta de material en medio de la ejecución, un comando *Hold* debe ser emitido mientras se configura un batch de una cantidad menor para completar el lote. Es decir, **en este caso se aplicaría el modelo de la figura 2.15** para el diseño de la lógica, obteniendo como resultado la detención de los equipos mientras el/los operario(s) resuelven la eventualidad ocurrida.

3. Capítulo III: Software de traducción al lenguaje IEC 61131-3 Petri Generator Code

Para el desarrollo del procedimiento descrito en el capítulo anterior que permite el diseño de modelos dinámicos de RdP con base al estándar ISA-88, se ha construido una aplicación software que transforma las RdP a código de programación definido por la IEC 61131-3. Este capítulo pretende ahondar sobre las características de esta herramienta software: tanto en el funcionamiento de la interfaz gráfica como en los algoritmos de transformación de RdP a lenguaje de programación de PLC (Ladder y SFC). Adicionalmente, se tratará su implementación en RsLogix 5000 por medio de los archivos L5K para la creación de la lógica de control de procesos batch en controladores industriales de alta gama.

3.1. Procedimiento para el diseño de la aplicación software Petri Generator Code (PGC)

A continuación se encontrarán descritos los requerimientos funcionales que permitieron la construcción de la herramienta software, y que fueron la base para representar las necesidades de usuario.

Requerimientos funcionales:

Principal:

- Implementar algoritmos computacionales para la transformación de RdP a código en lenguaje IEC 61131-3 para la plataforma RsLogix5000.

Secundarios:

- Se pueden asociar múltiples condiciones lógicas a una transición y múltiples acciones a un lugar.
- Trabaja con redes de Petri creadas en PIPE, CRP y Snoopy.
- Trabajar con el modelo de estados de fase de ISA-88.

Con esta información a la mano, se presentarán los diagramas UML que describirán tanto el funcionamiento de la aplicación software como la manera en la cual estos abordan las soluciones a los requisitos de usuario anteriormente presentados.

3.1.1. Diagramas UML

En la figura 3.1 se presenta un diagrama de casos de uso en el que se resume el alcance de la aplicación software.

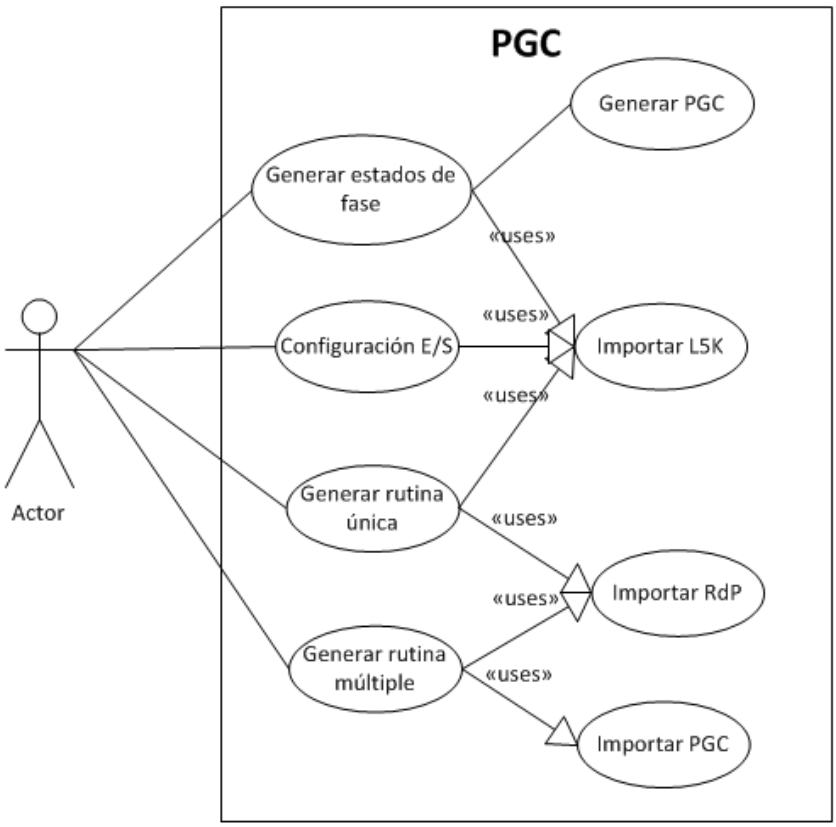


Figura 3.1 Diagrama de casos de uso. ***Fuente: Propia***

Descripción de los casos de uso

Nombre: Generar estados de fase
Función: Se podrá configurar los estados de fase obtenidos desde el formato L5K.
Descripción: Desde una interfaz se podrá realizar la asociación de un número determinado de estados de fase según el usuario con base al modelo de estados de fase de ISA-88. Requiere de la importación de un archivo L5K.

Nombre: Configuración E/S
Función: Permite al usuario realizar la configuración del módulo de E/S.
Descripción: Desde una interfaz se podrá realizar el cambio de nombre de las entradas y salidas discretas. Requiere de la importación de un archivo L5K.

Nombre: Generar rutina única
Función: El usuario podrá realizar la configuración de rutinas únicas.
Descripción: Desde una interfaz se podrá realizar la asociación de las entradas discretas en condiciones lógicas a una transición, y de las salidas discretas a lugares de una única RdP. Requiere de la importación de un archivo L5K y de una RdP en formato XML.

Nombre: Generar rutina múltiple
Función: El usuario podrá realizar la configuración de múltiples rutinas.
Descripción: Desde una interfaz se podrá realizar la asociación de las entradas discretas en condiciones lógicas a una transición, y de las salidas discretas a lugares de múltiples RdP. Requiere de la importación de un archivo L5K y de un archivo en formato PGC.

Para el desarrollo de estos casos de estudio, son presentados los diagramas UML que describen la aplicación software por medio de los diagramas de clases (ver figura 3.2), de estados (ver figura 3.3), de secuencias (ver figuras 3.4-3.7) y de actividades (ver figuras 3.8-3.10). Este modelamiento posteriormente dará pié a presentar las características de la herramienta desarrollada en el trabajo de grado dentro en la sección 3.2.

- **Diagrama de clases**

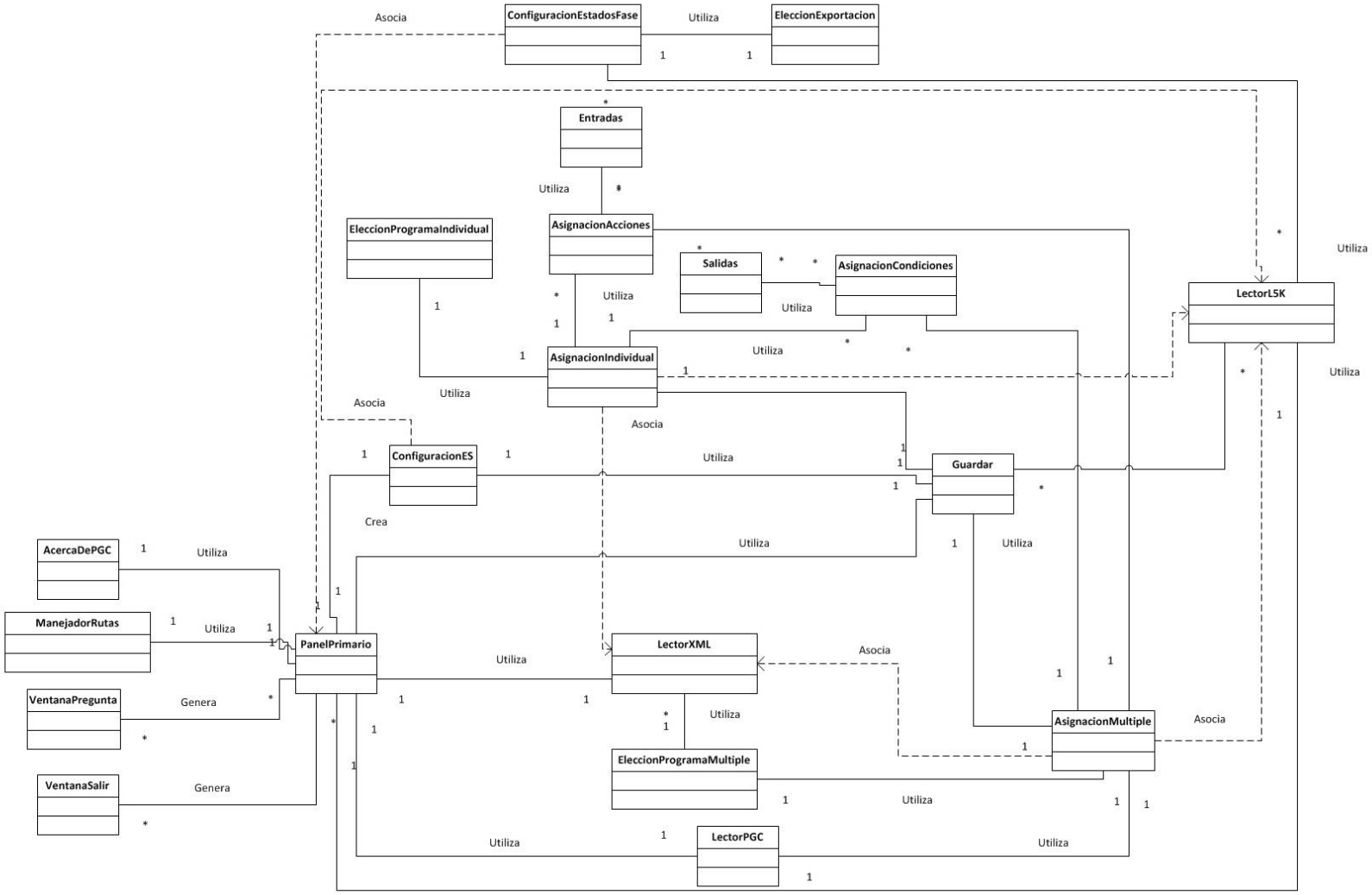


Figura 3.2 Diagrama de clases. ***Fuente: Propia***

- **Diagrama de estados**

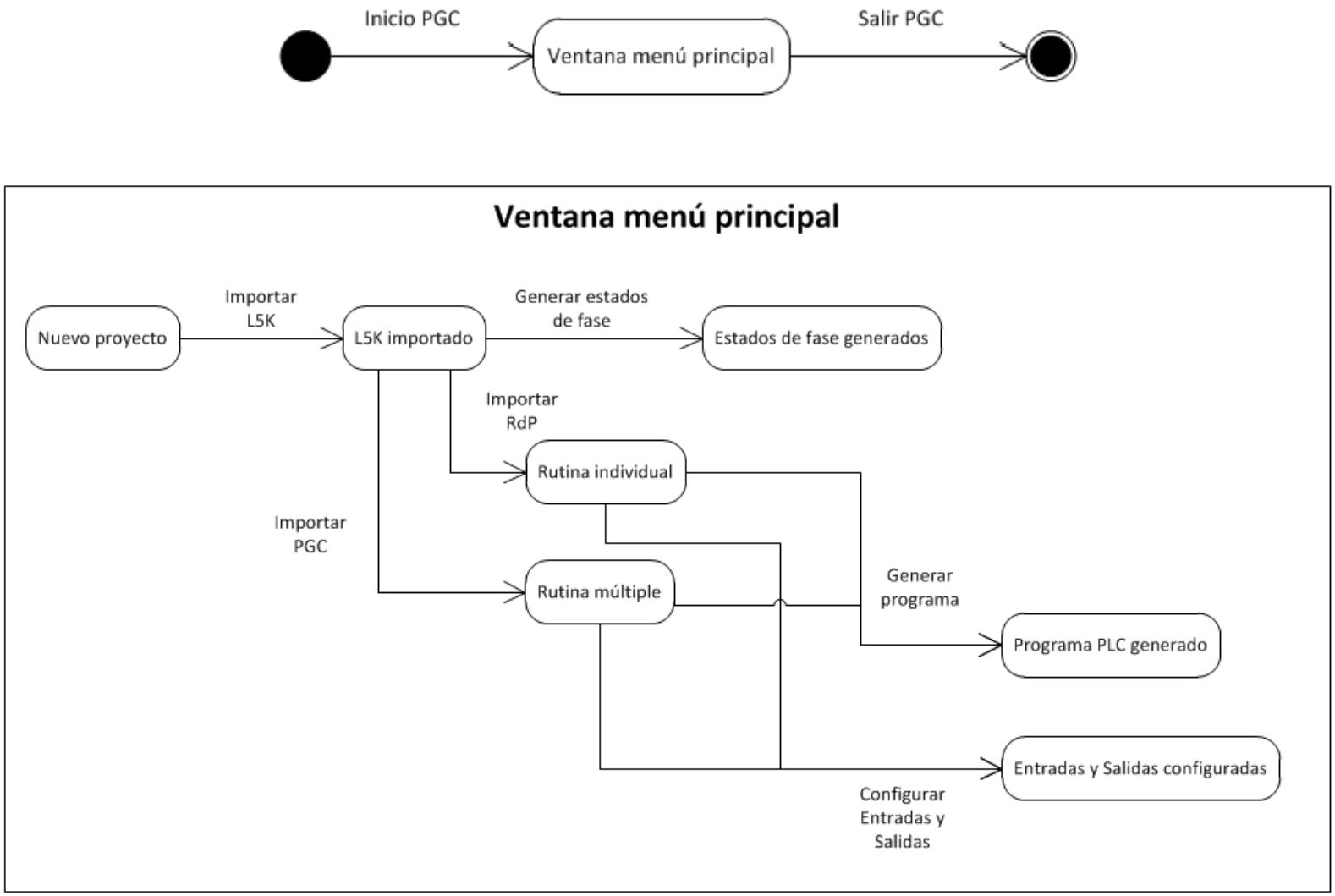


Figura 3.3 Diagrama de estados. ***Fuente: Propia***

- **Diagrama de secuencias**

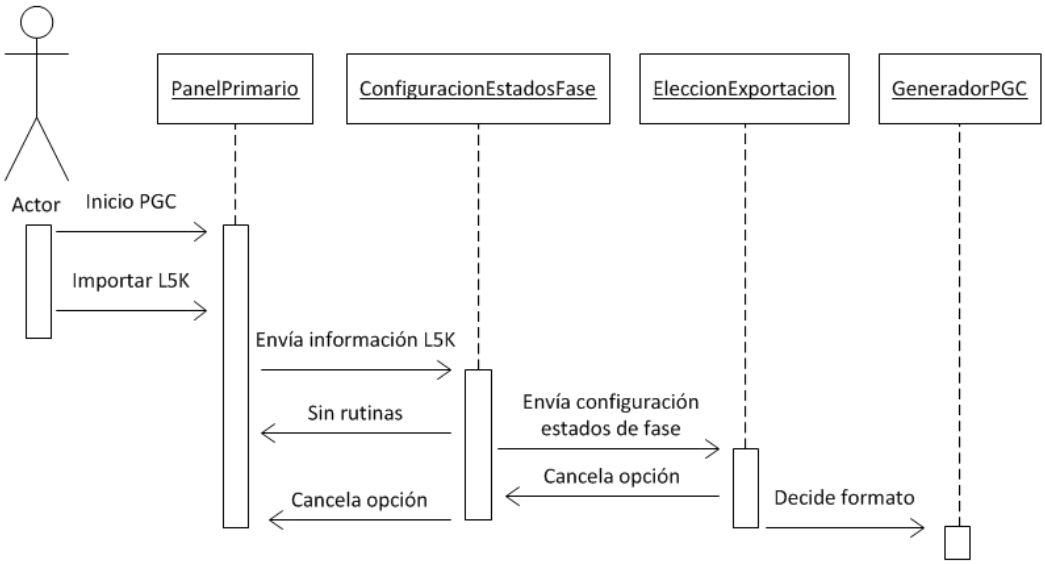


Figura 3.4 Secuencia de configuración y exportación de estados de fase. ***Fuente: Propia***

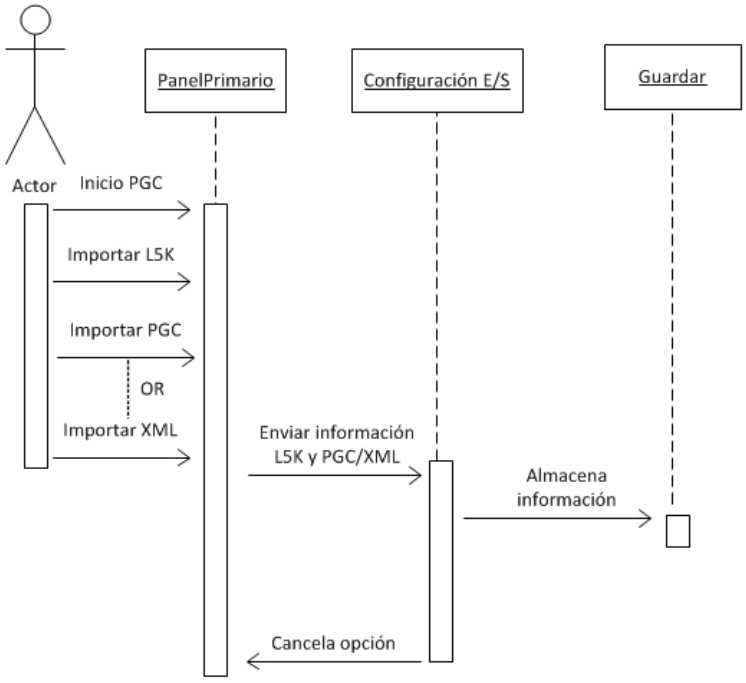


Figura 3.5 Secuencia de configuración E/S. ***Fuente: Propia***

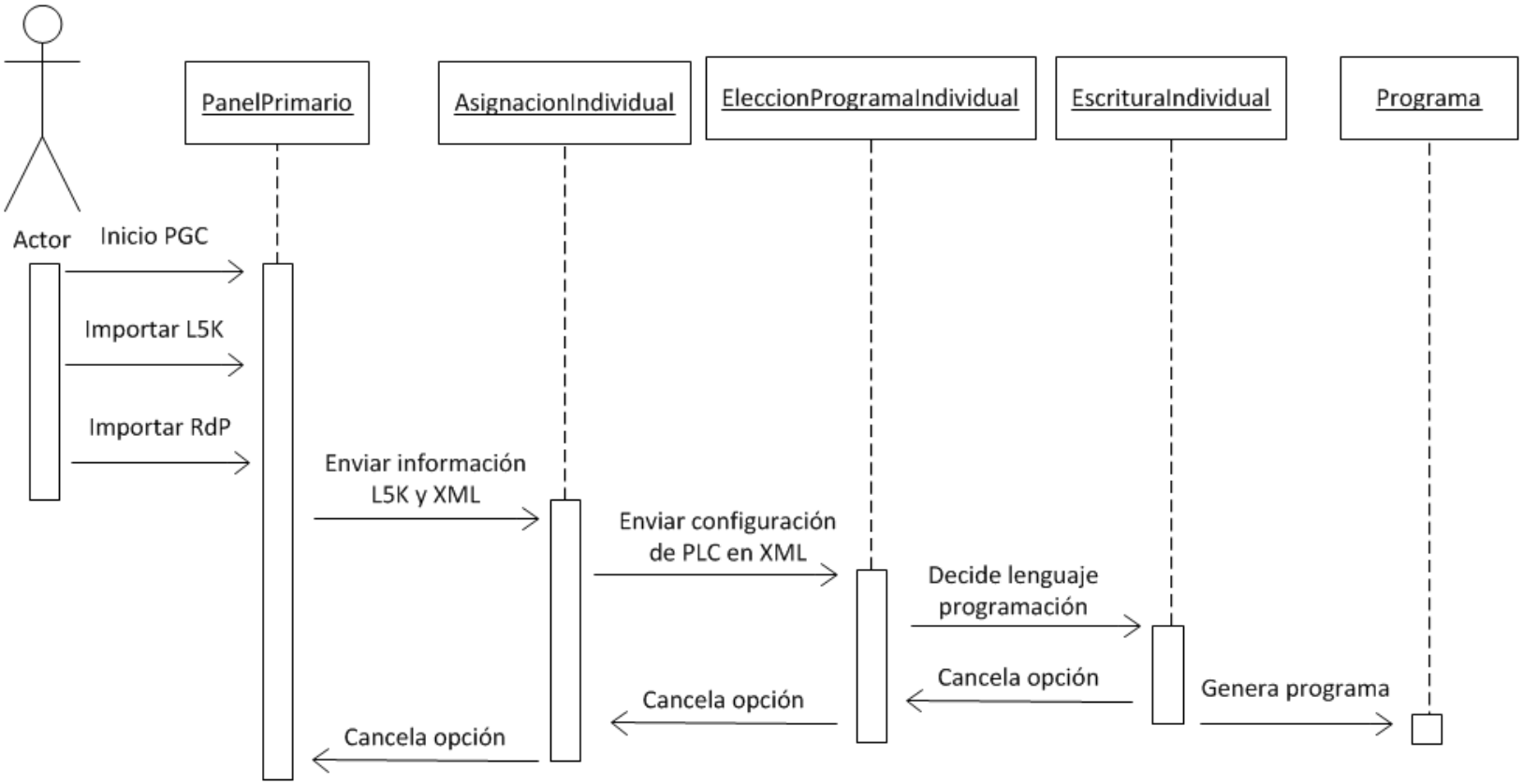


Figura 3.6 Secuencia de configuración y exportación de una rutina única en L5K. ***Fuente: Propia***

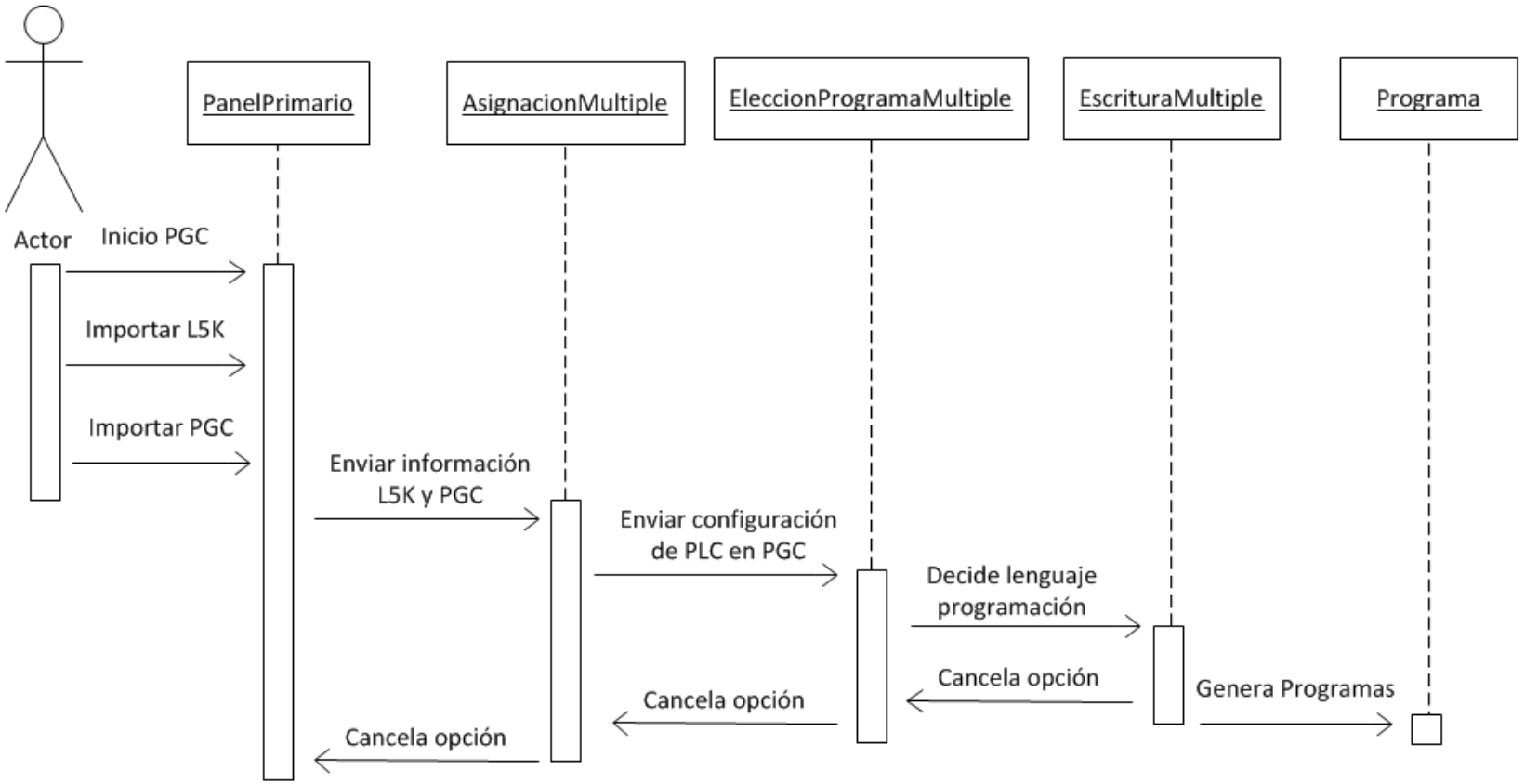


Figura 3.7 Secuencia de configuración y exportación de múltiples rutinas en L5K. ***Fuente: Propia***

- **Diagrama de actividades**

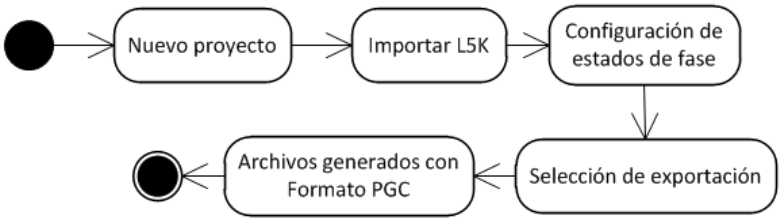


Figura 3.8 Generación de estados de fase. ***Fuente: Propia***

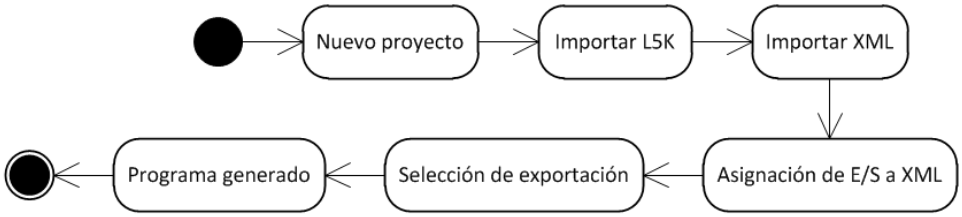


Figura 3.9 Generación de proyecto individual. ***Fuente: Propia***

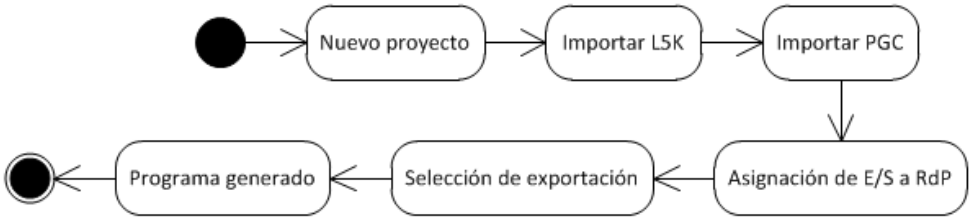


Figura 3.10 Generación de proyecto múltiple. ***Fuente: Propia***

3.2. Introducción a PGC

PGC es una herramienta software diseñada en Java que le permite al usuario realizar la traducción automática de una o varias redes de Petri a un proyecto en formato ASCII para la implementación de múltiples líneas de código para controladores programables.

El algoritmo de conversión del software le permite al usuario realizar la traducción automática de las redes de Petri en código para controladores programables definido por el estándar IEC 61131-3: (a) Ladder, y (b) SFC.

La aplicación está enfocada a la programación de procesos Batch ya que le permite al usuario realizar la configuración de los estados de fase a utilizar por cada módulo de equipo, aprovechando así las ventajas de los modelos definidos por el estándar ISA 88.

Características de la aplicación software:

- Paso de uno o varios archivos con redes de Petri en formato XML (construido en CRP, PIPE o Snoopy) a múltiples rutinas de programación en el lenguaje IEC 61131-3 de manera automática.

- Interfaz gráfica de usuario intuitiva y simple para la configuración de múltiples E/S (entradas y salidas) en la programación de Ladder y SFC.

- Uso de la técnica de TPL (Token Passing Logic) para la traducción de red de Petri a código Ladder.

- Uso de una técnica de traducción directa de red de Petri a código SFC para el manejo de programas simples en la lógica de control.

- Lectura y extracción de información de los proyectos realizados en formato L5K de la plataforma de RsLogix5000 de Rockwell Automation.

- Configuración de los estados de fase de cada módulo de equipo definidos en el entorno RsLogix 5000 de Rockwell Automation para la edición de la lógica de control en tres diferentes plataformas de diseño de redes de Petri: PIPE, CRP y Snoopy.

- Salvar configuraciones realizadas en la aplicación software bajo el formato PGC.

Requerimientos de la aplicación software:

- Sistema operativo: Windows8/Windows7/Vista/WindowsXP/Windows2003 Server.

- Espacio en disco: 1MB.

- Se requiere la instalación previa del Java Runtime Enviroment (JRE) ver. 1.7.

La herramienta PGC fue creada en Java ya que provee de una completa funcionalidad de una interfaz gráfica de usuario (GUI) y su rápida adaptación a la plataforma en la que es ejecutada. El diseño de la GUI utiliza elementos gráficos intuitivos y simples que facilitan su navegación a través de ella por medio del ratón utilizando componentes Swing derivados de Java. El código compilado se almacena en un archivo de extensión jar que podrá ser ejecutado en cualquier computador con JRE.

3.3. Uso de las técnicas de transformación de Petri Generator Code

En esta tesis hay dos vías para usar PGC. Una, si se ha construido una RdP individual que desea traducirse a código para PLC en cuyo caso la lógica de control formará parte de una sola rutina (Main routine) dentro de un proyecto en RsLogix5000 (este procedimiento es presentado en el Anexo B). En la segunda vía, se propone el uso de PGC para generar los programas de la lógica de control en código definido por la IEC 61131-3, de acuerdo a los modelos dinámicos en redes de Petri que parten del modelado de ISA-88 (procedimiento descrito en el capítulo II). Esta lógica de control es introducida en un proyecto de RsLogix5000 como rutinas, las cuales serán ejecutadas por medio de la aplicación software FactoryTalk Batch.

Esta segunda vía se encuentra descrita en esta sección, y por tanto, se sugiere que sea revisada la guía monográfica en [10] caso de que el lector desconozca el procedimiento de asignación de un proyecto con rutinas múltiples sincronizadas entre FactoryTalk Batch y RsLogix5000. Ya que en el momento en el cual el editor de equipos de FactoryTalk Batch es sincronizado con RsLogix5000, se generarán fases de equipo vacías en el proyecto de RsLogix5000 que deberán ser llenadas por medio de código de programación (ver figura 3.11). El trabajo de PGC será el de llenar con líneas de código estas fases con modelos dinámicos de Redes de Petri conservando la propuesta del modelo de estados de ISA-88.

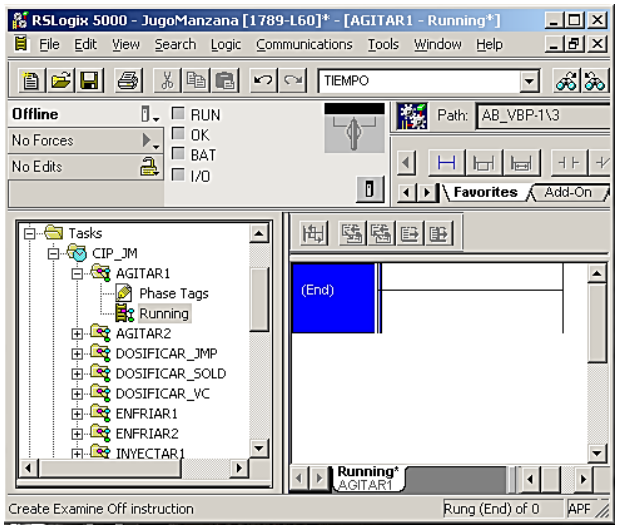


Figura 3.11 Proyecto con fases sincronizadas en RsLogix5000. ***Fuente: RsLogix5000***

El usuario deberá, por tanto, exportar un archivo L5K desde RsLogix5000 con la información de estas fases, ya que allí está contenida la información del proyecto en formato ASCII y será la manera en la cual PGC incide directamente en la modificación del

código del controlador programable. A continuación, se explicarán los pasos que permitirán el llenado de estas líneas en código de programación definido por la IEC 61131-3.

3.3.1. Generación de la lógica de control de un proyecto para RsLogix 5000 con rutinas múltiples

Desde la ventana principal en PGC se debe de crear un nuevo proyecto en blanco e importar el proyecto de RsLogix5000 en formato L5K (ver Figura 3.12).

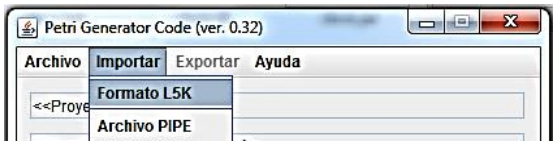


Figura 3.12 Importar proyecto de RsLogix5000 en PGC. ***Fuente: Propia***

Si el archivo es abierto correctamente, el programa PGC obtendrá una ventana de información sobre el proyecto generado. Aquí se indicará:

- El nombre con el que se identifica el controlador que implementará la lógica de programación, y su localización a nivel local.

- Los módulos del chasis que hacen parte del proyecto.

En la Fig. 3.13 ha sido importado el archivo L5K llamado “JugoManzana” el cual se compone de dos módulos: Aquel que contiene el procesador de la serie 1789-L60 (nombre, tamaño y ubicación en chasis), y otro que conserva las E/S del proceso junto con su ubicación en chasis. Esta referencia de la ubicación es importante, ya que de allí se obtendrán el nombre por definición bajo el cual se identificarán las entradas y salidas con las que interactúa el programa de PLC.

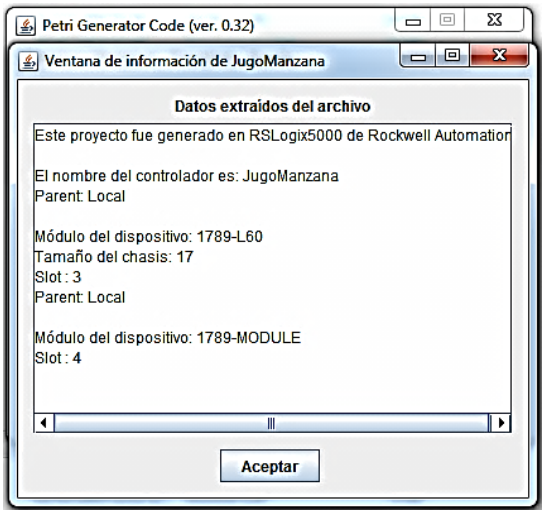


Figura 3.13 Resultados del archivo importado en RsLogix5000. ***Fuente: Propia***

Se procede a generar el número de estados que deberá contener cada fase, para lo cual, se requiere que sean exportados en formato XML compatible con un editor de RdP como PIPE, CRP o Snoopy. Por tanto, en el menú de Exportar en CRP, se encuentra la opción "Configurar Estados de fase". Al hacer click en ella, se puede observar una figura similar a la indicada en la Figura 3.14.

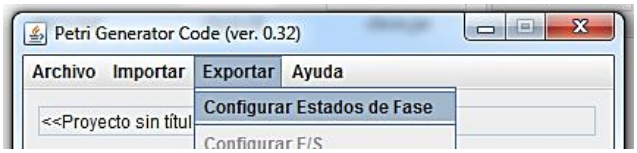


Figura 3.14 Configurar Estados de Fase. ***Fuente: Propia***

En la ventana de la figura 3.15, el usuario podrá indicar si un módulo de equipo deberá poseer un único estado de fase o por el contrario, el número de estados de fase que deberá utilizar es mayor o igual a dos. Vale la pena recordar que siempre será obligatoria la asignación de una rutina Running dentro de PGC ya que esta representa la ejecución del código de programación de la fase en condiciones normales.

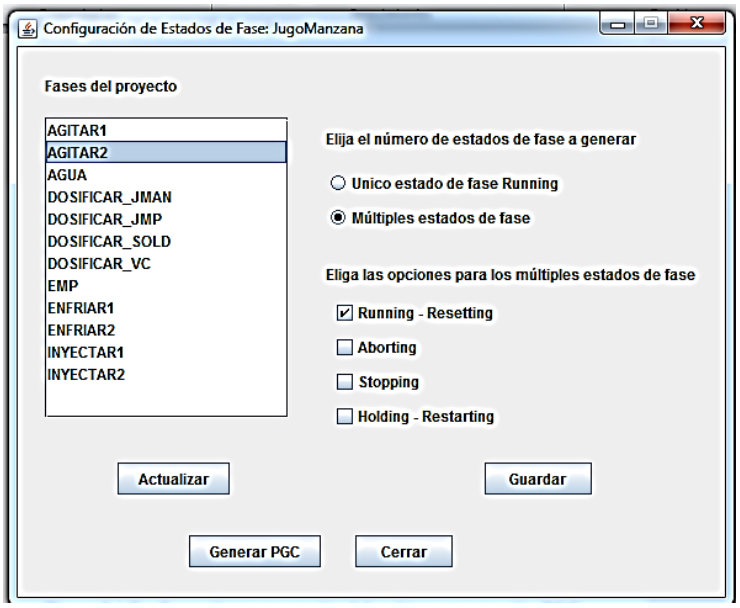


Figura 3.15 Configuración y exportación del archivo con extensión PGC. ***Fuente: Propia***

Al terminar la configuración de los estados de fase, se pulsará la opción de "Generar PGC". Esto lleva a una segunda ventana desde la que se podrá seleccionar el formato de exportación a RdP de la configuración anteriormente realizada (ver figura 3.16). Existen dos opciones posibles PIPE o Snoopy. El formato XML de almacenamiento de PIPE contiene la misma estructura al formato de CRP, por tanto desde ambas interfaces se puede abrir y modificar la RdP.

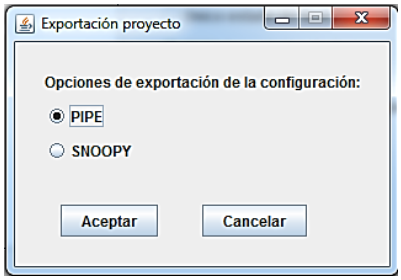


Figura 3.16 Opción de exportación de la configuración de estados de fase. ***Fuente: Propia***

Cuando se pulsa Aceptar. El usuario deberá indicar la carpeta de destino en la cual deberá ser guardado esta configuración. Se propone que el usuario destine una carpeta aparte con el nombre del proyecto dado en RsLogix5000. Se generan archivos en formato XML según sea el número de estados que existan por fase. Ej. Si una fase llamada Agitar contiene dos

estados, entonces se generarán dos archivos XML respectivamente (el número máximo de estados por fase es de seis). Independientemente de la opción pulsada entre PIPE o Snoopy, se generará un archivo adicional con formato PGC, el cual es un archivo de texto que contiene la siguiente información:

```
%   Descripción del archivo PGC para usuario   %
[INITDIRPATH]
%   Ubicación de la carpeta en la que es guardado el proyecto   %
[ENDDIRPATH]
[INITXMLRDP]
%   Ubicación de los diferentes archivos en XML con redes de Petri   %
[ENDXMLRDP]
[INITCONFMOD]
% Información sobre el número de estados configurados por cada módulo de equipos existente %
[ENDCONFMOD]
```

Esta información le permite a la aplicación software PGC retomar las direcciones de los archivos modificados en los editores de redes de Petri, y verificar aquellas que contienen información. En la figura 3.17 se podrá visualizar el resultado de exportación.

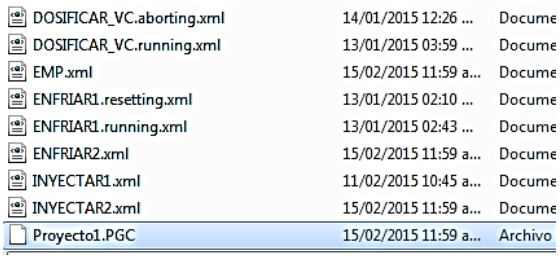


Figura 3.17 Resultado de la exportación de los estados de fase. *Fuente: Propia*

Hay dos tipos de archivos: (a) “nombreEquipo”, y (b) “nombreEquipo.estadoDeFase”. El primer tipo de archivo indica que el módulo de equipo solo utilizará una única rutina que representa su comportamiento en condiciones normales. Con el segundo tipo de archivo, el paso entre estados de una fase estará descrito por el diagrama de estados de ISA 88.

En este punto, se propone el modelado de fases según los procedimientos del capítulo II por medio de los editores de redes de Petri (en este caso de PIPE o CRP). A modo de ilustración, se presentará un posible diagrama en redes de Petri del estado Resetting de la fase AGUA en la figura 3.18.

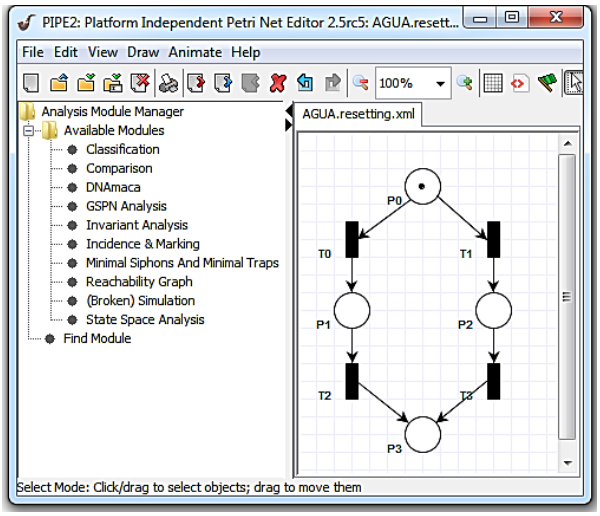


Figura 3.18 Diseño de un estado de fase por redes de Petri. *Fuente: Propia*

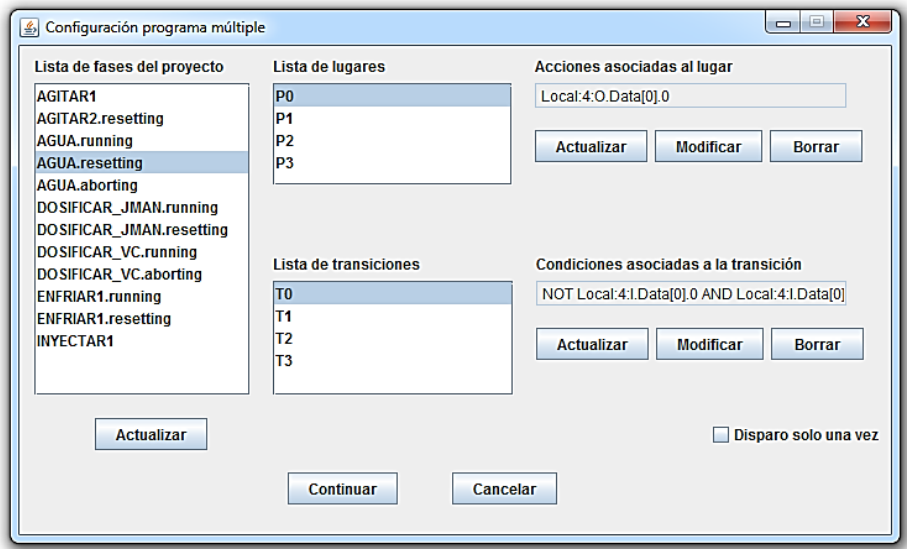
Después del modelado en redes de Petri de todos los estados de fase, se deberá importar el formato PGC, tal como se puede ver en la figura 3.19 (a), que para este caso es llamado Proyecto1.PGC. Se recuerda al lector, que en caso de haber cerrado la aplicación PGC, también deberá reimportarse el archivo L5K. Esto quiere decir que para acceder al paso siguiente en el menú Exportar, deberán encontrarse las opciones habilitadas de la figura 3.19 (b).

(a) (b)

Figura 3.19 (a) Importación del formato PGC (b) Visualización de los formatos importados en la GUI. *Fuente: Propia*

En el menú Exportar, se encuentra la opción de Generar programa(s), tal como está indicado en la figura 3.20 (a). Luego de hacer click, deberá aparecer una ventana similar a la de la figura 3.20 (b).

(a)



(b)

Figura 3.20 (a) Opción Generar programa(s) (b) Configuración de múltiples redes de Petri para asignación de acciones y de condiciones. *Fuente: Propia*

La ventana de la figura 3.20 (b) contendrá las siguientes opciones:

- Lista de fases del proyecto a la izquierda: Aparecerán el nombre de todos los archivos del proyecto que contengan redes de Petri internas. Aquellos archivos en formato XML que no tengan ninguna red de Petri interna no aparecerán listadas.

- Botón de Actualizar abajo a la izquierda: Al ser presionado se actualizarán las Listas de lugares y Lista de transiciones del archivo seleccionado en Lista de fases del proyecto.

- En la parte superior derecha se podrá asignar una o varias acciones (conectadas por una instrucción AND) a un lugar de la RdP. La lectura de la acción en la figura 3.20 (b) sería la siguiente “Cuando exista una marca en P0, las salida 0 del módulo de E/S se energizará”.

- De manera análoga, en la parte inferior derecha se realizará la asignación de una o varias condiciones lógicas (conectadas por una instrucción AND o por una instrucción AND NOT) a una transición de la RdP. La lectura de la condición lógica de la figura 3.20 (b) sería la siguiente "Para disparar una transición T0, se requiere que la entrada 0 esté desactivada y la entrada 2 esté activada, ambas del módulo de E/S".

Posteriormente al terminar la asignación de cada uno de ellos, se podrá pulsar el botón "Generar Programa". Desde aquí surgirá la siguiente ventana con la representación de la Figura 3.21.

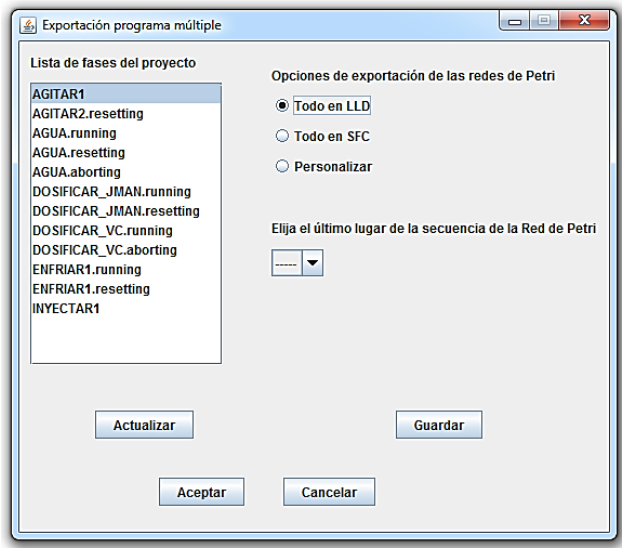


Figura 3.21 Opciones de exportación en SFC o Ladder del proyecto. ***Fuente: Propia***

Desde aquí, se puede indicar la manera de exportar el número de fases del proyecto al archivo L5K con las siguientes posibilidades:

- Exportar todas las fases del proyecto a lenguaje Ladder
- Exportar todas las fases del proyecto a Lenguaje SFC
- Personalizar las opciones de exportación.

Las dos primeras opciones bastan con estar seleccionadas para que al presionar el botón de Exportar, se pueda generar un archivo en formato L5K con todas las rutinas transformadas al lenguaje de programación indicado. Por otro lado, al seleccionar la opción "Personalizar", se podrá indicar para cada fase la opción que desee el usuario.

Independientemente del modo de exportación (Ladder o SFC), aparecerá una opción adicional que permitirá elegir el último lugar de la secuencia de la red de Petri. Esta opción indica si existe un lugar dentro del modelo dinámico en el cual si llegase a existir una marca, esta daría fin a la ejecución de la fase (ver Figura 3.22.)

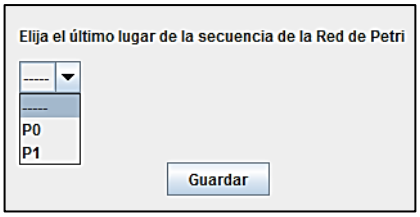


Figura 3.22 Opción de SFC de generación del código de programación. ***Fuente: Propia***

Al presionar el botón Exportar, todas las redes de Petri diseñadas serán transformadas a rutinas ordenadas en el proyecto L5K con el lenguaje de programación elegido por el usuario.

En resumen, PGC requiere de la información de un proyecto generado por RsLogix5000 (un programador de PLC) del cual obtendrá la información del número de rutinas a desarrollar y de los módulos de E/S añadidos al controlador. Posteriormente, según sea el número de estados configurados por el usuario este generará un número determinado de archivos en PIPE/CRP o Snoopy, los cuales podrán ser modificados por el usuario para modelar el comportamiento deseado por medio de redes de Petri. El usuario podrá asociar a los lugares y transiciones de cada modelo dinámico una entrada o salida del módulo del controlador programable. Finalmente, la aplicación genera código de programación en Ladder o SFC, con base al estándar IEC61131-3.

En las secciones 3.4 y 3.5, se profundizará sobre el procedimiento de traducción a código de programación Ladder y SFC respectivamente. En la sección 3.6, se hará referencia a la escritura de los programas dentro de los archivos L5K.

Por cuestiones de espacio, el uso de la aplicación Snoopy para el diseño de redes de Petri y su enlace con esta herramienta está descrito en el Anexo C.

3.4. Algoritmo de traducción de Red de Petri a código Ladder

En esta sección se describen las características implementadas en el algoritmo de traducción de redes de Petri a código Ladder. Adicionalmente, se describirán los pasos de ejecución del algoritmo de programación y las limitaciones de trabajo con esta técnica de traducción.

La metodología de traducción implementada en este proyecto está basada en el proyecto de grado de C. Buchely y F. Coque [22], quienes desarrollaron una aplicación software llamada CRP que permite el diseño de redes de Petri bajo la metodología de supervisores para sistemas a eventos discretos (SEDs). El procedimiento para diseñar redes de Petri con CRP culmina con la traducción del funcionamiento de la red a código Ladder para gamas bajas y medias de controladores lógicos programables (MicroLogix y SLC500 respectivamente) de la plataforma de RsLogix500 de Rockwell Automation.

Sin embargo, para esta monografía se propone aumentar la capacidad del algoritmo de traducción de redes de Petri a Ladder, y por tanto, el uso de redes de Petri con las siguientes características:

- Diseño de redes de Petri con arcos de inhibición (inhibitor edge) y de lectura o activación (read edge).

- Asociación múltiple de salidas y entradas de tipo discreto a una red de Petri.

Aunque estas extensiones sobre las RdP no serían estrictamente necesarias en una representación con RdP ordinarias, si ofrecen una ventaja en la generación de código Ladder ya que permiten reducir el número de lugares o transiciones necesarios para representar las acciones deseadas, con lo que el tamaño del código Ladder también será reducido.

Formalmente, estas redes de Petri con elementos adicionales han sido catalogadas como Red de Petri de Automatización (Automation Petri Nets) por Uzam y Jones en [33] y son representadas matemáticamente de la siguiente manera:

$$APN = (PN, In, En, x, Q, M_0)$$

Dónde:

$PN = (P, T, Pre, Post)$, representa una red de Petri ordinaria,

$In: (P \times T) \rightarrow N$, es una función de entrada inhibidora que define arcos inhibidores con peso de lugares a transiciones,

$En: (P \times T) \rightarrow N$, es una función de entrada de lectura que define arcos de lectura con peso de lugares a transiciones,

$x = \{x1, x2, \dots, xm\}$, grupo finito no vacío de condiciones de disparo asociado a las transiciones

$Q = \{q1, q2, \ldots, qm\}$, grupo finito de acciones que podría ser asignado a los lugares

$M_0: P \rightarrow N$, marcaje inicial de la red de Petri

Adicionalmente, para asegurar que el código Ladder generado cumpla con las especificaciones del lenguaje dadas en el estándar IEC 61131-3 este deberá ser probado dentro de una plataforma de programación de controladores que esté adherida a la norma. En el caso de esta monografía, el algoritmo generará instrucciones para un controlador programable del fabricante Rockwell Automation en RsLogix 5000, el cual tiene la capacidad de trabajar con los cinco lenguajes de programación del estándar, y permitir la descomposición de los programas dentro de tareas (task) y rutinas (routines) [34-36].

Vale la pena aclarar que este procedimiento de trasformación no está limitado a este fabricante, ya que la metodología TPL sobre la que se está trabajando ha sido aplicada en múltiples oportunidades sobre varios casos de estudio que utilizan controladores de proveedores diferentes [37, 38]. Esto por tanto, permite el uso de esta guía monográfica para trabajar en la implementación de este algoritmo hacia otros controladores de alta gama.

3.4.1. Pasos para la generación de código Ladder

Para trabajar con la técnica de traducción a Ladder se utiliza la siguiente información de las redes de Petri:

- Vector de marcado inicial.
- Matrices de incidencia hacia adelante, de incidencia hacia atrás, de inhibición y de activación.
- Asociación de entradas y salidas discretas a las transiciones y lugares de la red de Petri, respectivamente.
- Clasificación de una red de Petri como segura o no segura.

El algoritmo computacional realiza las siguientes tareas para la creación de código Ladder:

1. Determinar las relaciones existentes en las matrices de incidencia hacia adelante, incidencia hacia atrás, de inhibición y de activación.

Cada valor igual o superior a uno en cada una de estas matrices representa un valor de peso del arco de la red de Petri. Si la red de Petri es segura (el valor de marcaje de cada lugar de la red no es superior a 1) entonces cada número de la matriz será traducido por un

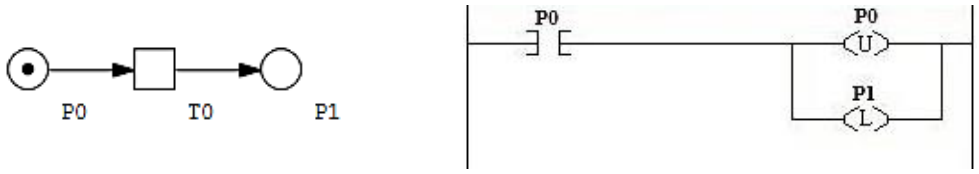
comparador de tipo booleano, por el contrario, si la red de Petri no es de tipo seguro (el valor de marcaje de cada lugar de la red puede llegar a ser superior a 1), entonces se utilizará un comparador de tipo entero para su representación.

Si las matrices son de incidencia hacia adelante, de incidencia hacia atrás o de activación, el comparador será un contacto normalmente abierto para redes de Petri seguras o un comparador entero de tipo “mayor o igual” para redes de Petri no seguras. En el caso contrario de una matriz de inhibición, el comparador será un contacto normalmente cerrado para redes de Petri seguras o un comparador entero de tipo “menor o igual” para redes de Petri no seguras.

Por último, para la representación del flujo de marcas de un lugar a otro se utilizarán bobinas para redes de Petri seguras, y contadores para redes de Petri no seguras. Las bobinas serán de tipo directo si la traducción es de una matriz de incidencia hacia adelante, mientras que las bobinas de tipo inverso se utilizarán para las matrices de incidencia hacia atrás. Para las redes de Petri no seguras, se utilizará una función de suma enteros si se está traduciendo el valor una matriz de incidencia hacia adelante o se utilizará una función de resta de enteros en el caso análogo de una matriz de incidencia hacia atrás. El valor base de suma de estas funciones depende exclusivamente del valor especifico que se esté trabajando en la matriz respectiva (que a su vez, es una representación del valor de peso del arco de la red de Petri).

Vale la pena recordar que las matrices de inhibición y de activación no realizan movimientos de marcas en una red de Petri, y por tanto, no requieren de la representación de bobinas ni de bloques de función de suma y resta.

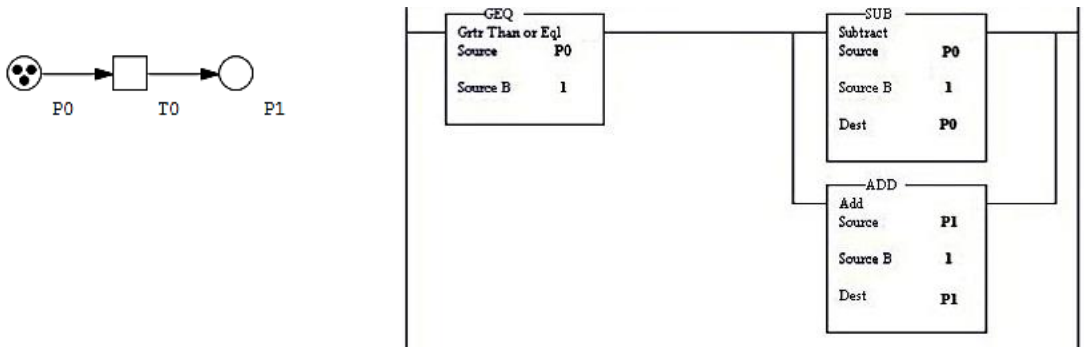
En la figura 3.23 (a) se puede observar un ejemplo de transformación de una red de Petri segura, y en la figura 3.23 (b) su correspondiente código Ladder en donde son utilizados bits para representar el flujo de marcas. En este caso en el cual no existen más condiciones asociadas a T0, el disparo de T0 depende únicamente de la existencia de una marca en el lugar P0. Por tanto, en lenguaje Ladder si P0 está energizado, la acción del controlador será la de encender P1 y apagar P0.



(a) (b)

Figura 3.23 Ejemplo de transformación de una red de Petri segura. (a) Red de Petri. (b) Código Ladder. ***Fuente: Propia***

En la figura 3.24 (a) se presenta un caso similar al anterior con la diferencia de que esta última es una red de Petri no segura, y se utilizan variables de tipo entero para representar el flujo de marcas en su respectivo código Ladder de la figura 3.24 (b). De manera análoga, no existen más condiciones asociadas a T0, y por tanto, su disparo solo depende del marcaje de P0. En el programa del controlador, se utiliza un bloque funcional denominado GEQ para comparar el peso del arco de P0 a T0 con el valor actual de marcaje, mientras que se utilizan bloques funcionales de resta y suma (SUB y ADD) para sustraerle al valor del lugar P0 el peso del arco de P0 a T0, y añadirlo al lugar P1 según el peso del arco de T0 a P1.

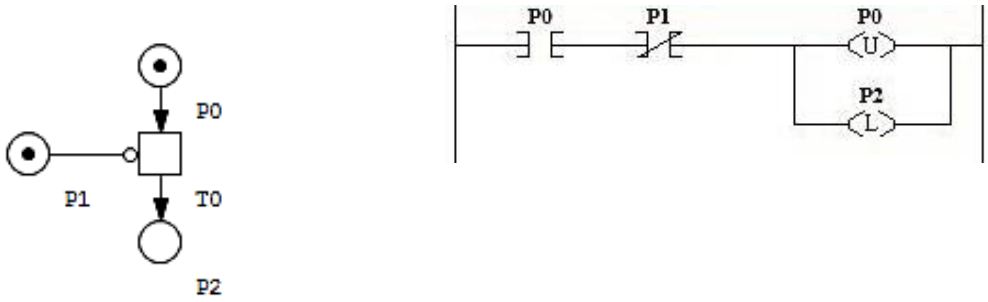


(a) (b)

Figura 3.24 Ejemplo de transformación de una red de Petri no segura. (a) Red de Petri. (b) Código Ladder. ***Fuente: Propia***

En la figura 3.25 se presenta la transformación de una red de Petri con arco de inhibición. Si bien, rigen los mismos principios utilizados en los dos ejemplos anteriormente descritos, se adiciona un contacto de tipo cerrado que representará al arco inhibidor que conecta el

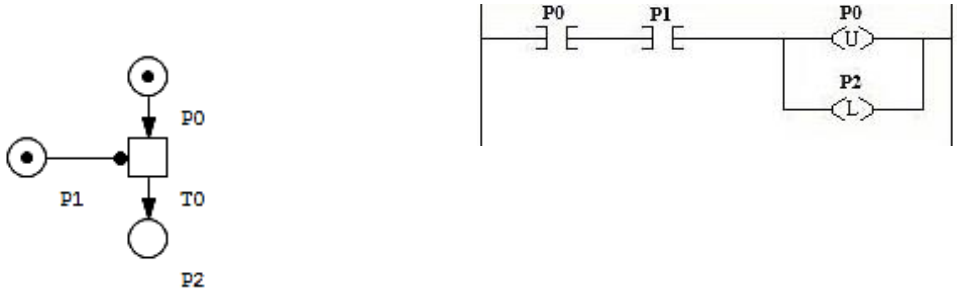
lugar P1 a la transición T0. Por tanto, si P1 se encuentra energizado, se evitará el disparo de la transición T0, y por ende del encendido de P2 y el apagado de P0.



(a) (b)

Figura 3.25 Traducción de una red de Petri con arco de inhibición. (a) Red de Petri. (b) Código Ladder. ***Fuente: Propia***

En la figura 3.26 se presenta la transformación de una red de Petri con arco de lectura. En esta ocasión, el contacto añadido es de tipo abierto que representará al arco de lectura que conecta el lugar P1 a la transición T0. Por tanto, es condición necesaria para el disparo de la transición T0 que P1 se encuentre energizado, y por ende, realice el encendido de P2 y el apagado de P0.



(a) (b)

Figura 3.26 Traducción de una red de Petri con arco de lectura. (a) Red de Petri. (b) Código Ladder. ***Fuente: Propia***

2. Escritura de las líneas que representen el valor inicial de marcaje de la red de Petri.

En este caso, el valor inicial de marcaje de cada lugar será representado por una bandera bit de solo dos valores (0 y 1) si la red de Petri es segura. En caso de que la red de Petri no sea

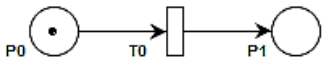
segura, se utilizarán variables de tipo entero para representar los lugares de la red de Petri con un marcaje superior a 1.

3. Configuran los valores iniciales de los temporizadores de la red de Petri.

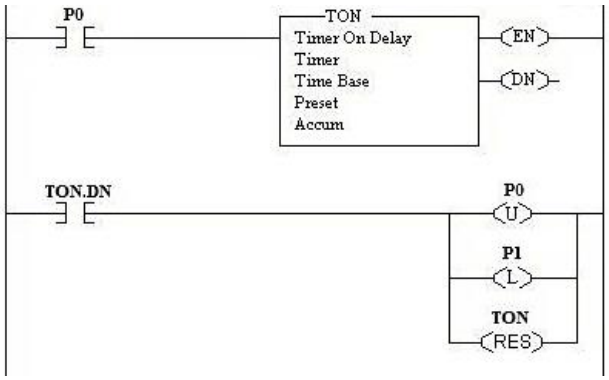
Por cada transición temporizada existente en la red de Petri, tendrá asociado dentro de la lógica Ladder un temporizador con su respectivo valor de temporización (Preset). La base de tiempo será de 1ms y el valor de temporización será igual a aquel número que indique la transición temporizada (dado en unidades de segundo).

Al culminar el conteo del temporizador, se activará una señal de finalización (DN) en el código Ladder que deberá energizar los elementos lógicos asociados a ella. Esta línea contendrá a su vez una instrucción de reseteo del contador (Reset) para permitir en una ocasión posterior su activación.

En la figura 3.27 (a) se presenta un ejemplo de transformación de una red de Petri con una transición temporizada y en la figura 3.27 (b) su correspondiente código Ladder.



(a)



(b)

Figura 3.27 Traducción de una red de Petri con transiciones temporizadas. (a) Red de Petri. (b) Código Ladder. ***Fuente: Propia***

Para este elemento temporizado, se requiere del uso de un bloque de función Timer que es activado en el momento en el que P0 sea energizado. En ese momento, el Timer empezará a correr hasta alcanzar el valor de Preset y activar un bit denominado TON.DN. Con la activación de este bit, se provoca el cambio de estado encendiendo P1 y apagando P0.

4. Configuración de las entradas discretas asociadas a las transiciones.

En varios procedimientos es común encontrar que determinados eventos activados de manera simultánea pueden desencadenar un cambio de estado en un sistema[33]. Esto hace referencia a la capacidad de asociación de una señal de entrada de un controlador a una transición en una red de Petri.

En la aplicación software PGC se pueden asociar varias condiciones de tipo discreto a una transición de una red de Petri. Una condición de este tipo puede contener una o varias entradas asociadas por medio de un operador AND y/o NOT AND (NAND) en cualquier caso, cada condición puede agrupar una o varias entradas de tipo discreto las cuales son señales provenientes del módulo de E/S discreto de un controlador programable, y por tanto, son representadas en la lógica Ladder por medio de un contactores normalmente abiertos cuando el operador asociado a la entrada es de tipo AND o es un contactor normalmente cerrado cuando el operador asociado a la entrada es de tipo NOT AND.

La figura 3.28 representar el paso de una pieza de trabajo de una zona A hacia una zona B, para lo cual se requiere de la activación de dos tipos de sensores I0.0 e I0.1. Ambos sensores representan respectivamente: “Transportador disponible” y “Lugar B disponible”.

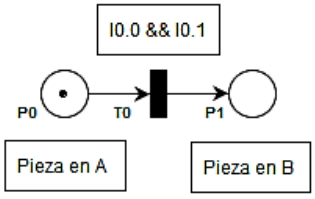


Figura 3.28 Representación del traslado de una pieza de A hasta B. ***Fuente: Propia***

En la figura 3.29 se presenta la transformación a código Ladder de la red en la figura 3.28. Como se puede observar, se encuentran incluidas las asociaciones de las entradas de tipo discreto I0.0 e I0.1 de un controlador programable como contactores de tipo abierto. Por tanto, al encontrarse energizadas las entradas del controlador junto al bit P0, se encenderá P1 y apagará P0.

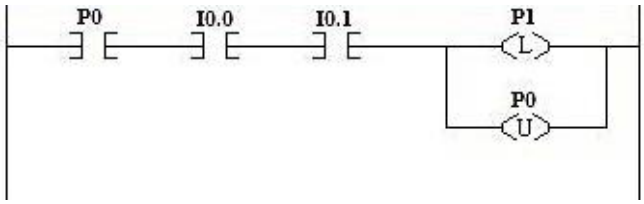


Figura 3.29 Traducción de una red de Petri con asociación múltiple del traslado de una pieza de A hasta B. *Fuente: Propia*

5. Configuración de las salidas discretas asociadas a las transiciones.

De manera análoga, un lugar de una red de Petri puede asociar varias acciones de tipo discreto a una transición de una red de Petri. Estas acciones permiten la energización de una o varias salidas provenientes del módulo de E/S discreto de un controlador programable. En este caso, su representación en la lógica Ladder es dada por un grupo de energizadores directos según sea el número de salidas asociadas a un lugar de la red de Petri.

En el siguiente caso presentado, la llegada de una marca en un lugar de una red de Petri puede significar la activación no solo de una, sino de dos o más acciones [33]. Se requiere de la capacidad de asociación de una o varias señales de salida de un controlador a un lugar de la red de Petri. En la figura 3.30 se está representando el llenado de un tanque por medio de dos motobombas “Motobomba1” y “Motobomba2” luego de activarse la señal de “Inicio Llenado”.

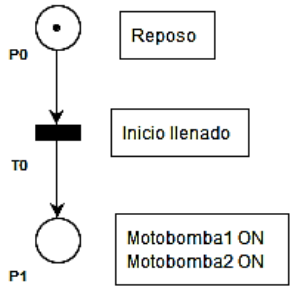


Figura 3.30 Representación del llenado de un tanque por medio de dos motobombas. *Fuente: Propia*

En la figura 3.31 se presenta la transformación de la RdP de la figura 3.30, en la cual se encuentra representado el comportamiento de una red de Petri con asociación múltiple del llenado de un tanque por medio de dos motobombas. En su transformación a Ladder, la señal discreta de entrada del controlador de Inicio Llenado es ubicada como requisito para la activación de P1 (además de activación de P0). Por otro lado, al momento de activarse el

bit P1, las señales discretas denominadas como Motobomba1 ON y Motobomba2 ON son energizadas por el controlador.

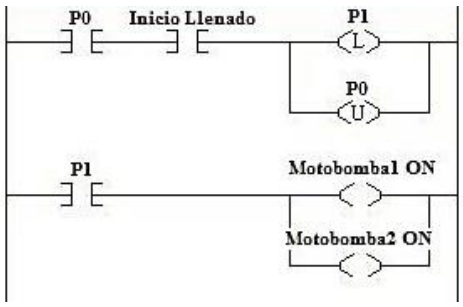


Figura 3.31 Traducción de una red de Petri con asociación múltiple del llenado de un tanque por medio de dos motobombas. ***Fuente: Propia***

6. Generación del número de líneas del Ladder.

El número de líneas de Ladder depende directamente del número de transiciones existentes en la red de Petri y del número de lugares que tienen acciones asignadas. Dentro de cada una de las líneas del código serán ubicados cada uno de los elementos gráficos ya construidos en los pasos anteriores.

En resumen, el desarrollo de este procedimiento inicia con la traducción de cada valor de las matrices de incidencia, matrices de inhibición y de activación en el cual, cada valor distinto de cero en cada una de las matrices representa un distinto elemento representable en lenguaje Ladder. Por tanto, como se pudo observar en este procedimiento, el objetivo fundamental de esta técnica es el de decodificar la información de una red de Petri para traducir a elementos gráficos dentro de la estructura lógica de tipo Ladder.

En [39] será posible encontrar tablas en las cuales se encuentra consignada la sintaxis utilizada por la aplicación RsLogix 5000 para la representación de código Ladder a nivel gráfico.

3.4.2. Adaptaciones y modificaciones con respecto al algoritmo de programación Ladder de CRP

A continuación se tratarán la forma en la cual fueron adaptadas las modificaciones de la metodología TPL realizadas por [22], lo cual servirá para abordar sobre las características que diferencian a un algoritmo de programación sobre otro. Esta comparación es meramente ilustrativa y servirá para aclararle al lector sobre las dimensiones en que profundiza tanto PGC como CRP.

Dentro de las modificaciones y extensiones realizadas por [22] a la metodología TPL para su implementación en programas desarrollados en RsLogix500 se encuentran los siguientes aspectos:

- Escritura del marcaje inicial desde el archivo de programa.
- Traducción de los eventos generados por un cambio de estado en el que las transiciones solo son disparadas una sola vez.
- Traducción de los eventos generados por entradas apagadas.

Para el caso del desarrollo de proyectos generados en RsLogix5000, estas características no revistieron de mayores modificaciones salvo en la ubicación del archivo L5K. Más concretamente, RsLogix5000 provee de la capacidad de incluir TAGs de tipo global y local en sus proyectos. Por tanto, esto requiere que cada nuevo programa traducido contenga los marcajes iniciales de red dentro de la estructura del L5K al inicio de cada rutina.

El segundo ítem describe el comportamiento de aquellos eventos en los que un cambio de estado, equivale a un solo disparo en la transición y luego requieren de un nuevo cambio de estado para poder dispararse. Esta característica de disparo único, se ha incluido dentro del software por medio de una instrucción "One Shot". Para seleccionar esta opción, es necesario que se seleccione esta opción dentro de la interfaz gráfica de PGC tal como se puede observar en la figura 3.32 dentro de la asignación de condiciones a una transición.

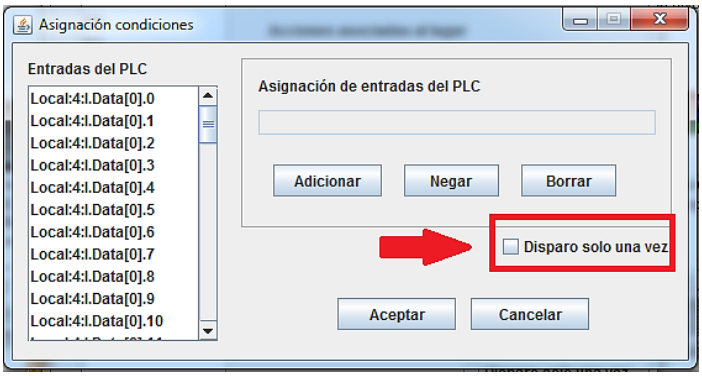


Figura 3.32 Asociación de un disparo único a una transición de una red de Petri por medio de PGC. *Fuente: Propia*

Finalmente, los eventos que requieran de la implementación de entradas apagadas podrán utilizar la interfaz PGC para la inclusión de esta característica tal como se puede observar en la figura 3.33. En la que a una transición de la red de Petri se asigna una condición de negación a una entrada discreta del programador controlable. Adicionalmente se puede

observar como se ha aumentado la capacidad de asociación de una transición para que pueda incluir una condición lógica que agrupe varias entradas, tal como se indicó en la figura 3.20 (b). Esto supera la asociación 1 a 1 que existía en CRP y que limitaba el diseño de las redes de Petri en caso de requerirse que varios eventos a la vez activasen un estado del sistema.

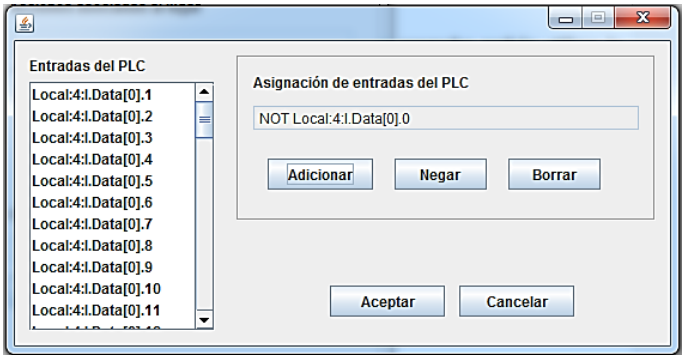


Figura 3.33 Asociación de una entrada apagada a una transición de una red de Petri por medio de PGC. *Fuente: Propia*

Todas las características adaptadas a PGC se encuentran resumidas en el esquema comparativo de la Tabla 3.1. En la columna de la izquierda se podrá observar aquellas características originales en CRP y a su derecha las adaptaciones realizadas estas características.

Tabla 3.1 Esquema comparativo de las adaptaciones realizadas a PGC. *Fuente: Propia*

Monografía C. Buchely y F. Coque	Monografía actual
Escritura del marcaje inicial desde el archivo de programa.	La escritura del marcaje inicial se realiza en la declaración e inicialización de las TAGs de cada programa.
Traducción de los eventos generados por un cambio de estado en el que las transiciones solo son disparadas una sola vez.	Las transiciones también pueden incluir el disparo una sola vez para eventos que así lo requieran por medio de una instrucción OSN (One shot).
Traducción de los eventos generados por entradas apagadas.	Los eventos de tipo discreto representados por entradas del programador controlable podrán agruparse como una condición lógica a cada transición de la red de Petri.

Hay que recordarle finalmente al lector, que este trabajo de tesis no busca reemplazar el uso de la aplicación CRP para la programación en lenguaje Ladder ya que su enfoque está en el diseño de supervisores por medio de algoritmos computacionales, los cuales, no fueron implementados en Java ya que requieren de una mayor carga computacional. En cambio, en

este trabajo se busca facilitar la implementación de la lógica de fases del modelo ISA-88, sacando provecho del modelado mediante RdP, tal el código de control que sea compatible con familias más actuales de PLCs.

3.4.3. Ventajas y limitaciones de la técnica de traducción a código Ladder

A continuación se ilustrarán las ventajas y desventajas de trabajar con esta técnica de traducción.

Ventajas de la técnica de traducción:

- La técnica de traducción TPL está establecida como la técnica más efectiva y eficaz para la traducción de redes de Petri.
- Permite la representación de redes de Petri seguras y no seguras.
- Permite la implementación de supervisores en redes de Petri.

Limitaciones de la técnica de traducción:

- Su interfaz gráfica no es fácilmente asimilable por el programador en comparación a los diagramas de bloques funcionales o al lenguaje SFC.
- Con extensas líneas de código de programación el lenguaje Ladder no es fácilmente depurable.

En la sección siguiente se abordará la opción de traducción a código SFC como complemento al diseño de lógica de control para contrarrestar estas carencias y limitaciones encontradas en este lenguaje de programación.

3.5. Algoritmo de traducción de Red de Petri a código SFC

Los diagramas SFC derivan de las redes de Petri y del lenguaje Grafcet como una herramienta para describir una secuencia de operaciones, y especificar relaciones entre entradas y salidas dentro de un proceso. Los modelos en SFC contienen una representación gráfica similar a un diagrama de flujo, la cual facilita la organización, control y programación de recetas para sistemas batch [40].

La inclusión del lenguaje SFC en la IEC61131-3 a inicios de la década de los 90, permitió la adaptación de muchos elementos de Grafcet con algunas diferencias (por ejemplo, en SFC solo se permite un estado inicial a diferencia de Grafcet que permite un número de estados iniciales mayor igual a uno) [41]. Sin embargo, en determinados casos es muy

común encontrar en la literatura un uso indistinto de ambos términos. Para el caso de esta monografía, SFC hará referencia al lenguaje de programación y Grafcet como una metodología de diseño a eventos discretos derivada de las redes de Petri [42].

Un diagrama en SFC permite una mejor documentación del proceso, y permite a las personas no expertas entender más fácilmente la lógica de control en comparación a otros lenguajes de programación que no poseen una fuerte estructura gráfica [43]. Por tanto, el objetivo de presentar un algoritmo de traducción a SFC es el de abarcar un lenguaje de programación que pueda representar de una manera mucho más próxima la dinámica de las redes de Petri [3].

A continuación se describe una técnica de traducción de las características gráficas y comportamentales de una red de Petri a un diagrama SFC, junto con una descripción paso a paso de la ejecución del algoritmo de programación, y las restricciones requeridas en el diseño de las redes de Petri.

3.5.1. Características del algoritmo de traducción a código SFC

Los modelos gráficos en SFC son construidos por etapas representadas como cuadrados, y transiciones dibujadas como líneas. En la ejecución del sistema, la etapa inicial será la única que deberá estar activa, y gráficamente se representa por un doble cuadro. Una etapa puede estar activa o inactiva, y de acuerdo a esto, determina la situación actual del sistema. Cada etapa podrá tener una o varias acciones asociadas, y estas se ejecutan cuando la etapa se activa.

Una transición es utilizada para conectar etapas. Una transición es activada si todas las etapas precedentes a la transición están activas. Cada transición tiene una receptividad (la cual puede ser una condición lógica o un evento), y cuando ésta es verdadera, la transición es disparada. Al dispararse la transición, las etapas precedentes a ésta se desactivan y las etapas posteriores a la transición son activadas.

Existen dos tipos de calificadores para las acciones: de nivel y de impulso. Una acción de nivel es modelada por una variable booleana y tiene una duración finita (el mismo tiempo que la etapa permanezca activa). Por otro lado, la acción de tipo impulso se activa tan pronto como la etapa pase de inactiva a activa.

Finalmente, la representación gráfica de los elementos de un diagrama SFC se puede observar en la figura 3.34.

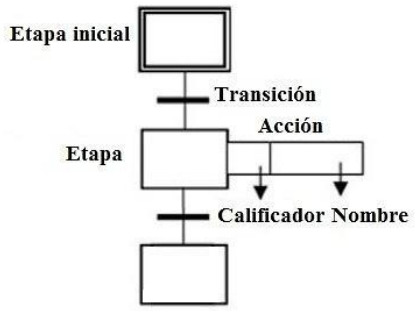


Figura 3.34 Elementos básicos de un SFC. *Fuente:* [40]

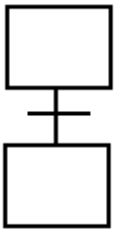
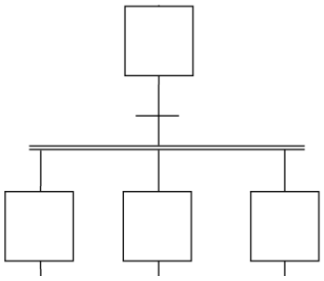
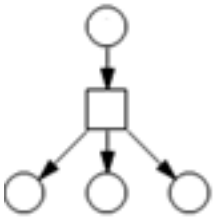
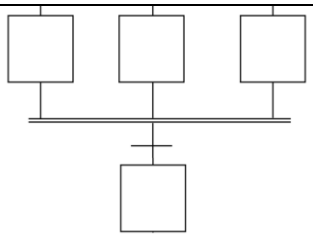
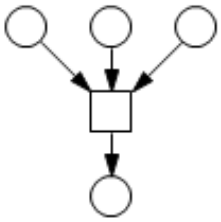
Para la transformación de una red de Petri a SFC, se requiere que ambas estructuras gráficas sean equivalentes. Se ha determinado que una red de Petri puede ser definida de tal forma que su comportamiento de entrada/salida sea igual al comportamiento de entrada/salida de un SFC [44]. Sin embargo, existen dos diferencias importantes en ambos formalismos que deben ser abordadas antes del proceso de traducción:

- Marcaje: una red de Petri puede contener un marcaje de números enteros positivos, en tanto SFC utiliza valores booleanos para determinar la activación de sus etapas.
- Regla de disparo ante conflictos: En una RdP no está definido el disparo simultáneo de transiciones, por lo que las implementaciones de una RdP deben considerar políticas de asignación de prioridades para estos casos. Por el contrario, en SFC una transición es disparada de acuerdo a su prioridad, o en caso de no existir prioridad, las transiciones se dispararán simultáneamente.

Cuando la red de Petri en cualquier marcaje alcanzable, los lugares solo pueden llegar a tener una marca (segura), la primera diferencia no existe. Por otro lado, si para cualquier marcaje alcanzable un grupo de transiciones en conflicto tiene una receptividad que no las hace verdaderas al mismo tiempo, entonces su comportamiento es determinista, y la segunda diferencia no existe. Por tanto, una red de Petri de tipo binaria, acotada con transiciones disparadas en forma determinista es considerada SFC [44].

En el esquema de la Figura 3.35 se encuentran consignados los elementos gráficos comunes entre ambas herramientas de diseño que permitirán posteriormente realizar la traducción de la metodología de diseño de redes de Petri al lenguaje de programación en SFC [43].

SFC	Red de Petri
	

Etapa inicial	*Lugar con marcaje inicial*
	
Etapa	*Lugar*
	
Condición	*Transición*
	
Conectores en un SFC	*Arcos de conexión en una red de Petri*
	
Rama divergente de simultaneidad	*Simultaneidad en redes de Petri*
	
Rama convergente de simultaneidad	*Sincronización en redes de Petri*

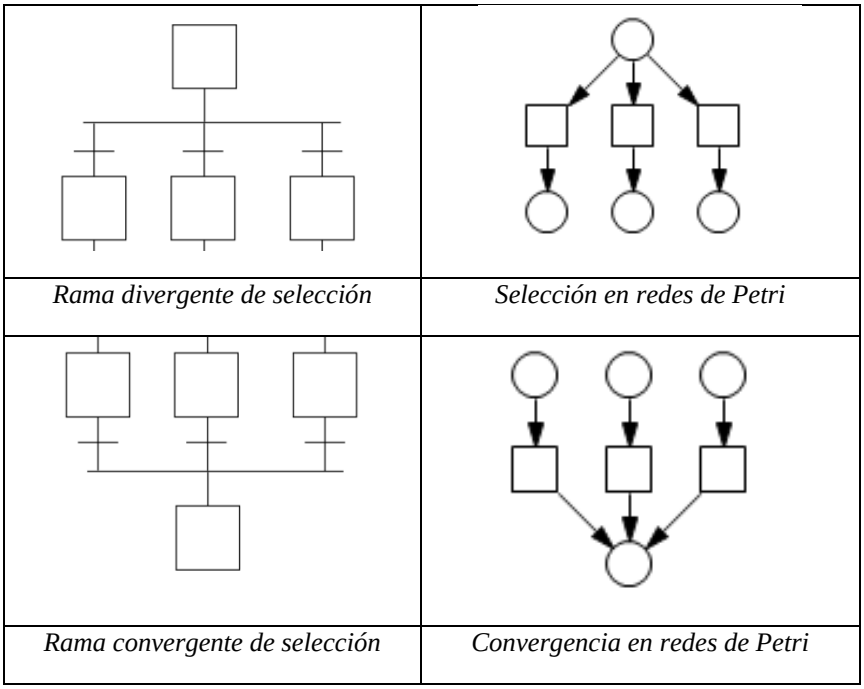


Figura 3.35 Elementos gráficos comunes entre SFC y Redes de Petri. *Fuente:* [43]

3.5.2. Restricciones a los modelos en redes de Petri para el paso a SFC

En esta monografía se proponen limitaciones en el diseño de las redes de Petri como una primera aproximación a una técnica de traducción que permita de manera clara y segura, la traducción equivalente en SFC. Se espera que en posteriores trabajos de grado, esta técnica de traducción se aborde con mayores elementos de alcance. Por el momento, los ítems mencionados a continuación hacen referencia a aquellos aspectos a tener en cuenta en el diseño de las redes de Petri:

- Las redes de Petri deberán estar compuestas por un lugar de inicio (source place) y un lugar final (sink place). Esta condición sirve para asegurar que existe un solo lugar con marca que representará la etapa inicial del SFC. Esto implica que el vector de marcado solo puede tener un único valor en 1, y los demás en 0.

- El marcaje de una red de Petri no podrá ser mayor a 1 en cada lugar, y tal como se analizó anteriormente, esto asegurará una equivalencia entre los SFC implementados y las redes de Petri anidadas a 1-token (seguras). Por ende, esto también implica el uso de arcos ordinarios con un valor de peso de 1.

- El algoritmo de traducción a SFC no posee la capacidad de traducir redes de Petri con jerarquías (uso de macroetapas) o redes de Petri con extensiones (arcos inhibidores y de activación), por tanto no podrán ser implementadas.

- La red de Petri deberá ser viva (ausencia de puntos muertos sin importar la secuencia de disparo elegida), con lo cual, se evita la transformación de redes de Petri con bloqueos.

Para este último caso, se podría plantear la transformación de redes de Petri con lugares de control tal como está definido en [45]. Sin embargo, en la programación, se requiere del uso de múltiples diagramas SFC implementados en programadores controlables diferentes. Por lo que la interacción entre SFC se realiza por medio de la sincronización de las transiciones de cada SFC (transiciones compuestas por idénticas condiciones de disparo). A nivel de programación se ha hecho referencia a la dificultad de alcanzar en la práctica un alto grado de sincronización por los retardos de comunicación entre los controladores. Por tal razón, el uso de supervisores para SFC no está contemplado en esta monografía.

Además, se podría requerir la transformación de redes de Petri no seguras a seguras. En [3] se ha desarrollado una técnica para la transformación de redes de Petri a SFC por medio del uso de variables enteras positivas que se van sumando o restando en cada ejecución de una etapa, los cuales requieren de la inclusión de un mayor número de lugares y transiciones en los diagramas SFC. Este método se espera que sea abordado en una posterior oportunidad a nivel de traducción.

3.5.3. Pasos para la generación de código SFC

Para trabajar con la técnica de traducción a SFC se utiliza la siguiente información de las redes de Petri:

- Vector de marcado inicial.
- Matrices de incidencia hacia adelante y de incidencia hacia atrás.
- Asociación de entradas y salidas discretas a las transiciones y lugares de la red de Petri, respectivamente.
- Vector de posiciones en (x, y) de los lugares y transiciones de una red de Petri (generados en PIPE o Snoopy).

A continuación, se describirán los pasos para la formación de código de programación en SFC:

1. Creación de las etapas, transiciones y acción del SFC.

Cada lugar de una red de Petri genera un equivalente en SFC llamado etapa (step), con la información de un ID, y de su posición (x, y) en el plano proveniente del editor de redes de

Petri. Cada acción tiene asignada una o varias acciones según la información proveniente de la interfaz gráfica de usuario (GUI) de PGC, las cuales indicarán la operación u operaciones que son ejecutadas cuando la etapa es activada. Finalmente, de acuerdo al vector de marcado inicial, se asignará una sola etapa de inicio a un SFC.

Análogamente, una transición de una red de Petri genera un equivalente en SFC llamada transición (transition), la cual posee la información de un ID, y de su posición (x, y) en el plano proveniente del editor de redes de Petri. Cada Transición tiene asignada una o varias condiciones de disparo según la información proveniente de la GUI de PGC, las cuales son de lógica booleana.

2. Creación de las convergencias y divergencias de tipo AND y OR.

Aquellas relaciones de simultaneidad en SFC están representadas con divergencias y convergencias de tipo AND, mientras que las relaciones de convergencia están representadas por medio de divergencias y convergencias de tipo OR.

Una divergencia de tipo AND es creada al existir dos o más unos en una columna de la matriz de incidencia hacia adelante. En una red de Petri, esto implica que el disparo de una transición extrae la marca de un lugar para ubicarla en dos o más lugares asociados a ella. Por tanto, la divergencia de tipo AND en SFC, permite la conexión de una transición con varias etapas, y cuando la transición sea disparada, todas las etapas conectadas a la divergencia serán activadas.

Una convergencia de tipo AND es creada al existir dos o más unos en una columna de la matriz de incidencia hacia atrás. En una red de Petri, esto implica que el disparo de una transición, extrae una marca de todos los lugares asociados a ella, y la ubica en un solo lugar. Por tanto, la convergencia de tipo AND en SFC, permite la conexión de varias etapas a una transición, y solo permitirá su disparo cuando todas las etapas conectadas a la convergencia sean activadas.

Una divergencia de tipo OR es creada al existir dos o más unos en fila de la matriz de incidencia hacia atrás. En una red de Petri, esto implica que un lugar está conectado a dos o más transiciones dentro de la red, y el disparo de una de ellas extraerá una marca de este lugar solo se logrará si cumple sus condiciones lógicas asociadas. Por tanto, la divergencia de tipo OR en SFC, permite la conexión de una etapa con varias transiciones, y solo una de ellas podrá ser disparada.

Una convergencia de tipo OR es creada al existir dos o más unos en fila de la matriz de incidencia hacia adelante. En una red de Petri, esto implica que un lugar está conectado a dos o más transiciones dentro de la red, y el disparo de una de ellas ubicará una marca en

este lugar solo si es activada antes que las demás transiciones. Por tanto, la convergencia de tipo OR en SFC, permite la conexión de varias transiciones a una etapa, y solo el disparo de una transición activa la etapa del SFC.

Por tanto, en este punto se generan las conexiones entre etapas y transiciones en SFC que, por otro lado, involucran simultaneidad y selección en redes de Petri.

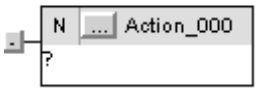
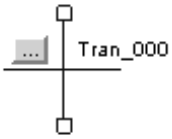
1. Creación de los arcos conectores de cada elemento gráfico.

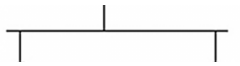
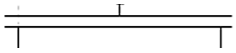
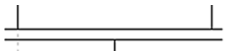
Para realizar la conexión entre los elementos restantes de etapas y transiciones que no hacen parte de una divergencia o una convergencia, se utilizan las matrices de incidencia hacia adelante y hacia atrás. Con la matriz de incidencia hacia adelante se generan los arcos de etapas a transiciones, y con la matriz de incidencia hacia atrás se generan los arcos de transiciones a etapas.

En la Tabla 3.2 se encuentra consignada la sintaxis utilizada por la aplicación RsLogix 5000 para la representación de los diagramas SFC. Cada elemento gráfico tiene asignado distintos atributos, los cuales se encuentran consignados en la columna Configuración del elemento.

Tabla 3.2 Instrucciones en RsLogix5000 para la generación de código SFC. *Fuente:* [39]

Nombre	Representación gráfica	Instrucción	Configuración del elemento
Etapa		**Step**	**ID**: Valor de identificación de la etapa. **Posición en X**: Indica la posición gráfica de la etapa en el eje x. **Posición en Y**: Indica la posición gráfica de la etapa en el eje y. **Operando**: TAG asignada a la etapa. **Ocultar descripción**: Indica la aparición de la descripción o no al usuario. **Descripción en X**: Indica la posición de la descripción en el eje x. **Descripción en Y**: Indica la posición de la descripción en el eje y. **Ancho de la descripción**: Describe la ocupación gráfica de la descripción.

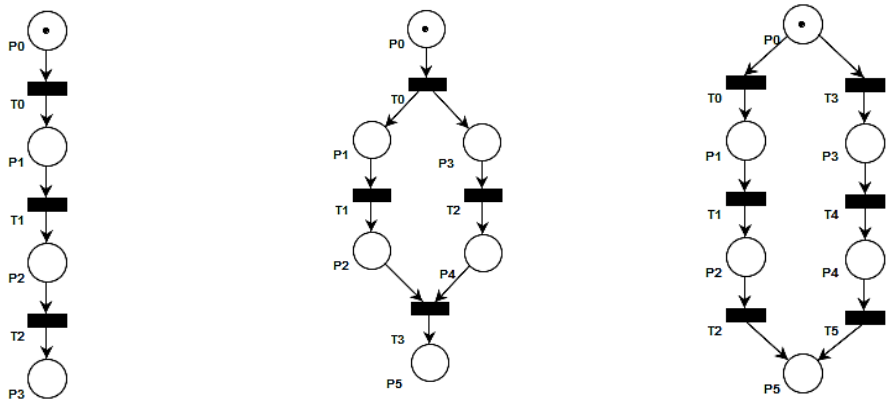
Acción		**Action**	**Operando**: TAG asignada al elemento. **Calificador**: Indica el modo en el que se ejecuta cada acción. En particular se deja la opción activa "N: Non-Stored" la cual ejecuta la acción en tanto la etapa esté activa. **Es booleano**: Indica si la acción es de tipo booleano o no. **Preset con expresión**: Indica si el contador interno de la acción tiene asociada una expresión matemática. **TAG indicadora**: Describe la presencia de una TAG de tipo indicador.
Transición		**Transition**	**ID**: Valor de identificación de la transición. **Posición en X**: Indica la posición gráfica de la transición en el eje x. **Posición en Y**: Indica la posición gráfica de la transición en el eje y. **Operando**: TAG asignada a la transición. **Ocultar descripción**: Indica la aparición de la descripción o no al usuario. **Descripción en X**: Indica la posición de la descripción en el eje x. **Descripción en Y**: Indica la posición de la descripción en el eje y. **Ancho de la descripción**: Describe la ocupación gráfica de la descripción. **Condición**: Se describen todas las condiciones lógicas asociadas a la transición.
Finalización		**Stop**	**ID**: Valor de identificación del elemento finalización. **Posición en X**: Indica la posición gráfica de la finalización en el eje x. **Posición en Y**: Indica la posición gráfica de la finalización en el eje y. **Operando**: TAG asignada al elemento finalización. **Ocultar descripción**: Indica la aparición de la descripción o no al usuario.

			Descripción en X: Indica la posición de la descripción en el eje x. **Descripción en Y**: Indica la posición de la descripción en el eje y. **Ancho de la descripción**: Describe la ocupación gráfica de la descripción.
Rama divergente de selección		**Branch Selection converge**	**ID**: Valor de identificación de la rama. **Posición en Y**: Indica la posición de la rama en el eje y. **Tipo de rama**: Escoge entre selección o simultaneidad. **Tipo de flujo**: Escoge entre divergencia o convergencia.
Rama divergente de simultaneidad		**Branch Selection diverge**	
Rama convergente de selección		**Branch Simultaneous converge**	
Rama convergente de simultaneidad		**Branch Simultaneous diverge**	

3.5.4. Ejemplos de generación de código SFC

Para mostrar el funcionamiento de la técnica de traducción a código SFC visto desde la plataforma de RsLogix 5000, se presentan tres representaciones elementales de redes de Petri:

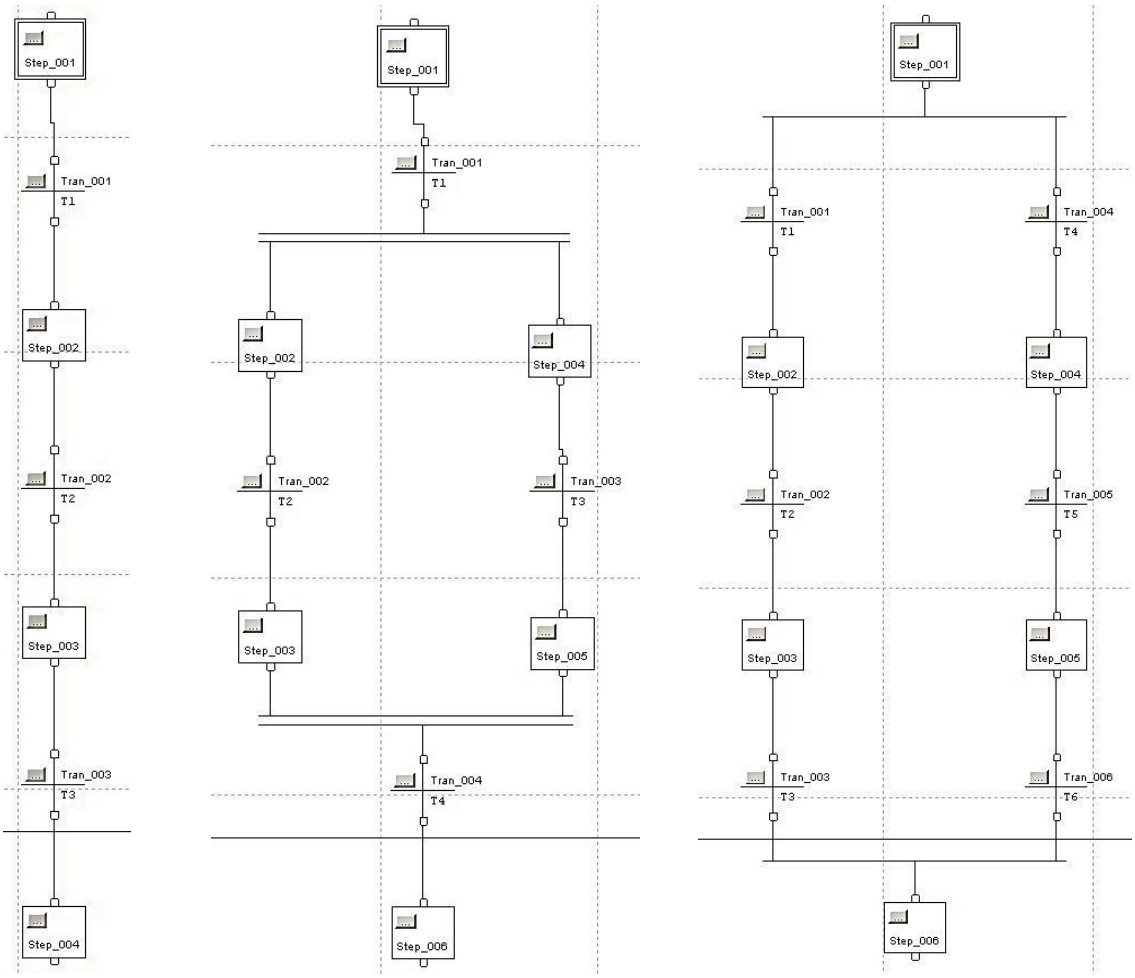
- Relación de precedencia en la figura 3.36 (a): Una secuencia simple con etapas seguidas de transiciones.
- Procesos en paralelo en la figura 3.36 (b): La ejecución puede tomar múltiples alternativas según sus condiciones externas.
- Procesos simultáneos en la figura 3.36 (c): Ejecución concurrente de varias etapas.



(a) (b) (c)

Figura 3.36 Representaciones elementales de las redes de Petri. (a) Relación de precedencia. (b) Procesos en paralelo. (c) Procesos simultáneos. ***Fuente: Propia***

Al observar la traducción del caso de la relación de precedencia de la figura 3.37 (a), se pueden denotar conectores entre cada bloque de forma secuencial. Tanto etapas como transiciones inician su conteo de ID desde uno, esta es la razón por la cual el lugar P0 es representado como Step_001, hasta P3 el cual es representado como Step_004. De igual forma, las transiciones T0 y T2 son llamadas Tran_001 y Tran_003 respectivamente. Sin embargo, en los casos de procesos en paralelo y simultáneos, se deben generar divergencias y convergencias que permitan conectar más de una transición a una etapa y viceversa.



(a) (b) (c)

Figura 3.37 Representaciones elementales de las redes de Petri. (a) Relación de precedencia. (b) Procesos en paralelo. (c) Procesos simultáneos. ***Fuente: Propia***

En la figura 3.38 se puede observar el análisis matemático de la red de Petri representativa del proceso en paralelo de la figura 3.37 (b). El algoritmo detecta en la matriz de incidencia hacia adelante una columna con dos unos asociada a la transición T0 (llamada Tran_001 en SFC), lo que determina la creación de una rama divergente OR. Análogamente, el algoritmo detecta en la matriz de incidencia hacia atrás una columna con dos unos asociada a la transición T3 (llamada Tran_004 en SFC), lo que determina la creación de una rama convergente OR.

	T0	T1	T2	T3
P0	0	0	0	0
P1	1	0	0	0
P2	0	1	0	0
P3	1	0	0	0
P4	0	0	1	0
P5	0	0	0	1

(a)

	T0	T1	T2	T3
P0	1	0	0	0
P1	0	1	0	0
P2	0	0	0	1
P3	0	0	1	0
P4	0	0	0	1
P5	0	0	0	0

(b)

Figura 3.38 Análisis matemático de los procesos en paralelo. (a) Matriz de incidencia hacia adelante. (b) Matriz de incidencia hacia atrás. *Fuente: Propia*

En la figura 3.39 se puede observar el análisis matemático de la red de Petri representativa del proceso simultáneo de la figura 3.37 (c). El algoritmo detecta en la matriz de incidencia hacia adelante una columna con dos unos asociada al lugar P0 (llamada Step_001 en SFC), lo que determina la creación de una rama divergente AND. Análogamente, el algoritmo detecta en la matriz de incidencia hacia atrás una columna con dos unos asociada a la transición P5 (llamada Step_006 en SFC), lo que determina la creación de una rama convergente AND.

	T0	T1	T2	T3	T4	T5
P0	0	0	0	0	0	0
P1	1	0	0	0	0	0
P2	0	1	0	0	0	0
P3	0	0	0	1	0	0
P4	0	0	0	0	1	0
P5	0	0	1	0	0	1

(a)

	T0	T1	T2	T3	T4	T5
P0	1	0	0	1	0	0
P1	0	1	0	0	0	0
P2	0	0	1	0	0	0
P3	0	0	0	0	1	0
P4	0	0	0	0	0	1
P5	0	0	0	0	0	0

(b)

Figura 3.39 Análisis matemático de los procesos simultáneos. (a) Matriz de incidencia hacia adelante. (b) Matriz de incidencia hacia atrás. *Fuente: Propia*

Finalmente, en el siguiente ejemplo se presenta el caso de una máquina de estados (ver figura 3.40) en la cual cada transición solo puede poseer un arco de entrada y de salida. Para verificar la sencillez de los modelos hechos en SFC en comparación a Ladder, en la Figura 3.41 se han representado en ambos lenguajes de programación la ejecución luego de su transformación por PGC.

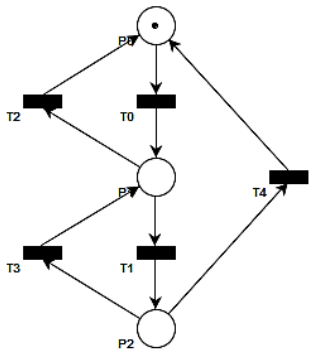
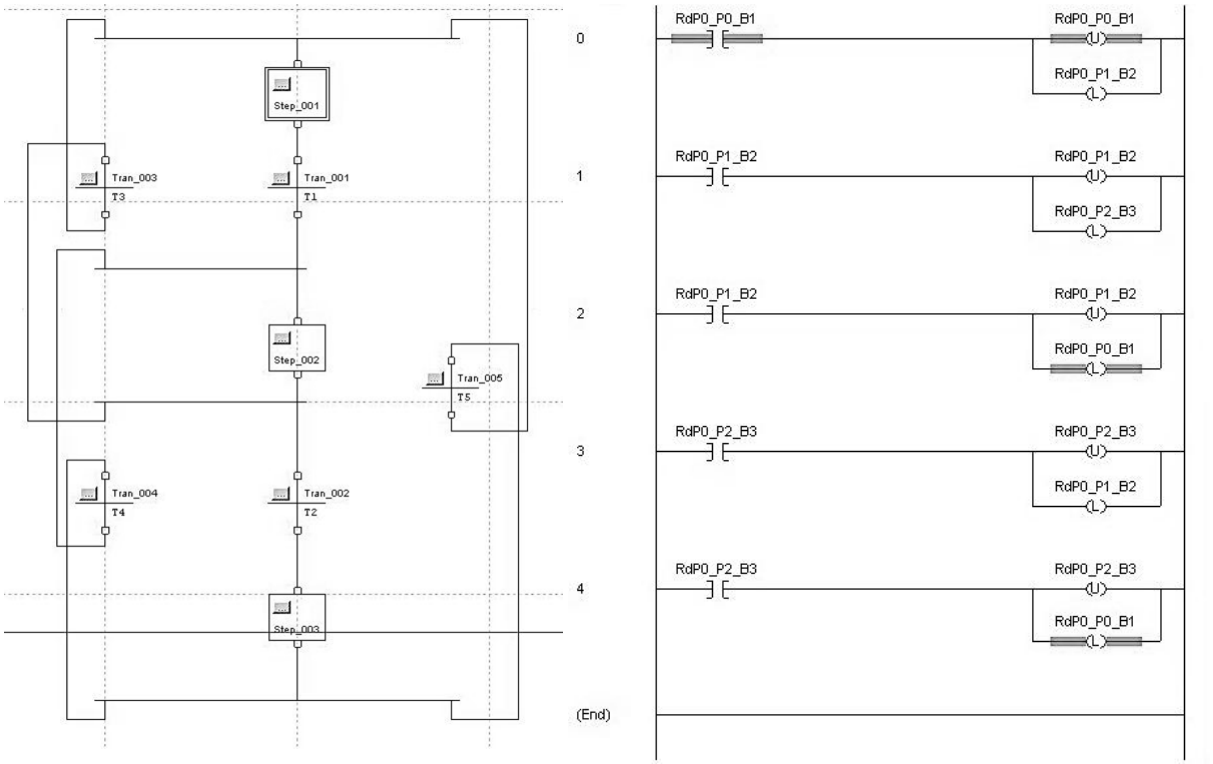


Figura 3.40 Red de Petri de una máquina de estados. ***Fuente: Propia***



(a) (b)

Figura 3.41 Transformación de la máquina de estados. (a) SFC (b) Ladder. ***Fuente: Propia***

Como se puede observar en la figura 3.41, en el paso de una máquina de estados modelada en red de Petri, se puede observar cómo los diagramas SFC conservan una correspondencia

más directa con el comportamiento de una red de Petri en comparación a Ladder. Evidentemente, esto no implica que los lenguajes de programación sean excluyentes. Por el contrario, ambos son complementarios dentro de la estructura de programación, y en el caso que uno no cumpla correctamente con los requerimientos del programador, se puede utilizar un lenguaje para suplir las limitaciones del otro.

3.5.5. Ventajas y limitaciones de la técnica de traducción a código SFC

A continuación se ilustrarán las ventajas principales de trabajar con esta técnica de traducción:

- La traducción de tipo directo genera un equivalente exacto de una red de Petri, mucho más asequible de analizar y modificar en comparación a otros lenguajes de programación.
- Facilidad en la documentación, modificación, ampliación y mantenimiento de la lógica de control.

Limitaciones de la técnica de traducción propuesta en la tesis:

- No permite la implementación de redes de Petri no seguras.
- No permite el desarrollo de supervisores.
- No permite la implementación de las extensiones mencionadas en el segmento de traducción a Ladder.

A pesar de estas considerables restricciones en la traducción de RdP a SFC, en esta sección se ha propuesto un algoritmo computacional para que en posteriores trabajos esta pueda ser aprovechada como base en el desarrollo de nuevas extensiones a esta técnica.

3.6. Comunicación con la aplicación software RsLogix 5000

RsLogix 5000 es una aplicación software para la configuración, programación y supervisión del funcionamiento de los controladores autómatas programables (PAC) de tipo Logix (ControlLogix y CompactLogix). PGC es compatible con la aplicación RsLogix 5000 y por consiguiente, con la solución software basada en FactoryTalk Batch de Rockwell Automation [46]. A pesar de que PGC no genera código compatible con otros fabricantes de controladores programables (Siemens, Schneider, Emerson Industrial Automation, etcétera), cualquier lenguaje de programación basado en IEC 61131-3 permite que todo autómata programable construido bajo las especificaciones de esta norma pueda implementar los códigos de programación generados en dicho lenguaje [47]. Por ende,

existen procedimientos documentados sobre la traducción de programas entre distintos fabricantes, en [47] por ejemplo, se puede encontrar un manual para la conversión de la lógica de control entre Step 5 de Siemens y RsLogix 5000 de Rockwell Automation, tal como se encuentra resumido en las equivalencias del esquema de la Tabla 3.20.

Tabla 3.3 Equivalencias entre distintos elementos de un proyecto de Step 5 y de RsLogix5000. *Fuente:* [47]

	Step 5 de Siemens	RsLogix 5000 de Rockwell Automation
A nivel de organización de un proyecto	Bloques de organización	Tareas
A nivel de programas	Bloques de programa	Rutinas
	Bloques de funciones	
A nivel de lenguajes de programación	Statement List (STL)	Texto estructurado
	Ladder Diagram (LAD)	Diagramas Ladder
	Control System Flowcharts	Diagrama de bloques funcionales
	GRAPH 5	Cartas de funciones secuenciales

3.6.1. Uso de los archivos con extensión L5K

Para realizar el paso del código en PLC a un proyecto generado en RsLogix 5000 se debe de realizar la modificación de un archivo en formato L5K derivado de la función de exportación de la aplicación. Este formato en ASCII está basado en el lenguaje de texto estructurado del estándar IEC 61131-3 para la definición del sistema operativo del controlador y de los tipos de datos utilizados [35]. A continuación, en la figura 3.42 se observa la estructura general de un archivo L5K en blanco.

```
(********************************************

  Import-Export
  Version   := %   Versión de la aplicación software RsLogix 5000   %
  Owner     := %   Nombre del usuario   %
  Exported  := %   Fecha de exportación del archivo   %
*********************************************)
%   Versión del entorno   %

CONTROLLER NombreDelControlador
        %   Configuración del controlador   %


        MODULE NombreDeMódulo
```

```
        %   Configuración del módulo   %
END_MODULE


TAG
        %   Aquí se asigna la información de las TAGs globales   %
END_TAG


PROGRAM NombreDelPrograma
        %   Aquí se asigna la información de los programas   %
END_PROGRAM


TASK NombreDeLaTarea
        %   Aquí se asigna la información de las tareas   %
END_TASK

%   Aquí se asigna la configuración restante del documento   %
```

Figura 3.42 Formato ASCII para el archivo L5K. ***Fuente: Propia***

Las primeras líneas de texto hacen referencia a las configuraciones de carácter general de un archivo, junto con la fecha de creación de este documento de exportación. Posteriormente estará ubicada la información relacionada con el controlador, los módulos asociados al proyecto y las TAGs de asignación de tipo global (estas últimas no son modificadas ya que la traducción asigna TAGs de tipo local). Para añadir líneas de código al formato ASCII referentes a los programas en Ladder y SFC, se procede a la modificación del contenido en PROGRAM, el cual contiene el formato visto en la figura 3.43.

```
PROGRAM NombreDelPrograma
        %   Configuración del programa  %
                TAG
                        %   Asignación de las TAGs locales   %
                END_TAG

                ROUTINE Running
                        %   Programa del PLC   %
                END_ROUTINE

END_PROGRAM
```

Figura 3.43 Configuración de PROGRAM en un archivo L5K. ***Fuente: Propia***

Ambos generadores de código PLC (tanto para Ladder como para SFC) ubican el código generado dentro de la configuración del programa. Por lo cual, al definir las variables de trabajo de tipo booleano y entero (boolean e integer, respectivamente), se utilizarán los

espacios TAG, mientras que para especificar la estructura del programa funcional se utiliza el espacio dentro de ROUTINE. El número de variables utilizados en TAG puede variar y ser diferente para cada programa. Cada Routine maneja un grupo de TAGs, pero cada Routine posee una funcionalidad única. Esto también refiere al hecho de que por un programa puede existir más de una Routine.

Las variables que se pueden encontrar en RsLogix 5000 son asignadas como TAGs. La gran ventaja de trabajar con estos elementos es su facilidad de anotación, no tienen restricciones de nombre y pueden tomar cualquier valor que no esté repetido. En ROUTINE existen 6 implementaciones diferentes basadas en el estándar ISA 88: Running, Stopping, Aborting, Resseting, Restarting. El programa PGC tiene una opción para indicarle al usuario la adición de una instrucción Fin de fase que detiene la rutina y permite el paso a otra. En Ladder es utilizada la instrucción PSC y en SFC se utiliza una instrucción de tipo Stop. La ubicación de esta opción depende de los requerimientos dados al usuario para el diseño de la lógica de control.

4. Capítulo IV: Caso de estudio

Los objetivos de este capítulo están orientados a la aplicación de los conceptos desarrollados en las pasadas secciones a un proceso batch caso de estudio. Para lograr los cometidos, se especificará la lógica para cada una de las fases del proceso en base a las definiciones de la parte 1 del estándar ISA-88 y los esquemas propuestos en el capítulo 2, y posteriormente ser traducidos a lenguaje para PLC.

4.1. El proceso de producción de NABAM

De forma general, las actividades para la producción de Nabam [48] el cual es un fungicida se especifican en la figura 4.1. Para efectos de desarrollo, en este trabajo únicamente se describirá la primera etapa del esquema funcional que corresponde a *síntesis*.

4.1.1. Descripción del proceso de reacción

La síntesis del Nabam puede llevarse a cabo de manera continua o por barch, y tiene lugar a partir de una reacción en medio acuoso de Etilendiamina (EDA) con adición de Bisulfato de carbono (CS2) y soda caustica (NaOH).

La dilución de EDA en agua es exotérmica, tal que con la adición de CS2, el pH tiende a disminuir. Con la adición de NaOH el pH y la temperatura tienden a aumentar, por tanto, es necesario una adecuada dosificación de los dos reactantes (CS2 y NaOH) para lograr el equilibrio en la reacción y así producir Etilenbisditiocarbamato de sodio (Nabam).

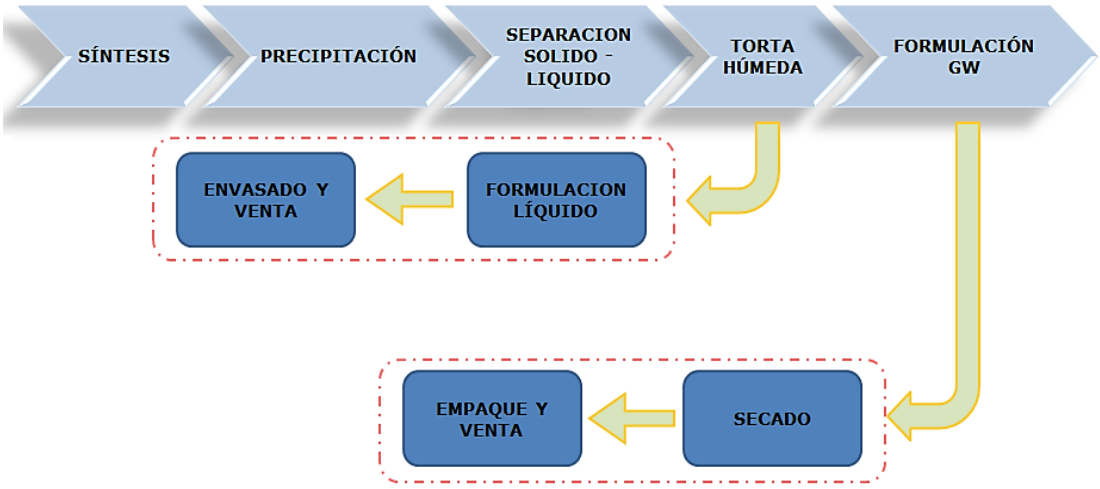


Figura 4.1 Esquema funcional del proceso de Nabam. ***Fuente:*** [48]

El proceso de reacción de Nabam comprende 3 etapas:

- Adicionar EDA y CS2.
- Adicionar agua, NaOH, y agitar.
- Adicionar agua, agitar y transferir.

En la figura 4.2 se muestran las etapas para la síntesis del Nabam.

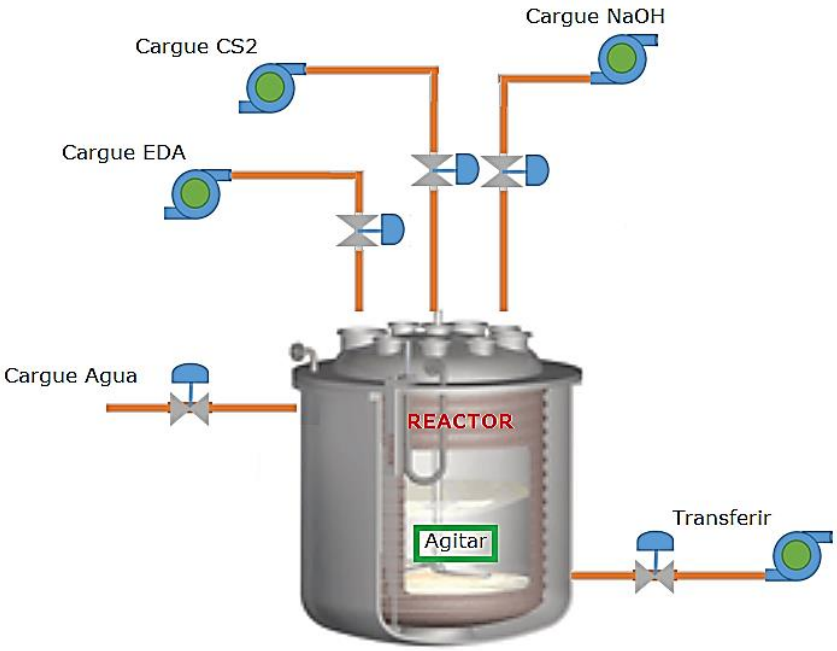


Figura 4.2 Diagrama de proceso. ***Fuente:*** **[48]**

4.2. Modelos del estándar ISA-88 aplicados al proceso de producción de NABAM

El estándar ISA-88 plantea una metodología de diseño jerárquica, con lo cual cada componente de esta estructura supone un grado de complejidad dependiendo del nivel al que pertenezca dentro del proceso productivo. Los modelos deben proveer la información necesaria para representar el proceso de producción, y lo que debe hacer a partir de la desagregación en partes funcionales.

En este ítem se definen las estructuras jerárquicas que representan el proceso de reacción para la obtención del Nabam como se muestra a continuación.

4.2.1. Modelo Físico

El modelo físico del proceso de reacción de Nabam representa de manera jerárquica los equipos que este involucra, teniendo en cuenta la pregunta ¿Qué es capaz de hacer el equipo? Para la definición del modelo, primero se identifica la Célula de proceso, la cual está conformada por el conjunto de equipos para la producción de un batch, en este caso la producción de **fungicidas**. A partir de esto se identifican los componentes inferiores (*Unidades, Módulos de Equipo y Módulos de Control*). La consolidación del modelo se puede observar en la figura 4.3, donde se muestra que la **unidad A**, en este caso un reactor, en el cual varias materias primas son agregadas para ser sometidas a cambios físicos, llevados a cabo por una secuencia de operaciones ejecutadas en conjunto por 6 módulos de equipo.

4.2.2. Modelo de control de procedimiento

El modelo de control de procedimiento, conocido en la práctica como receta, define una secuencia de actividades encaminadas a la obtención de un producto.

Como se mencionó en el capítulo 1, el modelo de control de procedimiento se puede desagregar hasta en cuatro niveles jerárquicos, por tanto no resulta necesario diseñar una receta con los cuatro niveles, dado que para algunos productos todo su procesamiento se lleva a cabo en una sola unidad, en dicho caso la receta se representaría en tres niveles: Procedimiento de unidad, Operaciones, y Fases. Dado que para el caso de estudio planteado solo se considera la unidad de reacción, se diseñará el modelo de control de procedimiento en función de lo antes mencionado.

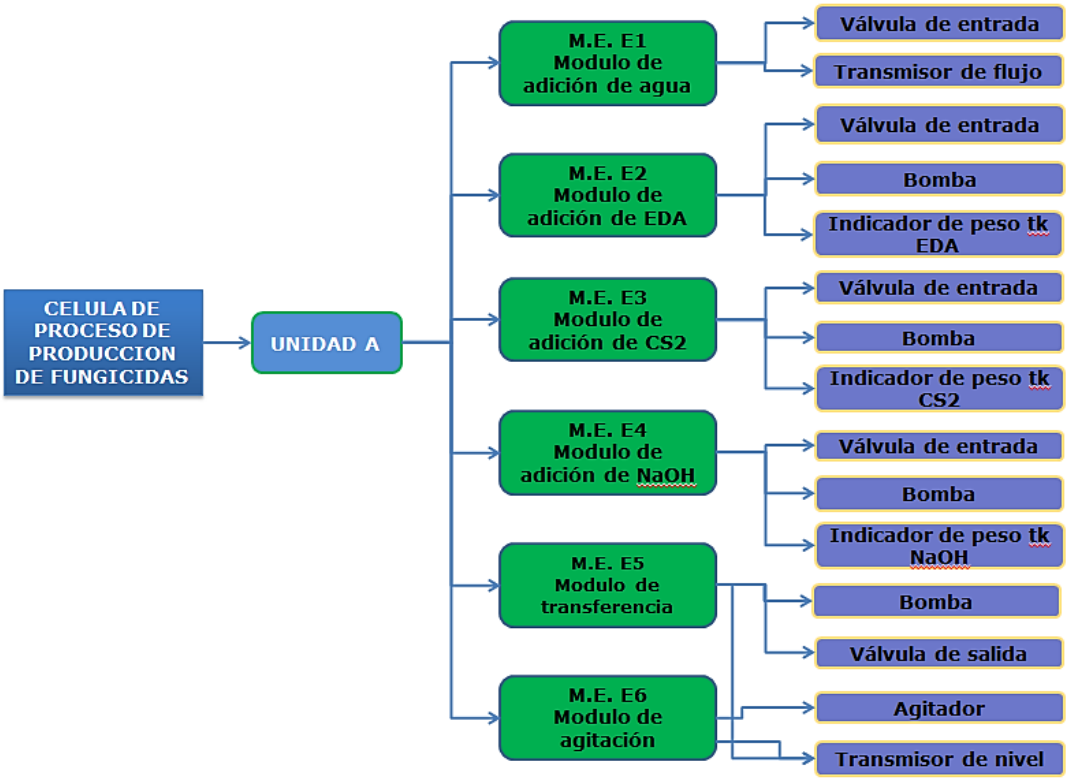


Figura 4.3 Modelo Físico de la Unidad de Reacción. ***Fuente: Propia***

Para la identificación de las fases se hace uso del modelo físico del proceso, es decir, las fases son equivalentes a los módulos de equipo o *capacidades de proceso* identificados para la unidad de reacción en la figura 4.3. Los cuales se definen a continuación.

4.2.2.1. Cargue de CS2

Para dar inicio al cargue de CS2 se debe verificar que la cantidad requerida de este componente está lista. En esta etapa de adición se deberán cargar 300 Kg de CS2, para lograrlo se deberá abrir la válvula de entrada al reactor asignada para este componente, y posteriormente se encenderá la bomba de llenado.

4.2.2.2. Cargue de EDA

Para dar inicio al cargue de EDA se debe verificar que la cantidad requerida, la cual es de 200 Kg, esté lista. Una vez se confirma la cantidad deseada, se deberá abrir la válvula de entrada al reactor asignada para este componente, y luego se encenderá la bomba de llenado. Esta fase deberá ser ejecutada en paralelo con la fase de cargue de CS2.

4.2.2.3. Cargue de la primera etapa de Agua

Para este cargue se deberá abrir la válvula de entrada asignada para este componente, e inmediatamente después se deberá encender la Bomba, y así agregar 300 Kg. Para comprobar que la cantidad de agua es la deseada, se tendrá un totalizador el cual está comparando en cada instante la cantidad de fluido adicionado, así una vez se igualen las cantidades, la válvula deberá cerrarse.

4.2.2.4. Cargue de NaOH

Para dar inicio al cargue de NaOH se debe verificar que la cantidad requerida de este componente esta lista. En esta etapa de adición se deberán cargar 500 Kg de NaOH, para lograrlo se deberá abrir la válvula de entrada al reactor asignada para este componente, y posteriormente se encenderá la bomba de llenado.

4.2.2.5. Agitación inicial

El agitador deberá encenderse para lo cual se establecerá un setpoint de 300Hz. esta agitación se ejecutara en paralelo con las dos etapas inmediatamente anteriores.

4.2.2.6. Cargue final de Agua

Para este cargue se deberá abrir la válvula de entrada asignada para este componente, y así agregar 100 Kg.

4.2.2.7. Agitación final

El agitador deberá encenderse para lo cual se establecerá un setpoint de 50Hz. esta agitación se ejecutara en paralelo con la etapa de Cargue final de agua.

4.2.2.8. Transferencia

Para esta etapa se deberá abrir la válvula de salida que comunica al reactor con el tanque de almacenamiento, e inmediatamente después se encenderá la bomba para transferir el producto. Una vez el transmisor de nivel indique cero la bomba se apagara y posteriormente se cerrará la válvula.

En las figuras 4.4, 4.5, y 4.6, Se desarrolla el modelo de control de procedimiento definiendo el cómo ejecutar las capacidades de procesos en conjunto bajo una secuencia específica de tal forma que se pueda obtener el producto deseado.

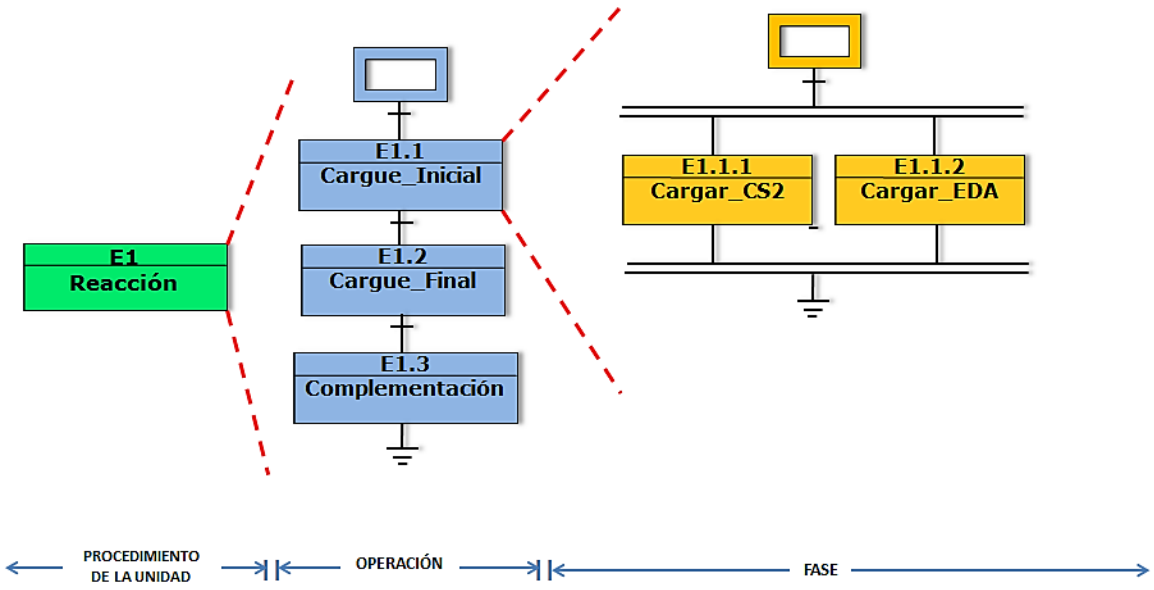


Figura 4.4 Modelo de control de procedimiento - Operación *Cargue Inicial*. *Fuente: Propia*

Tabla 4.1 Parámetros asociados a la operación de cargue inicial. *Fuente:* Propia

OPERACIÓN E1.1 Cargue Inicial	
FASE	**PARAMETRO**
Cargar_CS2	300 Kg
Cargar_EDA	200 Kg

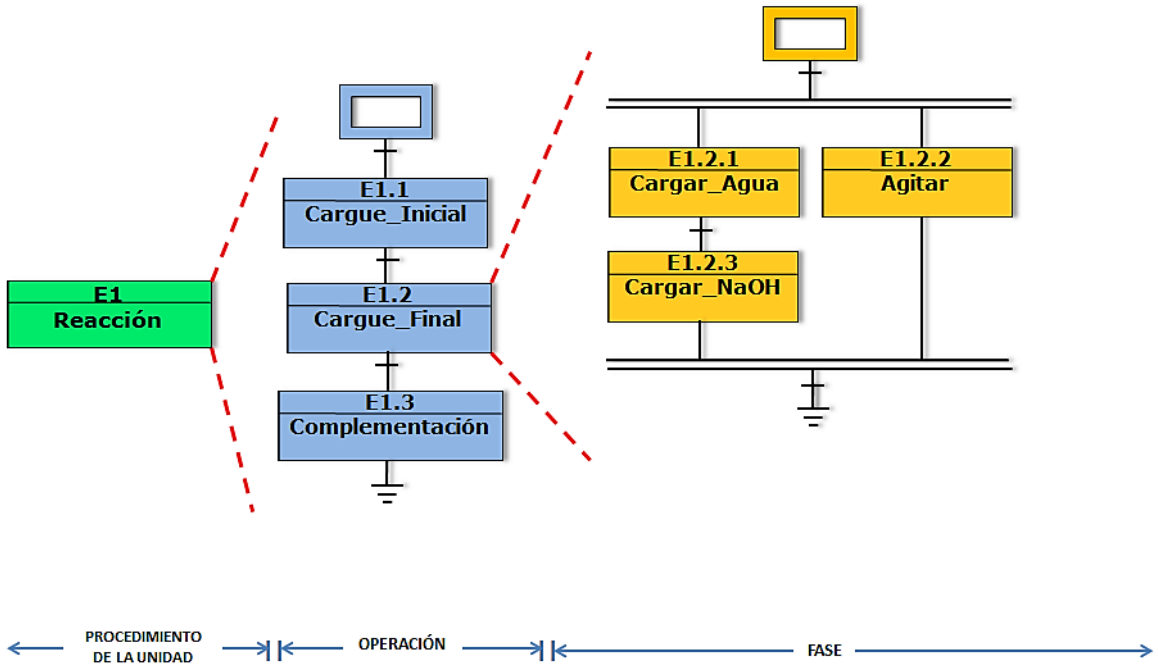


Figura 4.5 Modelo de control de procedimiento - Operación *Cargue Final*. *Fuente: Propia*

Tabla 4.2 Parámetros asociados a la operación de cargue final. *Fuente:* Propia

OPERACIÓN E1.2 Cargue Final	
FASE	**PARAMETRO**
Cargar_Agua	300 Kg
Agitar	300 Hz
Cargar_NaOH	500 Kg

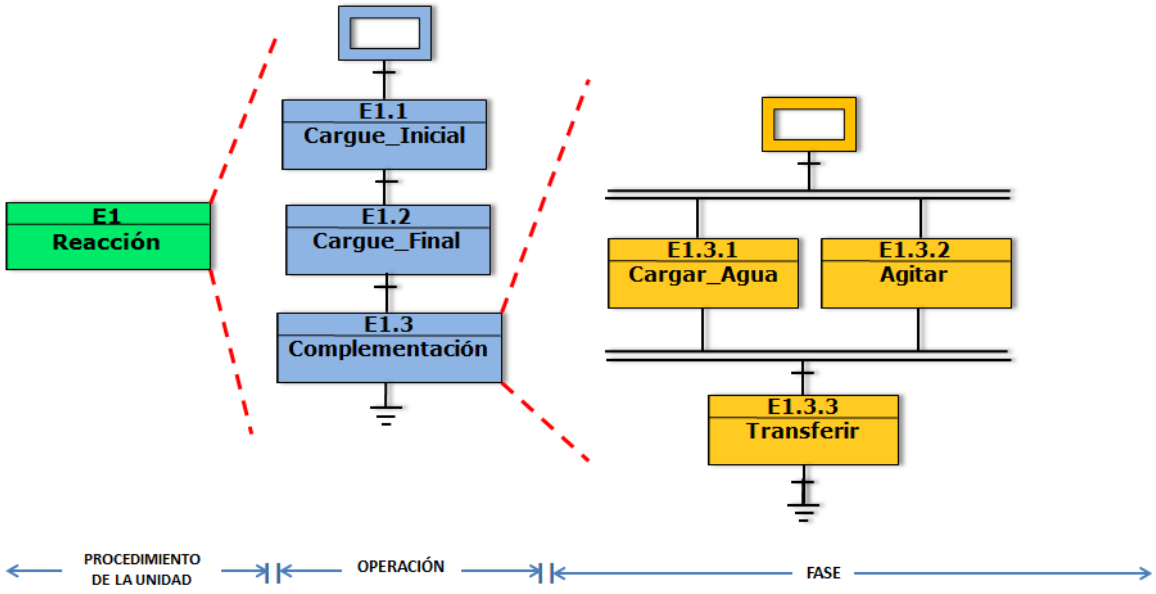


Figura 4.6 Modelo de control de procedimiento - Operación *Complementación.* *Fuente: Propia*

Tabla 4.3 Parámetros asociados a la operación de complementación. *Fuente:* Propia

OPERACIÓN E1.3 Complementación	
FASE	**PARAMETRO**
Cargar_Agua	100 Kg
Agitar	50 Hz
Transferencia	-

4.2.3. Modelo de proceso

El modelo de proceso especifica las actividades que deben llevarse a cabo para la obtención del producto en función de los activos definidos en el modelo físico, es decir de la correspondencia entre el modelo físico y el modelo de control de procedimiento se obtiene el modelo de proceso. Ver figura 4.7

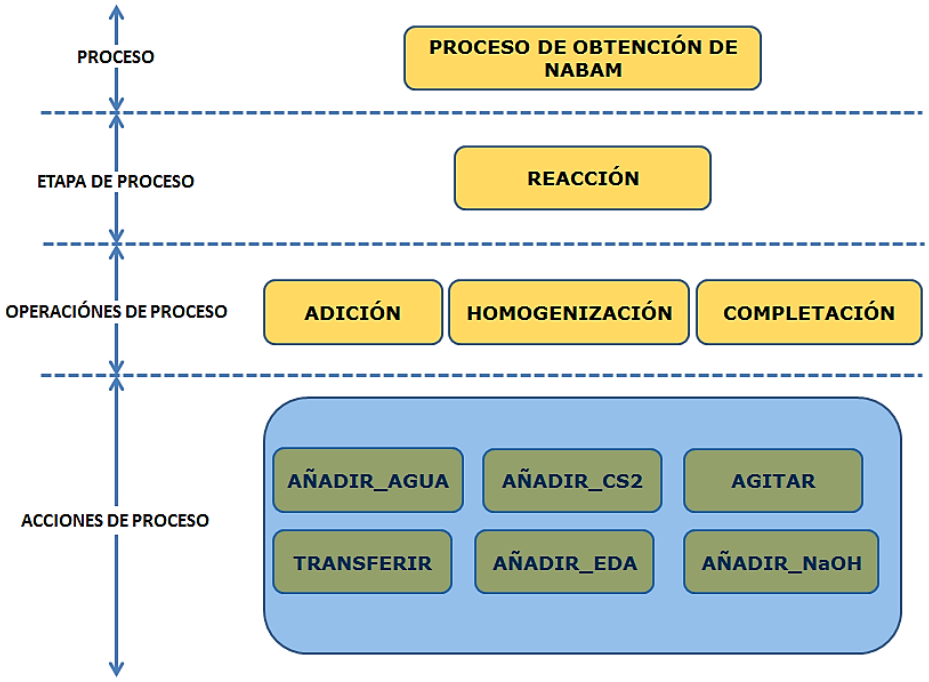


Figura 4.7 Modelo de proceso de reacción. ***Fuente: Propia***

4.3. Diseño de la lógica para los estados de las fases de la unidad de reacción

Desde el punto de vista práctico los términos ***Fase*** y ***Modulo de Equipo*** son iguales y hacen referencia a una capacidad de proceso, de tal forma que no hay diferencia en decir: *Módulo de equipo de agitación* a decir: *Fase de agitación*. Sin embargo, a nivel conceptual su diferencia radica en que cuando se habla de fase es a nivel de procedimiento (receta) y cuando se habla de módulo de equipo es a nivel del sistema de control, de tal forma que cada fase en un procedimiento (receta) tiene un módulo de equipo asociado en el sistema de control que ejecuta la lógica de funcionamiento. En otras palabras, un módulo de equipo es la instancia de una fase y en algunos casos para hacer la diferenciación se habla de Fase de Receta y Fase de Equipo. En este sentido las **fases de equipo** que definen el proceso de reacción de Nabam son:

- Cargar_Agua
- Cargar_CS2
- Cargar_EDA
- Agitar
- Cargar_NaOH
- Transferir

4.4. Resultados

A continuación se mostraran parte de los modelos en red de Petri para la fase ***Cargar_Agua***, a partir de los esquemas de procedimiento propuestos en el capítulo 2. Esto será implementado desde el software PGC, posteriormente será traducido a código ladder y finalmente se mostrara el resultado haciendo uso del administrador batch. La lógica de funcionamiento para esta fase es ejecutada por dos módulos de control: *Bomba 1* y *válvula1*. Los estados restantes para la fase cargar agua y para las fases Cargar_CS2, Cargar_EDA, Agitar, Cargar_NaOH, Transferir, su diseño y el código traducido están incluidos en el anexo A.

4.4.1. Implementación de la lógica de fase en el software PGC

Como se indicó en el capítulo 1 para el diseño de las fases de un proceso, se debe haber previamente configurado el FactoryTalk Batch el cual se enlaza con el sistema RsLoigix5000,ver figura 4.8, luego desde este último se hace la exportación del archivo con formato L5K, el cual contiene las fases previamente configuradas, ver figura 4.9.

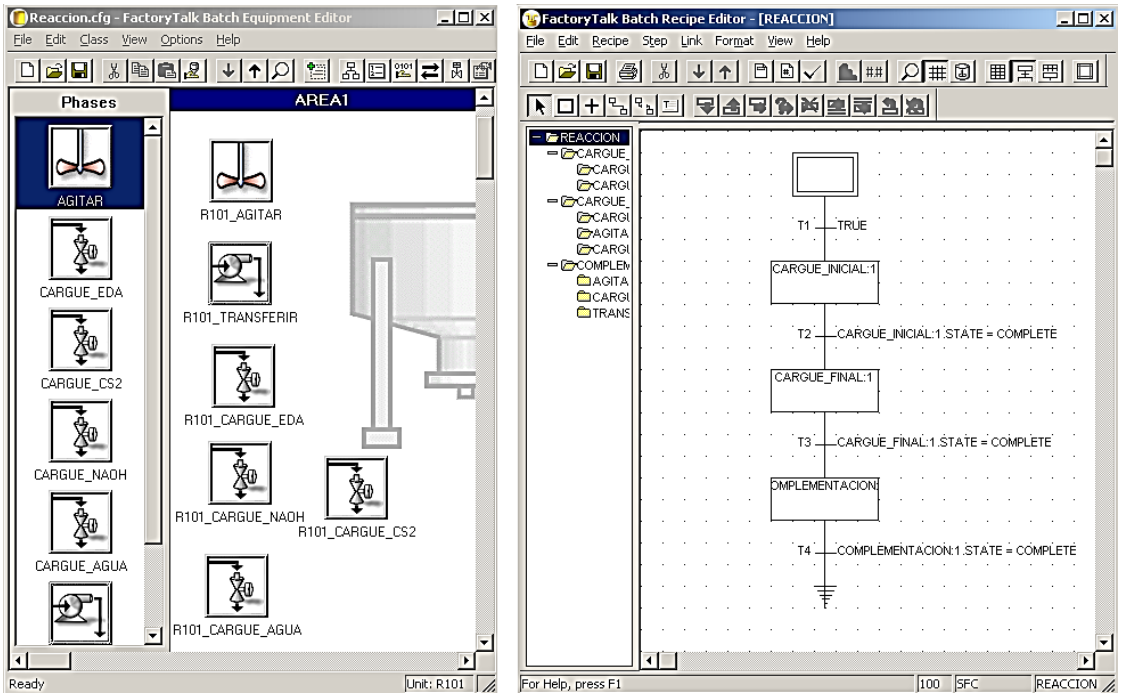


Figura 4.8 Editor de equipos y Editor de recetas. *Fuente: Propia*

Teniendo ya el archivo L5K, se procede a abrirlo desde el software PGC dando clic en la opción importar, ver figura 4.10, con lo cual se genera una ventana con la información del tipo de controlador usado, los módulos y su ubicación. Dar clic en *Aceptar*.

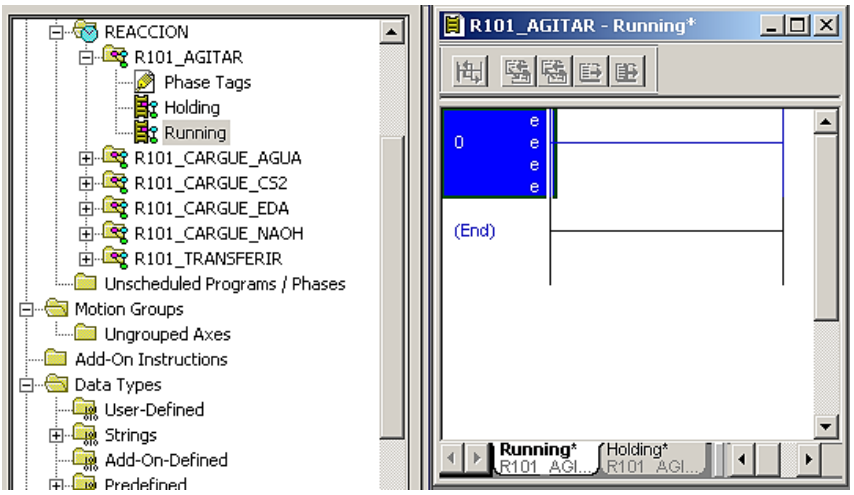


Figura 4.9 Fases del proceso de reacción. ***Fuente: Propia***

Para la configuración de estados de fase se da clic en la opción *Exportar / Configurar estados de fase.* En la ventana emergente, ver figura 4.11, se tiene la opción de seleccionar para cada fase los estados que sean requeridos. Luego se da clic en la opción *Generar PGC* generando para cada estado de fase un archivo XML que puede leído por el software PIPE, ver figura 4.12

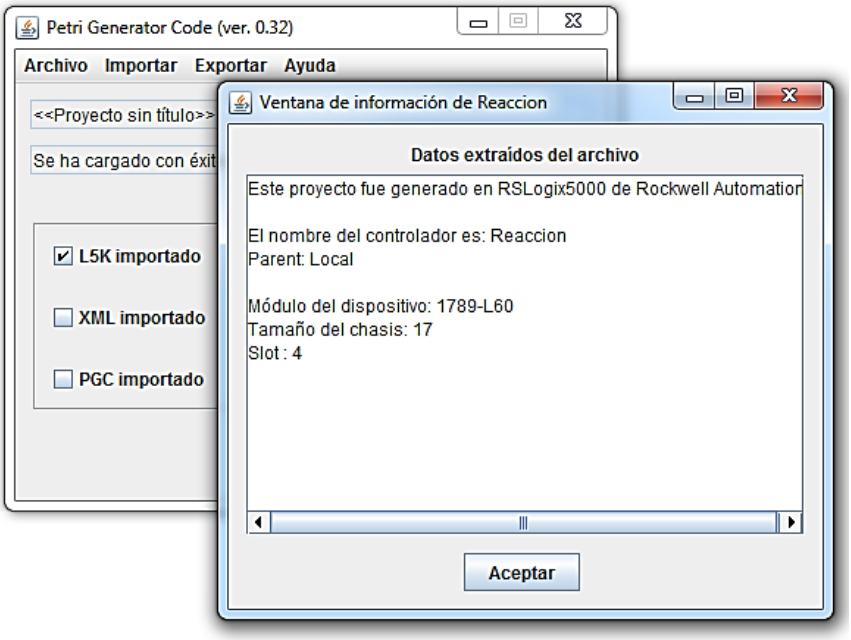


Figura 4.10 Importación del archivo L5K desde PGC. ***Fuente: Propia***

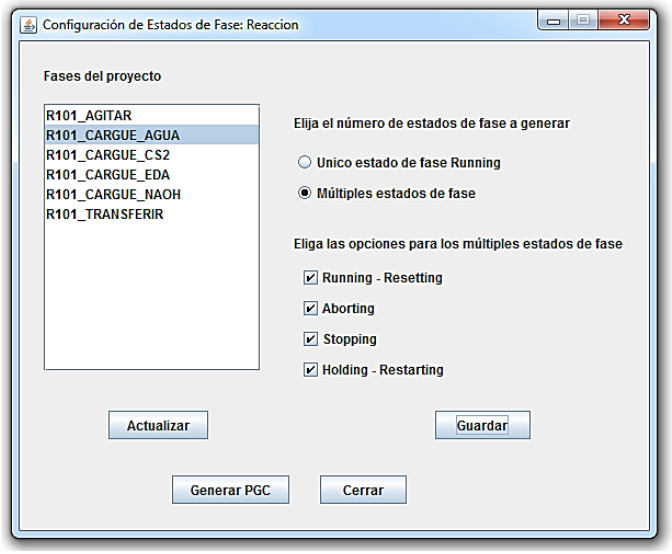


Figura 4.11 Configuración de estados de fase. ***Fuente: Propia***

Proyecto1.PGC	05/05/2015 02:52 ...	Archivo PGC	2 KB
R101_CARGUE_AGUA.aborting.xml	05/05/2015 02:52 ...	Documento XML	1 KB
R101_CARGUE_AGUA.holding.xml	05/05/2015 02:52 ...	Documento XML	1 KB
R101_CARGUE_AGUA.resetting.xml	05/05/2015 02:52 ...	Documento XML	1 KB
R101_CARGUE_AGUA.restarting.xml	05/05/2015 02:52 ...	Documento XML	1 KB
R101_CARGUE_AGUA.running.xml	05/05/2015 02:52 ...	Documento XML	1 KB
R101_CARGUE_AGUA.stopping.xml	05/05/2015 02:52 ...	Documento XML	1 KB

Figura 4.12 Archivos generados para cada estado de fase. ***Fuente: Propia***

Una vez que se tengan los archivos para cada estado de fase, el siguiente paso es generar la lógica en RdP, abriendo los archivos desde el software PIPE, ver figura 4.14, y diseñando la lógica con referencia a el procedimiento propuesto en el capítulo 2.

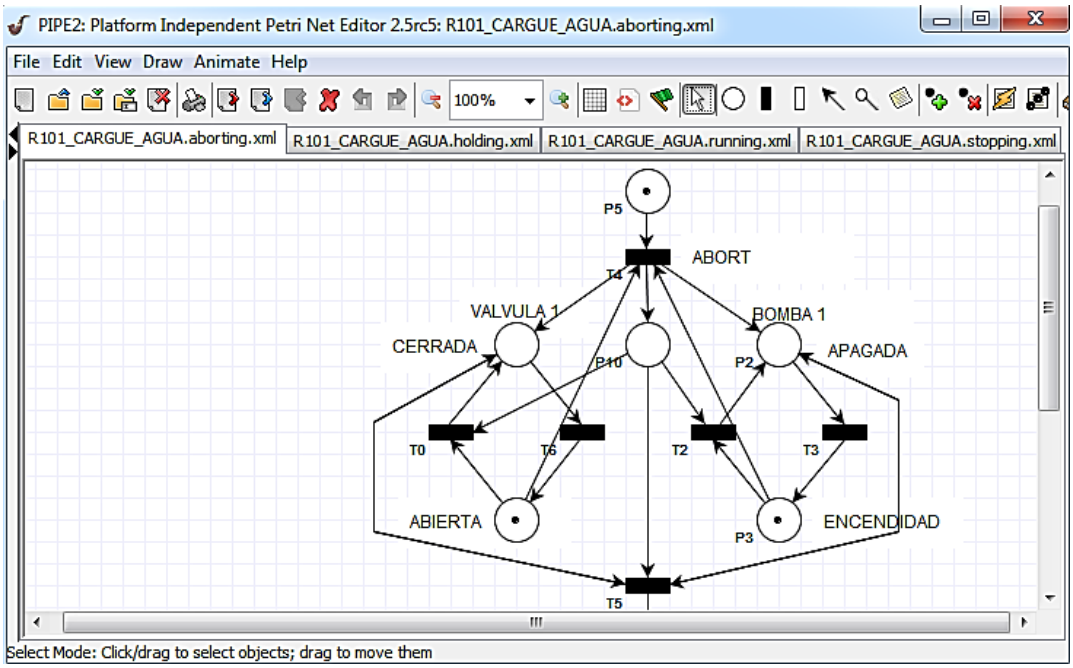


Figura 4.13 Diseño de estado de fase en el software PIPE. ***Fuente: Propia***

A continuación se muestra la lógica obtenida para el estado ***Holding***:

4.4.2. Lógica Holding para la fase *Cargar_Agua*

La condición para la detención de ésta fase, es que la bomba debe apagarse antes de que la válvula se cierre y así evitar daños. Para esto la ecuación que define la condición es:

$$v5 - v2 \leq 0$$

Al momento de ejecutar la red de Petri el marcaje avanza hasta la habilitación de la transición T5, la cual es un evento que permite la confirmación de la detención de los equipos, así para evitar que quede habilitada después de su disparo se plantea una nueva restricción:

$$v5 - v0 \leq 0$$

Y finalmente se implementa la condición que evita que los equipos se puedan volver a activar una vez el proceso haya concluido:

$$v6 + v3 \leq 0$$

Lo anterior se consolida en la figura 4.15

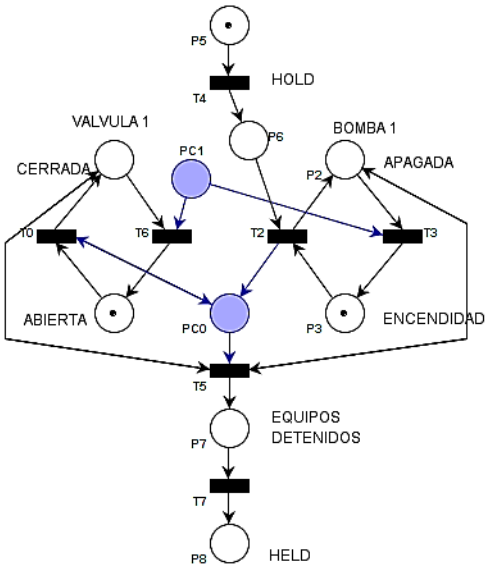


Figura 4.14 Lógica Holding. ***Fuente: Propia***

4.4.3. Traducción de red de Petri a código ladder

Para la traducción de la lógica en red de Petri a Ladder, se deberá cargar desde la aplicación software el archivo con nombre "Proyecto1.PGC" presentado en la figura 4.12, e ingresar a la opción *Exportar / Generar programa(s).* Desde aquí, se podrán visualizar los estados de fase existentes en la figura 4.15, para luego generar la exportación en rutinas dentro del proyecto L5K.

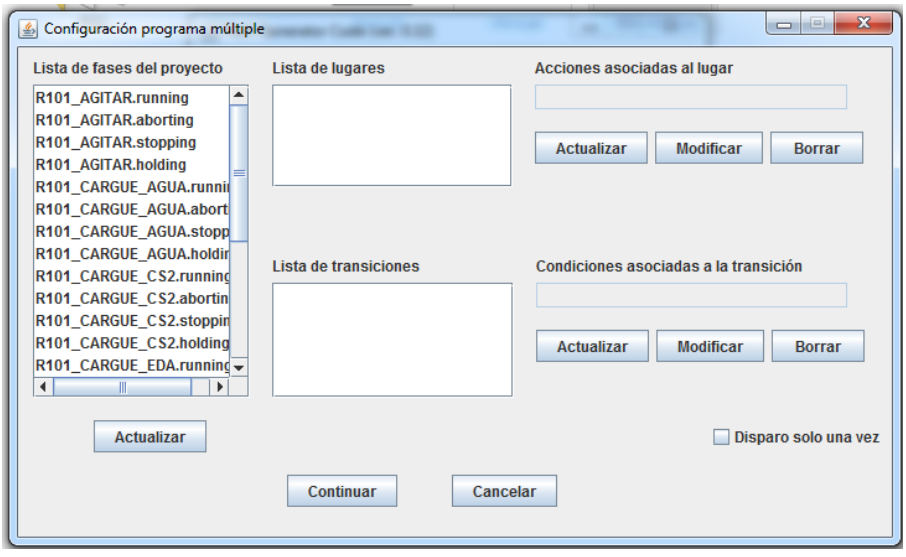


Figura 4.15 Configuración de fases. ***Fuente: Propia***

Al generar el nuevo L5K con la información de la lógica de control, bastará verificar la existencia del mismo dentro de RsLogix5000. En la figura 4.16 se encuentra la evidencia del código generado para la fase Cargar_Agua.

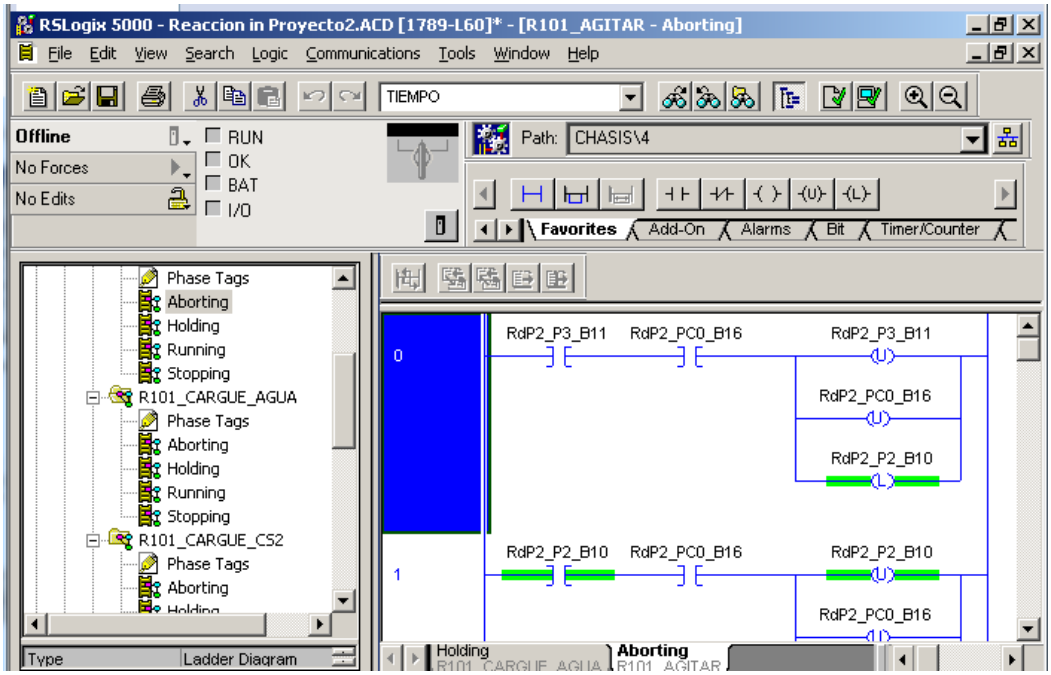


Figura 4.16 Evidencia del código generado para el caso de estudio. *Fuente: Propia*

A continuación en la figura 4.17 se muestra la lógica *Holding* traducida a código Ladder por el software PGC, implementado en RsLogix5000.

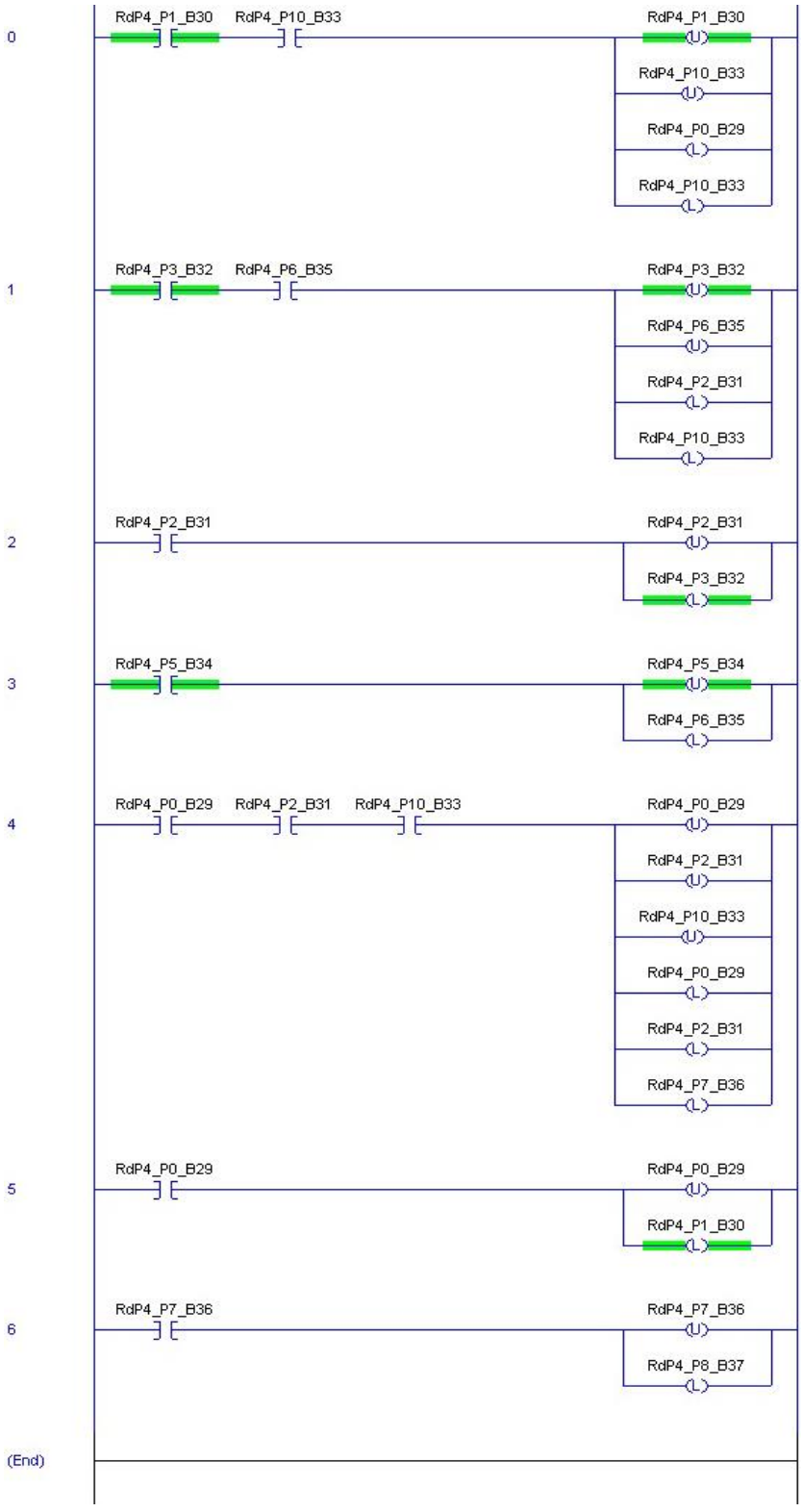


Figura 4.17 Codigo ladder para el estado *Holding. Fuente: Propia*

Para comprobar que la lógica cumple con las especificaciones de control, se hace uso de la herramienta FactoryTalk View Studio de Rockwell, usado para la implementación del HMI del proceso (*Human in Machine Interface*)

En la figura 4. Se observa el estado de la planta en el que las motobombas y válvulas para CS2 y EDA (salidas: 00, 01, 02, 03) están abiertas, después de haber recibido un comando de inicio (*entrada: 00*) para el estado *Running*.

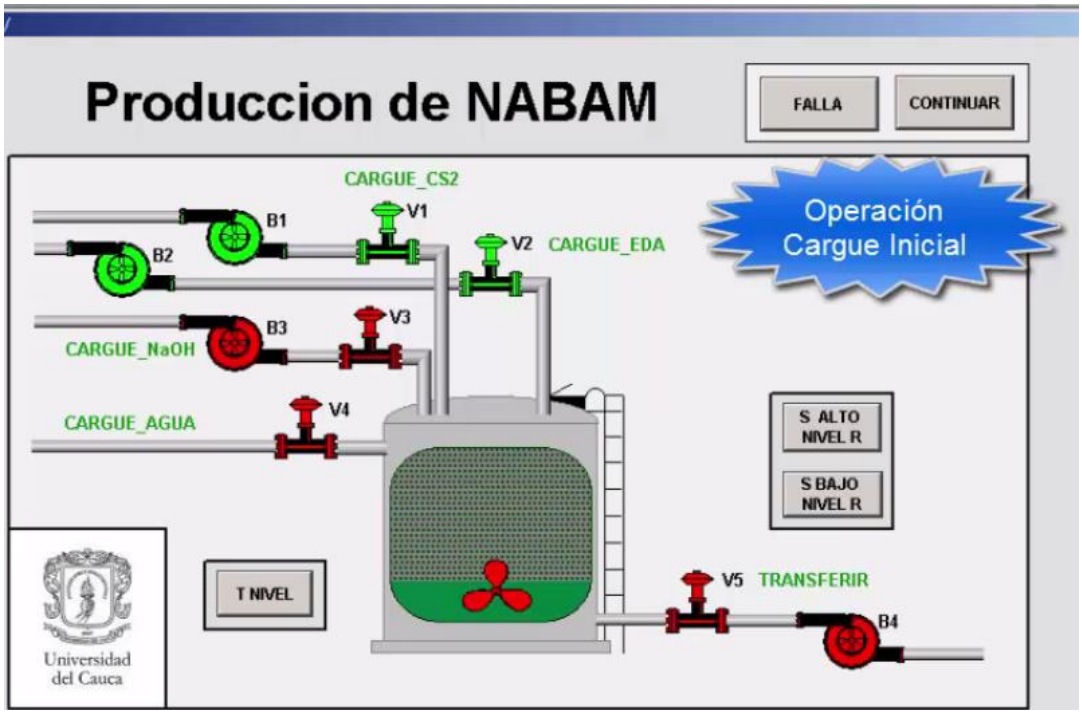


Figura 4.18 Estado del proceso donde: Valvula para CS2 y EDA abiertas. *Fuente: Propia*

5. Conclusiones

Al dividir la lógica de control en módulos más pequeños representados por fases de equipo, y posteriormente asignar a cada uno de ellos una red de Petri por medio de PGC, se simplifica la modificación y corrección de una fase, sin necesidad de realizar cambios en los demás diseños de fase manteniendo la independencia entre ellos, lo que finalmente disminuye la complejidad de diseño y aumenta el grado de descentralización de la lógica de control. Por tanto, el gran éxito de esta aplicación software consiste en integrar el diseño de la lógica de control con base a IEC 61131-3, los modelos de ISA-88 y el modelado formal de las redes de Petri.

La herramienta implementó un algoritmo computacional de traducción a IEC 61131-3, en el cual se enfatizó en las características de los programas creados en Ladder, tales como el uso de arcos de inhibición y de lectura, o un número mayor a uno de asociaciones entre lugares/salidas y transiciones/entradas. Además, se presentó una base para la implementación de redes de Petri en SFC que consiste en la transformación directa entre elementos como etapa/lugar y transición/transición. Además de la generación de ramas convergentes o divergentes según los valores inscritos en las matrices de incidencia hacia adelante y de incidencia hacia atrás.

Las redes de Petri como herramienta de modelado proveen una gran versatilidad, lo cual se pudo comprobar en el momento de diseñar los procesos tipo batch aun estando orientadas a los sistemas de eventos discretos, resultando para esto fundamental la consideración de los modelos de la parte 1 del estándar ISA-88, dada las propiedades de modularidad que proveen, haciendo mucho más sencillo el diseño de la lógica a través de partes que se pueden manipular fácilmente.

En el caso de estudio se logró observar como el código de programación generado se encuentra localizado en múltiples rutinas de programación para controladores programables de alta gama de Rockwell Automation. Este código fue validado dentro de la simulación

para el caso de estudio que fue modelado desde el estándar ISA-88, y de la descripción de las especificaciones de control desde el formalismo de redes de Petri. Sin el uso de PGC, la generación de código ladder no podría haberse realizado partiendo de un lenguaje de alto nivel como las redes de Petri, ni facilitar la distribución de la lógica de control por medio del modelo de estados de fase de ISA-88.

El modelo de estados propuesto por ISA-88 es de gran importancia para el manejo de la lógica de la fase, dado que contempla una estructura para el manejo de todas las situaciones que se pueden presentar en los procesos de producción, en este sentido el planteamiento propuesto para el diseño de la lógica de cada estado del modelo antes mencionado, logra potenciar las habilidades para un diseño más coherente sirviendo de guía para la definición de cada una de las etapas por las que deben pasar los estados de la fase.

BIBLIOGRAFÍA

[1] G. Zapata, J. W. Branch, L. F. Quintero "Metodología para el Modelado y Generación de Código de Control de Sistemas Secuenciales mediante Redes de Petri Jerárquicas," *Avances en Sistemas e informatica*, vol. 4, no. 1, pp. 59–66, 2007.

[2] PLCopen, "IEC 1131-3: un recurso de programación estandar," *Estandarizacion en la programacion de control industrial*, vol. 3.

[3] A. Dideban, M. Kiani, H. Alla, "Implementation of Petri nets based controller using SFC," *Control Eng. Appl. Informatics*, vol. 13, no. 4, pp. 82–92, 2011.

[4] A. Espuña, "Contribución al estudio de plantas quimicas multiproducto de proceso discontinuo," *Escola tècnica superior d'enginyers industrials*, p. 27, Barcelona, 1994.

[5] E. G. Hernandez, E. S. Puga, S. A. Foyo, J. A. Meda, "Modeling Framework for Automated Manufacturing Systems Based on Petri Nets and ISA Standards," *Studies in Informatics and Control*, vol. 22, no. 2, 2013.

[6] Capitulo 1. Caracterisiticas de los procesos batch

[7] R. Olsson, K. Arzén, "Exception Handling in Recipe-Based Batch Control," *Dept. of Automatic Control, Lund University*, December, 2002.

[8] K. R. Quintero, "Analisis y Validación del Esquema de Administración de Procesos ISA-88 para el Proceso de Produccion de Azúcar," *Universidad de los Andes*, Merida, Venezuela, Septiembre, 2009.

[9] iBatch, "iBatch Application Guide," 2003.

[10] G. Camacho, "Proyecto de Automatización II -Introducción al Editor de Modelos S88 Factory Talk Batch Pasos Previos."

[11] D. Flores, J. L. Pontón, D. Soria, J. Andino, "Normas S88," *Universidad de las Fuerzas Armadas*.

[12] International Society of Automatcion, "Batch Control Part 1 : Models and Terminology," October, 1995.

[13] R. Olsson, “Batch Control and Diagnosis,” *Dept. of Automatic Control, Lund Institute of Technology*, p. 244, Junio 2005.

[14] H. E. Salomone, “Métodos, estructuras y modelos para la simulación de procesos batch:,” *Modelado simulación y Optimizacion de procesos químicos*, pp. 741–766, 1999.

[15] F. L. Quintero, “Un Modelo de control inteligente para sistemas de manufactura basado en los paradigmas Holónico y Multi-Agente,” p. 150, 2009.

[16] G. Zapata Madrigal, “Propuesta Para la Planificación, Programación, Supervisión y Control de la Producción en Procesos Continuos Desde la Teoría del Control Supervisorio y el Enfoque Holónico,” *Universidad de los Andes*, Merida, Venezuela, Diciembre, 2011.

[17] G. Cohen, “Analisis y control de sistemas de eventos discretos: de redes de petri temporizadas al algebra,” 2001.

[18] J. E. Castellanos, “Sistemas a eventos discretos, una aproximacion a la teoria de redes de petri y grafcet,” *Universidad Militar Nueva Granada*, 2008.

[19] G. Zapata, J. Cardillo, and E. Chacón, “Aportes metodológicos para el diseño de sistemas de supervisión de procesos continuos,” *Universidad de los Andes*, Merida, Venezuela, vol. 22, no. 3, pp. 97–114, Octubre, 2011.

[20] M. Indriago, “Desarrollo de un Procedimiento para Diseño de Sistemas de Control en Procesos con un Modelo de Integracion Basado en Holones,” *Universidad de los Andes*, Merida, Venezuela, 2011.

[21] R. Olsson and K. E. Årzén, “Exception Handling in S88 using Grafchart,” *World Batch Forum North America, Conf. Woodcliff Lake, NJ*, pp. 1–8, 2002.

[22] C. D. Buchely, F. Ruiz, “Herramienta Basada en Redes de Petri para Diseño de Supervisores de Sistemas de Eventos Discretos” *Ingenieria en Automatica Industrial, Universida del Cauca*, 2012.

[23] Rockwell Automation, “Procedimientos comunes de los controladores Logix5000™,” Agosto, 2002.

[24] C. Johnsson,“A Graphical Languaje for Batch Control,” *Department of Automatic Control, Institute of Technoligy,* Lund, 1999.

[25] W. M. Hawkins, T. G. Fisher, "*Batch Control Systems*," 2006.

[26] D. Brandl, “Recipe/Equipment Separation,” pp. 17–40.

[27] A. Rovira, “Control de Procesos en Sistemas Hibridos e Integracion de Software S88,” Noviembre 2007.

[28] J. Parshall, L. Lamb, “Batch Control from a User's Perpective,” 2006.

[29] Japan Batch Forum, “Introduction to S88,” 2007.

[30] iBatch, "Phase Programming Guide," 2003.

[31] A. Creus, “Instrumentacion Industrial,” 1997.

[32] D. Brandl, “Master and Control Reipe Procedures,” pp. 41–66.

[33] M. Uzam, H. Jones, “A new Petri-net-based synthesis technique for supervisory control of discrete event systems,” *Turkish Journal of Electrical Engineering*, vol. 10, no. 1, 2002.

[34] Rockwell Automation, “Logix5000 Controllers IEC 61131-3 Compliance.”

[35] Rockwell Automation, “Logix5000 Controllers I / O and Tag Data.”

[36] Rockwell Automation, “Logix5000 Controllers Tasks , Programs , and Routines.”

[37] M. Uzam, H. Jones, N. Ajlouni, “Conversion of Petri net controllers for manufacturing systems into ladder logic diagrams,” *University of Salford,* vol. 2, pp. 649–655, 1996.

[38] J. Lee, P. Hsu, “A systematic approach for the sequence controller design in manufacturing systems,” *The International Journal of Management Science, Omega,* vol. 25, no. 7–8, pp. 754–760, 2005.

[39] Rockwell Automation, “Logix5000 Controllers General Instructions.”

[40] F. Haugen, “Article : Sequential Control,” pp. 1–8, August, 2009.

[41] R. David, “Grafcet: A Powerful Tool for Specification of Logic Controllers,” *IEEE Transactions on control systems technology*, Vol. 3, N° 3, September 1995.

[42] P. San Segundo, “Introducción al modelado GRAFCET,” *Universidad Politecnica de Madrid*, 1988.

[43] K. Collins, “Plc Programming for Industrial Automation,” p. 140, 2007.

[44] G. Muši˘, D. Matko, “BATCH PLANTS,” vol. 2, pp. 989–994, 1998.

[45] G. Music, D. Matko, “Petri net based control of a modular production system,” *Proceedings of the IEEE International Symposium on Industrial Electronics*, vol. 3, pp. 1383–1388, 1999.

[46] Rockwell Automation, "FactoryTalk Batch User's Guide."

[47] K. Heinz, M. Tiegelkamp, "IEC 61131-3: Programing Industrial Automation Systems", 2010.

[48] R. Arrieta, “Proceso de Manufactura y Tipos de Formulación del MANCOZEB,” Julio 2013.