Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
EPSJ

Trabajo Fin de Grado

SIMULADOR GRAFCET PARA ASIGNATURAS DE AUTOMATIZACIÓN

Alumno: Alejandro Estepa González

Tutor: Elisabet Estévez Estévez, Alejandro Sánchez García

Dpto: Ingeniería Electrónica y Automática

Noviembre, 2022

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don **Alejandro Sánchez García**, **Doña Elisabet Estévez Estévez**, tutores del Proyecto Fin de Carrera titulado: **Simulador Grafcet para asignaturas de automatización**, que presenta **Alejandro Estepa González**, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, noviembre de 2022

El alumno:

ALEJANDRO ESTEPA GONZÁLEZ

Los tutores:

ALEJANDRO SÁNCHEZ GARCÍA

SANCHEZ GARCIA ALEJANDRO - 77321466C

Firmado digitalmente por SANCHEZ GARCIA ALEJANDRO - 77321466C
Fecha: 2022.11.18 11:32:19 +01'00'

ELISABET ESTÉVEZ ESTÉVEZ

ESTEVEZ ESTEVEZ ELISABET - 72578245C

Firmado digitalmente por ESTEVEZ ESTEVEZ ELISABET - 72578245C
Fecha: 2022.11.18 12:28:52 +01'00'

Contenido

Figuras

1 Motivación, objetivos y estructura

La pandemia y el desarrollo de la enseñanza online de los últimos años han obligado a los alumnos a aprender en otro entorno diferente y, a veces, más complicado de cara a afrontar las prácticas de distintas asignaturas. Esta falta de opciones sugiere la necesidad de desarrollar aplicaciones que faciliten el aprendizaje y optimicen otros recursos que ya se tenían y se aplicaban a dichas prácticas.

Esto mencionado daría la posibilidad de poder trabajar a distancia y simular procesos de automatización sin necesidad de trabajar con máquinas reales, o de comprobar previamente a trabajar con estas si la secuencia de etapas propuestas es coherente, evitando errores antes de trabajar con un equipo físico y suponiendo un gran avance en cosas como las que se han comentado anteriormente.

Este proyecto se centra en resolver tanto los problemas de automatización que se dan en los primeros cursos de todos los grados de Ingeniería Industrial en la Universidad de Jaén en la asignatura de Automática Industrial. De igual manera, también se considera de interés para asignaturas de automatización en últimos cursos de Grado en Ingeniería Electrónica Industrial y Másteres como, por ejemplo: Ingeniería Industrial y Máster Universitario de Ingeniería Mecatrónica.

Centrándose en el caso de la asignatura de Automática Industrial, el alumnado no tiene conocimiento ni para diseñar el proyecto de automatización ni mucho menos para su codificación. Con este trabajo se pretende ayudar a diseñar el proyecto de automatización centrándose únicamente en el conocimiento de léxico y sintaxis de GRAFCET (Gráfico Funcional de Control de Etapas y Acciones). Los alumnos que continúen en la rama de electrónica y automática van a necesitar ampliar estos conocimientos, haciendo uso de estos Grafcets, y programándolos en TIA portal y viendo el funcionamiento del automatismo físico.

Por tanto, el objetivo principal del TFG consiste en el diseño y desarrollo de una herramienta que permita:

- Diseñar el proyecto de automatización de cualquier sistema secuencial a través de GRAFCET.
- Conectar dicho proyecto con la parte operativa representada bien con una planta física real o una representación digital (Gemelo Digital).
- Simular el proyecto de automatización que permita ver la situación (estado) en la que se encuentra el sistema automatizado en todo momento.

Esta herramienta, además, tiene que ser válida para emular el proyecto de automatización, independientemente del PLC y lenguaje de programación donde y como se vaya a implementar. Por ello, es necesario hacer uso del concepto de Ingeniería Conducida por Modelos o Model Driven Engineering (MDE) que permite facilitar el diseño de sistemas complejos. Concretamente, será necesario definir un Meta-Modelo que represente el léxico y sintaxis que siga todo proyecto de automatización. Dicho meta-modelo será utilizado por la herramienta por lo que deberá ser implementado con un lenguaje de marcado. Este entorno además se debe caracterizar por seguir fielmente la simbología del GRAFCET. Por otro lado, la interconexión con la planta (física o virtual) será exclusivamente a través de sus Entradas/Salidas. Así, la herramienta deberá tener dicha información, que se podrá proporcionar a través de un fichero XML (eXtensible Markup Language). (school of information technology, 2022)

La estructura del trabajo es la siguiente: un segundo apartado estará centrado en detallar tanto el léxico y sintaxis de todo GRAFCET para poderlo finalmente plasmar en un XML Schema. Los apartados tres y cuatro se centrarán en el diseño y desarrollo de la herramienta propuesta. Se presentará un ejemplo de uso en el apartado cinco. La memoria concluye con un sexto apartado centrado en las conclusiones y trabajos futuros.

2 Modelado y diseño del proyecto de automatización

Para el diseño de un proyecto de automatización lo primero será definir el concepto de Grafcet. Un Grafcet es un modelo de representación gráfica, nacido en 1977 en Francia de la necesidad de unificar y racionalizar el lenguaje para describir los sistemas lógicos, particularmente, la parte secuencial de los mismos y desarrollado por la Asociación Francesa para la Cibernética Económica y Técnica (AFCET). Este describe todo sistema cuyas evoluciones puedan expresarse secuencialmente y puede ser utilizado para los procesos combinatorios, además, ya que permite tratarlos secuencialmente.

Representa una sucesión de estados de un sistema, donde cada uno puede estar asociado a su vez con una o varias acciones que se realizarán en el momento que este estado esté activo. A su vez habrá también transiciones para pasar de un estado a otro y la condición se llamará receptividad. Para avanzar en las distintas etapas del Grafcet se debe tener una de estas activa y la condición o receptividad de la transición a la que precede la etapa ser verdadera. Esto quiere decir que cada GRAFCET estará basado en el elemento fundamental de un proceso, la operación o etapa y a partir de esto dicho proceso debe dividirse en macroetapas, y estas, en etapas más elementales hasta reducir toda la complejidad a una dependencia de las acciones en relaciones combinacionales entre las entradas y salidas del sistema.

Además, el funcionamiento de cualquier Grafcet es cíclico, es decir, las etapas pueden estar asociadas mediante una transición a un estado anterior del sistema, repitiendo el ciclo o pasando por otras etapas que den lugar a un desarrollo distinto pero siempre volviendo a una etapa anterior para que el Grafcet quede cerrado. Esto implica también la necesidad de tener siempre al menos dos estados o etapas, ya que, en caso contrario sería imposible tener un proceso cíclico al no estar permitido volver a la etapa desde una conexión directa con esta misma.

Los siguientes subapartados se van a centrar en detallar léxico y sintaxis así como simbología de GRAFCET. Dado que la herramienta a realizar debe almacenar de alguna manera dicha información, se ha de definir dicho léxico y sintaxis a través de un Meta-Modelo y para ello se ha elegido la tecnología XML schema. Por tanto, un segundo subapartado se centra en la definición de dicho XML schema. Finalmente, un tercer subapartado donde se presenta un ejemplo básico y cómo se almacena en un fichero XML.

2.1 Léxico y sintaxis de GRAFCET

A la hora de llevar a cabo un GRAFCET se deben comprender los distintos conceptos básicos relacionados con el léxico y sintaxis utilizados. En lo referente al léxico, todo GRAFCET se compone de:

- **Etapa** – Proporciona información del estado del sistema. Se simboliza con un cuadrado con un número entero en su interior. En el caso de la etapa inicial el cuadrado estará marcado con una doble línea. Cada etapa tiene un valor único que se asigna al irlas añadiendo al GRAFCET. Por último, una etapa puede estar activa o inactiva, simbolizándose, en el caso de que esté activa con un punto en la parte superior del símbolo de dicha etapa.

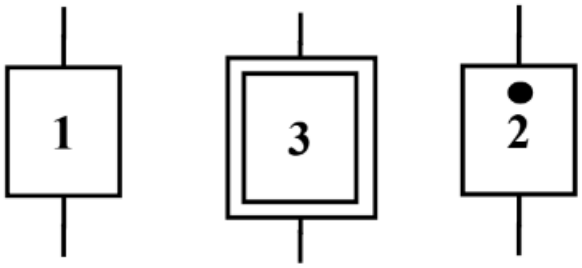


Figura 1. Representaciones gráficas de etapa

(a) Símbolo genérico (b) Etapa inicial (c) Etapa activa

En la Figura 1 se representan los símbolos que se pueden ver referidos a las etapas, siendo el de la izquierda el símbolo genérico, el del medio, con una doble línea, marca que la etapa corresponde con la inicial en el GRAFCET, y la de la derecha se trataría de una etapa genérica en el caso de que se encuentre activa, simbolizada con el punto como se ha comentado anteriormente.

- **Acción** – Órdenes del sistema de control, por lo que van asociadas a las etapas. Las acciones pueden ser externas o internas. En el primer caso, se corresponden a la activación de salidas del sistema de control (e.g. expansión del vástago de un cilindro). En el segundo caso, se corresponden con el lanzamiento de temporizadores o el uso de contadores (carga o actualización). Se representan gráficamente con un rectángulo y a la derecha de cada etapa. En el caso de haber dos o más acciones a realizar en una misma etapa se colocan una debajo o a la izquierda de otra. Es importante destacar, que todas las acciones vinculadas a una etapa, se van a ejecutar mientras dicha etapa esté activa.

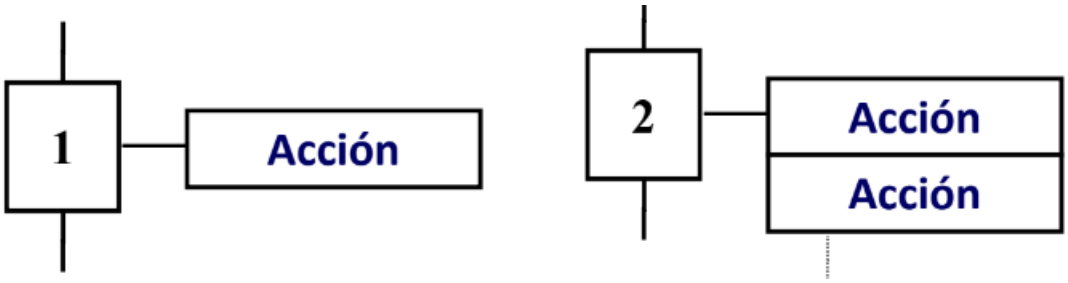


Figura 2. Símbolo de acciones

- **Transición** – Indica la posibilidad de evolución entre etapas. Toda transición tiene vinculada una condición lógica (receptividad) que condiciona la evolución de una etapa a la siguiente. Dicha evolución se lleva a cabo franqueando la transición y esto se puede hacer únicamente si la(s) etapa(s) desde donde nace la transición está(n) activa(s) y la condición vinculada es verdadera. El caso contrario sería que la transición no esté validada, en ese caso la condición asociada puede cumplirse, pero no se disparará la etapa siguiente.

- **Receptividad** – Condición a satisfacer para franquear una transición y por tanto permitir la evolución del GRAFCET. La receptividad puede ser una combinación lógica de las entradas del sistema de control (externa) o puede denotar el estado de un contador, temporizador, incluso el estado activo o inactivo de una etapa (interna). Está asociada a una transición y se define como la proposición lógica, verdadera o falsa, que se debe cumplir para pasar del estado que precede a dicha transición, a las etapas posteriores. Entre toda la información disponible en un momento determinado, la receptividad agrupa únicamente la necesaria para el franqueo de la transición a la que está asociada.

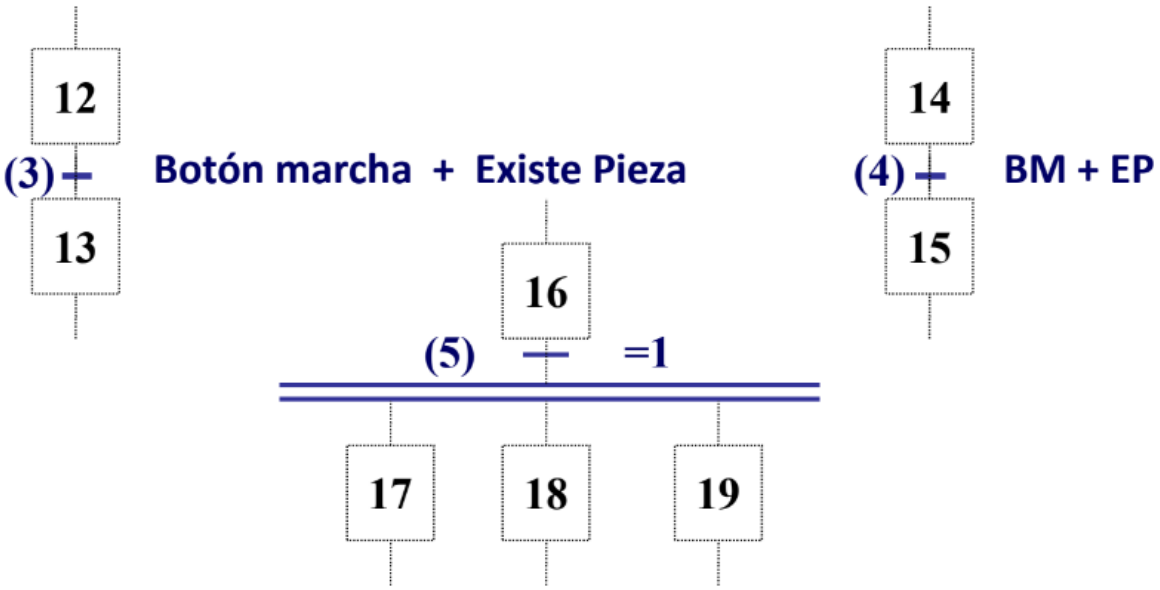


Figura 3. Representaciones gráficas de transiciones y receptividades asociadas

En la Figura 3 se observan distintos tipos de transiciones, simbolizados con una línea vertical con una horizontal más marcada dividiendo a la anterior por la mitad. El número entero a la derecha marca la transición y la condición lógica de la derecha hace referencia a la receptividad asociada a dicha transición.

- **Divergencia y convergencia selectora** – Cuando se desea plasmar caminos alternativos, es decir, el equivalente a IF-ELSE o SWITCH …CASE en lenguajes de alto nivel, GRAFECT ofrece las divergencias y convergencias selectoras. Así, de una etapa saldrán tantas transiciones como caminos alternativos se tengan. La convergencia, es para representar el fin de dichos caminos alternativos. Esto implica que una etapa podrá ser activada por diferentes vías, eso sí, nunca por más de una a la vez. Por ello, es importante destacar que las condiciones asociadas a las transiciones de cada rama deben ser excluyentes.

En la Figura 4 no se aprecian las transiciones, que se deben colocar al principio de cada rama y seguidas, al menos, de otra etapa.

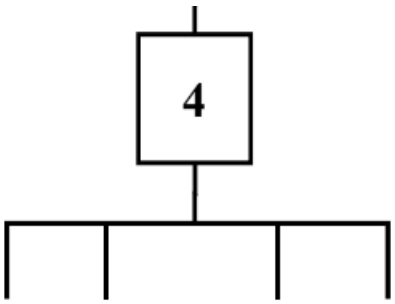


Figura 4. Símbolo de divergencia

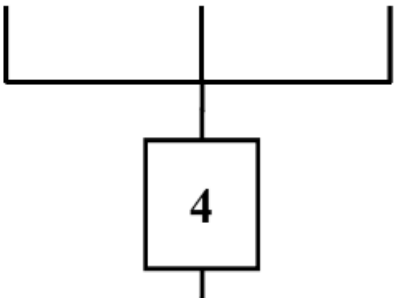


Figura 5. Símbolo de convergencia

En cada una de las ramas que se aprecian en la Figura 5 debe ir situada una transición como último elemento de la rama. Solo una de las ramas tendrá etapas activas como se comentó en la descripción de divergencia selectora, ya que, las condiciones son excluyentes. De esta forma, una de las tres transiciones, en este caso, estará validada y al hacerse verdadera su receptividad asociada se activará la etapa cuatro que vemos en el ejemplo.

Hay que destacar, además, que los distintos caminos iniciados como divergencia deben confluir en uno o más puntos de convergencia y en la estructura no pueden existir caminos abiertos, debemos tener una estructura globalmente cerrada.

- **Divergencia y convergencia de simultaneidad** – GRAFCET ofrece este recurso para reflejar la ejecución de subprocesos de forma paralela y por tanto simultánea. Por ello, una divergencia permite pasar de una etapa a la activación de n etapas, la primera etapa de cada subproceso a realizar en paralelo. Habrá una única transición encargada de arrancar simultáneamente la primera etapa de las distintas ramas al superar su correspondiente flanco. Solamente cuando se hayan terminado los subprocesos se podrá converger. Para ello, es necesario añadir en cada rama una etapa que refleje que se ha terminado (una etapa de espera o sincronización). Por ello, toda convergencia tiene vinculada una única condición que se comprobará solamente cuando las n ramas a converger hayan terminado (es decir cuando la última etapa de cada rama a converger esté activa).

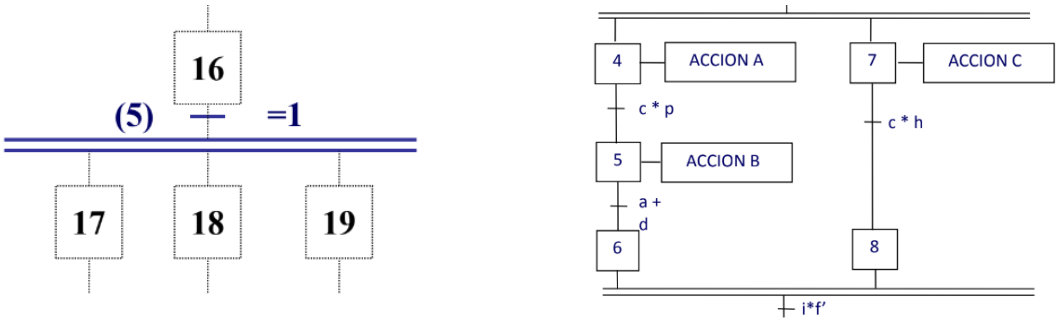


Figura 6. Representaciones de divergencia

Igual que en el caso de las divergencias selectoras debe haber una o más confluencias de los caminos formados en este tipo de divergencia, dando lugar a una estructura cerrada.

En cuanto a la sintaxis, se debe respetar siempre la alternancia etapa-transición y transición-etapa, es decir, dos etapas nunca van a poder estar unidas directamente ni dos transiciones tampoco pueden presentar una unión directa entre ellas, siempre debe haber un elemento del otro tipo intercalado.

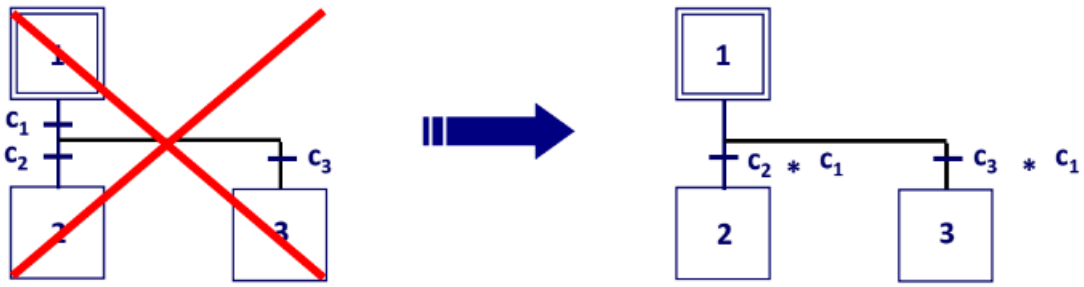


Figura 7. Ejemplo sintaxis de GRAFCET en transiciones

En el caso de la Figura 7, se colocan dos transiciones seguidas, mientras que la opción correcta sería colocar en cada una de las transiciones de la divergencia una transición cuya receptividad representa una expresión lógica combinacional de la complejidad que sea necesaria.

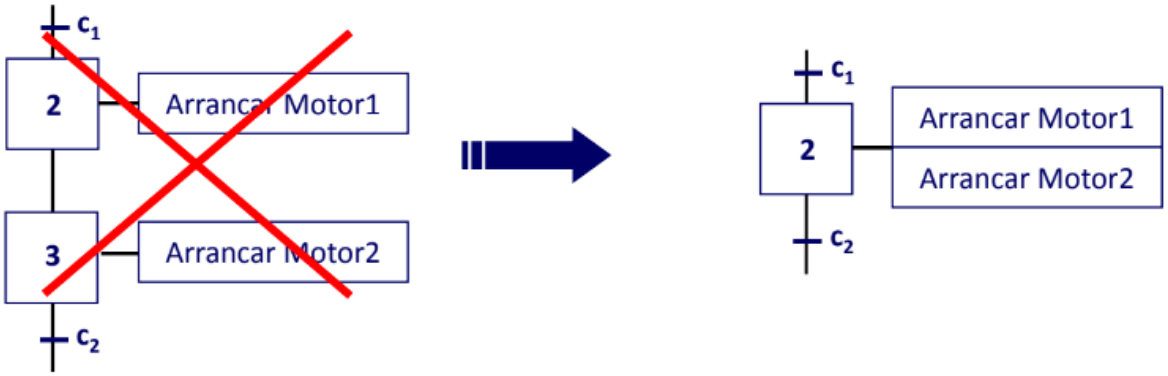


Figura 8. Ejemplo sintaxis de GRAFCET en etapas

El mismo caso para las etapas y acciones del GRAFCET, si es necesario colocar dos acciones se deberán colocar en la misma etapa y en ningún caso podrán colocarse como acciones asociadas a dos etapas unidas directamente.

Hay que destacar de aquí también que no es posible la unión de una etapa mediante una transición con ella misma ya que se entraría en un bucle infinito que rompería la coherencia del GRAFCET.

En cuanto a la evolución del GRAFCET, cada etapa va a tener asociada una variable de estado con formato “Xi” y del tipo booleano, además de dos posibles estados, activo, representado con un 1, o inactivo, con un 0. La etapa marcada con la doble línea normalmente iniciará la simulación del GRAFCET, pero esto solo será si se da un arranque en frío, es decir, que el proceso se inicie sin ningún guardado de la situación anterior. Su contraparte es el arranque en caliente, en el que haya registros en la memoria de alguna situación anterior. En el caso de codificarlo con TIA portal se utilizan el bloque OB100 y OB1 para el arranque en frío y producción normal respectivamente.

Como se ha comentado anteriormente, en la evolución vista desde el punto de las etapas, se tiene que una etapa que no sea inicial se activará cuando la etapa anterior esté activa y la condición entre ellas sea verdadera. Además, al cumplirse esta condición, la etapa anterior, que se encontraba activa, se desactivará, pasando a estar activa la que se encuentra después de la transición que hay entre ambas.

Desde el punto de vista de las transiciones se pueden diferenciar cuatro casos atendiendo a la evolución en el sistema:

- Transición no validada: se da cuando la etapa inmediatamente anterior a esta no se encuentra activa y, por tanto, no se evaluará la condición asociada a dicha transición, siendo una situación que no es dinámica en el proceso ya que no hay posibilidad de avanzar a la etapa siguiente si la transición anterior no está validada.

- Transición validada: se da al estar activa la etapa que precede a la transición. La condición asociada a dicha transición ya es evaluable pero aún no se cumple por lo que no se va a dar una evolución en el GRAFCET.

- Transición franqueable: la etapa anterior a dicha transición está activa y, además, la condición asociada es verdadera, por lo que se puede dar una evolución, pero en este instante aún no ha sucedido, por lo que se dice que la transición está en un estado franqueable pero aún no ha sido franqueada completamente disparando la etapa siguiente a esta.

- Transición franqueada: se corresponde con el estado siguiente al anteriormente comentado, la condición es verdadera y la transición ya ha sido franqueada, por lo que se desactiva la etapa anterior y se activa la que se encuentra inmediatamente después de la transición que acaba de ser evaluada.

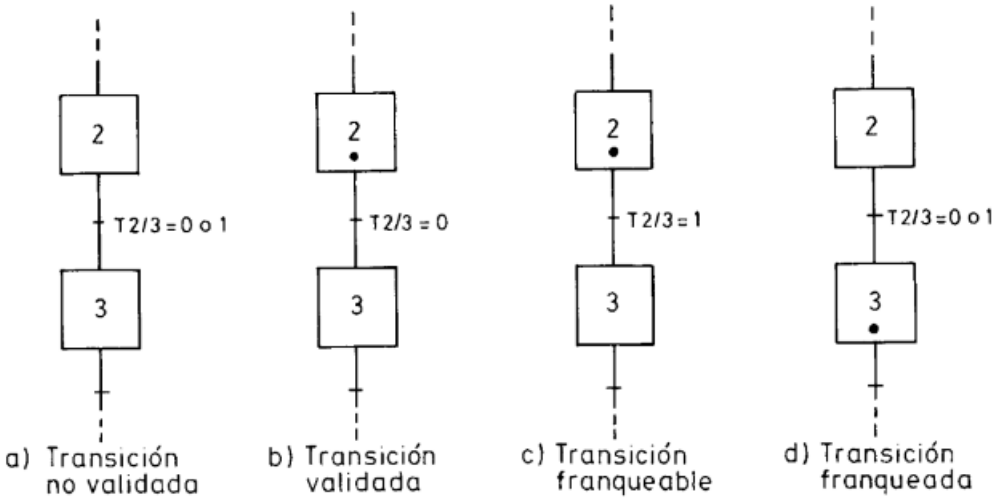


Figura 9. Evolución en transiciones

2.2 XML schema de un grafcet

Otro de los objetivos principales para los que se está elaborando este proyecto es la posibilidad de conectar el simulador, con una planta física o un gemelo digital. Para esto es necesario hacer uso de un lenguaje de marcado, que permita interpretar a la máquina la representación gráfica como una combinación de elementos con sus respectivos atributos.

Gráficamente la estructura de este documento XML se representa como un árbol expansible que ayuda a dar una visión general de todos los elementos que componen cualquier GRAFCET. El proyecto de automatización que se quiera realizar estará definido por su nombre e incorporará al menos un proyecto GRAFCET, este tendrá como mínimo dos etapas, indicando cuál de ellas es la inicial y cumpliendo con la necesidad de que este sea cíclico.

Entre cada una de las etapas debe haber una transición, por lo que todo proyecto de automatización deberá constar al menos de una. Toda transición se identifica con su id y su receptividad que será evaluada cuando dicha transición esté siendo validada.

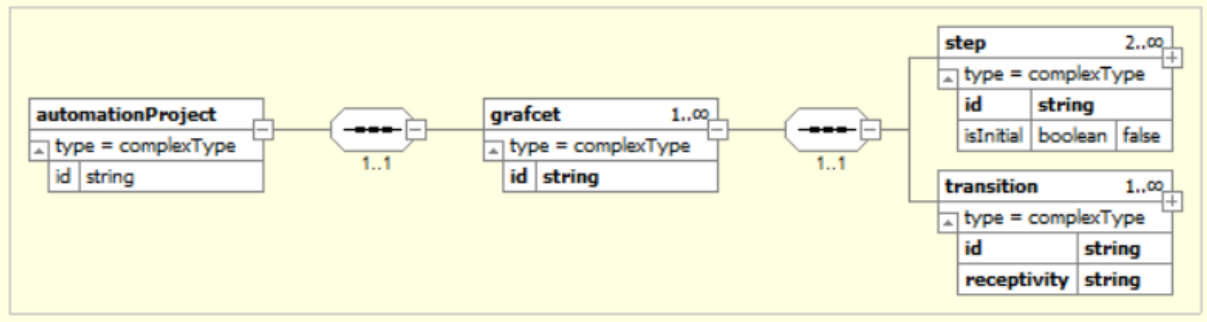


Figura 10. Estructura XML de un proyecto GRAFCET

En la Figura 11 se observan el resto de elementos como son las acciones, relacionadas con la parte de las etapas, y las propiedades asociadas a las transiciones. Estas propiedades ayudan a explicar el concepto de divergencias selectoras y de simultaneidad. Asociada a cada transición y según la sintaxis que se ha descrito en el GRAFCET, debe haber al menos una etapa antes y una después de cada transición, así se crean las propiedades que se reflejan en el documento XML y que ayudarán a identificar si hay divergencias o convergencias. En el caso de una divergencia de simultaneidad una misma transición tendrá como destino tantas etapas como ramas se tengan en la simultaneidad. Por otro lado, en las divergencias selectoras habrá n transiciones, correspondientes a las n ramas de la divergencia, que tengan su origen en una misma etapa, que es de la que nace dicha divergencia. En el caso de las convergencias ocurre al contrario, la transición que cierra la divergencia de simultaneidad tendrá como origen la misma cantidad de etapas como ramas se tenga en la divergencia y la etapa en la que convergen las ramas de una divergencia selectora será a su vez el destino de las n transiciones que se tengan en dicha divergencia.

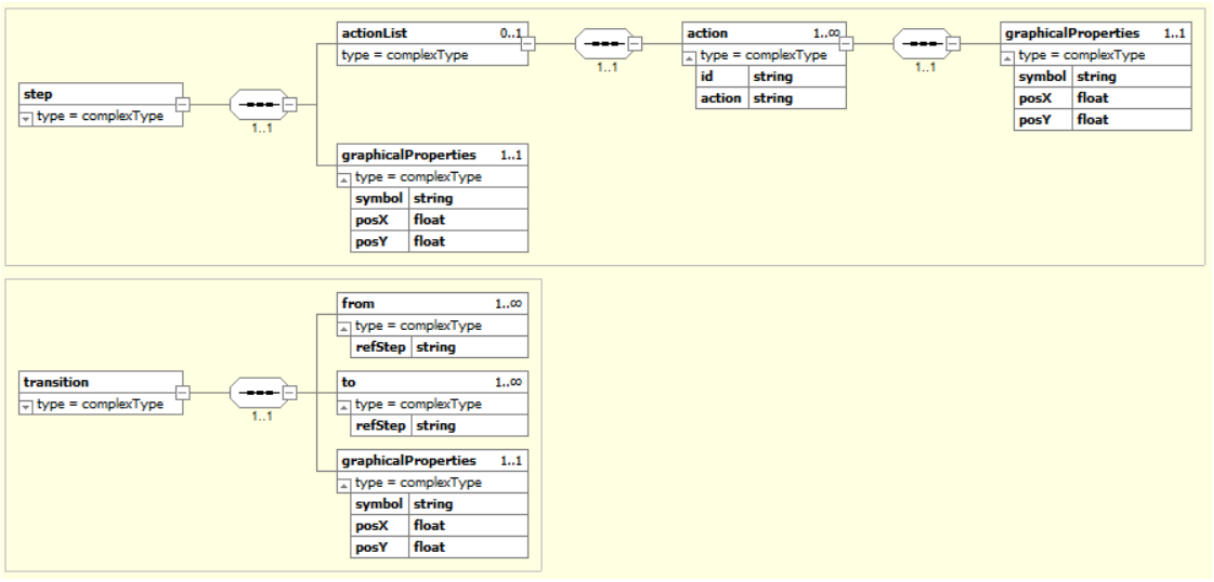


Figura 11. Árbol XML de etapas y transiciones

Cada etapa en su interior va a tener una serie de elementos en el árbol XML, entre los que se van a encontrar una lista de acciones, que puede o no tenerse, ya que no es necesario en todos los casos que una etapa vaya acompañada obligatoriamente de una acción asociada. En caso de tener dicha lista de acciones esta puede ir desde una acción hasta las que se quieran incluir para que se realicen en el momento en el que la etapa con la que se hayan relacionado estas acciones esté activa. El nombre de esta casi siempre irá resumido, por ejemplo, en el caso de un cilindro A que se quiera expandir la acción asociada puede llamarse “A+”.

Para que el software pueda incluir estos elementos de los que se ha hablado en la interfaz se han incluido las propiedades gráficas de cada uno, con sus coordenadas en “X” e “Y”.

Desde el punto de vista de las transiciones se sabe que, según las reglas de sintaxis, que no se pueden colocar dos seguidas, por lo que al menos irán precedidas por una etapa y, además tendrán como origen etapa diferente.

Aprovechando esta característica se establecen las propiedades “to” y “from”, se pueden definir las divergencias y convergencias del GRAFCET de la forma que se ha explicado anteriormente. Así, una divergencia selectora se define como aquel conjunto de transiciones que tienen como “from” la misma etapa. Una convergencia selectora se define como aquellas transiciones que tienen como “to” la misma etapa.

Las divergencias de simultaneidad son una transición que tiene una única etapa en “from” y múltiples etapas destino “to”. Finalmente, una convergencia de simultaneidad se define como aquella transición que tiene múltiples etapas “from” y sólo una etapa destino o “to”.

Las propiedades gráficas informan de la posición en “X” e “Y” que tiene cada transición, necesarias para que el software que se quiere diseñar las pueda colocar en el lugar correspondiente.

2.3 Ejemplo proyecto de automatización

A la hora de abordar un problema con un automatismo real el primer paso será hacer un resumen de las entradas y salidas del sistema y, una vez se tenga claro se procederá a realizar el esquema en GRAFCET del automatismo haciendo uso de estas para definir las etapas y transiciones del proyecto. Para simplificar el esquema se usarán nombres acortados de las entradas y salidas indicando siempre a cuál se refiere cada uno de estos (en otras palabras se hará uso del nombre de la variable que representa la I/O en cuestión). En el caso de las entradas es muy común encontrar un pulsador de marcha, encargado de dar luz verde a la ejecución de un ciclo del proceso, y que se suele denotar con “PM”.

Se va a analizar como ejemplo en este apartado un caso sencillo de automatismo como es el movimiento del vástago de un cilindro A de doble efecto, con capacidad para expandirse (A+) y retraerse (A-). Un sensor nos informará cuando el cilindro se encuentre en posición de reposo, es decir, totalmente retraído (a0), y cuando se encuentre completamente expandido, accionando el fin de carrera (a1). Por tanto, las entradas del sistema serán el pulsador de marcha (PM), la señal del sensor cuando el cilindro se encuentre totalmente expandido (a1) y la señal, también dada por el sensor, que nos informe de que el vástago se encuentra retraído (a0).

Por otro lado, las salidas serán la orden para expandir el cilindro (A+) y la orden que comenzará el movimiento inverso y llevará el vástago a la posición de reposo, (A-).

El vástago inicialmente debe encontrarse completamente retraído y al accionar el pulsador de marcha debe comenzar a expandirse hasta tocar el final de carrera, una vez el sensor informe que el vástago está completamente expandido comenzará el movimiento inverso, este se detendrá en la posición que ocupaba al inicio dando información de esta el sensor y comenzando un nuevo ciclo en el que se volverá a esperar que se accione el pulsador de marcha para repetir el proceso.

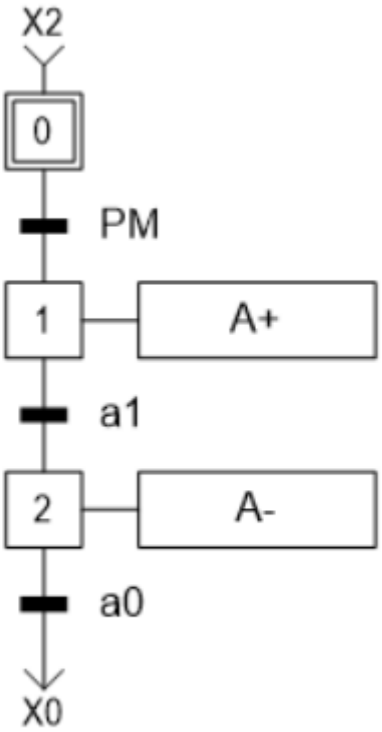
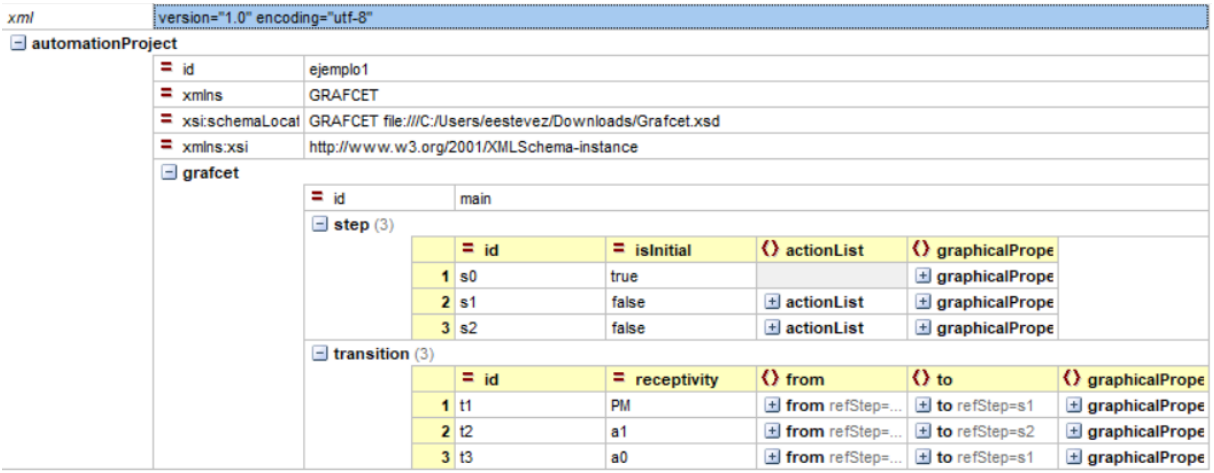


Figura 12. GRAFCET ejemplo en sfcEdit

Una vez se saben las entradas y salidas del sistema y conociendo el proceso que debe realizar el automatismo se puede descomponer en una serie de etapas, en las que se realicen las acciones que vendrán determinadas por lo que se le pide al automatismo en cada momento, y transiciones que controlen la activación de las etapas en el momento que sea necesario. El GRAFCET de este automatismo, como se puede ver en la Figura 12, consta de tres etapas con sus respectivas tres transiciones intercaladas, en este caso va a ser lineal ya que no se va a tener necesidad de incorporar ninguna divergencia, y tampoco se va a hacer uso de acciones internas como el uso de contadores o temporizadores.



xml	version="1.0" encoding="utf-8"					
automationProject						
	id	ejemplo1				
	xmlns	GRAFCET				
	xsi:schemaLocat	GRAFCET file:///C:/Users/eestevez/Downloads/Grafcet.xsd				
	xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance				
	grafcet					
		id	main			
		step (3)				
			id	isInitial	actionList	graphicalPrope
			1 s0	true		graphicalPrope
			2 s1	false	actionList	graphicalPrope
			3 s2	false	actionList	graphicalPrope
		transition (3)				
			id	receptivity	from	to	graphicalPrope
			1 t1	PM	from refStep=...	to refStep=s1	graphicalPrope
			2 t2	a1	from refStep=...	to refStep=s2	graphicalPrope
			3 t3	a0	from refStep=...	to refStep=s1	graphicalPrope

Figura 13. Ejemplo de la estructura de un automatismo básico

La primera etapa (s0), va a ser la inicial, hará la función de etapa de espera hasta que se accione el pulsador de marcha, asociado como condición a la transición que le sigue. Cuando esta transición sea flanqueada la etapa inicial se desactivará y se activará la etapa que se encuentra delante de dicha transición, en la Figura 13 se observa que se trata de la etapa uno o s1, a la que está asociada la acción de expandir el vástago. La transición que le sigue ahora está validada y su receptividad asociada es “a1”, que como se ha comentado, es la señal que manda el fin de carrera del cilindro. El paso a la última etapa se dará cuando esta transición se encuentre flanqueada, desactivando la etapa inmediatamente anterior. El objetivo y acción asociada a esta última etapa será la de retraer el vástago hasta la posición inicial activando la señal del sensor que informará cuando dicha posición se alcance y constituyendo la última condición, asociada a la tercera transición, encargada de cerrar el ciclo, desactivando la tercera etapa y devolviendo el foco a la etapa inicial, que de nuevo se encontrará a la espera de la acción de “PM”.

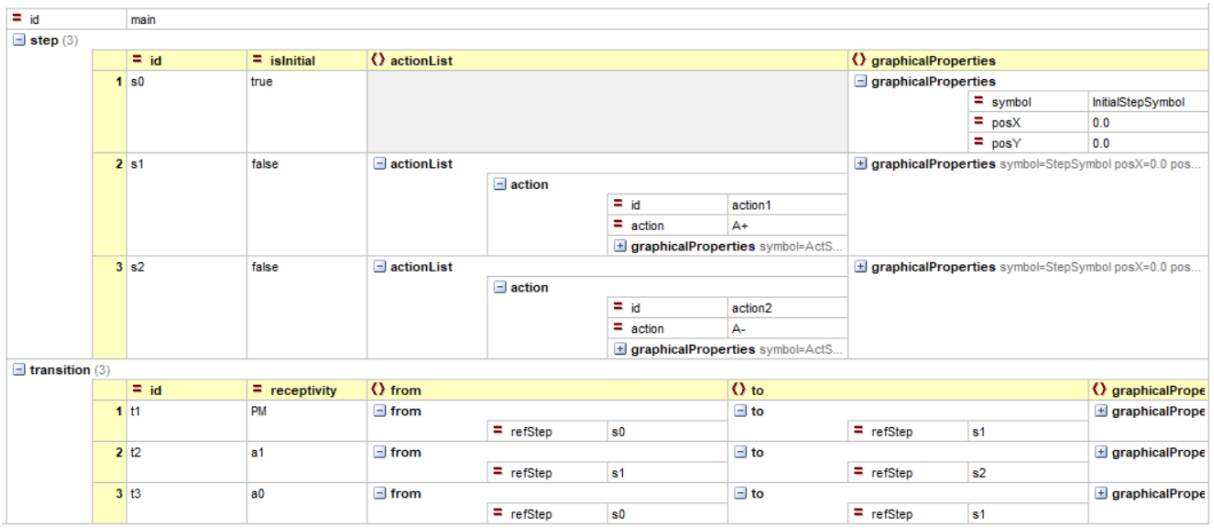


Figura 14. Estructura de etapas y transiciones en un automatismo básico

Por último se puede observar que cada elemento tiene un apartado de propiedades gráficas con la finalidad de ubicar en el proyecto la posición de cada etapa, acción y transición

3 Diseño de simulador grafcet

Para el diseño de una herramienta que simule tanto la parte gráfica como el comportamiento de un automatismo se han tomado como ejemplo softwares como sfcEdit, una herramienta de edición de GRAFCET, pero que no permite la simulación. Este apartado se centra en el diseño del simulador GRAFCET propuesto en dicho TFG, que permite visualizar y actualizar en todo momento el estado en el que se encuentra el automatismo.

Para ello, el motor de dicha herramienta se programará con el lenguaje de programación Java, con objeto de que pueda ser utilizado en cualquier plataforma. Se trata de un software algo más restrictivo, pero a su vez ofrece todos los recursos para poder implementar todas las funciones mencionadas anteriormente, añadiendo características que hagan lo más sencillo posible el diseño de GRAFCETs para el usuario y permitan, además de tener la parte esquemática, analizar el comportamiento que tendría con un modo de simulación.

Con respecto a la parte gráfica de cada elemento de GRAFCET, como se ha comentado anteriormente, es necesario, diseñar el símbolo gráfico correspondiente a cada uno de los elementos que podemos encontrar a la hora de elaborar un GRAFCET. Por ejemplo, a la hora de añadir una etapa se debe diseñar el cuadrado con el número entero correspondiente a dicha etapa en su interior, y en el caso de que esta sea inicial, se debe dibujar con una doble línea. También es necesario el diseño y la edición de los elementos asociados a las etapas y transiciones como son las acciones y las receptividades, en el caso de las acciones será más complejo ya que irán representadas con otro símbolo gráfico y se debe tener en cuenta que, al añadir más de una en una misma etapa, se debe dibujar debajo de la anterior. Por último, tanto las divergencias como las convergencias deben tener un símbolo gráfico distintivo, diferenciando además cuando se trate de divergencias o convergencias de simultaneidad o selectoras. Además, se tiene la posibilidad de conectarse a las I/O de una planta real o virtual y emular el ciclo scan, cogiendo estas entradas y salidas que en programas conocidos como TIA portal se corresponde con la tabla de símbolos.

Los siguientes subapartados están orientados en primer lugar en mostrar la estructura en lenguaje de marcado que van a adquirir las entradas y salidas del sistema de control. Este documento será el utilizado tanto al cargar en el programa dichos elementos del sistema como para conectarse a una planta física o gemelo digital. Se va a continuar con un apartado que describirá la estructura general que se ha utilizado para desarrollar el proyecto, mostrando las herramientas que han servido como ayuda a la hora de la programación del software. Por último, se describirá un flujograma general del funcionamiento del entorno, ordenando los procesos que se producen en este y mostrando un acercamiento a la aplicación.

3.1 Fichero con las I/Os del Sistema de Control

Las I/Os o entradas y salidas de un sistema de control son fundamentales a la hora de definir las acciones correspondientes a las etapas y las receptividades necesarias para garantizar la evolución del automatismo. Como se ha comentado anteriormente las acciones definidas a través de salidas del sistema de control, así como las receptividades como una combinación lógica de entradas se conocen como acciones y receptividades externas. Esta, a diferencia de las internas (relacionadas con el uso de contadores y temporizadores) depende exclusivamente del Hardware del sistema a automatizar. Estas entradas y salidas dan información del sistema y son fundamentales para definir las combinaciones lógicas que se van a poder usar como acciones y receptividades.

A la hora de crear el programa y empezar el diseño del GRAFCET es fundamental conocer las I/Os del sistema con el que se va a trabajar. Para ello se va a incorporar un archivo XML que las contenga para poder usarlas posteriormente a la hora de configurar las acciones y receptividades. La estructura de dicho fichero es que tanto las entradas como las salidas se caracterizan por tres propiedades (identificador, dirección de memoria, así como su valor)

xml version="1.0" encoding="utf-8"

AP_IOList

input (3)

	id	address	value
1	a0	I0.0	true
2	a1	I0.1	false
3	PM	I0.2	false

output (2)

	id	address	value
1	A+	Q0.0	false
2	A-	Q0.1	true

Figura 15. Estructura del archivo XML de entradas y salidas

La Figura anterior, recoge la lista de I/Os utilizadas para el ejemplo anterior, donde se manipulaba un vástago de doble efecto y el objetivo del automatismo era que al accionar el pulsador de marcha el vástago se expanda hasta tocar el fin de carrera, y cuando este sensor informe de que se ha tocado, volver a retraerlo hasta la posición inicial.

A continuación, se analizará una por una las diferentes entradas y salidas que se tendrían en este caso. Como entradas se tiene el pulsador de marcha, denotado como PM. Se trata de un pulsador normalmente abierto cuyo fin es iniciar la expansión del cilindro al accionarlo. Al ser normalmente abierto su valor va a ser siempre falso a no ser que se accione, en ese instante y durante el tiempo que se mantenga pulsado su valor cambiará a verdadero, es decir, dejará pasar la corriente y esto será traducido a un 1 binario. Además del pulsador se tiene un sensor de fin de carrera, que informa de la posición inicial y retraída del cilindro mediante la variable “a0” y detecta, haciendo verdadera la variable “a1” cuando el cilindro toca el fin de carrera. Como se puede ver en la Figura 15 inicialmente el pulsador de marcha está sin accionar y el cilindro se encuentra retraído, justo como se explica en el ejemplo de un proyecto de automatización.

Como salidas y posibles acciones se tiene la expansión del cilindro, denotada en el documento como “A+” y la retracción de éste, que figura como “A-”. Estas van acompañadas también del valor verdadero o falso, según si se está realizando la acción en el caso de que la etapa correspondiente esté activa o no se esté realizando en caso contrario.

Por último, se tiene asociada una dirección de memoria, encabezada por una “I” en caso de las entradas, o por una “Q” en caso de las salidas. Este procedimiento es el que se lleva a cabo en TIA portal al elaborar la tabla de símbolos.

3.2 Estructura general del entorno

Como se ha comentado antes, se ha decidido desarrollar la aplicación en Java. Teniendo esto en cuenta se podría haber utilizado un entorno de programación que facilite la tarea como puede ser Eclipse o Netbeans. En este caso se ha optado por el segundo con pequeños matices, como librerías incorporadas para optimizar la creación del software de la forma que se tenía pensada, ya que gráficamente no es tan potente y la creación de interfaces de usuario incorporando dibujos y el resto de recursos que se piensan utilizar puede derivar en demasiadas líneas de código, además de demasiado tiempo invertido cuando se tienen opciones que facilitan esta tarea. (Java | Oracle, 2022)

3.2.1 JDOM2

El primer paso que se necesita hacer a la hora de realizar un ejercicio con un proyecto de automatización va a ser identificar las entradas y salidas del sistema de control por lo que para el funcionamiento del software y a la hora de programar lo primero debe ser asegurarse que el proyecto puede interpretar o se puede pasar un documento con un lenguaje de marcado para incorporar estas entradas y salidas al mismo. Estas entradas y salidas se identifican con las acciones y receptividades externas. En este programa no se va a hacer uso de las internas como uso de contadores, pero en caso de que se fueran a utilizar habría que programarlas internamente, como su nombre indica, en vez de incorporarlas desde un documento externo al programa.

JDOM se trata de una biblioteca de código abierto hecha para la manipulación de datos XML optimizados para Java, incluyendo sobrecarga de métodos, colecciones, etc. Se ha incorporado como una dependencia al proyecto para desarrollar la función principal de resumir en XML lo que se tenga en la pantalla principal del proyecto, es decir, las etapas y transiciones con sus distintas propiedades, todo en el formato de árbol que se describe en anteriores apartados. También se ha aprovechado para transformar un documento XML con dicho formato en un esquema grafcet en pantalla, es decir, también desarrolla la función inversa. (JDOM, 2022)

Todo esto abriendo un explorador de archivos para poder elegir donde queremos guardar el documento o qué documento es el que se quiere abrir, capturando un error si no se cumple el formato o si no se encuentra el archivo.

3.2.2 Processing

En un proyecto en el que se quieren representar distintos elementos dentro de una estructura general, formando un esquema en conjunto, va a ser necesaria una herramienta gráfica que permita incorporar con facilidad dichos elementos que necesitamos a la hora de ir creando el GRAFCET. Además será necesario un menú para editar las acciones del proyecto y las transiciones para poder añadir combinaciones lógicas de las entradas que hemos incorporado con JDOM en el apartado anterior. Otra función que cobra importancia en cualquier aplicación, como por ejemplo un editor de textos, es una barra de herramientas o el uso de botones, características que ya vienen incorporadas en las clases de Java que ya podemos encontrar creadas por otros usuarios.

Si se habla de botones y GUI, Java parece una buena opción para desarrollar toda la base del código del software, pero dada la poca potencia gráficamente hablando que posee Java, sería necesario incorporar o hacer uso de otras bibliotecas o programas que estén orientados a desarrollar proyectos más visuales o que den más facilidad a la hora de crear interfaces, añadiendo estas características a las que ya se pueden utilizar en Java y dando como resultado una opción interesante para el software que se quiere crear.

Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto, de fácil aprendizaje y utilización y además centrado en la producción de proyectos multimedia e interactivos. No precisa de grandes conocimientos de programación, ya que hace uso de elementos sencillos y una sintaxis simplificada con un modelo de programación de gráficos. Al estar basado en Java se convierte en una herramienta que hereda toda su funcionalidad y puede resultar un complemento perfecto para el proyecto permitiendo aprovechar la facilidad a la hora de programar elementos gráficos que posee. (Processing, 2022)

Se ha hecho uso de esta herramienta incorporando las librerías correspondientes a Netbeans, aprovechando la gran cantidad de bocetos y funciones incorporados a su subclase PApplet, de la que mediante el concepto de herencia en Java se va a basar la clase principal del software.

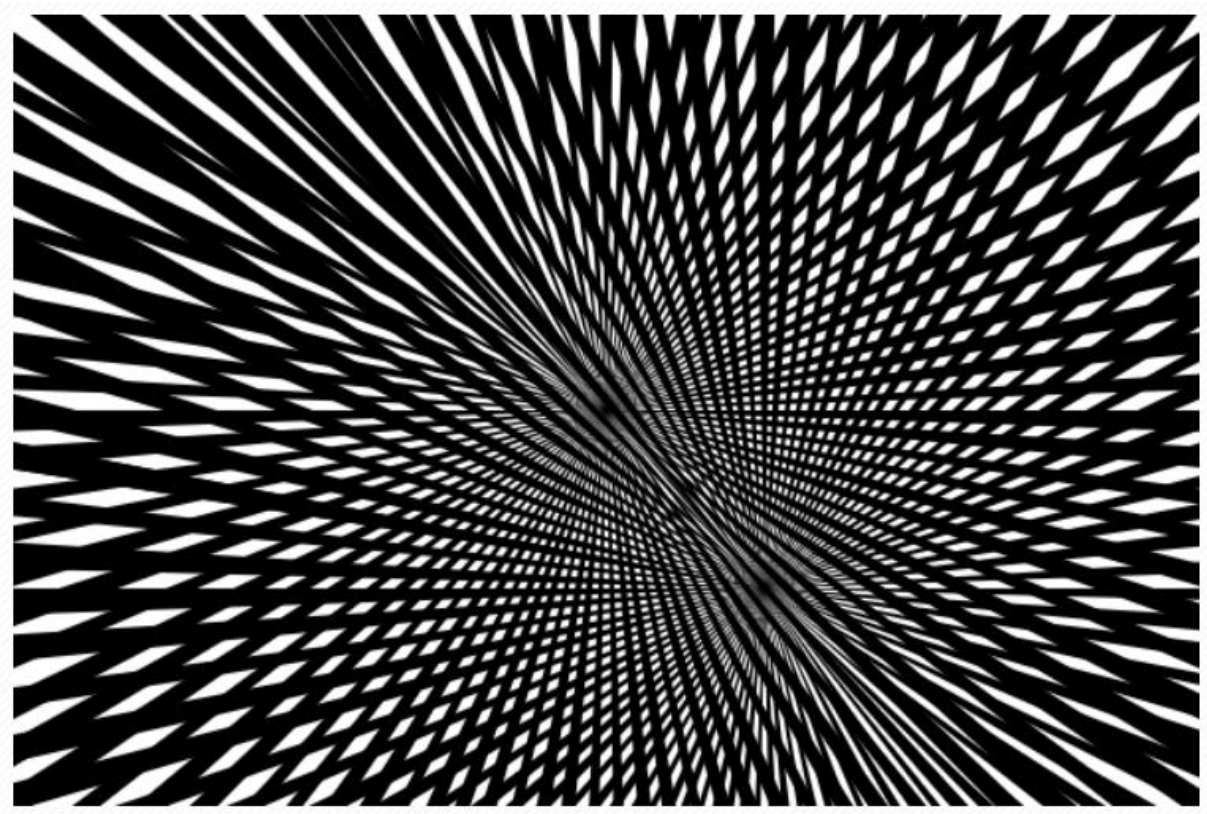


Figura 16. Ejemplo gráfico en Processing

Además en la página de Processing se encuentran varios ejemplos de programas gráficos que se pueden desarrollar con esta herramienta, un ejemplo es el que se puede ver en la Figura 16, en el que se forma un mosaico que cambia al deslizar el ratón por la ventana principal de la aplicación. (Processing, 2022)

Una vez descrito el entorno y visto un ejemplo de cómo se pueden aprovechar los recursos que presenta, se puede pasar a describir la forma en la que se va a hacer uso de dichos recursos en el desarrollo de la aplicación. A la hora de representar elementos se tiene la facilidad de poder dibujar una figura en una única línea de código introduciendo como parámetros las posiciones y dimensiones de esta. En este caso en concreto se tiene la ventaja, además, de que no es necesario hacer uso de figuras simples para representar otras más complejas, dado que sólo hará falta el uso de cuadrados y líneas rectas, además de incluir texto.

Para representar una figura determinada se tienen herramientas con funciones como establecer unos ejes de coordenadas en el lugar que se desee, cambiar el grueso de la línea con la que se va a dibujar o, si se prefiere, que no haya línea. Se podrá además variar el color de esta o del relleno de la figura usando escala de grises o RGB (Red, Green, Blue). Estos son básicamente los recursos usados a la hora de dibujar cualquier elemento de GRAFCET.

Usando como ejemplo una etapa, esta debe tener como símbolo un cuadrado (en caso de que esta sea inicial un doble cuadrado) con un número entero en su interior que la represente y, además, al seleccionarla, deben cambiar de color los bordes para diferenciarla de otra que no esté marcada y al simular el GRAFCET dichos bordes deben ser de otro color diferente al anterior para marcar que la etapa está activa, diferenciándola así de las que no lo estén. Así se tendría un ejemplo sin código de cómo se pueden aprovechar los recursos gráficos de Processing en la creación de los elementos que se van a utilizar.

De cara a marcar elementos, moverlos, o diseñar GUI esta herramienta es también una gran opción ya que posee un abanico de recursos y funciones que permiten detectar cuando se mueve el ratón, cuando se hace clic o cuando se pulsa el teclado, identificando el carácter que ha sido pulsado, entre otras cosas. Estas funciones se han utilizado por ejemplo al diseñar la barra de desplazamiento, ya que necesita detectar dónde se encuentra el ratón para deslizar la barra, o para seleccionar una etapa, cuando el ratón se encontraba sobre ella y se hacía clic, dando también la posibilidad de moverla siguiendo la posición de este.

3.2.3 Netbeans

El programa principal sobre el que se basa el proyecto y el encargado de incorporar las bibliotecas de los anteriores apartados debe ser un entorno de desarrollo que facilite la escritura del código del proyecto, que como se ha comentado va a ser en Java. Un entorno de desarrollo integrado o IDE es una aplicación informática que consiste normalmente en un editor de código fuente, herramientas de construcción automática y un depurador con el objetivo de facilitar el trabajo de programación a un desarrollador. En muchos casos estos tienen auto-completado inteligente de código (IntelliSense). (Ramos Salavert & Lozano Pérez, 2000)

Netbeans es un IDE que permite crear aplicaciones a partir de módulos, que no son más que archivos Java que contienen una serie de clases creadas por el usuario. Estos módulos pueden ser extensibles por otros usuarios agregando unos nuevos, lo que facilita la extensión de las aplicaciones creadas en este entorno de desarrollo.

Java es un lenguaje de programación orientado a objetos por lo que trataremos a cada elemento que queramos añadir a nuestro programa como una clase a partir de la cual crearemos un objeto con las propiedades o atributos necesarios para cumplir la función que queremos que tenga dicho elemento. Por ejemplo, cada etapa del grafcet será tratada como un objeto del tipo Etapa con atributos como ancho y alto. (Domínguez Dorado, 2005)

Para poder utilizar Netbeans se debe instalar la máquina virtual de Java. En este caso se ha decidido utilizar la versión de jre 1.8 y además se debe tener un jdk o Java development kit, que incluye el intérprete y las clases de Java creadas por otros usuarios anteriormente, aparte del depurador y otras herramientas de desarrollo. Se ha decidido usar la versión 11.0.16 de este último ya que, pese a que es una versión algo antigua, es la última versión que incluye un motor que interpreta código de Javascript, necesario para una de las funciones del programa en la que se debe interpretar el texto que se encuentra dentro de las receptividades para evaluarlo y así saber si la transición es franqueable o no. (Java | Oracle, 2022)

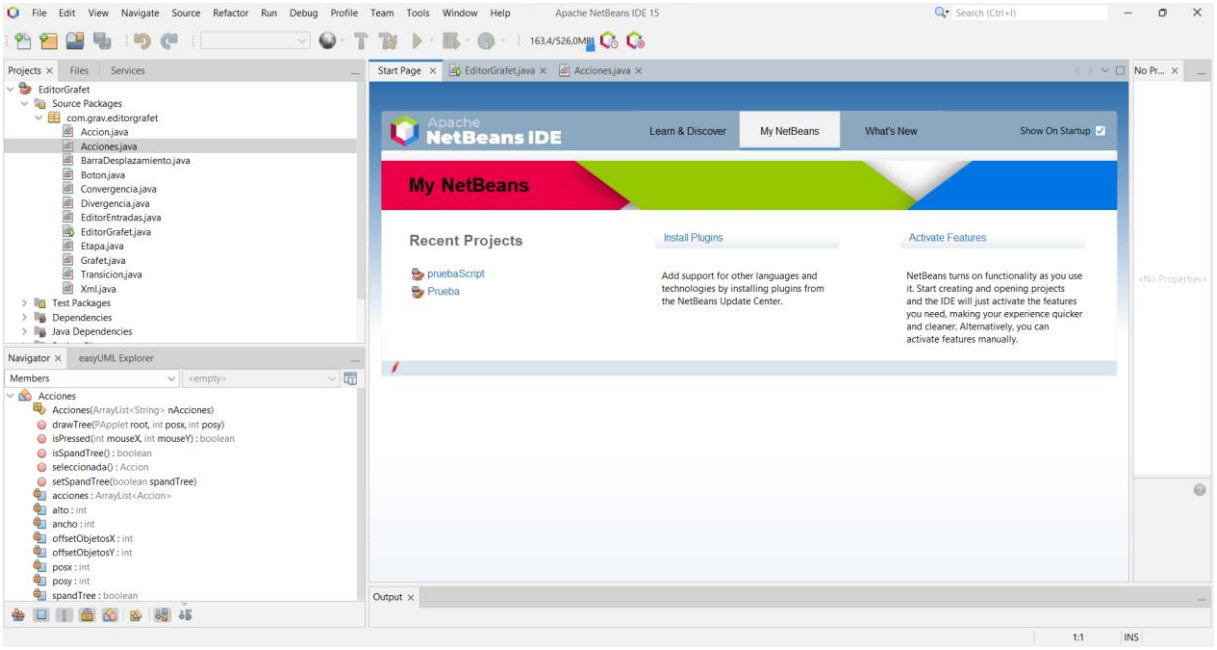


Figura 17. Interfaz de Netbeans

La interfaz de Netbeans es la que se puede ver en la Figura 17, con una vista a la izquierda para los proyectos, en la que se puede observar que cada uno de ellos se divide en paquetes y dependencias. En la parte de paquetes se encuentra el código de cada una de las clases en las que se divide el programa mediante modularización, que no es más que la estrategia de dividir dicho programa en un conjunto de subprogramas que además faciliten la comprensión del código para otros usuarios que quieran interpretar el contenido de la aplicación. Cada subprograma o clase es interpretada como un objeto para Java, es decir, es como si se tuviera una clase principal que hiciera referencia a un coche y el conjunto de clases de apoyo fueran los elementos de este, como ruedas, chasis y elementos que pueden llegar a ser optativos pero que pertenecen a los elementos que tiene un coche por lo general como es puede ser la climatización. (Apache, 2022)

A la derecha se tiene la ventana de código, en la que se programan las diferentes clases de programa y justo debajo en la ventana de output se muestra el resultado al compilar el código en el programa. En caso de imprimir algo por pantalla se hará en esta ventana de abajo.

Justo a la izquierda abajo se tiene el navegador, necesario para moverse por las clases del programa de una forma más sencilla, a parte de dar una vista global de los métodos y atributos que se tienen en dicha clase.

Por último y, como en la mayoría de los programas se tiene una barra de herramientas en la parte superior, con las opciones para crear proyecto, guardarlo y compilar el programa, entre otras funciones.

3.3 Flujograma general del engine del entorno

Ya definidas las entradas y salidas del sistema de control y la estructura que se va a tener en el proyecto, se va a definir el funcionamiento y el orden en el que se llevan a cabo los procesos dentro del proyecto mediante un flujograma. Cómo se ha comentado a la hora de llevar a cabo un proyecto de automatización, lo primero será identificar las entradas y salidas del sistema de control que se tenga, en este caso reflejadas en dos documentos, uno para las entradas que corresponderán con las receptividades externas del proyecto, y otro para las salidas, que se identifican como las acciones externas en un GRAFCET, ambos en lenguaje XML para que pueda ser interpretado en el programa. Una vez se tengan identificadas estas entradas y salidas y se tenga claro el funcionamiento del automatismo se puede elaborar el GRAFCET con la secuencia de etapas y transiciones, siempre respetando el orden de etapa-transición-etapa y cerrando el mismo ya que siempre debe ser cíclico.

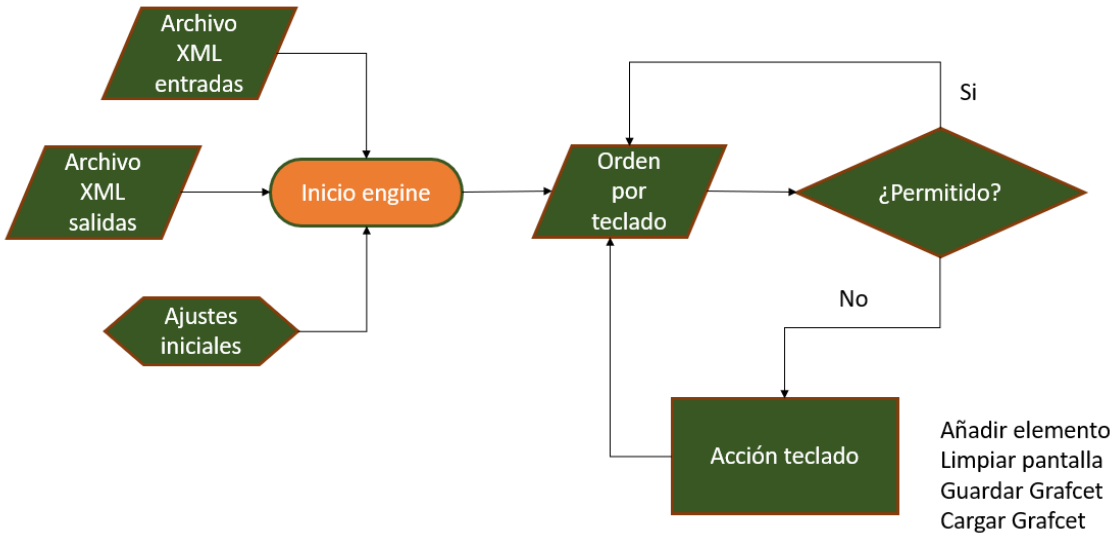


Figura 18. Flujograma del proyecto

En la Figura 18 se caracteriza el inicio con el símbolo correspondiente y un color distintivo, a la izquierda, los procesos que tienen lugar antes y a la derecha, las opciones una vez iniciado el engine del entorno.

Simbolizados como entradas, se tienen los archivos XML de entradas y salidas, responsables de inicializar los editores correspondientes con las entradas y salidas del sistema, así, en el editor de acciones sólo tendremos las correspondientes a las que se han cargado con el archivo y el editor de transiciones sólo dejará colocar como receptividades una combinación lógica de las entradas que se hayan cargado con el documento correspondiente. También se encuentran a la izquierda del inicio los ajustes iniciales, que se encargan de dar tamaño a la ventana del programa y de inicializar los elementos necesarios como listas de array, variables, etapa inicial, etc.

Una vez la ventana de la interfaz se haya hecho visible el programa está preparado para recibir órdenes por teclado. Estas pueden ser colocar una etapa, una transición, una divergencia o cargar un Grafcet anterior, entre otras. El programa evaluará si la orden que se le ha dado es coherente en ese punto del Grafcet y tomará dos caminos posibles: que la orden concuerde con la sintaxis básica de Grafcet y, por tanto, se pueda realizar o, que no pueda llevarse a cabo en ese punto porque la sintaxis no se cumpla.

4 Desarrollo del simulador Grafcet

Para el desarrollo del programa haciendo uso de Netbeans ha sido necesaria la creación de varias clases, una por cada elemento, a parte de una para leer los documentos XML y escribir en ellos según la estructura que se ha determinado. Todas estas sirven como apoyo al programa principal en el que se tiene todo el código utilizado para los settings de la interfaz, la forma de añadir los elementos y el relacionado con las diferentes interacciones que se pueden realizar en la ventana del software.

Para una mejor representación se va a hacer uso de la Figura 19, que representa el diagrama de clases del proyecto.

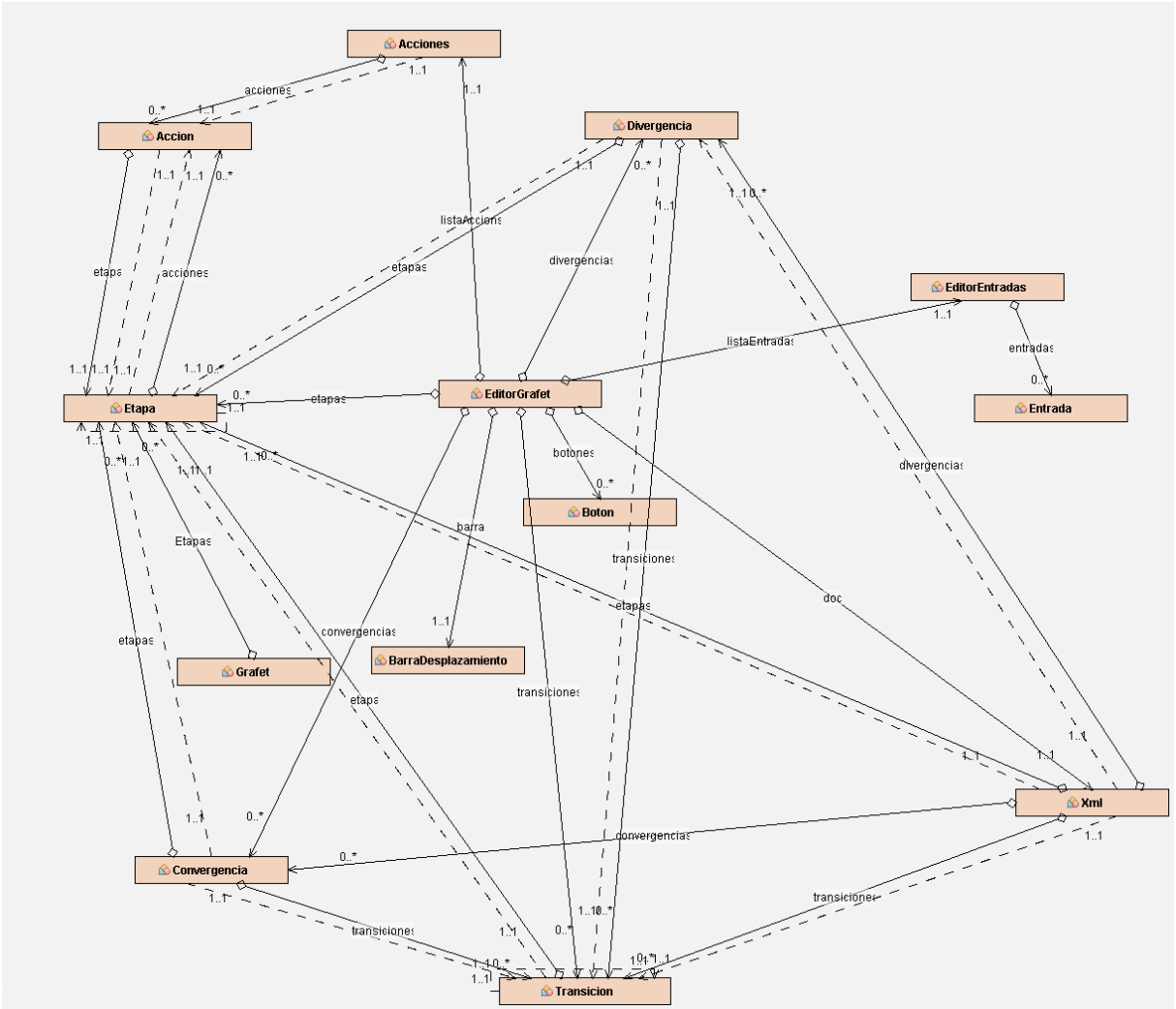


Figura 19. Diagrama de clases del proyecto

Una clase es un conjunto de objetos con los mismos atributos, operaciones, métodos, relaciones y semántica, por tanto, se van a agrupar las distintas clases y mediante flechas se van a trazar las relaciones entre estas.

Existen tres tipos de relaciones posibles entre clases pero en este programa:

- Asociación: describe la relación entre dos clases mediante la incorporación de un objeto de una clase a otra distinta, utilizando lo que se conoce como instancia de clase. Para entenderlo mejor se tomará como ejemplo la relación entre un cliente y una cuenta en un banco. En este caso Cliente y Cuenta serán las dos clases. Cada cliente que llegue al banco será diferente y cada cuenta de las que se creen también va a ser única, pero siempre se va a tener la misma relación descrita: un cliente puede crear una o varias cuentas en dicho banco. Esta relación que se forma es la que se describe en este apartado. Por ejemplo, en este programa se ha creado un array de la clase Etapa en el programa principal. Estas relaciones se marcan con una línea continua.

- Dependencia: se trata de una relación en la que una clase debe hacer uso de un método de otra, como si se tratara de una relación entre un cliente y un proveedor, en la que el cliente utiliza un servicio dado por el proveedor. Cliente es el objeto que solicita el servicio y Proveedor es el objeto que provee el servicio solicitado, hablando más en términos de Java. Se simboliza con una línea discontinua.

- Herencia: se trata de cuando objetos de distintas clases tienen atributos similares y exhiben comportamientos parecidos. Como ejemplo se puede tomar el caso de Animales y Mamíferos. Los mamíferos tienen los atributos y comportamientos generales que presenta un animal, además de los suyos específicos que solo presentan los mamíferos y que, por ejemplo, no tendría un reptil. Esta relación se representa como una línea continua terminada en flecha señalando la clase que hereda de la otra.

Una vez que se sabe esto ya se puede interpretar el diagrama de clases de la Figura 19**,** correspondiente al diseño y distribución de clases que se ha decidido hacer para el programa.

La creación del proyecto comenzó por la clase principal EditorGrafcet, a la que se le fueron añadiendo relaciones gracias al resto de clases creadas para referenciar a los elementos de un Grafcet.

Para los editores se han creado dos clases, Acciones y EditorEntradas, con un atributo destinado a ser rellenado con los nombres de las entradas y salidas del sistema, leídas de documentos XML que proporciona la clase con este mismo nombre, en orden de facilitar la edición del GRAFCET con opciones para las acciones y receptividades que sean posibles en el sistema que se esté trabajando.

Por último, la clase XML, programada para poder leer y crear documentos XML, necesarios para guardar el estado del GRAFCET como un documento de este formato, poder leer documentos de este tipo y transformarlos en el esquema correspondiente a los datos que haya almacenados.

Con este diseño y relaciones entre clases se incluyeron métodos en la clase principal, que hereda de PApplet e importa las librerías de processing, capaces de interpretar las acciones que se pueden introducir por teclado y representar los elementos correspondientes evaluando en todo momento si es posible y dando lugar a un esquema Grafcet simulable.

4.1 Elementos del GRAFCET

Para la creación de cada uno de los elementos de GRAFCET se ha incorporado una clase con el nombre del elemento que se quiere incluir y dentro de dicha clase se añaden los atributos como el grosor de línea o la posición en “X” y en “Y”. Pertenecen a este subapartado las siguientes clases:

4.1.1 Etapa

Para este elemento se deben crear atributos tales como la longitud del lado del cuadrado que representa su símbolo o el número entero que debe ir en el interior del mismo y que debe ir aumentando una unidad con cada etapa añadida al esquema. También la posición en la que se encuentra el centro del cuadrado como ayuda para dibujar el elemento, el grosor de la línea o el tamaño de letra. Y como variables lógicas unas que indiquen que parte de la etapa está seleccionada, la entrada, la salida o la etapa en sí. Una vez definidas las propiedades principales que identifican a una etapa se pasa a los métodos que se utilizaran.

La encapsulación es un concepto importante en la programación, al encapsular una propiedad o atributo de una clase A el resto de clases (B, C, D…) del programa ven este atributo como una caja negra y no pueden acceder a su valor. Aquí nace la importancia de crear métodos que permitan obtener o cambiar los valores de los atributos de una clase. Estos son llamados setters y getters y su función tiene lugar cuando se crea un objeto de una clase A en otra distinta, B. Por ejemplo, se va a necesitar tener etapas en otras clases del programa. Se sabe que cada etapa va a tener, por ejemplo, una posición en X, pero no se quiere que esta posición se pueda manipular fácilmente, por lo que se encapsula y para el resto de clases del programa esta posición en X es una “caja negra” a la que no se puede acceder. La función de los métodos setters y getters, programados en la clase A, es la de obtener o cambiar el valor de dichos atributos a los que no se podía acceder desde otras clases (B, C, D…). Así se obtiene una forma de modificar estos atributos de una forma que no sea directa.

Una vez definidos los métodos setter y getter necesarios es necesario definir un método que detecte cuando se está pulsando con el ratón encima de una etapa y se encargue de que una vez se pulse aparezca como seleccionada. Por último un método que dibuje, según el tamaño de letra, la longitud de las líneas predefinida y el ancho del trazo, el símbolo de cada etapa con el número que corresponde en su interior.

4.1.2 Transición

Para las transiciones se tendrán atributos similares, como grueso de línea y tamaño de letra, para representar la receptividad asociada. Se tendrá también la longitud que ocupa la línea de la transición y las posiciones en “X” y en “Y”. También una variable lógica con la misma misión que las de Etapa, saber cuándo se ha seleccionado la transición mediante un método que captura en qué posición se ha hecho clic y comprueba si la transición se encontraba en dicha posición. Como se ha hecho referencia en apartados anteriores es necesaria una propiedad que indique las etapas que preceden y sucesoras. Esta propiedad se define como un vector, ya que es posible que haya desde una hasta infinitas etapas inmediatamente antes o después de cada transición. Directamente relacionado con esto se tendrán métodos setter y getter, entre ellos el método setter que permite editar las receptividades asociadas a cada una de las transiciones.

Se destaca un método que permite encontrar, dado un vector con transiciones, si una determinada etapa, definida por el número entero que le corresponde, precede directamente a una de las transiciones del vector o se encuentra inmediatamente después. Esto hace posible una de las características más importantes a la hora de poder colocar elementos en el GRAFCET, ya que si una transición ya tiene etapas relacionadas tanto antes (from) como después (to) suya no será posible añadir elementos para no romper la sintaxis de GRAFCET.

Por último, es necesario un método que haga uso, al igual que con las etapas, de las propiedades necesarias para representar cada transición y la dibuje en la ventana del programa.

4.1.3 Acción

Como en anteriores casos esta clase presenta los atributos necesarios para dibujar el elemento correspondiente, en este caso, la acción. Estos son el alto y ancho del rectángulo, el grosor de línea o el tamaño de letra. En este caso no se tienen atributos para la posición ya que se utiliza como atributo la etapa a la que están relacionada las acciones, tomando de esta su posición ayudándose de los métodos setter y getter de Etapa. También se ha utilizado una variable lógica para saber cuando está seleccionada la acción y cuando no.

Como método, a parte de todos los setter y getter, necesarios en cualquier clase, se tiene el método para dibujar el elemento con el nombre de la acción a realizar dentro del rectángulo que lo simboliza.

4.1.4 Divergencia

Para las divergencias también se ha creado una clase ya que será necesario dibujar el símbolo y diferenciar entre los tipos que hay. Esto se ha hecho con una variable “enum” o tipo enumerado que puede tomar dos valores, OR en caso de que la divergencia sea selectora y AND en caso de que sea de simultaneidad. La anchura de la divergencia está marcada por dos variables, una que representa el offset a la derecha y otra a la izquierda, siendo manipulables con el objetivo de ensanchar la divergencia. El número de ramas de la divergencia también se almacena en una variable que se pedirá al momento de crear una divergencia. También se ha añadido como variables dos vectores que se van a rellenar con los primeros elementos de cada rama. Se rellenará un vector, con el mismo número de etapas que de ramas tenga la divergencia, en caso de divergencia AND y siendo cada una la primera etapa de cada rama, o de transiciones en el caso de la divergencia OR, para más tarde añadir un método que permita ampliar la anchura de la divergencia si se mueve uno de los elementos que se encuentren en dicho vector.

Como resumen de los métodos se encuentran los setter y getter de las variables que se quieren que sean manipulables desde otras clases del programa y el método que se encarga de dibujar la divergencia atendiendo los atributos como el grueso de línea y la posición donde se quiere dibujar. En este caso el elemento no será seleccionable como lo era en las clases anteriores.

4.1.5 Convergencia

Para las convergencias se han utilizado los mismos atributos que en las divergencias: posiciones, número de ramas, tipo de convergencia, offsets variables según la amplitud de las convergencias que se quiera y los vectores de etapas y transiciones al igual que se explicó antes, dependiendo del tipo de convergencia que se trate. Como métodos se tienen los correspondientes setters y getters de los atributos que se quiere que sean manipulables desde otras clases y el método que será el encargado de dibujar la convergencia, haciendo uso de los atributos definidos para cada convergencia que se quiera crear, y siempre teniendo en cuenta que para poder añadir una convergencia debe de haber una divergencia abierta.

4.2 Ficheros XML

Una de las partes fundamentales del programa es el desarrollo de una clase capaz de escribir o interpretar un archivo XML de un formato determinado. Las funciones para las que fue pensada esta clase fueron: cargar la lista de entradas y salidas, leyendo desde un XML, que dependerá del sistema automático que se quiera simular y guardar un proyecto GRAFCET mediante los elementos que haya en pantalla y los atributos necesarios para replicar dicho GRAFCET, leyendo el archivo XML que ha sido generado al guardar un proyecto anterior.

Esta clase está estrechamente relacionada y tiene relaciones de asociación con todas las clases anteriormente nombradas excepto acciones, ya que van incorporadas a las etapas. Por tanto, un vector para cada elemento de cada tipo será suficiente para poder almacenar y cargar la información incluida en archivos XML. También se crean como atributos tres vectores necesarios a la hora de leer las entradas, ya que cada una de estas tendrá un nombre, una dirección de memoria y un valor, verdadero o falso.

Como resumen de métodos en el programa se encuentran los encargados de la escritura y lectura del XML, responsables de guardar y cargar el contenido del GRAFCET, y el método encargado de leer las entradas y salidas del sistema de control. Cuando se quiera guardar en un documento un diagrama creado en el software se hará uso del método que crea un archivo XML y gracias a JDOM se guardan los elementos siguiendo la estructura presentada en apartados anteriores. Al revés con el método que lee; este aprovecha los archivos XML creados por el método anterior para cargar un diagrama con todos sus elementos y atributos. Por último un método para leer las entradas y salidas y cargarlas en el programa utilizando también un archivo XML pero esta vez con una estructura diferente y también definida anteriormente en la memoria. A este último habrá que pasarle desde otra clase un vector de acciones que va a rellenar con las salidas del sistema de control.

4.3 Editores de acciones y transiciones

Otra de las funciones que debe presentar el software es un editor para las etapas que permita añadir combinaciones de las salidas del sistema como acciones a realizar cuando dicha etapa se encuentre activa. Además, un editor para las transiciones que permita editar las receptividades de estas, encargadas de marcar la evolución de los estados del GRAFCET durante la simulación.

Al ser dos clases muy parecidas y que se han diseñado siguiendo la misma idea se van a resumir en un único apartado, marcando las diferencias entre una y otra. Los atributos serán los mismos en las dos clases, ya que los editores van a tener el mismo formato: un árbol desplegable en el que se muestren todas las opciones. La única diferencia será a la hora de cargar el árbol con dichas opciones, para ello el editor de acciones necesitará un vector con las salidas y el editor de transiciones un vector con las entradas.

Aquí se debe aclarar una cosa, y es que antes no se ha hablado de la clase entrada ya que es una clase interna del editor de transiciones. Cada entrada va a tener los atributos que se recogían en el apartado de XML y que son el nombre, la dirección de memoria y el valor, que será verdadero o falso, inicialmente.

Los atributos de los editores, salvando esta diferencia comentada, serán los mismos: una posición en “X” e “Y” donde se situará el editor dentro de la interfaz del programa, el ancho y alto de la pestaña inicial del editor, un atributo lógico que informará de si el árbol está desplegado o no y dos offsets, uno para cada coordenada, que se utilizarán para separar los distintos elementos del árbol.

Cada árbol debe ser expansible haciendo clic en él, por lo que se debe diseñar un método en cada editor que expanda el árbol al colocar el ratón sobre este y hacer clic. También un método que devuelva el elemento del árbol que se ha pulsado al estar expandido, como recurso para poder editar las acciones o receptividades. Por último un método común en las dos clases para dibujar los dos árboles, salvando las diferencias entre estos, y el resto de métodos setter y getter, que al tener atributos muy similares, serán también muy similares entre ellos.

El editor de transiciones es algo más complejo, por lo que se deben añadir dos métodos adicionales relacionados con la simulación del GRAFCET, un método que permita cambiar el valor de las entradas, es decir, si tenía valor verdadero, hacerlo falso y viceversa y un método que transforme la expresión que se tenga en la receptividad de cada transición, haciéndola evaluable de forma lógica y pudiendo obtener un resultado verdadero o falso al evaluarla en conjunto.

4.4 Scrollbar

Como en la mayoría de programas, como editores de texto y programas donde se necesite desplazarse por la ventana principal, se debe tener una barra deslizante que permita ver todos los elementos del GRAFCET. Los atributos que debe tener esta clase son los relacionados con el espacio que debe ocupar la barra y su cursor, la posición que ocupa dicho cursor en el eje de coordenadas “Y”, la posición mínima y máxima del deslizante, y dos variables lógicas que indiquen cuando se está sobre el deslizante y cuando se pulsa sobre él, activando la opción de moverlo.

Como métodos, a parte de los setters y los getters, se encuentra un método encargado de variar la posición del deslizante, siempre dentro de los valores marcados como máximo y mínimo que se definen a la hora de crear la scrollbar, y el método que se encarga de dibujar la barra y el cursor.

4.5 Clase principal

Se debe tener una clase principal en la que se va a ejecutar todo el código encargado de hacer funcionar el programa. Al principio de esta clase principal se van a incorporar las librerías y variables que sean necesarias para el desarrollo correcto de la aplicación.

Como atributos se tiene un vector para cada elemento del GRAFCET, es decir, un vector que agrupará etapas, otro para transiciones, divergencias, acciones, etc. Estos vectores se irán rellenando al ir incluyendo elementos en el GRAFCET, y simultáneamente se irán dibujando de forma que se cumpla siempre la sintaxis. También se tiene un atributo lógico para la simulación, que tendrá valor falso inicialmente y al hacerlo verdadero desde el programa se activará el modo simulación, señalando la etapa activa de color verde.

Centrándose en los métodos, se utiliza por primera vez el concepto de herencia explicado anteriormente en las relaciones entre las clases. La clase principal hereda de una clase que incorpora métodos de Processing necesarios para el programa, y, al heredar de esta clase, todos sus métodos pasan también a la clase principal y se podrá hacer uso de ellos. En primer lugar se tienen los métodos que se encargan de inicializar la aplicación, dando tamaño a la ventana principal, añadiendo los editores, una vez leído el archivo XML de entradas y salidas, y añadiendo la barra de desplazamiento y la etapa inicial, entre otras cosas.

Cuando se hayan ejecutado los métodos que inicializan la aplicación se podrá hacer uso del resto de métodos que permiten añadir elementos al GRAFCET, siempre respetando la sintaxis. Estos elementos se van añadiendo a los vectores que se tienen como atributos y se dibujan mediante otra función que se ejecuta de forma cíclica. También es posible mover dichos elementos con otro método, a partir de la posición del ratón. Por último se permite guardar el GRAFCET o cargar uno hecho anteriormente y también entrar en modo simulación para comprobar el comportamiento que se tendría antes de trabajar con una planta real.

5 Ejemplo

Se va a tomar como ejemplo el explicado en el apartado 2.3, se tendrá un sistema automático de expansión y retracción de un cilindro. Inicialmente el cilindro se encontrará en una posición de reposo esperando el accionamiento de un pulsador de marcha, PM, que accione el automatismo y expanda el cilindro lo suficiente hasta tocar la posición final, marcada por un sensor de fin de carrera, que informará cuando este ya haya alcanzado dicha posición y comenzará la retracción del vástago hasta volver a la posición de reposo, esperando de nuevo la señal del pulsador de marcha, normalmente abierto.

Como entradas del sistema y, por tanto, receptividades externas en el GRAFCET se tienen las relacionadas con los sensores y el pulsador de marcha. Se denota como “PM” la variable relacionada con el pulsador de marcha y como “a0” y “a1” las relacionadas con la posición del cilindro marcada por el sensor, correspondiéndose con la posición inicial del cilindro y la marcada por el final de carrera, respectivamente.

Como salidas del sistema se tienen las variables que caracterizarán las acciones externas del GRAFCET. Estas son las relacionadas con el movimiento que realiza el vástago del cilindro, teniendo solo dos acciones, la correspondiente a la expansión del cilindro “A+”, y la que se corresponde con la retracción del mismo, denotada como “A-”.

Tanto entradas como salidas tienen un estado inicial que puede ser verdadero o falso y que viene dado por el documento XML que se tenga, interpretando el enunciado del problema que se quiera resolver. El valor verdadero en las variables de entrada se corresponde con que dicha variable será verdadera al colocarla en una receptividad y si la combinación lógica de entradas es verdadera en su totalidad y dicha receptividad se encuentra validada será franqueable, causando la activación de la etapa siguiente a la transición asociada a esta receptividad. Por otro lado, el valor verdadero en las salidas quiere decir que la acción correspondiente a esta salida se está realizando, por ejemplo, en el momento que “A+” sea verdadero querrá decir que la acción de expandir el cilindro está activa y, por tanto, el cilindro está avanzando hacia la posición del fin de carrera.

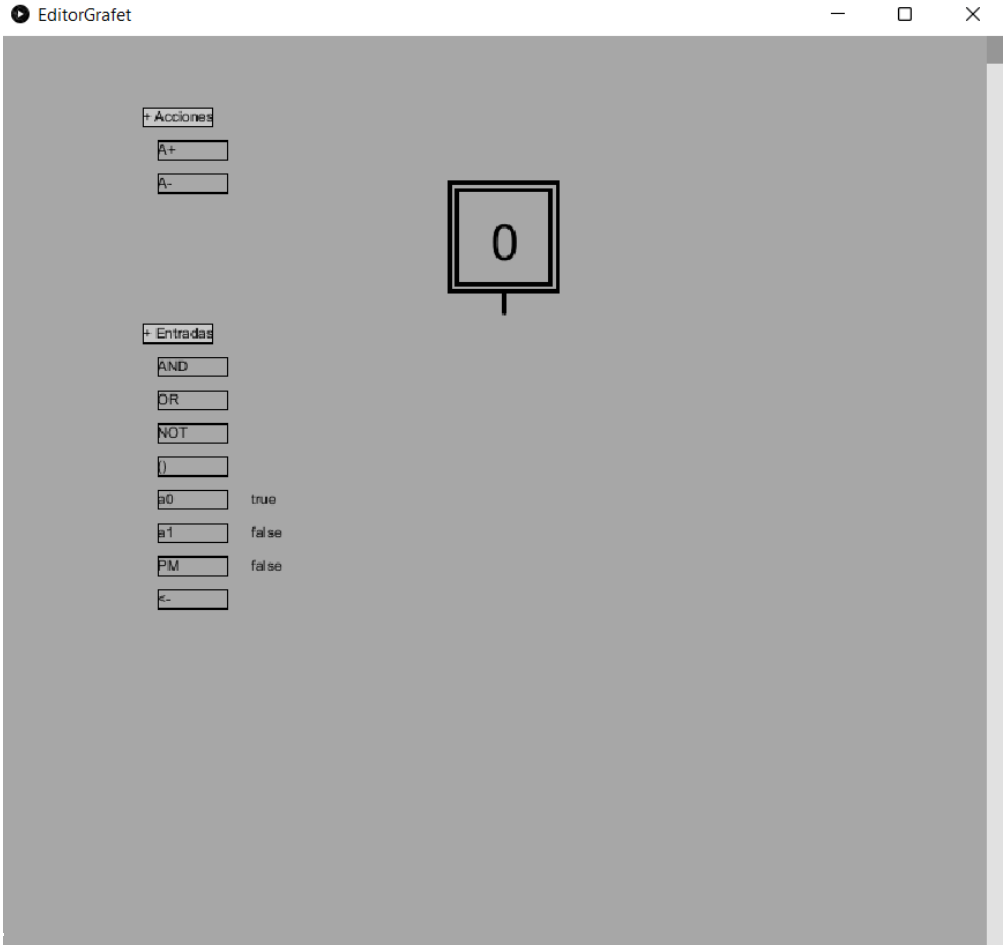


Figura 20. Estado inicial del software

El editor de acciones ya incorpora la expansión, “A+”, y la retracción del cilindro, “A-” y, el editor de transiciones presenta las entradas del sistema, acompañadas a la derecha con su valor booleano inicial, que se puede cambiar haciendo clic en la entrada, de cara a ver como varían los estados del sistema al simular. Además, se añade la etapa inicial remarcada con una doble línea y con el número cero en su interior.

Se debe añadir una transición cuya receptividad imponga la condición de que se deba activar el pulsador de marcha para activar el siguiente estado del sistema y pasar de la etapa cero. Lo siguiente a añadir será una etapa que accione el mecanismo del cilindro para que se expanda cuando esté activa y, con la condición en la transición que le va a seguir de que no se active la siguiente etapa hasta que el sensor del cilindro no detecte el final de carrera, es decir, que el cilindro no esté totalmente expandido.

La última etapa del Grafcet tendrá como acción retraer el vástago hasta tocar el sensor que indique que se encuentra de nuevo en la posición inicial y, de esta forma, cerrar el ciclo.

Como se puede ver en la Figura 21, el Grafcet queda como se ha comentado, cumpliendo la sintaxis con tres etapas intercaladas por transiciones y, en este caso, solamente una acción en las que correspondía. En este ejemplo tampoco se ha visto una combinación lógica asociada a las transiciones, simplemente una condición sencilla, que al hacerse verdadera disparaba la siguiente etapa y la acción relacionada en consecuencia.

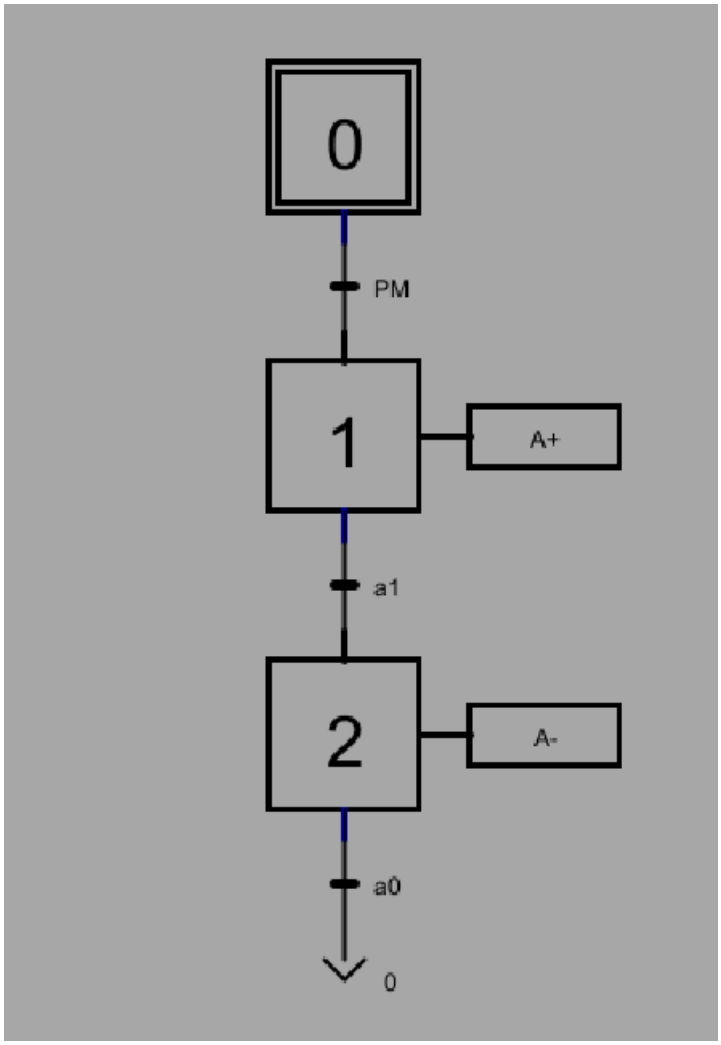


Figura 21. GRAFCET completo del ejemplo

En el modo simulación se marca de color verde la etapa activa y se evalúa la condición de la transición que sigue a la etapa. Si esta es verdadera se activa la siguiente etapa, coloreando sus bordes de verde, y se desactiva la anterior.

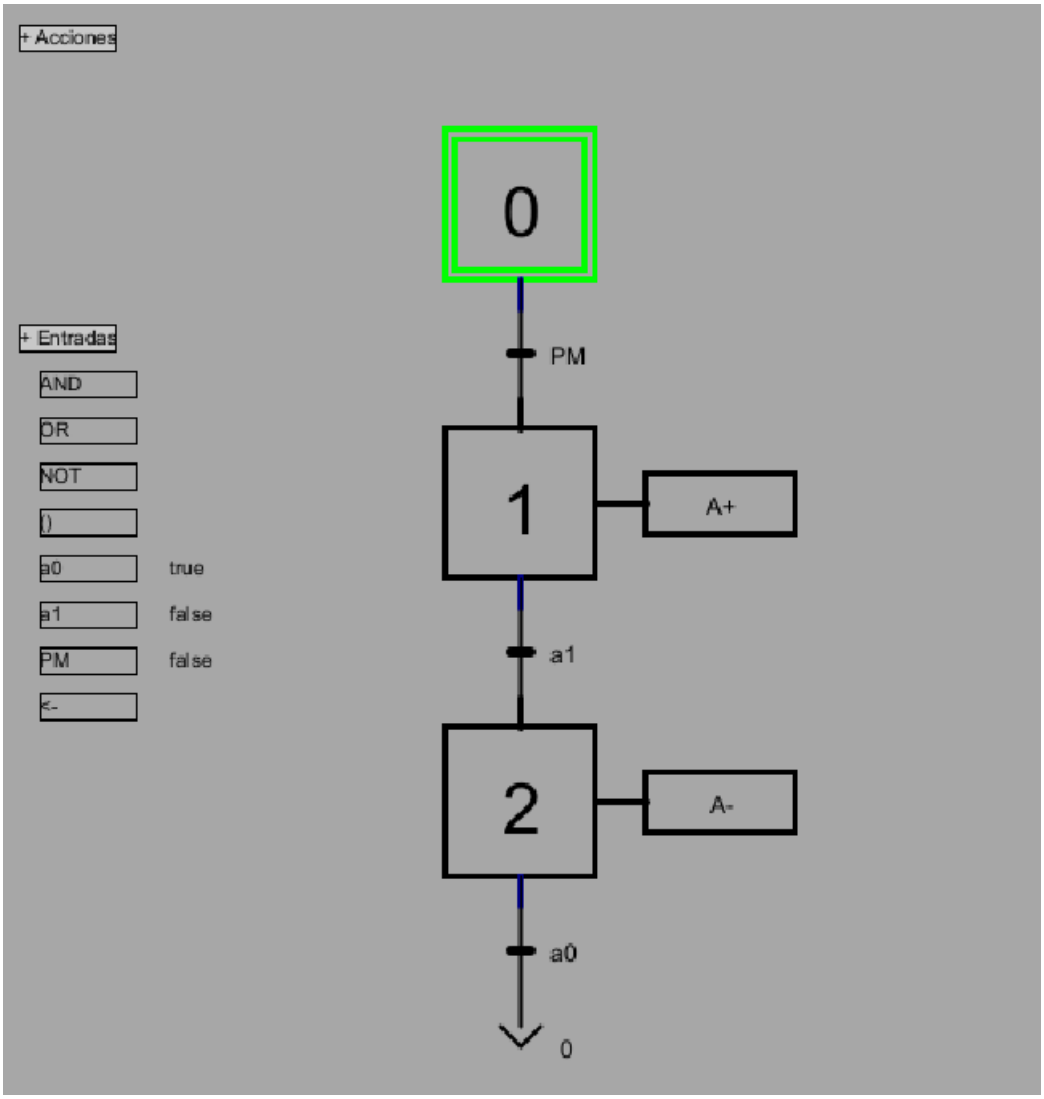


Figura 22. Simulación del GRAFCET ejemplo

Como se observa en la Figura 22, esta activa la etapa inicial pero la condición, en este caso “PM” no es verdadera por lo que no se puede pasar a la etapa uno. En el momento que se haga “true” la variable evaluada, se activará la etapa uno y se realizará la acción de expandir el vástago del cilindro.

Por último se puede ver el proceso de forma gráfica en el siguiente enlace: https://drive.google.com/file/d/1fEa-8KahykBJBoqfZ4MbJ1ZtjLojaX8x/view?usp=sharing

6 Conclusiones y trabajos futuros

A lo largo del desarrollo del proyecto se han debido tomar decisiones en cuanto a la estructura del programa y a las herramientas que era más adecuado utilizar para resolver el problema planteado inicialmente, obligando a hacer uso de todos los conocimientos disponibles o incluso aprender de nuevas herramientas que faciliten o que se adapten mejor a la tarea que se quiere desempeñar.

Las conclusiones a nivel de diseño han sido varias. En primer lugar, el uso del entorno de programación adecuado, con el que se esté familiarizado, para hacer uso de las distintas herramientas y de IntelliSense como apoyo, entre otras ventajas de programar en un IDE. Para la arquitectura del software también es clave hacer uso de las librerías adecuadas, en este caso las de processing, entre otras que han sido incorporadas para hacer el camino lo más rápido posible a la hora de ir solucionando los distintos problemas que iban surgiendo. Una vez ya se había elegido tanto el entorno de programación como las librerías que se van a incorporar a este, se debía ir creando el proyecto en torno a una clase principal y desarrollando siempre primero la funcionalidad, teniendo claro lo que se pedía del proyecto y de qué manera se podía estructurar para que se cumplieran las condiciones en cuanto a funcionamiento. La toma de decisiones en cuanto a la creación de una clase y los métodos que se elaboraban en esta para usarlos como apoyo a la hora, por ejemplo, de dibujar un elemento del Grafcet o de modificar atributos dentro de una clase, por ejemplo modificar la posición de una etapa, desde la clase principal ha sido otra de las decisiones fundamentales a la hora de cumplir con lo que se pedía en cada momento.

En general se han cumplido los objetivos y se ha conseguido una herramienta que va a servir para el aprendizaje de uno de los conceptos en los que se unifican varias ramas de la ingeniería, dando un software que cumple tanto con los objetivos didácticos para asignaturas como Automática Industrial o Automática Avanzada como con los objetivos de llevar esto algo más lejos y conectar el programa a una planta real.

En cuanto a un trabajo futuro, deja abiertas opciones como la incorporación de herramientas más avanzadas como contadores y temporizadores que doten de mayor funcionalidad al proyecto y haga posible la utilización de este software para realizar Grafcets más complejos, teniendo como opción la combinación, como acciones incluidas en las etapas, de salidas internas y externas del sistema. También abre un camino para otras aplicaciones que puedan sacar partido de esta y la complementen, dando cada vez más opciones para trabajar de forma que no implique estar físicamente en un laboratorio.

7 Bibliografía

Apache. (2022). *netbeans.apache.org*. Obtenido de https://netbeans.apache.org/download/archive/index.html

Domínguez Dorado, M. (2005). Todo Programación Nº13. Madrid: Iberprensa.

Java | Oracle. (2022). *www.Java.com*. Obtenido de https://www.java.com/es/

JDOM. (2022). *jdom.org*. Obtenido de http://jdom.org/

Processing. (2022). *processing.org*. Obtenido de https://processing.org/

Ramos Salavert, I., & Lozano Pérez, M. D. (2000). Ingeniería del software y bases de datos: tendencias actuales. En M. D. Lozano Pérez, *Ingeniería del software y bases de datos: tendencias actuales* (pág. 78). Castilla La Mancha.

school of information technology. (29 de 07 de 2022). *www.techtitute.com.* Obtenido de https://www.techtitute.com/informatica/blog/ingenieria-dirigida-modelos#