XXXVIII Jornadas de Automática
https://doi.org/10.17979/spudc.9788497497749.0892

GENERACIÓN DE CÓDIGO IEC 61131-3 A PARTIR DE DISEÑOS EN GRAFCET

M.L. Alvarez
Departamento de Ingeniería de Sistemas y Automática, Escuela de Ingeniería de Bilbao, UPV/EHU
marialuz.alvarez@ehu.eus

I. Sarachaga, A. Burgos, N. Iriondo, M. Marcos
Departamento de Ingeniería de Sistemas y Automática, Escuela de Ingeniería de Bilbao, UPV/EHU
isabel.sarachaga@ehu.eus, arantzazu.burgos@ehu.eus, nagore.iriondo@ehu.es
marga.marcos@ehu.eus

Resumen

Actualmente la colaboración entre aplicaciones es crucial en la ingeniería de automatización. En este trabajo se presenta un método de generación de código de control en texto estructurado de acuerdo con la norma IEC 61131-3 a partir de los GRAFCET de diseños obtenidos con la metodología MeiA y generados con la herramienta que le da soporte. El prototipo desarrollado se ha utilizado en la generación del código correspondiente a un proceso de transferencia de piezas. A la reducción considerable del tiempo y de los errores de implementación de los programas de PLCs que proporciona este trabajo, hay que añadir la simplificación de las futuras tareas de mantenimiento.

***Palabras Clave**: GRAFCET, IEC 61131-3, ST, PLC.*

1 INTRODUCCIÓN

Actualmente, la colaboración entre aplicaciones es crucial en la ingeniería de automatización. El diseño de los sistemas de control requiere la participación de un conjunto multidisciplinar de expertos, que precisan un entorno de diseño capaz de gestionar la colaboración interdisciplinar.

En los sistemas de automatización industrial el Controlador Lógico Programable (PLC) es la tecnología de control establecida. La asociación independiente de fabricante PLCopen se fundó en 1992 con la misión de resolver aspectos relacionados con la programación del control así como promover y dar soporte al uso de estándares en el área. IEC (International Electrotechnical Comission) promueve el uso de sistemas abiertos en el campo del control industrial. El estándar IEC 61131-3 [8] propone un modelo de software y lenguajes de programación textuales (ST - Structured Text, IL - Instruction List) y gráficos (LD - Ladder Diagram, FBD - Function Block Diagram, SFC - Sequential Function Chart). De entre estos lenguajes de programación, el texto estructurado (ST) es el que mejor aborda la creciente complejidad de la programación de sistemas de control y ofrece mayor garantía de interoperabilidad entre herramientas de programación de PLCs [18].

En el diseño de sistemas de automatización, los lenguajes de modelado más utilizados son las Redes de Petri (RdP) [14],[13] y GRAFCET [9]. Numerosos trabajos y publicaciones han demostrado la utilidad práctica de las RdP en muchas áreas de aplicación [13], pese a que el estudio realizado en [7] concluye que las RdP son un lenguaje de modelado complejo. De hecho, el lenguaje de modelado GRAFCET se define sobre la base de las RdP acotándolas para representar eventos discretos secuenciales, además de proporcionar claridad y normalización a la representación gráfica. En este sentido, [4] realiza un estudio comparativo entre las RdP y SFC (Sequential Function Chart) concluyendo que GRAFCET y SFC se pueden ver como una Red de Petri de tipo binaria acotada, donde las transiciones se disparan de forma determinista.

La mayor aceptación de GRAFCET en el campo de la automatización es debida fundamentalmente a su intuitiva representación gráfica, sencillez y claridad para la representación de características como el paralelismo estructural [1].

En este trabajo se presenta un método de generación de código de control en texto estructurado de acuerdo con la norma IEC 61131-3 a partir de los GRAFCET de diseños obtenidos con la metodología MeiA [1] y generados con la herramienta que le da soporte [2]. La metodología MeiA (MEthodology for Industrial Automation systems) ofrece guías de diseño y plantillas que capturan la experiencia de analistas y diseñadores, incluyendo no sólo aspectos estructurales claves, sino también aspectos relacionados con la flexibilidad, modularidad y extensibilidad de los diseños que pueden ser

reutilizados en otros sistemas. Siguiendo las fases y pasos de la metodología se obtienen los diseños en GRAFCET. La generación de código de control a partir del diseño obtenido con la herramienta reduce esfuerzo y tiempo de implementación.

La estructura del artículo es la siguiente: en el apartado 2 se realiza una revisión de trabajos relacionados con la generación de código de control. El apartado 3 presenta el método para la generación del proyecto de automatización, que se implementa en el prototipo descrito en el apartado 4. El apartado 5 ilustra la generación de código correspondiente al proceso "Transferencia de Piezas" utilizando el prototipo. Finalmente, las conclusiones se exponen en el apartado 6.

2 TRABAJOS RELACIONADOS

La colaboración entre aplicaciones, y más en concreto, la colaboración entre herramientas de diseño y programación que permitan la generación de software de control de manera automática reduce de forma considerable el tiempo de implementación de los programas de PLCs y produce una significativa reducción de los errores de implementación.

En la bibliografía se encuentran numerosas propuestas con este objetivo. Algunos enfoques utilizan Desarrollo Basado en Modelos (MDD) para la generación automática de código de control [20], [11]. Éstos presentan métodos y herramientas que son poco aceptadas en la programación práctica de PLCs en la industria.

Utilizando métodos y estándares más próximos al entorno industrial se presentan numerosos trabajos donde se utilizan las RdP para el diseño de sistemas de control, y diferentes algoritmos y métodos para la traducción a código fuente de PLC. En [6] los autores proponen un conjunto de técnicas de transformación de los diseños creados con RdP, denominadas IOPT, al lenguaje LD (Ladder Diagram). En [12] una herramienta implementa un método de transformación semiautomática de FIPN (Fuzzy Interpreted Petri Net) a texto estructurado (ST).

En los trabajos [17] y [10] se presenta la generación de código de PLCs partiendo de los diseños realizados en GRAFCET. En [17] seleccionan SFC como lenguaje de programación, pero requiere una normalización previa de los GRAFCETs antes de realizar las transformaciones debido a que SFC no permite representar las estructuras jerárquicas. En [10], los autores presentan una aproximación de transformación a lenguaje estructurado ST manteniendo las estructuras jerárquicas (macro-etapas y enclosed-grafcets) especificadas en GRAFCET. Los autores seleccionan ST como lenguaje más adecuado para la transformación de los diseños en base a criterios de equivalencia semántica, legibilidad, portabilidad y facilidad de mantenimiento. La razón reside en que aún siendo SFC el lenguaje más cercano semánticamente a GRAFCET, GRAFCET sencillos pueden dar lugar a SFCs complejos al incluir jerarquía.

La generación del código de control que proporcionan estos trabajos reduce considerablemente el tiempo y los errores de implementación. El presente trabajo pretende, además, que el código generado tenga una estructura familiar para los programadores de PLCs, lo cual facilitará el mantenimiento del código.

3 GENERACIÓN DEL PROYECTO DE AUTOMATIZACIÓN

En los procesos de Diseño, la metodología MeiA incorpora el lenguaje de modelado GRAFCET para generar las unidades de organización de diseño (DOUs - Design Organization Units). Se distinguen tres tipos de DOUs:

- DOUs de decisión que organizan el arranque y la parada de los distintos modos de funcionamiento, y coordinan todos los posibles estados del sistema.
- DOUs de producción que realizan las operaciones de producción, coordinación de operaciones, selección de parámetros de producción, etc.
- DOUs auxiliares que realizan procedimientos de inicialización, preparación, paros, avisos, etc.

La jerarquía, sincronización y coordinación de los GRAFCETs se realiza en base a señales de control generadas por los diferentes DOUs. No se utiliza la estructuración mediante macroetapas ni encapsulación. Además, dichos diseños cumplen las siguientes reglas:

- Toda etapa tiene un precedecesor y un sucesor. Una etapa puede tener como precedecesores: una transición o una divergencia en Y o una convergencia en O. Una etapa puede tener como sucesores: una transición o una convergencia en Y o una divergencia en O.
- Toda transición tiene un precedecesor y un sucesor. Una transición puede tener como precedecesores: una etapa o una convergencia en Y o una divergencia en O. Una transición puede tener como sucesores: una etapa o una divergencia en Y o una convergencia en O.
- Una divergencia en Y tiene como predecesor una transición y como sucesor tantas estapas como ramas en la divergencia.

- Una convergencia en Y tiene como predecesor tantas etapas como ramas confluyan y como sucesor una transición.
- Una divergencia en O tiene como predecesor una etapa y como sucesor tantas transiciones como ramas en la divergencia.
- Una convergencia en O tiene como predecesor tantas transiciones como ramas confluyan y como sucesor una etapa.

Partiendo de los diseños realizados en GRAFCET y desarrollados a nivel tecnológico, se generará la programación en la norma IEC 61131-3, distinguiendo: la **Parte Secuencial** que proporciona la secuencia de activación y desactivación de las distintas etapas y la **Parte Combinacional** que implementa las acciones del sistema en función de las etapas activas en cada momento. También se debe tener en cuenta la definición de todas las **Variables**.

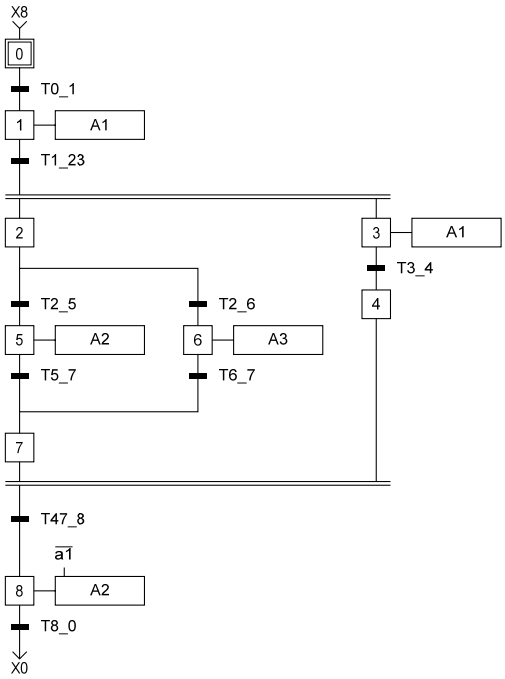

Figura 1: Ejemplo de GRAFCET (GEjemplo)

La Parte Secuencial se implementa con POUs de tipo bloque funcional; por cada DOU se genera un POU. Para ello, en primer lugar, se precisa construir la "Tabla de Set-Reset de las etapas" donde se presentan las condiciones que se deben cumplir para la activación y desactivación de cada etapa del GRAFCET. Una etapa se activa cuando está activada la etapa o etapas anteriores y se cumplen las condiciones de transición entre ambas. Cualquier etapa se desactiva cuando se cumplen las condiciones de transición a la-s siguiente-s etapa-s y dicha transición se haya efectuado (activadas etapas sucesoras).

Tabla 1: Tabla Set-Reset de las etapas

	Set (Activación de Etapas)	Reset (Desactivación de Etapas)
X0	SX0 = X8 . T8_0 + **Init**	RX0=X1 + **Reset**
X1	SX1 = X0 . T1_23	RX1=X2.X3 + **Init** + **Reset**
X2	**SX2 = X1 . T1_23**	**RX2=X5 + X6 + Init + Reset**
X3	SX3 = X1 . T1_23	RX3=X4 + **Init** + **Reset**
X4	SX4 = X3 . T3_4	RX4=X8 + **Init** + **Reset**
X5	SX5 = X2 . T2_5	RX5=X7 + **Init** + **Reset**
X6	SX6 = X2 . T2_6	RX6=X7 + **Init** + **Reset**
X7	SX7= X5.T5_7+X6. T6_7	RX7=X8 + **Init** + **Reset**
X8	SX8 = X4 . X7 . T47_8	RX8=X0 + **Init** + **Reset**

La Tabla **1** presenta la "Tabla Set-Reset de las etapas" para el GRAFCET (GEjemplo) de la Figura 1. Por ejemplo, la etapa 2 (X2) se activará cuando estando activa la etapa 1 (X1) se cumpla la condición asociada a la transición T1_23 y se desactivará cuando se active la etapa 5 (X5) o cuando se active la etapa 6 (X6) dependiendo de la evolución del GRAFCET. La variable **Init** se incluye con el fin de poder activar las etapas iniciales de los GRAFCETs y la variable **Reset** para desactivar todas las etapas. Dichas funcionalidades son requeridas para la puesta en el estado inicial de los GRAFCETs y para la implementación del paro de emergencia.

Para la programación, a cada etapa se le asocia una variable de estado **Xi** (donde i representa el número de etapa) de tipo booleana. Para cada etapa se programan dos estructuras de tipo IF-THEN: una para evaluar las condiciones que activan la etapa y otra para evaluar su desactivación.

```
(*---GEjemplo---*)
X0   : BOOL;
X1   : BOOL;
X2   : BOOL;
...
```

```
(* Set -Reset _____ X2*)
IF ( X1 AND T1_23 ) THEN
    X2:=1;
END_IF;
IF ( X5 OR X6 OR Init OR Reset ) THEN
    X2:=0;
END_IF;
```

Figura 2: Programación de la etapa 2 (X2) del ejemplo, resaltada en la tabla1

Cada transición lleva una condición asociada denominada receptividad, siendo el resultado de la evaluación de la fórmula lógica de tipo booleano. Las receptividades en los DOUs tecnológicos se enuncian como expresiones lógicas que pueden ser traducidas directamente al lenguaje ST.

La **Parte Combinacional** se implementa en el POU principal de tipo Programa, en el cual se instanciará

cada uno de los POUs que implementan la parte secuencial.

Para la programación de las acciones se distinguen distintos tipos. Para la programación de cada una de las señales relacionadas con los actuadores del proceso (salidas) se realiza una búsqueda en las acciones de todos los GRAFCET con el fin de identificar todas las etapas en las que aparece, considerando si es una acción continua, si la acción es condicional o se trata de una asignación (set, reset o valor). Por ejemplo, la salida A2 del GEjemplo, se activará cuando esté activa la etapa 5 (X5) o cuando se active la etapa 8 (X8) si la señal a1 no está activada (1).

```
A2:= X5 OR (X8 AND NOT a1);          (1)
```

Las señales de control que coordinan y sincronizan los DOUs son programadas de la misma forma que las salidas.

Los contadores, temporizadores, pulsos, etc., se implementan instanciando los bloques funcionales que proporciona la norma IEC 61131-3. Las operaciones relacionadas con la inicialización y actualización de los mismos se programan de manera similar a las salidas.

4 PROTOTIPO

Uno de los editores de GRAFCET más extendido es SFCEdit © [19]. SFCEdit proporciona un interfaz gráfico para desarrollar diagramas GRAFCET 100% conforme a la norma IEC 60848 [9]. Por lo tanto, los diseñadores sólo necesitan centrarse en el contenido y no en el léxico y la sintaxis de GRAFCET porque SFCEdit ofrece las directrices necesarias.

SFCEdit permite exportar GRAFCETs a los formatos XML, WMF y EMF. En el presente trabajo el archivo XML exportado es el punto de partida para la generación del código de control correspondiente en el lenguaje de programación ST de acuerdo a la norma IEC 61131-3.

Como marco de desarrollo, ha sido seleccionado el IDE de Eclipse [5]. Este entorno de desarrollo de código abierto y multiplataforma, proporciona un potente editor visual, un excelente motor de depuración y una interfaz gráfica de usuario flexible. Pero también es un sistema de plug-in extensible.

Aunque Eclipse soporta múltiples lenguajes, Java y XML se han utilizado para el desarrollo del prototipo, ya que Java proporciona código portable, mientras que XML proporciona datos portables. Como parser para documentos XML en Java se ha utilizado SAX(Simple API for XML) [16].

En los siguientes subapartados se presenta el esquema XML de SFCEdit y la generación del proyecto de automatización.

4.1 ESQUEMA XML DE SFCDEDIT

El léxico y sintaxis de dicho fichero se presenta en la Figura 3. El elemento raíz es el proyecto *(project)*. Cada proyecto está compuesto al menos por un GRAFCET *(grafcet)* caracterizado por un identificador, tipo (normal, macro o enclosure) y opcionalmente un comentario.

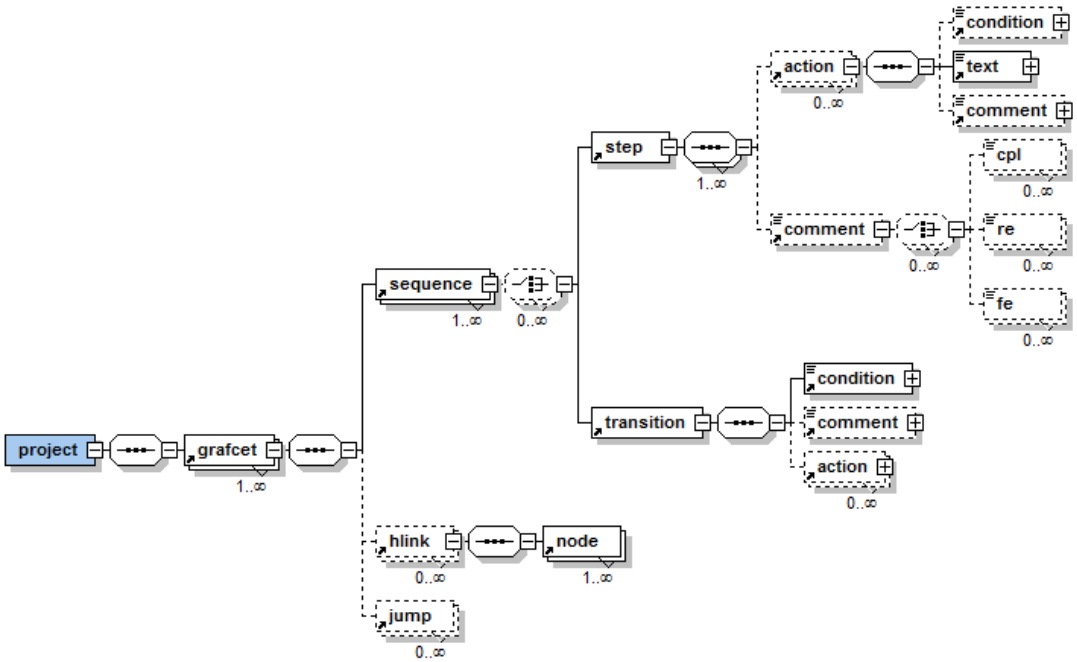

Figura 3: Léxico y sintaxis del editor SFCEdit

Un grafcet está formado por un conjunto de secuencias *(sequences)* y puede incluir enlaces horizontales *(hlink)* y saltos *(jump)* entre secuencias.

Normalmente las secuencias están formadas por un conjunto ordenado de pasos *(step)* y transiciones *(transition)*. Un paso se caracteriza por su identificador, tipo (inicial, final, macro, task, enclosing o inicial enclosing), y opcionalmente un comentario *(comment)* y una lista de acciones *(action)*. Las acciones incluyen el tipo (normal, a la activación, a la desactivación, condicional, forzado, por evento) dependiendo del modo y el instante en el que se deben ejecutar, la descripción textual *(text),* y opcionalmente comentarios *(comment)* y la condición *(condition)* en caso de tratarse de una acción condicional. Las transiciones deben tener obligatoriamente una condición *(condition),* y opcionalmente un comentario *(comment)* y una lista de acciones *(action)*.

Los enlaces horizontales identifican la secuencia que enlazan mediante el atributo "seqid" y la lista de nodos *(node)* que al menos debe contener la identificación de dos secuencias "seqid". Se pueden identificar cuatro tipos de enlaces horizontales: divergencia en Y *(div and)*, divergencia en O *(div or)*, convergencia en Y *(conv and)* y convergencia en O *(conv or)*.

Los saltos deben identificar la secuencia origen "seqid_from" y la secuencia destino "seqid_to".

```
<grafcet type="normal" owner="" name="GExample" comment="">
    <sequence id="1">...
    </sequence>
    <sequence id="2">
        <step type="normal" name="X2" />
    </sequence>
    <sequence id="3">...
    </sequence>
    <sequence id="4">...
    </sequence>
    <sequence id="5">...
    </sequence>
    <sequence id="6">
        <step type="normal" name="X7" />
    </sequence>
    <sequence id="7">...
    </sequence>
    <hlink type="div and" seqid="1">...
    </hlink>
    <hlink type="div or" seqid="2">...
    </hlink>
    <hlink type="conv or" seqid="6">...
    </hlink>
    <hlink type="conv and" seqid="7">...
    </hlink>
    <jump seqid_from="7" seqid_to="1" />
</grafcet>
<grafcet type="normal" owner="" name="Grafcet" comment="">...
</grafcet>
```

Figura 4: XML del GRAFCET GEjemplo

En la Figura 4 se presenta el fichero XML generado con SFCEdit para el GRAFCET GEjemplo y en la Figura 5 se identifican gráficamente las distintas secuencias (en elipses), los tipos de enlaces horizontales (DY, CY, DO y CO) con sus conexiones y el código XML correspondiente a una secuencia, un enlace horizontal (convergencia en Y) y un salto.

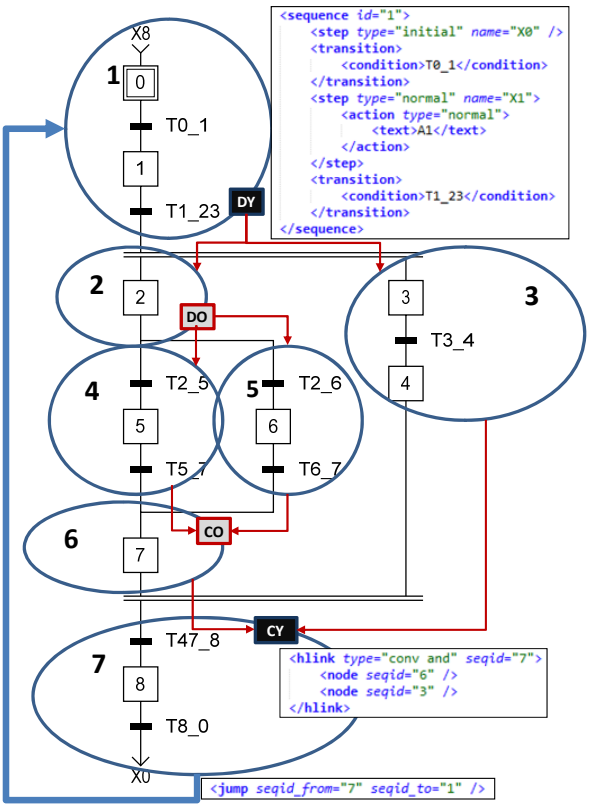

Figura 5: Relación entre el GRAFCET GEjemplo y los elementos definidos en el fichero XML

4.2 GENERACIÓN DE CÓDIGO

La generación del código de control comienza con la validación del fichero XML con el esquema ValidarGrafcet.xsd que presenta dos restricciones al esquema propuesto por SFCEdit: (1) siempre va a existir al menos un salto, y (2) las transiciones no disponen de lista de acciones.

Partiendo del fichero validado se genera el proyecto de automatización, que consta de un conjunto de POUs (bloques funcionales) correspondientes a la parte secuencial, un POU principal (programa) de la parte combinacional y la definición del conjunto de variables del proyecto.

En lo que respecta a la parte secuencial, en primer lugar hay que construir la "Tabla de Set-Reset de las etapas".

La parte correspondiente al Set se realiza identificando los predecesores de cada etapa. Si la etapa es el primer elemento de una secuencia, el predecesor puede ser un enlace horizontal (divergencia en Y o convergencia en O) o un salto,

teniendo que buscar la-s transición-es y etapa-s en secuencia-s predecesora-s. Pero, si la etapa no es el primer elemento de la secuencia, la transición predecesora se encuentra en la propia secuencia y la-s etapa-s predecesora-s de esta transición se pueden encntrar en la propia secuencia o en secuencia-s predecesora-s.

La parte correspondiente al Reset se realiza identificando los sucesores de cada etapa. Si la etapa es el último elemento de una secuencia, el sucesor puede ser un enlace horizontal (convergencia en Y o divergencia en O) o un salto, teniendo que buscar la-s etapa-s en secuencia-s sucesora-s. Pero, si la etapa no es el último elemento de la secuencia, la-s etapa-s sucesora-s se encuentran en la propia secuencia o en secuencia-s sucesora-s.

Con la información recopilada se programa un bloque funcional por cada GRAFCET, que consta de la declaración de variables y un conjunto de sentencias IF-THEN que evalúan la activación y desactivación de cada una de las etapas.

En cuanto a la parte combinacional, el programa consta de un conjunto de variables, las instancias a cada uno de los POUs que implementan la parte secuencial y la programación de las acciones.

Para cada acción relacionada con las salidas se programa una única sentencia de asignación, en la cual figuran todas las etapas en las que aparece dicha acción (condicionada si es el caso) enlazadas por el operador OR.

Para las acciones relacionadas con contadores se programan la carga de los contadores y su actualización (incremento, decremento) identificando las etapas que suscitan dichas acciones. De la misma forma se tratan las acciones relacionadas con la carga y activación de los temporizadores.

En cuanto a las variables del proyecto, en primer lugar se realiza su identificación en el fichero XML: las variables de etapa (booleanas) son los identificadores de las mismas; las salidas, entradas y señales de control se identifican en las condiciones de las transiciones y en las acciones; las variables asociadas a los temporizadores, contadores, etc. se identifican teniendo en cuenta los prefijos específicos que se utilizan en su denominación.

Una vez identificadas, hay que definirlas indicando el tipo de variable (entrada, salida, memoria, constante y sistema) y tipo de dato (bool, Word, doubleword, etc.). Del mismo modo es necesario indicar el tipo de contador (CTD, CTU o CTDU) y de temporizador (TON, TOF o TP). La Figura 6 muestra el interfaz para realizar la definición de variables.

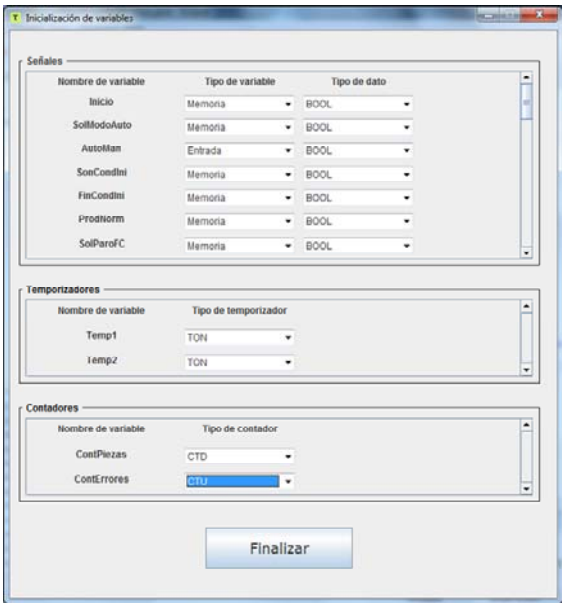

Figura 6: Interfaz de usuario para la definición de variables.

5 CASO DE ESTUDIO

El prototipo de la herramienta [15] se ha aplicado en la generación del código de todos los ejemplos que están disponibles en el curso abierto "MeiA. Metodología para ingeniería de Automatización. Nivel de Diseño" [3]. A modo de ejemplo se presenta la generación de código correspondiente al proceso "Transferencia de Piezas".

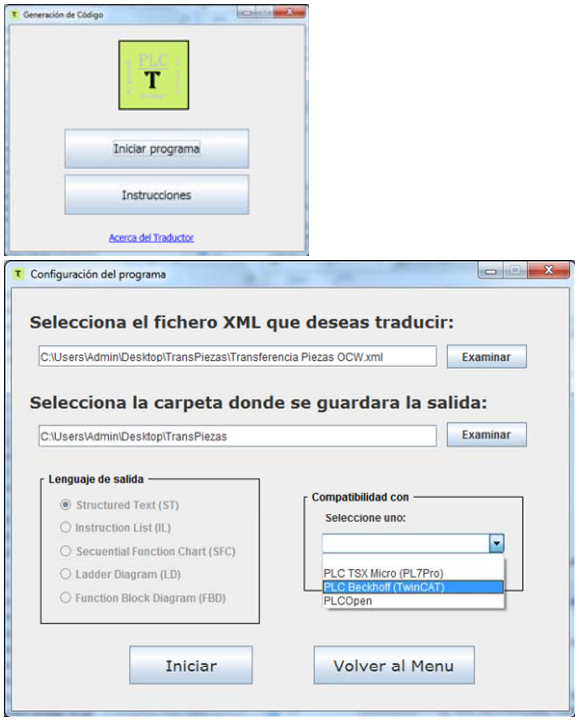

Figura 7: Interfaz inicial del prototipo.

Tal y como se muestra en la Figura 7, la generación de código comienza con la selección del fichero XML, del directorio destino de los ficheros resultantes y el PLC destinatario. A continuación se realiza la definición de variables tal y como se muestra en la Figura 6.

A modo resumen, la Figura 8 presenta el DOU correspondiente a la Secuencia Principal, el fichero XML generado por SFCEdit y la generación de código correspondiente que incluye el bloque funcional, el programa y la definición de variables para dicho DOU.

6 CONCLUSIONES

En este trabajo se presenta un método de generación del proyecto de automatización a partir de los GRAFCET de diseños desarrollados a nivel tecnológico, que se han obtenido con la metodología MeiA y generado con la herramienta que le da soporte. Para validar el método se ha desarrollado un prototipo que genera un conjunto de POUs (bloques funcionales) correspondientes a la parte secuencial, un POU principal (programa) de la parte combinacional y la definición del conjunto de variables del proyecto. El código de control se ha generado en texto estructurado de acuerdo con la norma IEC 61131-3.

La generación del código de control permite reducir de forma considerable el tiempo de implementación de los programas de PLCs y proporciona una significativa reducción de los errores de implementación. Además, los POUs generados presentan una estructura sencilla de entender por los programadores de PLCs, lo cual facilita las futuras tareas de mantenimiento.

El trabajo futuro contempla la ampliación del prototipo para poder generar código de control en los restantes lenguajes de programación de la norma IEC 61131-3. También se pretende utilizar el enfoque de la ingeniería basada en modelos para realizar un prototipo que pueda ser integrado en el entorno MeiA.

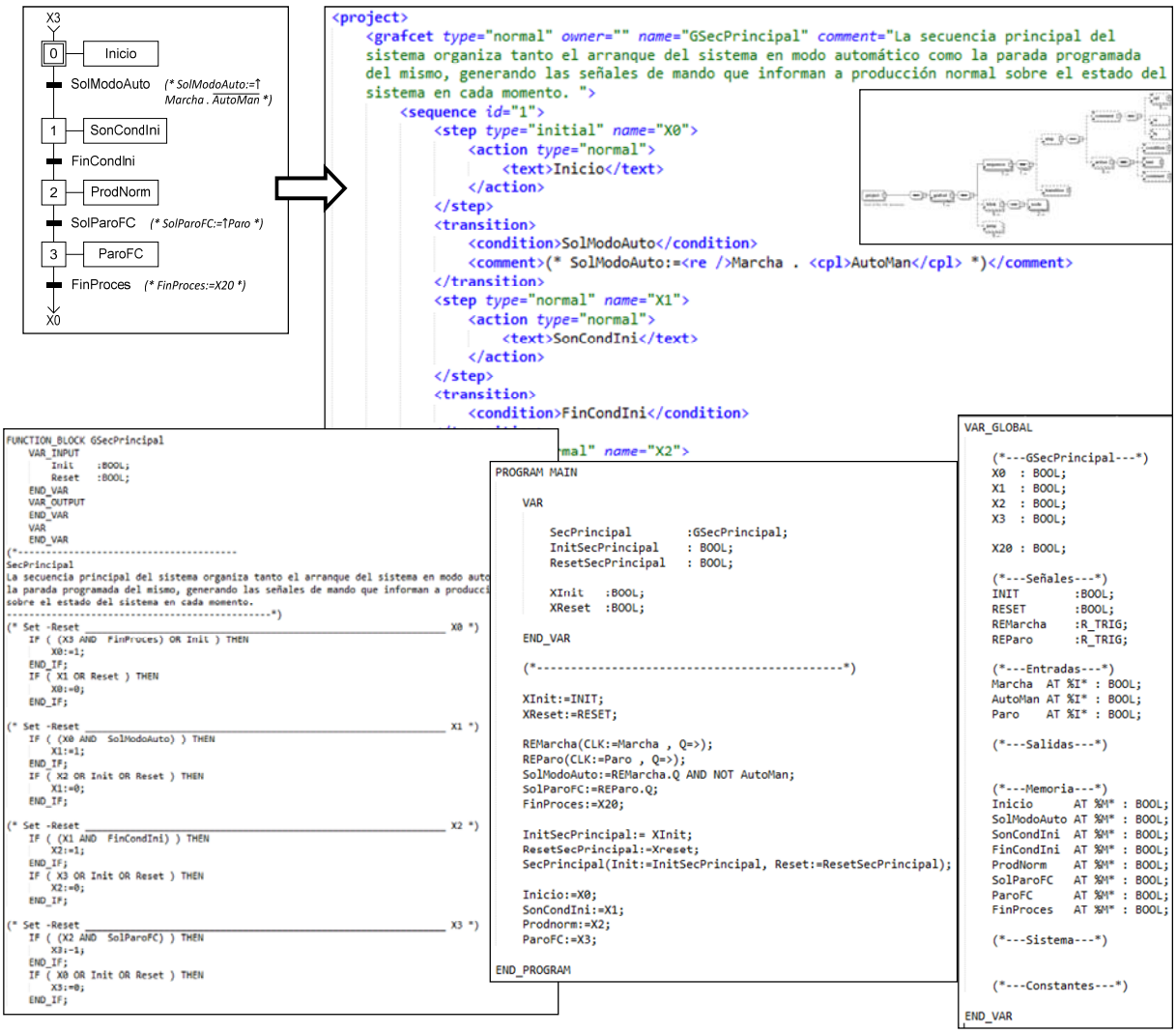

Figura 8: Generación de código del DOU Secuencia Principal del ejemplo Transferencia de Piezas

Agradecimientos

Este trabajo ha sido financiado por el proyecto DPI2015-68602-R (MINECO/FEDER, UE), por la UPV/EHU en el marco del proyecto PPG17/56 y GV/EJ en el marco de grupos de investigación reconocidos IT914-16.

Referencias

[1] Álvarez, M. L., Estévez, E., Sarachaga, M. I., Burgos, A., Marcos, M., (2013) A novel approach for supporting the development cycle of automation systems. International Journal of Advanced Manufacturing Technology, 68, pp 711-725.

[2] Álvarez, M. L., Sarachaga, I., Burgos, A., Estévez, E., Marcos, M., (2016) A Methodological approach to model-driven design and development of Automation Systems. IEEE Transactions on Automation Science and Engineering, pp. 1-13.

[3] Álvarez, M.L., Burgos, A., Sainz de Murieta, J.A., Sarachaga, M.I,. (2017) Curso OCW: MeiA. Metodología para ingeniería de Automatización. Nivel de Diseño. https://ocw.ehu.eus/course/view.php?id=415

[4] David, R., (1995) Grafcet: A powerful tool for specification of logic controllers. IEEE Transaction on Control Systems Technology, 3 (3), pp. 253-2.

[5] Eclipse. (2003) Eclipse IDE. Retrieved from http://www.eclipse.org/

[6] Feio, R., (2017) Translating IOPT Petri net models into PLC ladder diagrams.. Industrial Technology (ICIT), IEEE International Conference

[7] González, V. M., Mateos, F., & Ng, A. (2004). MLAV: the Object-oriented Methodology of the Virtual Automation Lab. in Proc. of the IEEE International Conference on Robotics and Automation, pp. 5153-5158.

[8] IEC, (2003) IEC 61131-3 Programmable Controllers — Part 3: Programming languages, *International Standard Edition 3.0*,.

[9] IEC 60848, (2013) International Electrotechnical Commission - GRAFCET specification language for sequential function charts.

[10] Julius, R., Schürenberg, M., Schumacher, F., Fay, A., (2017) Transformation of GRAFCET to PLC code including hierarchical structures. Control Engineering Practice.vol 64, pp. 173-194.

[11] Lukman, T., Godena, G., Gray, J., Hericko, M., Strmcnik, S., (2013) Model-driven engineering of process control software – beyond device-centric abstractions Control Engineering Practice, 21 (8), pp. 1078–1096.

[12] Markiewicz, M., Surdej, L., Gniewek, L., (2016) Transformation of a fuzzy interpreted Petri net diagram into structured text code. 21st International Conference on Methods and Models in Automation and Robotics (MMAR),

[13] Murillo, L., (2008) Redes de Petri: Modelado e implementación de algoritmos para autómatas programables. Tecnología en Marcha, 21 (4), pp. 102-125.

[14] Panjaitan, S., (2010) A Formal Design of Automation Systems based on Operation Modes using High-level Petri Net. Second International Conference on Advances in Computing, Control, and Telecommunication Technologies , pp. 129-131.

[15] Paz, H., (2016) Prototipo de la Herramienta de generación de código. https://github.com/HPaz001/translator

[16] SAX (Simple API for XML). http://www.saxproject.org/.

[17] Schumacher, F., Fay, A., (2014) Formal representation of GRAFCET to automatically generate control code Control Engineering Practice, 33, pp.84–93.

[19] SFCEdit, (2010) Ingénierie en automatisme et informatique industrielle

[20] Thramboulidis, K., Frey, G., (2011) Towards a model-driven IEC 61131-based development process in industrial automation Journal of Software Engineering and Applications, 4 (4), pp. 217–226.