PLC-based implementation of supervisory control for flexible manufacturing systems using theory of regions

Sadok Rezig, Chekib Ghorbel, Zied Achour, Nidhal Rezg

PLC-based implementation of supervisory control for flexible manufacturing systems using theory of regions

Sadok Rezig*

Laboratoire de Génie Informatique de Production
et de Maintenance (LGIPM),
Université de Lorraine,
Metz, France
Email: sadok.rezig@univ-lorraine.fr
*Corresponding author

Chekib Ghorbel

Advanced System Laboratory,
Polytechnic School of Tunisia,
BP 743, 2078, La Marsa, Tunisia
Email: chekib.ghorbel@enit.rnu.tn

Zied Achour and Nidhal Rezg

Laboratoire de Génie Informatique de Production
et de Maintenance (LGIPM),
Université de Lorraine,
Metz, France
Email: zied.achour@univ-lorraine.fr
Email: nidhal.rezg@univ-lorraine.fr

Abstract: The supervisory control is a common theory for the synthesis of Petri net (PN) supervisors for discrete event systems given a PN model and a control specification for the maximum permissive behaviour. The theory of regions as one of control synthesis method generates a PN controller to satisfy the control specification. Though the theory of regions has for over a decade received a considerable attention in academy, still very few applications exist. The real cause of this seems to be a contradiction between the abstract controller and its physical implementation. This is evident in particular when the implementation is supposed to be based on a programmable logic controller (PLC), as is the case for flexible manufacturing systems. Indeed, since the synchronous PLC is based on signals, the PN supervisor remains asynchronous; this explains its implementation difficulty. In this work, a control synthesis method using the theory of regions is implemented with a Java application on PLCs of a flexible manufacturing system (FMS) installed in our research laboratory in the University of Lorraine in Metz, France.

Keywords: Petri nets; signal interpreted Petri net; theory of regions; program logic controller; supervisory control; Petri net controller; reachability graph; algebraic methods; flexible manufacturing systems; FMSs; deadlock prevention.

Reference to this paper should be made as follows: Rezig, S., Ghorbel, C., Achour, Z. and Rezg, N. (2019) 'PLC-based implementation of supervisory control for flexible manufacturing systems using theory of regions', *Int. J. Automation and Control*, Vol. 13, No. 5, pp.619–640.

Biographical notes: Sadok Rezig graduated from the National Engineering School in Tunis in Electrical Engineering (September 2013). He is an Associate Professor at the University of Lorraine and author of many scientific journals and conferences.

Chekib Ghorbel received his Engineer Diploma degree in Electrical Engineering in 2004, Master's degree in 2005 in Automatic Control from the National School of Engineers of Sfax (ENIS), Tunisia, and Doctorate degree in 2010 from the National Engineers of Tunis (ENIT), Tunisia. He is currently an Associate Professor at the National Engineers of Carthage (ENICarthage), Tunisia. His areas of interest include nonlinear identification, stability and stabilization of the complex systems. Currently, he is a researcher in Advanced System Laboratory (LSA) in Polytechnic School of Tunisia (EPT), La Marsa.

Zied Achour is an Associate Professor at the University of Lorraine. After obtaining his Master's degree in 2002 and PhD in October 2005 in Automatic, he started his current position at the university and conducted his research in discrete event systems.

Nidhal Rezg is a Doctor of Industrial Automatic from the National Institute of Applied Sciences (INSA) in Lyon in 1996. He obtained his Accreditation to Supervise Research (HDR) at the University of Metz in 2003. He currently holds the position of Professor of universities. He has been the director of LGIPM Laboratory since October 2006.

1 Introduction

Discrete event systems (DES) are event systems with discrete states. Their evolution is made in accordance with the occurrence of the asynchronous events characterising the change of the system state. DES does not care about the continuous process of the phenomena, of their logical dynamic or temporal sequence. Examples of DES are computer systems, queuing networks, transport systems and manufacturing systems. The first investigations for DES led Ramadge and Wonham (1987, 1989) to the important results on the existence and synthesis of supervisors for DES (Azar and Zhu, 2015; Azar and Vaidynathan, 2014, 2015; Azar et al., 2015).

The proposed approach of Ghaffari et al. (2002a, 2002a) in supervisory control problem is based on the theory of regions (Badouel and Darondeau, 1998). This theory computes a PN supervisor by solving a linear system composed of three types of conditions; the reachability conditions, the marking/transition separation instance (MTSI) conditions and the basic cycle equations. However, there are very few guidelines for how to implement the synthesised PN supervisor, once the abstract controller model has been

calculated by the theory of regions. Therefore, the step to a physical implementation is not necessarily simple. In the special example of FMS, where the PLC-control is of capital importance, the gap between the event-based asynchronous automata world and the synchronous signal-based PLC world has to be bridged (Vieira et al., 2006; Bolton, 2015).

PLCs remain the main automatic equipment in all industrial sectors. The ever-increasing complexity of PLC programming software as well as user imposed security and operational requirements claim new methods to satisfy control specifications. Petri nets (PNs) are a very interesting and useful formalism for complex system specifications. Indeed, its description power allows us to implement causality, parallelism and synchronisation. However, PNs remain up to now non-autonomous and does not interact with the external environment. Consequently, one cannot manipulate the internal signals of the effectors and sensors of the PLC (Fabian and Hellgren, 1998; Peng and Zhou, 2004; Grobelna et al., 2016).

It is necessary then to find an effective way beyond how to use PNs to communicate with the external environment. Thus, the signal interpreted Petri net (SIPN) guarantees this communication. The SIPNs are an extension of PNs and allow the explicit processing of the internal signals of the PLC. Thereby, the PN supervisor can be synthesised using the theory of regions and its implementation on the PLC is ensured through a PN-SIPN translation of the calculated supervisor.

According to the IEC 61131-3 industry standard defined by the international electronic commission, there are five PLC programming languages; The ladder diagram (LD) (Fen and Ning, 2006) the Grafcet or sequential function chart (SFC) (Vyatkin and Instrument Society of America, 2007), the functional block diagram (FBD) (Blocks, 1998) and the instruction list (IL) (Asensio et al., 2013; Zhu and Azar, 2015; Azar and Vaidyanathan, 2016). These five PLC programming languages are widely used in the industrial world. Furthermore, one can wonder about the performance of these languages for describing the controlled system. Indeed, the LD and FBD languages do not perfectly correspond to the sequential or concurrent description of the controller. As a result, there is no way to make a visual description of the control algorithm (Ioannides, 2004). Further, the SFC-based Grafcet language provides sequential and concurrent description of the control algorithm but it is used to design powerful controllers where the execution time and program size is important (Frey, 2003). This hampers the use of the SFC language.

The main contribution of this work is to implement a PN controller on a PLC of a FMS using the theory of regions. Consequently, we introduce the SIPN as another programming language which compensates the programming languages limits of the industrial standard IEC 61131-3.

The rest of the paper is organised as follows. Section 2 defines the basic notions of PN, SIPN and the theory of regions for control synthesis. Next, in Section 3 we outline our new policy of monitoring and implementing PN supervisor on PLCs via a Java application. Next, a FMS example will be given in Section 4 in order to illustrate the contribution of the proposed approach. Finally, conclusions and future work are provided in Section 5.

2 Background

2.1 Petri nets

A Petri net is a graphical and mathematical modelling tool for many systems, especially for DES. From an informal point of view, a PN is a directed graph with two types of nodes (place and transition) and has a dynamic behaviour. The places and transitions are connected by oriented arcs. An arc connects either a place to a transition or a transition to a place but never connects a place to a place or a transition to a transition. Moreover, from a formal point of view, a PN is a bipartite graph: $PN = <P, T, Pre, Post>$; P is a set of places. T is a set of controllable and uncontrollable transitions $T = T_c \cup T_u$ (Murata, 1989). $Pre: P * T \to \mathbb{N}$ represent a pre-incidence function that specify weighted arcs from P to T. $Post: P * T \to \mathbb{N}$ is the post-incidence function that specially weighted arcs from T to P ($\mathbb{N}$ is a set of nonnegative integers). Let the set $p^{(t)}$ (respectively ${}^{(t)}p$) be the output transitions (respectively input transitions) of a place p. Indeed, a PN can be expressed by an indexed matrix C such that $C(p_i, t_j) = w(t_j, p_i)$ if $t_j \in {}^{(t)}p_i$ and $C(p_i, t_j) = -w(t_i, p_j)$ if $t_j \in p_i^{(t)}$, else 0; where $\omega: F \to \mathbb{N}$ is a valuation function of arcs (the finite set of arcs $F \subseteq (P * T) \cup (T * P)$. Moreover, let $t^{(p)}$ (respectively ${}^{(p)}t$) be the set of output places (respectively input places) of a transition t. The reachability graph constructed from the initial marking M_0 is denoted by $G(N, M_0)$. The set of generated markings in $G(N, M_0)$ is denoted by M. A transition is enabled from a marking $M \in$ M (denoted by $M[> t$) if and only if $M \geq Pre(., t)$. An enabled transition may fire yield a new marking M' such that $M' = M + C(., t)$ this expression can be defined by $M[t > M'$. A new marking M' is reachable from a marking M if a firing sequence $\sigma = t_1, t_2, \ldots t_n$ exists by firing σ, any marking M' reachable from the initial marking M_0 satisfies the following PN state equation: $M' = M_0 + C.\vec{\sigma};$; where $\vec{\sigma}: T \to \mathbb{N}$ is a vector of nonnegative integers called the occurrence of t_i in σ. A PN is said k-bounded if the number of tokens in each place $p_i \in P$ does not exceed k. A PN is said live if every transition is live. A PN is reversible, if from any reachable state M', an enabled sequence σ exists transforming M' into M_0.

2.2 Signal interpreted PNs

A SIPN is a type of PNs with binary markings. The exact definition of a SIPN is a direct extension of the binary PNs. Formally, a SIPN is a 9-tuple:

$$SIPN = (P, T, F, m_0, I, O, \varphi, \omega, \Omega)$$

(P, T, F, m_0) is an ordinary PN with binary markings, P is a set of places, T is a set of transitions, F is a set of arcs and m_0 is the initial binary marking. $I = (i_1, i_2, \ldots, i_I)$ represent the logical input signals, $O = (o_1, o_2, \ldots, o_O)$ represent the logical output signals. $I \cap O = \phi$. φ associate each transition $t_i \in T$ with a firing condition $\varphi(t_i) = f(i_1, i_2, \ldots, i_I)$ which represents a Boolean function. ω associates each place $p_i \in P$; $\omega(p_i) \in \{0, 1 -\}$, – means an undefined output. Ω is the output function combining all the outputs ω of marked places; Ω: $\{0, 1\}^p \to \{0, 1, -, c\}$ c means the contradiction in the case where the same signal is set to one in a given place and 0 in another place. To calculate $\Omega(m_a)$, one can use $m_a : \Omega(m_a) \prod_i w(p_i)$.

Similar to a PN, the places, the transitions and the tokens of a SIPN are graphically represented by circles, bars and dots. In addition, the connections between circles and bars are performed by arcs. Moreover, the transitions of a SIPN are labelled with their firing conditions and the places are labelled with their output functions. For more convenience, given a large number output signals, one can represent the output not by the vector form $\omega(p_i) = (o_1 = 1, o_2 = 0 \ldots o_O = 1)$ but by specifying directly the influence of the variables i.e., $\omega(p_i) = (1, 0 \ldots 1)$.

It is well known that, one of the disadvantages of PN is their growing complexity when the system is large. However, one of the approaches to managing complexity is to decompose the system into hierarchical subsystems.

As shown in Figure 1, PN can be hierarchical by inserting a sub-network into the places or transitions. Figure 2 illustrates a hierarchical refinement of the PNs. A SIPN can be hierarchical by replacing the places by the sub-networks (SIEMENS, 2012a).

Figure 1 Graphic representation of a SIPN

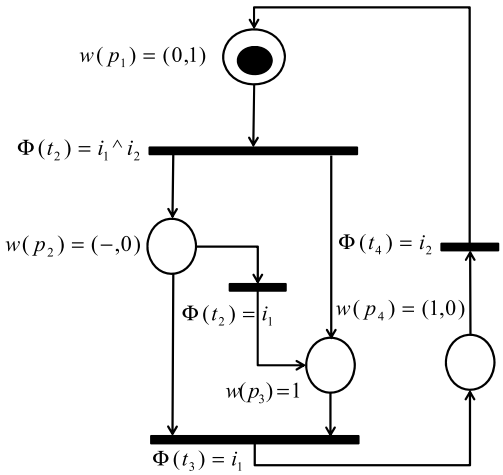
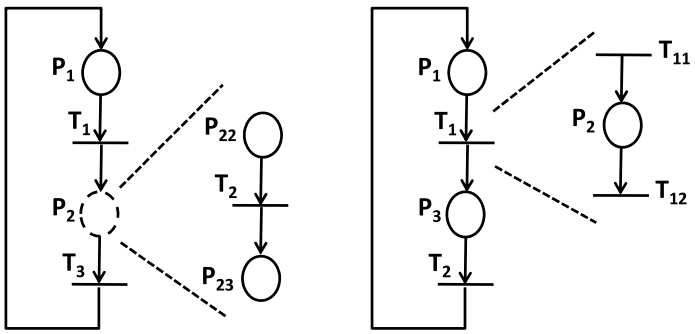


Figure 2 Correlation between PN and SIPN



The sub-networks in a hierarchical SIPN are subordinated to hierarchical places. The sub-networks can also contain hierarchical places and so on.

To take advantage of the theory of regions effectiveness, we will proceed by transforming the PN controller into a SIPN supervisor. Then, the controlled SIPN model is implemented using a Java programming language by developing a graphical interface with the help of the Eclipse environment.

2.3 Theory of regions

The theory of regions in control synthesis was proposed by Ghaffari et al. (2002a, 2002b, 2003a, 2003b) using the properties of PN tools for adding a PN controller to the initial PN model with initial marking $M_0(P_c)$ and its incidence vectors $C(P_c,.)$. This theory is represented by a linear system composed of three types of conditions; the reachability conditions, the MTS) conditions and the cycle equations. Its resolution leads to the design of PN supervisor. This control place P_c has to fulfil the reachability conditions, i.e.:

$$M_0(P_c) + C(P_c,.)\vec{\Gamma}_M \geq 0 \tag{1}$$

M_0 represent the initial state of the generated graph/state space and M is a given reachable marking of the graph.

$\vec{\Gamma}_M$ is the path between M_0 and M.

In addition, each PN controller to add should solve at least one prohibited event (M, t) in the set Ω of prohibited state transitions. The MTSI condition relative to the couple (M, t) is:

$$M_0(P_c) + C(P_c,.)\vec{\Gamma}_M + C(P_c,t) < 0 \tag{2}$$

$C(P_c, t)$ represent an incidence vector leading to the corresponding forbidden marking.

Finally, P_c has to satisfy cycle equations:

$$\sum_{t\in T} C(P_c,t)\cdot\vec{\sigma}[t] = 0 \tag{3}$$

$\vec{\sigma}[t]$ is the algebraic sum of occurrences of t in σ. $\sum_{t\in T} C(P_{ci},t_i)$ The sum of the incidence vectors form the basic cycle equations of the theory of regions.

Notably, many MTSI conditions may obtain the same solutions by resolving the linear system composed of equations (1), (2) and (3). As a result the number of PN controllers will be much smaller than the set of MTSI conditions.

The following example of Figure 3 is an application of the theory of regions for control synthesis (Zhou and Venkatech, 1999; Yu et al., 2016).

For this forbidden state problem, we want to prohibit the following event separation instance $\left(M_5 \xrightarrow{T_1} M_7\right)$ [i.e., see Figure 3(b)]. Based on the theory of regions, there is:

- Seven reachability conditions:

$$M_0(P_c) \geq 0$$

$$M_1(P_c) = M_0(P_c) + C(P_c,T_1) \geq 0$$

$$M_2(P_c) = M_0(P_c) + C(P_c, T_1) + C(P_c, T_2) \geq 0$$

$$M_3(P_c) = M_0(P_c) + C(P_c, T_1) + C(P_c, T_2) + C(P_c, T_3) \geq 0$$

$$M_4(P_c) = M_0(P_c) + 2C(P_c, T_1) + 2C(P_c, T_2) + C(P_c, T_3) \geq 0$$

$$M_5(P_c) = M_0(P_c) + 2C(P_c, T_1) + 2C(P_c, T_2) \geq 0$$

$$M_6(P_c) = M_0(P_c) + 2C(P_c, T_1) + C(P_c, T_2) \geq 0$$

- One cycle equation:

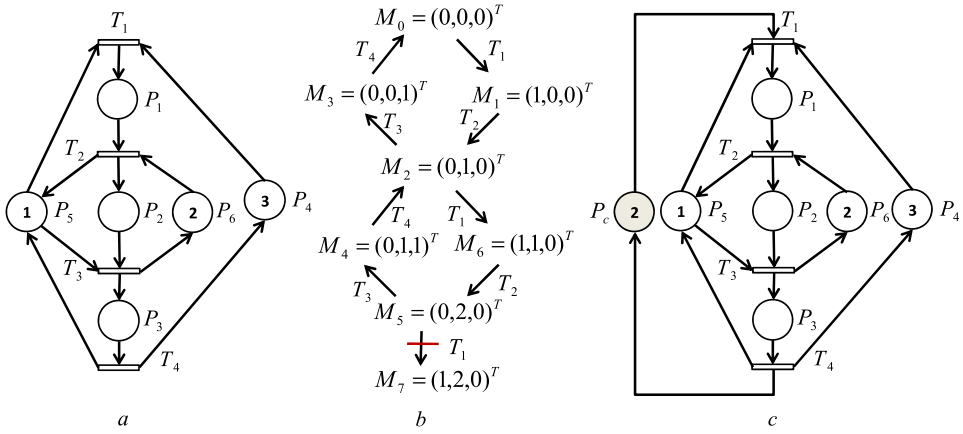
$$C(P_c, T_1) + C(P_c, T_2) + C(P_c, T_3) + C(P_c, T_4) = 0$$

- One MTSI condition:

$$M_7(P_c) = M_0(P_c) + C(P_c, T_1) + 2C(P_c, T_2) + C(P_c, T_1) < 0$$

The linear system of the theory of regions contains nine equations. Its resolution leads to the controlled PN [see Figure 3(c)]: $M_0(P_c) = 2$; $C(P_c,.) = (-1, 0, 0, 1)$.

Figure 3 Application of the theory of regions, (a) PN model (b) reachability graph (c) controlled PN (see online version for colours)



3 Implementing supervisors

A controller is typically described by a finite state machine, so that controller implementation is basically a matter of making the PLC behave as a state machine (Rezig et al., 2016). Nevertheless, there are a number of problems associated with this, both with regard to the underlying assumptions on the supervisor and with regard to the asynchronous nature of an event-driven state machine. Then, to communicate with the external environment one can use the SIPN which guarantees this communication (Frey, 2003; Shieh, 2014).

Thanks to our new control policy, one can synthesise the PN supervisor using the advantages of PN on the one hand and the effectiveness of the theory of regions for supervisory control on the other hand (Rezig et al., 2017). Therefore, we firstly model the FMS using PNs, and then one can apply the theory of regions to easily synthesise the PN controllers. Finally the controlled PN model can be transformed into a controlled SIPN model to implement the supervisor in the PLC.

Places in a SIPN are interpreted as situations. A situation is a local state of the PLC. It is divided into two categories: active or inactive. While it is active, it can affect the environment of the control, namely, the tuning actuation signals in the process or the input signal in another algorithm via a corresponding output. The state of a SIPN is given by the marking, namely the combination of active situations.

The transitions in a SIPN specify under what circumstances a situation ends and a new situation becomes active. The condition for triggering the transition is specified when the change of situation is allowed to occur. With these semantic interpretations, it is clear that crossing a transition can take a very short time. More generally, in the SIPNs, each place (respectively transition) is associated with a Boolean function which involves the output (respectively input) signals (Frey and Minas, 2000; Minas and Frey, 2002).

In the following example (i.e., see Figure 5), we present an application of the theory of regions with a transformation of the conceived controller into a SIPN supervisor. The given PN models two machines/two processes sharing one resource.

The following GMEC must be fulfilled:

$$M(M_1)+M(P_3)+M(M_2)+M(P_4)\le 1$$

The application of the theory of regions gives the linear system below:

$$M_0(P_{ci})\ge 0$$

$$M_0(P_{ci})+C(P_{ci},t_1)\ge 0$$

$$M_0(P_{ci})+C(P_{ci},t_1)\ge 0$$

$$M_0(P_{ci})+C(P_{ci},t_4)+C(P_{ci},t_5)<0$$

$$M_0(P_{ci})+C(P_{ci},t_1)+C(P_{ci},t_2)<0$$

$$C(P_{ci},t_3)+C(P_{ci},t_4)+C(P_{ci},t_5)=0$$

$$C(P_{ci},t_1)+C(P_{ci},t_2)+C(P_{ci},t_6)=0$$

Solving this set of equations gives us the designed SIPN controllers to add to the initial PN of the Figure 5:

$$P_{c1}: M_0(P_{c1})=1;\ C(P_{ci},t_i)=(-1,-2,0,1,2,0)$$

$$P_{c2}: M_0(P_{c2})=1;\ C(P_{ci},t_i)=(-1,0,0,1,0,2)$$

- After having designed the controllers, a Java program will be able to generate a LIST code capable of being executed by the PLC in order to satisfy the control specification.

Figure 4 Execution process of a SIPN

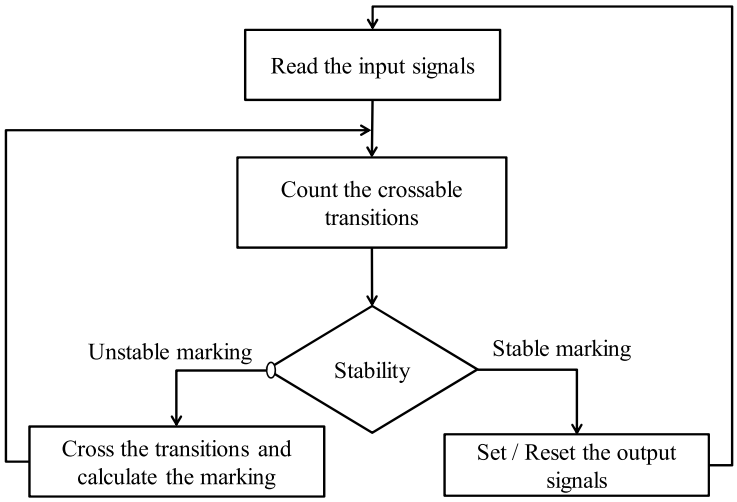
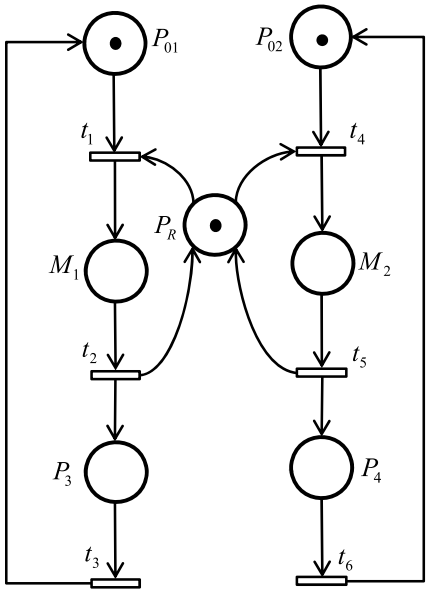


Figure 5 A simple PN example

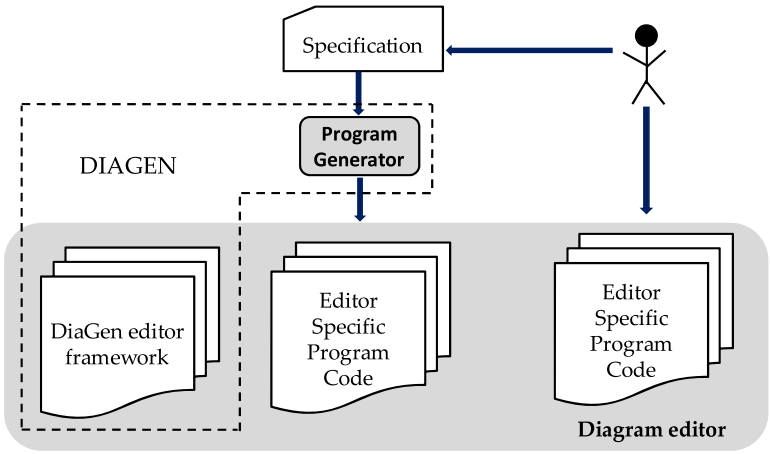


3.1 Methodology

In our control policy, the controlled SIPN model is implemented using the Java programming language by developing a graphical interface with the help of the Eclipse environment. This computer oriented-programming language is chosen in this work for a main reason; is that the applications developed with Java are easily brought on several operating systems such as UNIX, Windows, MacOS or GNU Linux. Therefore, our new application is usable by any computer.

In our application, we used Diagen which offers an environment to develop the diagram editors. It is completely implemented in Java and consists of a Framework and a Programmer. Diagen is an innovative workbench for generating graphics editors. Figure 6 shows the Diagen structure and the process of using this software to quickly develop an editor.

Figure 6 Diagram editor generation with Diagen (see online version for colours)



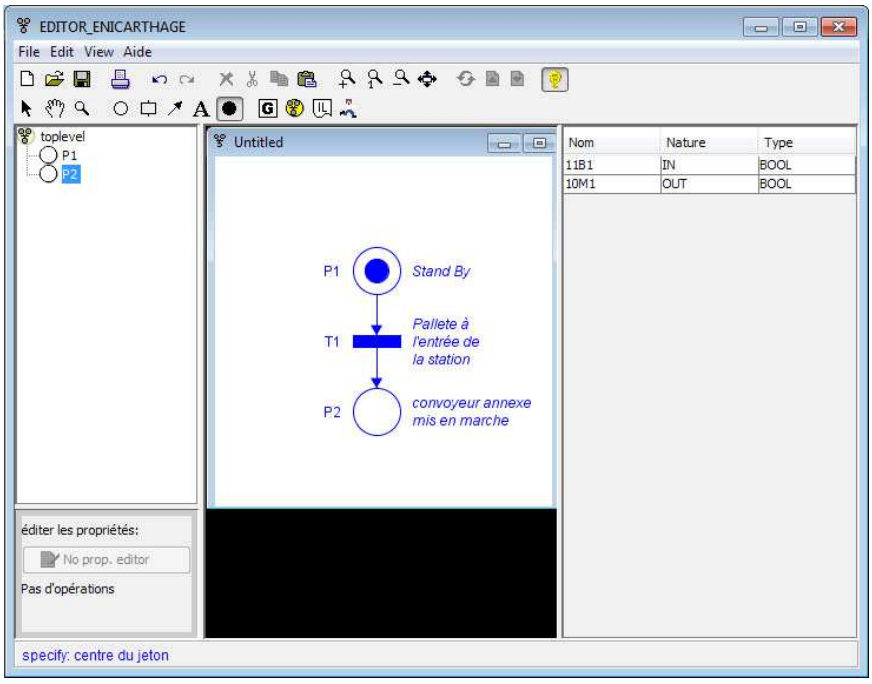
The Diagen framework is a set of Java classes that provide the general functionality needed to analyse and edit a diagram (Verma and Pandey, 2016). Indeed, to create our own editor for a programming language in a diagram, we must firstly give some specifications i.e., (circles, arcs, zooming, centring, deleting, copying, pasting, etc.). Once these specifications are given, the generator program intervenes to transform its specificities into Java classes. The editors developed by Diagen can always be modified, so the editor user can arbitrarily create/delete and modify chart components (places, transitions, arcs, and tokens) as with any drawing tool.

3.1.1 The graphical interface

As shown in Figure 7, the interface is split into four parts: the explorer on the left, the list of sensors and actuators on the right, a comment box at the bottom and the editor in the interface centre.

Indeed, the explorer find out the various open projects as well as the project elements i.e., the places, the transitions, etc. Therefore, a tree structure is chosen as in most file explorers. Furthermore, on the right side of the interface one can list the different sensors and actuators, specify their natures and add their corresponding addresses in the PLC. The design of the SIPN is performed thanks to our editor in the interface centre. Obviously, the user can add elements anywhere; we just have to select the element and then draw it on the editor.

Figure 7 Snapshot of the developed editor (see online version for colours)



3.1.2 PLC program generation

The designed diagram is simple and has only one actor. Thus, the user can enter the PN model, add a code using LIST language for the places/transitions; for a place, the code is executed when the latter is activated, whereas for a transition the execution depends on the specified firing condition (Nagata and Hirose, 2016; Capito et al., 2016). The user can also manipulate the internal signals of the PLC and define their names/addresses. Finally, once the SIPN controller model is entered, one can generate a LIST code supported by the PLC. The different steps for generating the LIST code imported into the PLC are shown in Figure 8.

The application strength is the tokens translation of the SIPN controller from the designed PN monitor because it plays an important role in the code generation imported into PLC. The main idea is to represent each place P_i of the PN with a Boolean variable. If the place P_i is marked, then $P_i = True$ else if P_i is unmarked then $P_i = False$.

Regarding to transitions, two tests are performed for each transition t_i; firstly, we check if the transition is activated and if the firing condition is verified. To optimise the code, the firing condition is only checked for activated transitions. To implement this optimisation, a conditioned jump to the next transition is programmed just after the verification of the transition activation. After having carried out these calculations, the accumulator is tested. If it is equal to 1, then all the conditions are satisfied and then the transition is fired. The crossing of the transition removes tokens from the places ${}^{(p)}t_i$ and adds tokens to the set $t_i^{(p)}$. The set/reset commands are only executed when the accumulator is equal to 1.

Figure 8 Activity diagram of the application

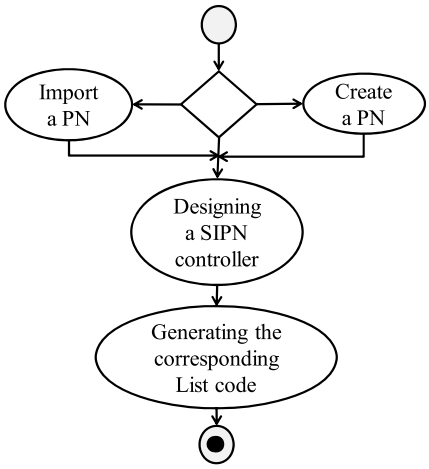
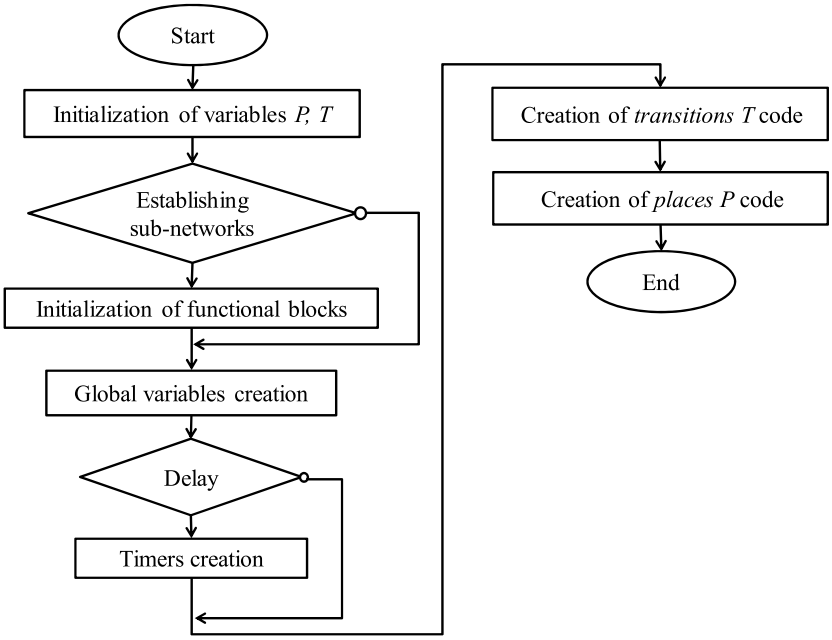


Figure 9 Main class chart



Eventually, the places model the supervisor state. If a place is marked then there is automatically an influence on the external environment, and the corresponding code is executed immediately (SET/RESET of one of the outputs). If a place is unmarked then a conditional jump is programmed to the next place label and the code is not executed. The Jump can be omitted since the SET/RESET statements are executed only if the accumulator is set to one. For an undefined output value, the PLC code keeps the last value given. The order in the PLC code is taken into account; if an output is set to 1 and 0

simultaneously, the action taken first will be taken into consideration. The latter two cases must be avoided.

Figures 8–11 present the flowcharts of the main class of the program as well as the transformations of the SIPN model into variables and the code generation for places and transitions.

Figure 10 Flowchart transforming a SIPN into variables

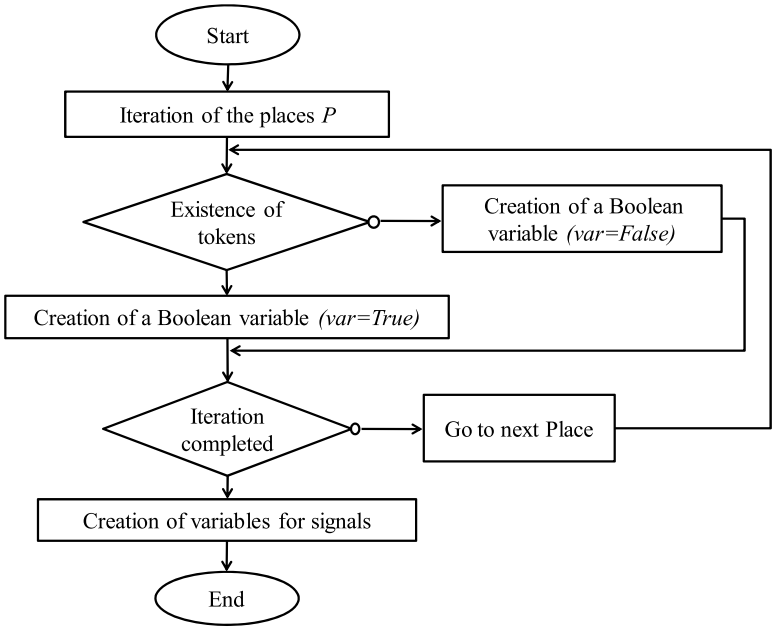
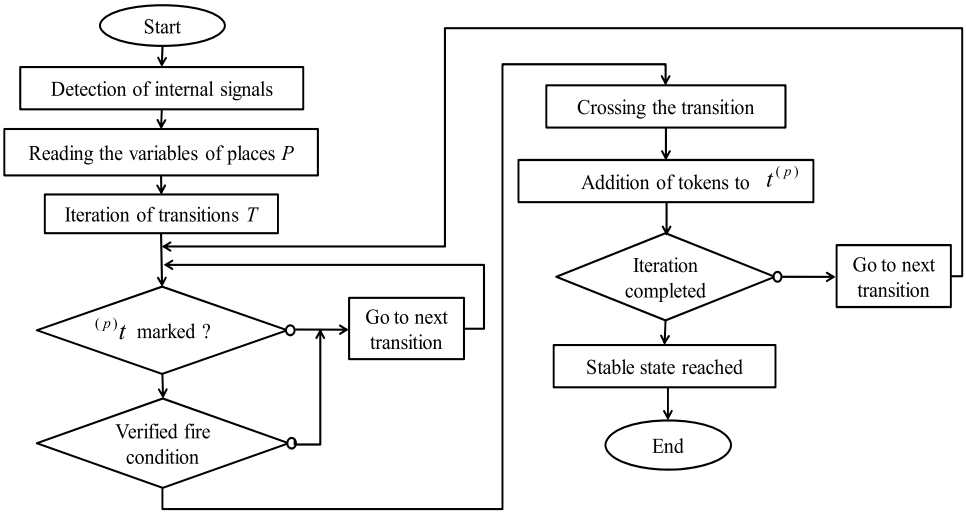


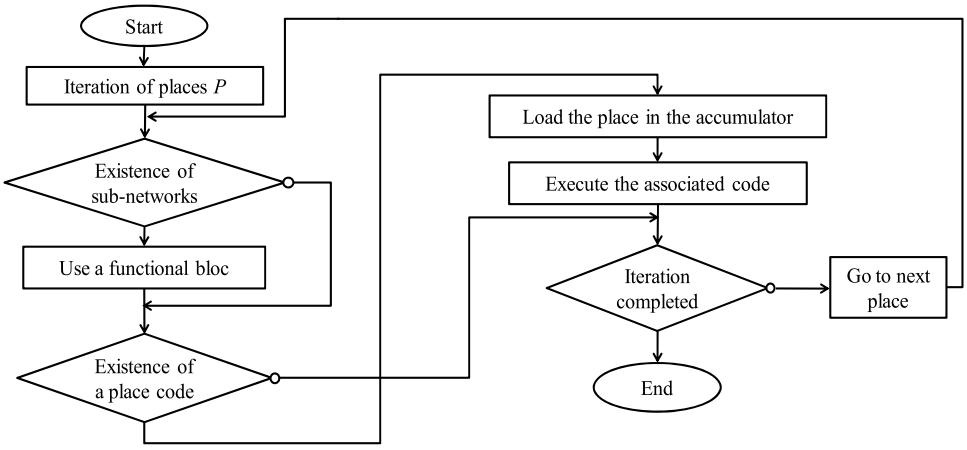
Figure 11 Flowchart generating transitions code



4 FMS example

In this section, we will apply the theory of regions on a FMS installed in the University of Lorraine in Metz, FRANCE. The PN controller to be synthesised must satisfy a control specification applied to two assembly stations shown in the Figure 12 (Rakhshan et al., 2016).

Figure 12 Flowchart generating places code



The first assembly station is modelled by the place P_2 while the second station is represented by P_3. The place P_1 models our initial production stock. The transitions (t_2, t_3) and (t_4, t_5) represent respectively the events of entry and exit of the product in the stations through their annexes conveyors; otherwise the pallet can continue its way into the main conveyor by t_1.

In an industrial context, we have imagined the following scenario; if the first station models our self-production whilst the second assembly station represents a subcontractor, we aim to maximise our production at the expense of the subcontractor (Salloum et al., 2016). This control specification can be expressed by the following general mutual exclusion constraints:

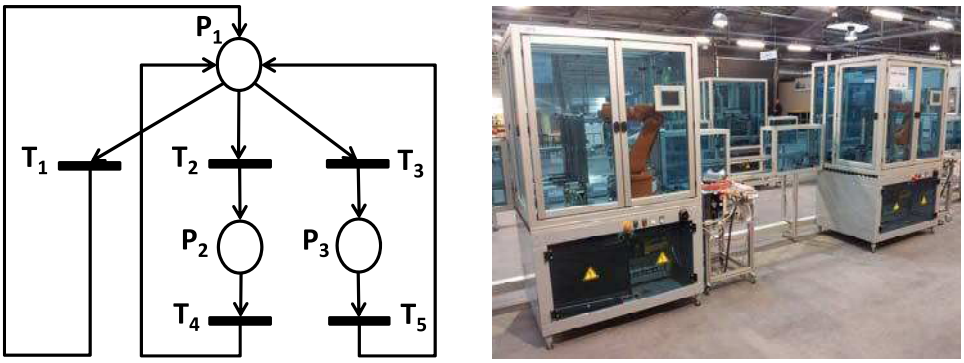
$$GMMEC : M(P_2) + 2M(P_3) \leq 3$$

4.1 Application of the theory of regions

In the reachability graph shown in the Figure 13, one can notice the existence of five MTSI:

$$\Omega = \left\{ M_2 \left[t_2 > M_4 ; M_5 \left[t_2 > M_8 ; M_5 \left[t_1 > M_9 ; M_{10} \left[t_2 > M_{13} ; M_6 \left[t_2 > M_p \right\} \right. \right. \right. \right. \right.$$

Figure 13 Experimental environment and PN model (see online version for colours)



4.1.1 The reachability conditions

$$M_1(P_{ci}) \geq 0$$

$$M_2(P_{ci}) = M_1(P_{ci}) + C(P_{ci}, t_2) \geq 0$$

$$M_3(P_{ci}) = M_1(P_{ci}) + C(P_{ci}, t_3) \geq 0$$

$$M_5(P_{ci}) = M_1(P_{ci}) + C(P_{ci}, t_2) + C(P_{ci}, t_3) \geq 0$$

$$M_6(P_{ci}) = M_1(P_{ci}) + 2C(P_{ci}, t_3) \geq 0$$

$$M_{10}(P_{ci}) = M_1(P_{ci}) + 3C(P_{ci}, t_3) \geq 0$$

4.1.2 The MTSI conditions

$$M_4(P_{ci}) = M_1(P_{ci}) + 2C(P_{ci}, t_2) < 0$$

$$M_9(P_{ci}) = M_1(P_{ci}) + 2C(P_{ci}, t_3) + C(P_{ci}, t_2) < 0$$

$$M_8(P_{ci}) = M_1(P_{ci}) + 2C(P_{ci}, t_2) + C(P_{ci}, t_3) < 0$$

$$M_{13}(P_{ci}) = M_1(P_{ci}) + 3C(P_{ci}, t_3) + C(P_{ci}, t_2) < 0$$

4.1.3 The basic cycle equations

$$C(P_{ci}, t_3) + C(P_{ci}, t_4) = 0$$

$$C(P_{ci}, t_2) + C(P_{ci}, t_5) = 0$$

$$C(P_{ci}, t_{51}) = 0$$

This linear system of the theory of regions is solved using the CPLEX software. Therefore, the PN monitors are listed as follows (i.e., see Figure 14):

$$P_{c1} : M_0(P_{c1}) = 1; C(P_{ci},.) = (0, -1, -1, -1, 1)$$

$$P_{c2} : M_0(P_{c2}) = 1; C(P_{ci},.) = (0, -1, 0, 0, 1)$$

In our case study, the supervisor is made up of two control places P_{c1} and P_{c2}. This PN controller will subsequently be transformed into a SIPN controller. Next, the synthesised model can be entered in the input interface. The developed application assures the LIST code generation imported into the PLC.

This controller will subsequently be translated into SIPN. Figure 15 presents the Petri net supervisor used in our application.

Indeed, we observe that many other features exist in our application; the used inputs and outputs are effectively associated with their addresses in the PLC, the insertion of the comments associated to the places/transitions and the addition of a code associated to each place to define the behaviour of the system (i.e., place P_7 contains a sub-network modelled in Figure 16).

The application was developed to mainly ensure the generation of a LIST code supported by the PLC. Then it is obvious to make sure that the generated code is correct. In the following Figure 17 is presented the generated LIST code of our control specification. It remains to verify the accuracy of this code. For this fact, we have used STEP7, the SIEMENS PLC development software (SIEMENS, 2012b). We created a STL source file, so we can insert the generated source code afterwards. Figure 18 shows the compiling result of the generated code. It is well noticed that the code is functional since the STEP7 compiler did not detect any error.

Figure 14 The reachability graph (see online version for colours)

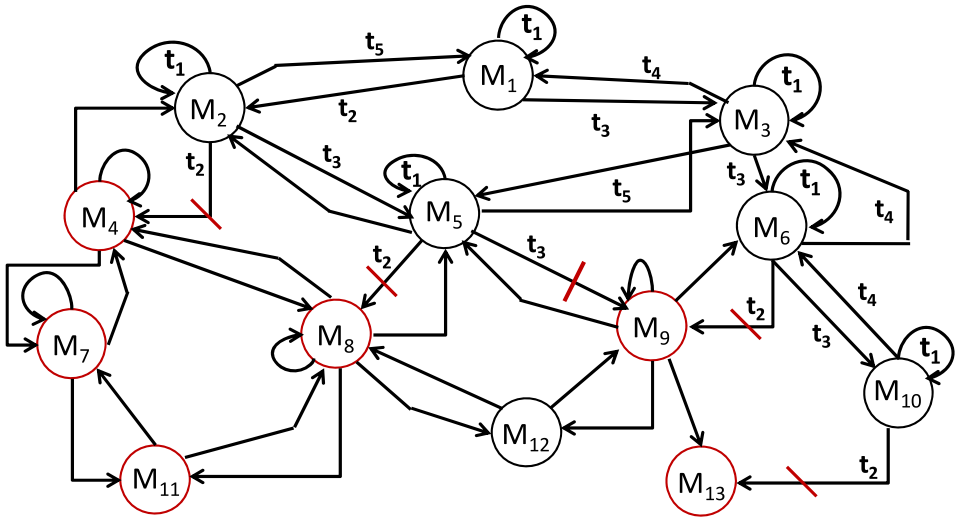


Figure 15 The controlled PN model

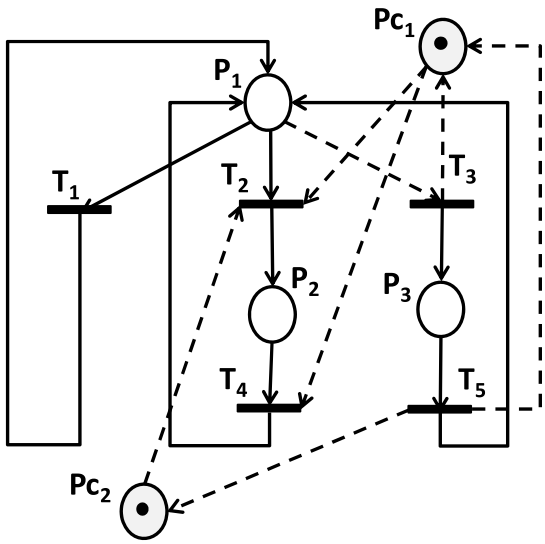


Figure 16 Snapshot of a part of the generated SIPN supervisor (see online version for colours)

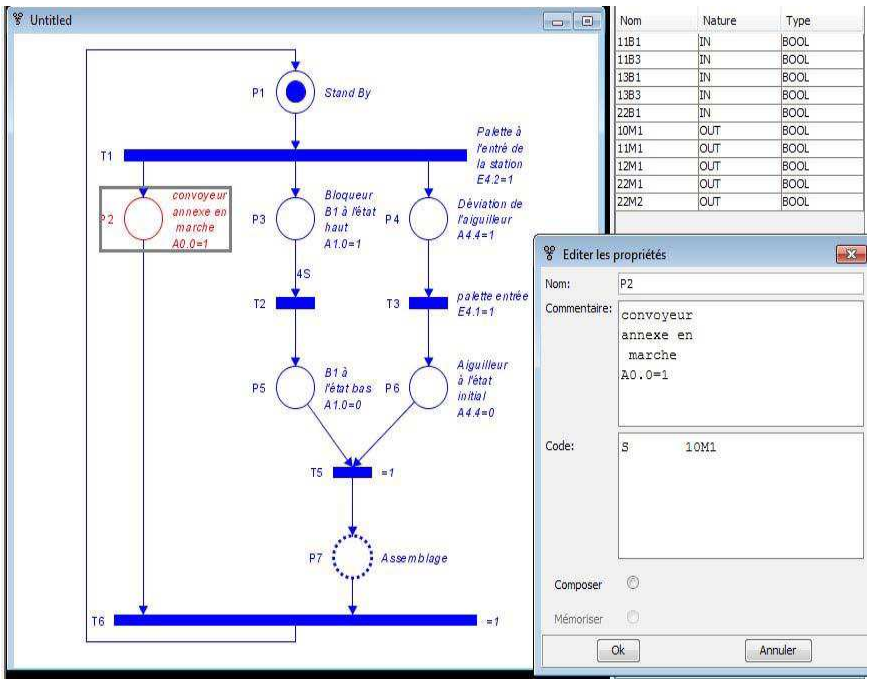


Figure 17 Snapshot of the sub-network of P_7 (see online version for colours)

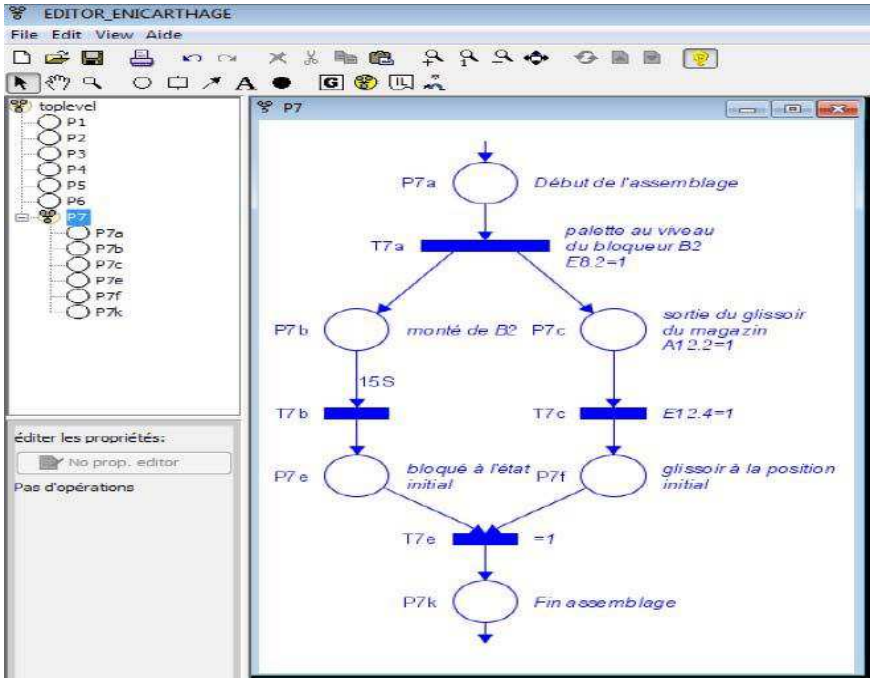


Figure 18 Snapshot of the generated code (see online version for colours)

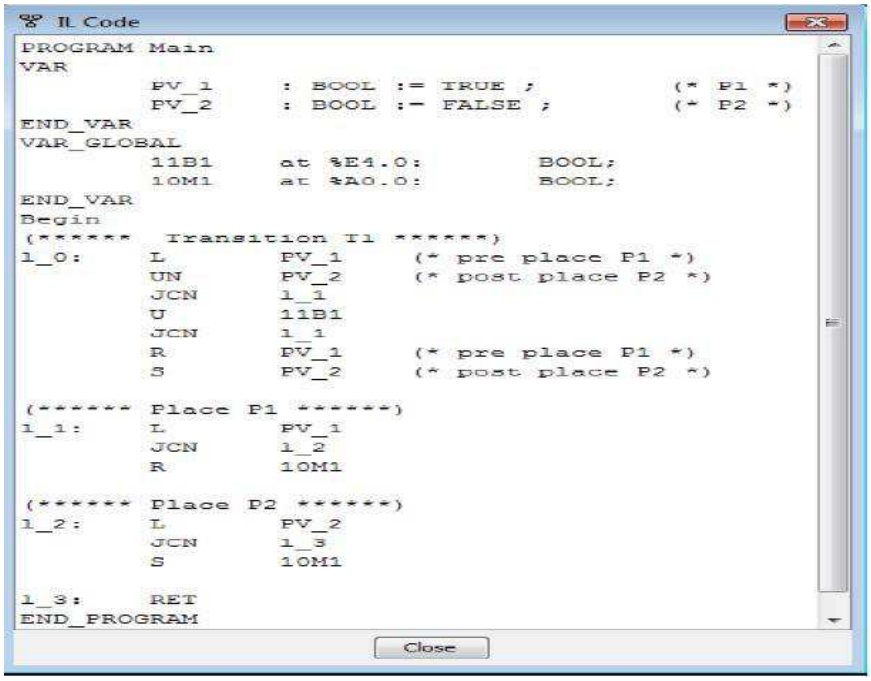
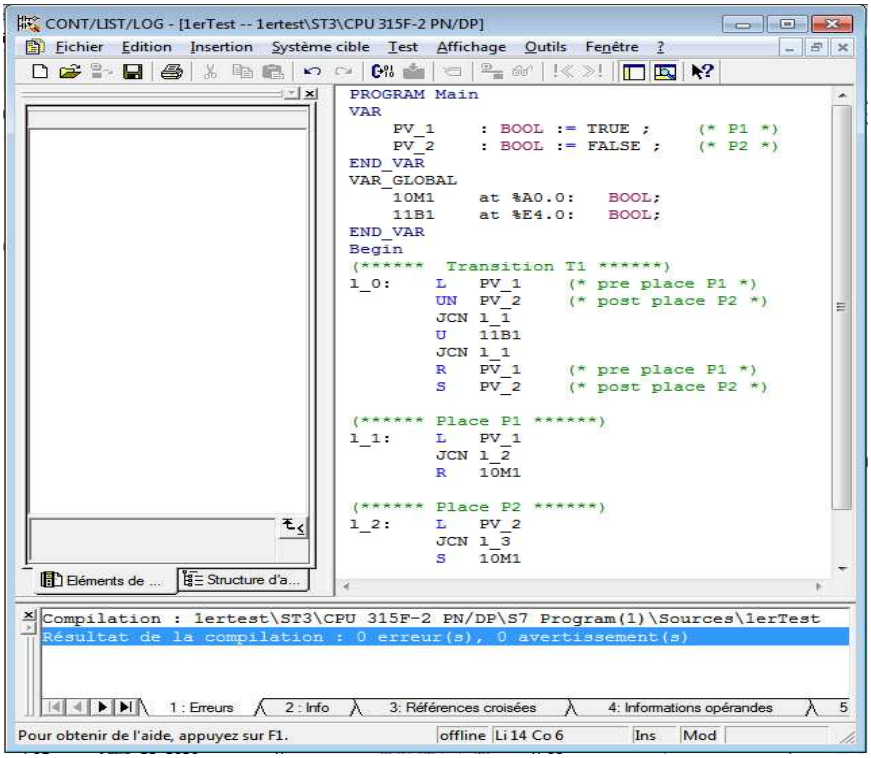


Figure 19 The code tested with STEP7(see online version for colours)



5 Conclusions and future work

The proposed control policy is implemented in a FMS. A Java application is developed wherein a PN model has been edited and then transformed into a comprehensible code supported by the PLC. Unlike the PN models, the use of the SIPN was essential in our work. After a thorough study of the PNs and the theory of regions for control synthesis, it was not obvious to directly implement a PN supervisor on a PLC since it is non-autonomous and does not interact with the external environment. However, the use of the SIPN was the alternative chosen, developed and implemented since one can manipulate the internal signal of the PLC.

Once the internal input/output signals of the PLC are detected, one can transform the PN model into a language supported by the PLC. The LIST language of IEC norm 61131-3 was chosen. In fact, a Java application was developed to model any PN controller and then generate the corresponding LIST code that will be sent to the target in a second time.

The interface is functional from the creation of the project until the code generation. It is possible to save a project, to model a SIPN as in any editor and to associate the inputs to their addresses in the PLC. Then, the generated code will be tested using the SIEMENS STEP7 compiler.

In perspective, the graphical interface of the application can be improved whether visually or functionally. An added feature that would also be very useful is the ability to compile and visualise directly the generic code on the PLC. This would allow the interface to be complete and no longer require any other software.

References

Asensio, J., Ortuño, F., Damas, M. and Pomares, H. (2013) 'Industrial automation programming environment with a new translation algorithm among IEC 61131-3 languages based on the TC6-XML scheme', *International Journal of Automation and Control Engineering*, Vol. 2, No. 2, pp.47–55.

Azar, A.T. and Vaidyanathan, S. (Eds.) (2014) *Computational Intelligence Applications in Modeling and Control*, Vol. 575, Springer, Sousse, Tunisia.

Azar, A.T. and Vaidyanathan, S. (Eds.) (2015) *Chaos Modeling and Control Systems Design*, Springer, Germany.

Azar, A.T. and Vaidyanathan, S. (Eds.) (2016) *Advances in Chaos Theory and Intelligent Control*, Vol. 337, Springer, Bale, Switzerland.

Azar, A.T. and Zhu, Q. (Eds.) (2015) *Advances and Applications in Sliding Mode Control Systems*, Springer International Publishing, Cham.

Azar, A.T., Vaidyanathan, S. and DeMarco, A. (Eds.) (2015) *Handbook of Research on Advanced Intelligent Control Engineering and Automation*, Engineering Science Reference, Florida, USA.

Badouel, E. and Darondeau, P. (1998) 'Theory of regions', in: *Lectures on Petri Nets I: Basic Models*, pp.529–586, Springer Berlin Heidelberg.

Blocks, L. (1998) 'Functional block diagram', *MATRIX*, Vol. 8, No. 8, p.8.

Bolton, W. (2015) *Programmable Logic Controllers*, Newnes, Amsterdam.

Capito, L., Proaño, P., Camacho, O., Rosales, A. and Scaglia, G. (2016) 'Experimental comparison of control strategies for trajectory tracking for mobile robots', *International Journal of Automation and Control*, Vol. 10, No. 3, pp.308–327.

Fabian, M. and Hellgren, A. (1998) 'PLC-based implementation of supervisory control for discrete event systems', in *Proceedings of the 37th IEEE Conference on Decision and Control*, IEEE, Vol. 3, pp.3305–3310.

Fen, G. and Ning, W. (2006) 'Transformation algorithm between ladder diagram and instruction list based on AOV diagraph and binary tree', *Journal of Nanjing University of Aeronautics and Astronautics*, Vol. 6, No. 2, p.19.

Frey, G. (2003) 'Hierarchical design of logic controllers using signal interpreted Petri nets', *IFAC Proceedings*, Vol. 36, No. 6, pp.361–366.

Frey, G.and Minas, M. (2000) 'Editing, visualizing, and implementing signal interpreted Petri nets', in *Proceedings of the AWPN 2000*, October, pp.57–62).

Ghaffari, A., Rezg, N. and Xie, X. (2002a) 'Algebraic and geometric characterization of Petri net controllers using the theory of regions', in *Proceedings. Sixth International Workshop on Discrete Event Systems*, IEEE, pp.219–224.

Ghaffari, A., Rezg, N. and Xie, X. (2002b) 'Live and maximally permissive controller synthesis using theory of regions', in: *Synthesis and Control of Discrete Event Systems*, pp.155–166, Springer US.

Ghaffari, A., Rezg, N. and Xie, X. (2003a) 'Design of a live and maximally permissive Petri net controller using the theory of regions', *IEEE Transactions on Robotics and Automation*, Vol. 19, No. 1, pp.137–141.

Ghaffari, A., Rezg, N. and Xie, X. (2003b) 'Feedback control logic for forbidden-state problems of marked graphs: application to a real manufacturing system', *IEEE Transactions on Automatic Control*, Vol. 48, No. 1, pp.18–29.

Grobelna, I., Wisniewski, R., Grobelny, M. and Wisniewska, M. (2016) 'Design and verification of real-life processes with application of Petri nets', *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.

Ioannides, M.G. (2004) 'Design and implementation of PLC-based monitoring control system for induction motor', *IEEE Transactions on Energy Conversion*, Vol. 19, No. 3, pp.469–476.

Minas, M. and Frey, G. (2002) 'Visual PLC-programming using signal interpreted Petri nets', in *Proceedings of the 2002 American Control Conference*, IEEE, May, Vol. 6, pp.5019–5024.

Murata, T. (1989) 'Petri nets: properties, analysis and applications', *Proceedings of the IEEE*, Vol. 77, No. 4, pp.541–580.

Nagata, T. and Hirose, K. (2016) *US Patent No. 9,395,710*, US Patent and Trademark Office, Washington, DC.

Peng, S.S. and Zhou, M.C. (2004) 'Ladder diagram and Petri-net-based discrete-event control design methods', *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, Vol. 34, No. 4, pp.523–531.

Rakhshan, M., Vafamand, N., Shasadeghi, M., Dabbaghjamanesh, M. and Moeini, A. (2016) 'Design of networked polynomial control systems with random delays: sum of squares approach', *International Journal of Automation and Control*, Vol. 10, No. 1, pp.73–86.

Ramadge, P.J. and Wonham, W.M. (1987) 'Supervisory control of a class of discrete event processes', *SIAM Journal on Control and Optimization*, Vol. 25, No. 1, pp.206–230.

Ramadge, P.J. and Wonham, W.M. (1989) 'The control of discrete event systems', *Proceedings of the IEEE*, Vol. 77, No. 1, pp.81–98.

Rezig, S., Achour, Z. and Rezg, N. (2017) 'Theory of regions for control synthesis without computing reachability graph', *Applied Sciences*, Vol. 7, No. 3, p.270.

Rezig, S., Achour, Z., Rezg, N. and Kammoun, M.A. (2016) 'Supervisory control based on minimal cuts and Petri net sub-controllers coordination', *International Journal of Systems Science*, Vol. 47, No. 14, pp.3425–3435.

Salloum, G., Mbayed, R. and Monmasson, E. (2016) 'Generic optimal control of synchronous motors', *International Journal of Automation and Control*, Vol. 10, No. 4, pp.389–406.

Shieh, C.S. (2014) 'Segmented control based on fuzzy logic control for non-linear inverted pendulum swing-up', *International Journal of Automation and Control*, Vol. 8, No. 1, pp.88–97.

SIEMENS (2012a) *Communication Industrielle*, Cathalogue ik pi.

SIEMENS (2012b) *Siemens S7*, Statement List (STL), PhD thesis.

Verma, C. and Pandey, R. (2016) 'An implementation approach of big data computation by mapping Java classes to MapReduce', in *3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016, IEEE, March, pp. 282–287.

Vieira, A.D., Cury, J.E.R. and de Queiroz, M.H. (2006) 'A model for PLC implementation of supervisory control of discrete event systems', in *IEEE Conference on Emerging Technologies and Factory Automation ETFA'06*, IEEE, September, pp.225–232.

Vyatkin, V. and Instrument Society of America (2007) *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design (p. o3neida)*, ISA-Instrumentation, Systems, and Automation Society.

Yu, M., Yan, C. and Li, C. (2016) 'Event-triggered control for heterogeneous multi-agent systems with time-varying delays when using the second-order neighbours' information', *International Journal of Automation and Control*, Vol. 10, No. 3, pp.286–307.

Zhou, M. and Venkatesh, K. (1999) *Modeling, simulation, and control of flexible manufacturing systems: a Petri net approach*, Vol. 6, World Scientific, London.

Zhu, Q. and Azar, A.T. (Eds.) (2015) *Complex System Modelling and Control Through Intelligent Soft Computations*, Springer, Germany.

List of abbreviations/acronyms

DES	Discrete event systems
FMS	Flexible manufacturing system
GMEC	General mutual exclusion constraint
PN	Petri net
PLC	Programmable logic controller
SIPN	Single interpreted Petri net
MTSI	Marking/transition separation instances