VOL. 6, NO. 2, FEBRUARY 2011 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences


www.arpnjournals.com

PETRI NET TO LADDER LOGIC DIAGRAM CONVERTER AND A BATCH PROCESS SIMULATION

Mostafa M. Gomaa
Department of Computer and Systems Engineering, Ain Shams University 1 El_Sarayatte St., Abbassia, Cairo, Egypt
E-Mail: mostafa.gomaa@eng.asu.edu.eg

ABSTRACT

Discrete-event dynamic systems (DEDS) are characterized by a set of states which the system can take, and by the set of asynchronous events that cause the state changes at discrete time points. Programmable logic controllers (PLCs) are still important special purpose computers used to automate the DEDS in industry. Ladder logic diagrams (LLDs) are still the most popular graphical programming tools of the PLCs; but the major problem is that programming is done heuristically and the LLDs are difficult to be used for both analysis and performance evaluation. Petri nets (PNs) are nowadays the most effective modeling environment for both the design and implementation of DEDS. This paper proposes a PN to LLD conversion tool, used for graphical editing of a PN net model of a DEDS controller and for converting this PN into the equivalent LLD for programming a PLC. The conversion algorithm is presented, considering many types of transitions, places, and arcs with generality that many types of PNs can be considered. This paper also presents a simulation of a batch process, on a personal computer from one side, interfaced with a real PLC from the other side, that is programmed using a LLD obtained from the conversion of a suitable PN model using the proposed conversion tool. Compared with a LDD got heuristically, the LLD got from a PN conversion is simpler, understandable, and meeting all the characteristics obtained from the PN analysis.

Keywords: Petri nets, ladder logic diagrams, batch processes, programmable logic controllers, simulation**.**

1. INTRODUCTION

Discrete event dynamic systems (DEDS) are characterized by a set of states which the system can take, and by the set of events that cause the state changes at discrete time points. The events may take place asynchronously as opposed to the synchronous nature in a discrete time system. The change of states and occurrence of events are the essence of the DEDS dynamic behavior. A primary task of the DEDS theory is creating a DEDS model. Without such a model it would be impossible to analyze and control DEDS [1]. Examples of DEDS are: production systems, manufacturing systems, batch processes, mobile robotics, industrial systems that contain changements of structure, commutations, on/off switching elements, non-linearities modeled by piece-wise linear form (e.g. thresholds, saturation, dead-band, etc.). Batch processes are evident examples of DEDS at higher level of abstraction, where they involve a sequence of phases that are carried out on a discrete quantity of material within a piece of operating equipment (e.g. reactor, blender) [2-3].

In DEDS, time is not explicitly expressed and only the precedence relations for states and events are given. The tool used for modeling a DEDS should achieve the following needs [4]:

- Describe the sequence of states of the DEDS.
- Take into account the concurrency, where some systems may be partially independent.
- Have a clear understanding of the input-output behavior of the DEDS.
- Given a state of the system, only a few inputs may affect the state, and a few outputs may change. Then, describe only the behavior corresponding to this input changes.

Famous DEDS modeling tools are: Automata, Ladder logic diagram (LLD), Grafcets, and the Petri nets (PNs).

An *automaton* takes input from a finite sequence of symbols, which is called a word and it contains a finite set of states. During each time instance of some run, automaton has to be in one of its states. At each time step when automaton reads a symbol, it jumps or transits to next state depending on its current state and the read symbol. *Ladder logic* can be thought of as a rule-based language, rather than a procedural language. A *"rung"* in the ladder represents a rule. When implemented with relays and other electromechanical devices, the various rules *"execute"* simultaneously and immediately. When implemented in a PLC, the rules are typically executed according to a scan cycle [5]. A *Petri net* is a directed bipartite graph, in which the nodes represent transitions (i.e., events that may occur, signified by bars) and places (i.e., conditions, signified by circles). The directed arcs describe which places are pre- and/or post conditions for which transitions (signified by arrows) [6-7]. A *Grafcet* (or Sequential Function Chart (SFC)) is evolved from Petri nets (*safe PN*). It is organized into a set of steps and transitions connected by directed links [4].

The PN tools proved to verify the stated needs above [4]. The Petri nets present two interesting characteristics:

- Ability to model and visualize types of behavior having parallelism, synchronization, and resource sharing;
- Ability to provide many properties describing the modeled DEDS, e.g. liveliness, deadlock, conflict, boundness, and safety.

A PLC is a special purpose industrial computer used to automate industrial processes. It can be connected to several inputs and outputs, and can be programmed to control the state of the outputs depending on the configuration of the inputs and its internal state [8]. The PLC execution follows a scan cycle consisting of: i) read and store the inputs; ii) execute the entire user program code; iii) write the outputs. It repeats as long as the PLC is running.

A PLC can be programmed using any of the five languages, defined in the standard IEC 1131-3, which are: Instruction List (IL), Structured Text (ST), Function Blocks Diagrams (FBD), SFC and LLD. Except for SFC, the semantics of these languages is not strictly defined; certain definitions are missing or contain ambiguities [9].

LLD is the most used language for programming PLCs. It is a graphical language where the basic elements are based on an analogy to physical relay diagrams, so it is familiar to the people working in the industry. But the disadvantages of LLD are: i) LLD becomes cumbersome both to design and debug when applied to complex sequential tasks in DEDS; ii) programming using LLD is done heuristically with an increasing difficulty with increasing the order of complexity; iii) lack of *flexibility, reusability* and *good maintainability*; iv) When a problem is detected in the system, cause-tracing and locating becomes extremely difficult, costly, and sometimes even a threat to human safety; v) LLDs don't provide any tools for analysis and performance characteristics of the system.

In [5], a comparative study between PNs and LLDs is carried out in terms of *design complexity*, *graphical complexity*, *adaptability for change in specifications*, and *response time*. This study was in the favor of PNs.

According to the stated PN advantages mentioned earlier and to the disadvantages of the LLDs, many researchers analyze and verify the LDDs using PNs [8, 10] and others begin the modeling, design and analysis using PNs and then convert the PNs into corresponding LLDs or IL [9, 11-15].

In [8], they define a time Petri net (TPN) semantics for LLD programs through an ATL (ATLAS Transformation Language) model transformation. They then generate behavioral properties over the LLD models as LTL (Linear Temporal Logic) formulae which are then checked over the generated TPN using the model checkers. In [10], they build PN net state equation for the LLD by using the firing condition functions. The transition values as used in the conventional PN state equation are replaced with transition functions.

In [9], they convert a control interpreted Petri net (CIPN) type into LLD based on the state equation of the PN and the structure of the PN using its incidence matrix. In [11], they convert a control Petri net (CPN) into LLD where, based on the firing regulation of transition, the relationship of places, conditions, and events are formulated with Boolean functions. These functions are converted into LLD. In [12], they employed token passing logic (TPL) methodology to convert from PN into a Petri net controller (PNC) and then into LLD. They use counters to store place tokens and use on delay timers to model timed places or timed transitions. In [13], they introduce a special type of PNs, the signal interpreted Petri net (SIPN). They introduce also a tool for graphical editing and compilation of the SIPN into instruction list (IL) for programming the PLC directly. In [14], they introduce fuzzy formalism via a new type of PN, fuzzy automation Petri net (FAPN). They then use the TPL as in [12]. In [15], he constructs a Petri net-based logic and sequence control model of a photo mask transport. Then he generates the LLD from the Petri net-based control model.

The problems in the approaches of conversion of PNs into LLDs or ILs are: i) some aspects of the conversion are not clear; ii) a general conversion algorithm is not presented; iii) dealing with specific types of PNs.

This paper adopts the approach that begins with the modeling, design and analysis of DEDS using PNs and then converts them into LLDs for PLC programming. Contributions of this paper are: i) proposal of a PN to LLD conversion tool, providing graphical editing of the PN, considering many types of places, transitions and arcs. The conversion algorithm is presented; ii) modeling a batch process using a PN and obtaining its equivalent LLD; iii) implementing a simulation of a batch process to test the PLC program, through microcontroller based interfacing with a PC.

The rest of this paper is organized as follows. In Section 2, the proposed PN to LLD converter is presented. In Section 3, a batch process and its PN control model are given. Section 4 presents the PLC LDD and real-time simulation scheme. Final conclusions are given in Section 5.

2. PETRI NET TO LADDER LOGIC DIAGRAM CONVERTER

The PN to LLD conversion tool is implemented using Java. It provides the interface shown in Figure-1.

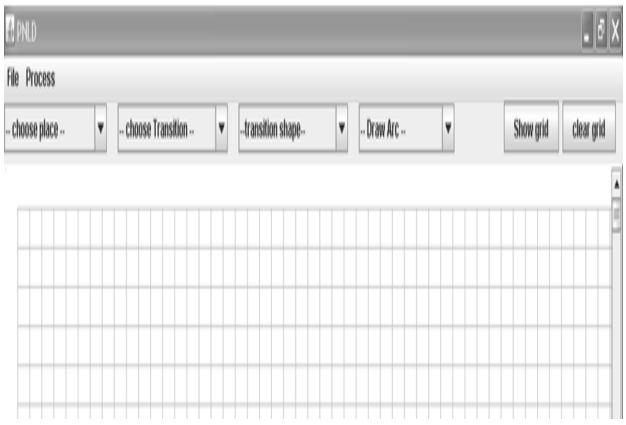

Figure-1. GUI of the proposed PN to LLD converter.

Where it is possible to choose place type, transition type, transition shape, arc type, show/hide the grid. Through menu option *File*, it is possible to open/save the PN model. Through the *Process* option, it is possible to show

the PN place/transition information, convert it to the equivalent LLD, or print the LLD. Four types of places and seven types of transitions are available as given in Figures 2a and 2b, respectively. E.g. in the graphical PN editing area, at certain position, when an Action and Timed *Place* is chosen, a dialogue box appears to require its initial marking value, the time value and the action label. When a *Timed Event Transition* is chosen, the time value and the event label are edited through a suitable dialogue box. Automatic place and transition numbering is followed; i.e., the first place is p1, the second place is p2 and so on; similarly the transitions. During drawing the transition, it is possible to choose its shape (Figure-2c): vertical, horizontal, diagonal with 45°, or diagonal with 135°. An arc may be normal, having weight more than or equal to one, or inhibitor whose weight is always one.

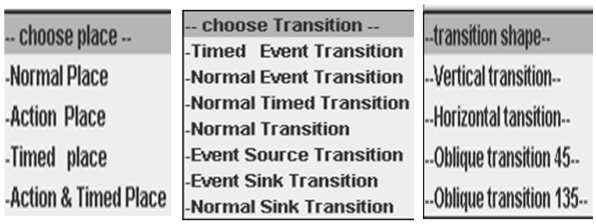

Figure-2. Place types, transition types and transition shapes.

When drawing an arc, it is edited its input place (transition) and its output transition (place). In Figure-3, it is given an example of edited PN part involving a single transition and six places with the shown place marking and arc weights. This PN part place/transition information is given in Table-1. The generated equivalent LLD is shown in Figure-4. In Figure-3, empty places have zero marking. The place p3 is input to and output from the transition t1, which represent state reading. The unity arc weight is not indicated. The places p1, p2, and p3 are of *Normal* type; the place p4 is of *Action and Timed* type; the place p5 is of *Action* type; the transition t1 is of *Timed Event* type.

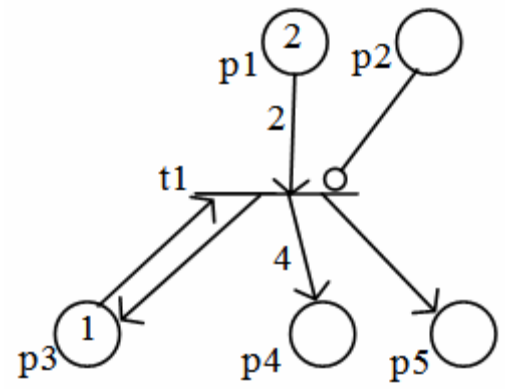

Figure-3. A PN part as edited in the graphical editor.

Table-1. Place/transition information of the PN of Figure-3.

Place/transition	Associated time	Actions/events
p1	---	---
p2	---	---
p3	---	---
p4	20 units	a1
p5	---	a2
t1	30 units	e1

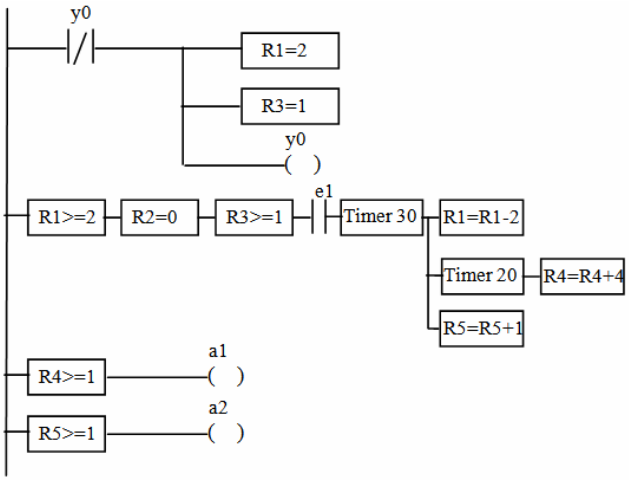

Figure-4. The LLD equivalent to the PN part of Figure-3.

In Figure-4, the marking of a PN place Pi is stored into a PLC register Ri. The first rung represents an initial marking setting; the unmentioned register has zero initial value; yo is a PLC internal variable. The second rung is equivalent to the transition t1 firing, where when ($R1 \geq 2$ and $R2 = 0$ and $R3 \geq 3$ and event e1 occurs and after *on delay timer* 30 units elapsed) then (subtract 2 from R1, add 1 to R5, and after *on delay timer* 20 units elapsed add 4 to R4). The 3rd and 4th rungs correspond to action places p4 and p5, when they contain at least one token. Other place and transition types are treated in a similar manner as given in the previous example.

Given the PN structure and initial marking, captured in suitable matrices and vectors, during PN editing, the PN to LLD conversion algorithm is summarized as follows:

a) Draw the initialization rung, considering only non-zero registers.
b) FOR each transition ti //add a rung in the LLD.

Start a new rung
SELECT CASE ti type
CASE "Timed Event":
FOR each input place pj to ti
IF arc pj→ti is normal THEN
add Rj >= *arc weight* rectangular component.
ELSE //i.e., inhibitor arc.
add Rj = 0 rectangular component.

END IF
END FOR
add ei label above two vertical bars component.
add *on delay time* component of ti associated time.
FOR each input place pj to ti
IF arc pj→ti is normal THEN
branch the rung;
add *Rj = Rj-arc_weight* rectangular component.
END IF
END FOR
FOR each output place pk from ti
IF there is no self loop between Pk and ti THEN
IF Pk has *timed* feature THEN
branch the rung;
add *on delay time* component of pk time;
add *Rk=Rk+arc_weight* between ti and pk.
ELSE
branch the rung;
add *Rk=Rk+arc_weight* between ti and pk.
END IF
END IF
END FOR
. //other transition types are handled similarly
. // with small changes; e.g a sink transition has
. //no output places; a source transition has no
. // input places.

END SELECT
c) FOR each action place pi
Start a new rung;
add Ri>=1 rectangular component;
add the place pi action label (aj) above two parenthesis output component.
END FOR.

3. BATCH PROCESS AND ITS PN MODEL

The batch process handled in this paper is shown in Figure-5, during simulation in real-time. The following components are operated in an ON/OFF fashion: the four valves VA, VB, VC, and Vop, and the two pumps P1 and P2, and the heater in Tank 2. The operation sequence is as follows:

a) After pressing start button, valves VA and VC open.
b) When 2/3 maximum level in tank 1 is reached, VA closes, VB opens and the mixer in tank 1 turns on. When maximum level in tank 2 is reached, VC closes and the heater is turned on.
c) When maximum level in tank 1 is reached, VB closes. When temperature in tank 2 reaches 60 °C, the heater is turned off.
d) Now, the two pumps P1 and P2 are ON. When minimum level in tank 3 is reached, the mixer in tank 3 turns on.
e) When minimum level in tank 1 is reached, both mixer in tank 1 and pump P1 stop. When minimum level in tank 2 is reached, pump P2 stops.
f) When maximum level in tank 3 is reached, both pumps stop. Wait for 60 seconds.
g) The valve Vop opens for outputting the product.
h) When minimum level in tank 3 is reached, both mixer in tank 3 and the valve Vop stop and a new batch can be started.
i) At any moment, pressing stop button leads to process re-initialization.

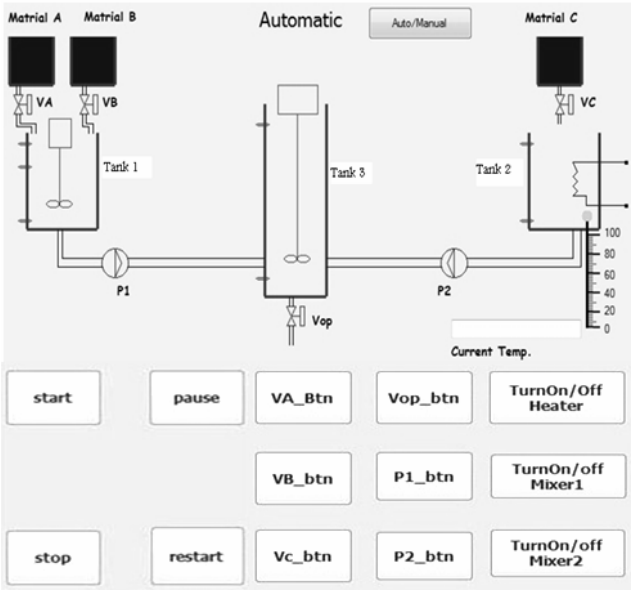

Figure-5. A batch process simulation GUI.

The PN model achieving the control recipe just mentioned is given in Figure-6. Some places represent partial states: p1: initial state; the places p2, p3, p9, p10, p13, p14, and p15 are intermediary places. Some places represent resources: p4: the valve VC; p5: the valve VA; p6: the heater; p7: the valve VB; p8: the mixer in tank 1; p11: the pump P1; p12: the pump P2; p16: the mixer in tank 3; p17: the valve Vop. A token in a place representing a resource indicates that this resource is on; a non-token indicates that this resource is off.

Transition t9 and t14 are of normal type. Transition t15 is of normal timed type with time 60 seconds. Other transitions are associated with events, where a transition ti is associated with an event ei. With Lj is the level in tank j, for j=1, 2, and 3, the event descriptions are as follows: e1: start; e2: L2<L2min; e3: L1<L1min; e4:L2≥L2max; e5: L1≥2/3 L1max; e6: Temperature in tank 2≥ 60°C; e7: L1≥L1max; e8: (L1<L1min OR stop); e10: (stop AND L1≥L1min); e11: (stop AND L2≥L2min); e12: (L3≥L3max OR L1<L1min); e13: (L3≥L3max OR L2<L2min); e16: L3<L3min; e17, e18, e19, e20, e21: stop.

Simulation of the PN in Figure-6, using PN state equation, and the obtainment of the reachability graph, proved the desired control recipe and that there is no deadlock states.

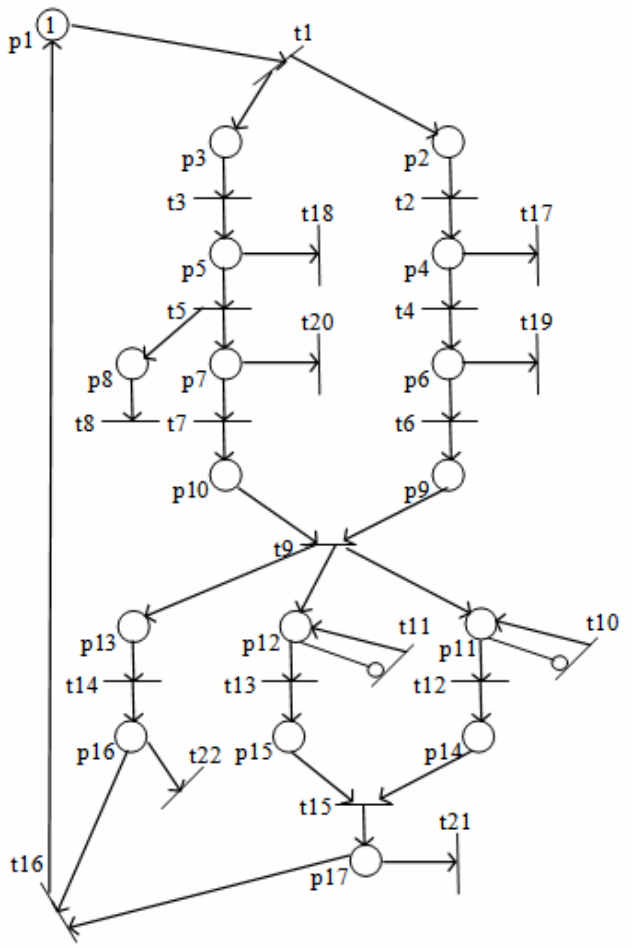

Figure-6. The PN model of the batch process.

4. PLC PROGRAMMING AND BATCH PROCESS SIMULATION

The LLD, obtained from the conversion of the PN of Figure-6, is shown in Figure-7. It is shown only the initialization rung, the rungs corresponding to transitions t1 and t12 for example, where other transition rungs are similar as explained in the conversion algorith, and the output rungs.

Programming a PLC with the LLD of Figure-7 is carried out. The PLC is *Fatek FBS* type, having 24 inputs/16 outputs. The simulation of the batch process is carried out on a PC using Visual C#. The interfacing between the PC and the PLC is achieved using a microcontroller based circuit as shown in Figure-8.

Simulation GUI of the batch process is shown in Figure-5. Through *Automatic mode*, the control recipe is carried out through the interaction with the PLC actions in real time. Tank levels animation is shown; valves, pumps, and heater status is also shown through color change (red/green); mixers status is shown via rotation/no rotation. It is also available to pause/ restart the simulation. Through *Manual mode*, it is available to start/stop any of the nine on/off resources. Through simulation interacting with the real PLC, the LLD is examined and proved to carry out the desired control recipe.

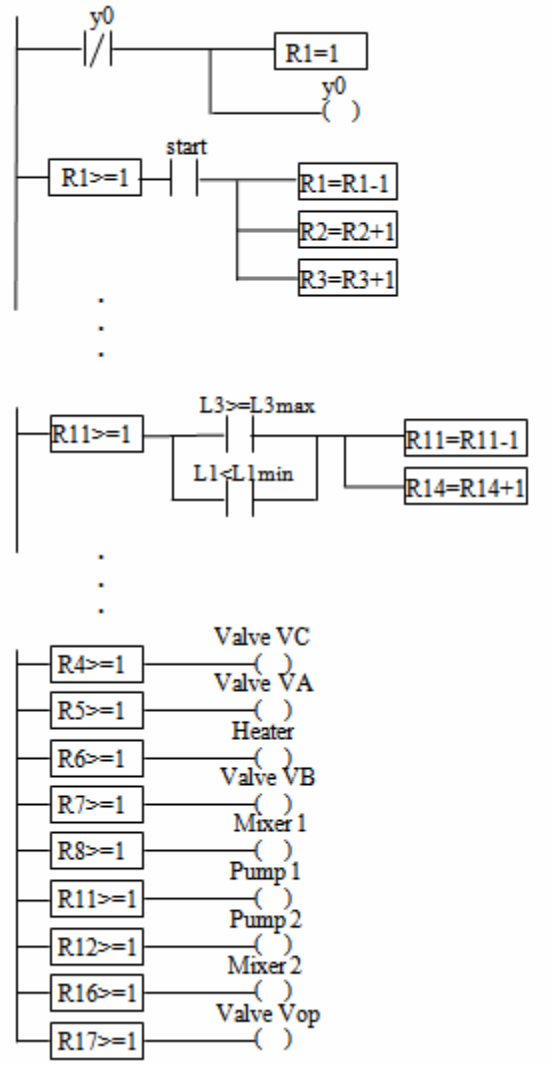

Figure-7. The LLD obtained from PN conversion.

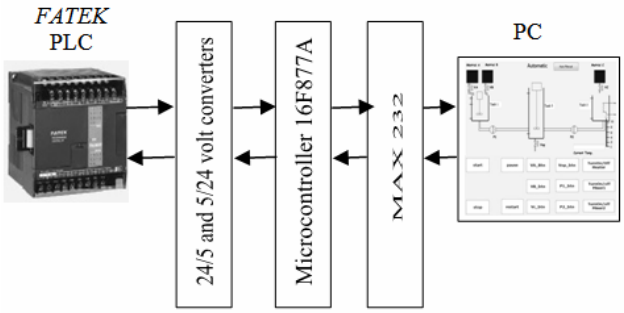

Figure-8. PLC-PC interfacing scheme.

5. CONCLUSIONS

This paper presents a clear and straightforward conversion tool from Petri nets (PNs) to ladder logic diagrams (LLDs) used for programming a programmable logic controller (PLC). Compared with a LLD obtained heuristically, the LLD obtained from PN conversion is readable, understandable, and verifying all the properties and performance got from the PN analysis. Starting with PN model/controller synthesis is effective, modular, and

consistent for the discrete event dynamic systems (DEDS); then the LLD obtained from PN conversion is programmed directly on a suitable PLC. This paper also handles an example of a batch process: the PN model representing the control recipe is given; its equivalent LLD is given; simulation of the batch process and interaction with the real PLC is carried out. In future work, a programming tool using the PN directly is to be implemented.

REFERENCES

[1] B. Hrúz, M. C. Zhou. 2006. Modeling and control of discrete-event dynamic systems with Petri nets and other tools. Springer.

[2] H. Yu, T. C. Yang, S. Cang, W. Chen. 2004. Modeling and analysis of a batch plant using Petri nets. Control 2004, University of Bath, UK, ID-074, pp. 1-6.

[3] L. Ferrarini, L. Piroddi. 2003. Modular design and implementation of a logic control system for a batch process. Journal of Comp. and Chem. Engineering, Elsevier. 27: 983-996.

[4] R. David. 1995. Grafcet: a powerful tool for specification of logic controllers. IEEE Trans. on Control Sys. Tech. 3(3): 253-268.

[5] K. Venkatesh, M. C. Zhou, R. J. Caudill. 1994. Comparing ladder logic diagram and Petri nets for sequence controller design through a discrete manufacturing system. IEEE Transactions on Industrial Electronics. 41(6): 611-619.

[6] T. Murata. 1989. Petri nets: properties, analysis and applications. Proc. of IEEE. 77(4): 541-580.

[7] R. David, H. Alla. 2005. Discrete, continuous, and hybrid Petri nets. Springer.

[8] D. F. Bender, B. Combemale, X. Crégut, J. M. Farines, B. Berthomieu, F. Vernadat. 2008. Ladder Metamodeling and PLC Program Validation through Time Petri Nets. Model Driven Architecture-Foundations and Applications (ECMDA 2008), Berlin-Germany. pp. 121-136.

[9] M. V. Moreira, D. S. Botelho, J. C. Basilio. 2006. Ladder diagram implementation of control interpreted Petri nets: a state equation approach. IFAC DES Design, Gandia Beach, Spain. pp. 85-90.

[10] G.B. Lee and J.S. Lee. 2002. Constructing Petri Net State Equation for Ladder Diagram. Journal of Intelligent Systems. 12(2): 69-92.

[11] G. B. Lee, H. Zandong, J. S. Lee. 2004. Automatic generation of ladder diagram with control Petri Net. Journal of Intelligent Manufacturing. 15(2): 245-252.

[12] M. Uzam, A. H Jones, N. Ajlouni. 1996. Conversion of Petri net controllers for manufacturing systems into ladder logic diagrams. Proc. of the 5th IEEE International Conference on Emerging Technologies and Factory Automation, Hawaii, USA. pp. 649-655.

[13] M. Minas, G. Frey. 2002. Visual PLC-programming using signal interpreted Petri nets. Proc. Of the American Control Conference (ACC2002), Anchorage, Alaska. pp. 5019-5024.

[14] P. R. Venkateswaran, J. Bhat, S. Meenatchisundaram. 2009. Formalism for fuzzy automation Petri nets to ladder logic diagrams. ARPN Journal of Engineering and Applied Sciences. 4(10): 83 92.

[15] Y. Liu. 2009. A Petri net-based ladder logic diagram design method for the logic and sequence control of photo mask transport. Proc. of the 5th Int. Conf. on Emerging Intelligent Computing Technology and Applications. pp. 774-783.