Design of Industrial Automated Systems Via Relay Ladder Logic Programming and Petri Nets

MengChu Zhou, *Senior Member, IEEE*, and Edward Twiss, *Member, IEEE*

***Abstract*—For the past decades, programmable logic controllers (PLC's) using relay ladder logic (RLL) programming have been the workhorse for controlling event-driven industrial automated systems. RLL proved flexible compared to the hardwired RLL control implementation, due to its feature of software implementation. As automated systems become more complex, they also become more difficult to understand and maintain. It takes tremendous effort to accommodate specification changes, which become frequent, to meet today's flexible and agile automation needs. Several methods emerge to overcome the shortcomings of RLL. Petri nets (PN's), initially proposed as a modeling tool, have been developed as such a method. This paper adopts an industrial scale system to compare RLL and PN design methods so that the advantages of PN-like approaches are fully recognized. The criteria are the understandability that relates to the ability to evaluate the programmed logic, to verify its correctness, and to maintain the control system as well as the flexibility that relates to the easy modification of logic when the specification changes. To the authors' knowledge, this is the first work that takes an existing industrial system, conducts discrete event control designs by using both RLL programming and PN methods, and performs a comparative study on them. Together with the previous comparison results through small-scale systems, the results of this study support that PN-like advanced discrete event control design methods are better than RLL, in terms of the understandability and flexibility of a resulting control design.**

***Index Terms*—Design method, discrete event systems, industrial automation, Petri net, relay ladder logic.**

I. Introduction

As PRODUCT lifecycles become shorter, factories are pushed to develop small batches of many different products. The need for highly flexible control systems has become a necessity. The majority of existing automated industrial systems are controlled by programmable logic controllers (PLC's). In most cases, the control programs for PLC's are developed by using relay ladder logic (RLL) [3]. RLL is a graphical programming language consisting of software devices (i.e., relays, timing relays, drum sequencers, and programmable counters) to achieve a control strategy. RLL's graphical representation of physical switching devices does not capture the underlying sequential, asynchronous, and concurrent events that drive the process to be controlled. It is difficult to determine the original design specification/sequence of operation from an RLL program. For this reason, control software in RLL is difficult to understand and verify for its correctness and lacks the high degree of flexibility required for agile automation.

Petri net (PN) theory was developed in 1962 by C. A. Petri. PN's are a graph theoretic and a visually graphical tool designed for modeling, analysis, performance evaluation, and control of discrete event systems (DES's) [4], [22]. PN's are capable of modeling sequential, asynchronous, and concurrent events that drive an industrial process. Since they are a proven tool for modeling these systems, they should prove to be a valuable tool for controlling these processes [4], [18]. Several PN-based control design methods and laboratory demonstrations are reported in [1], [2], [6], [15], and [17]. The related CAD tools and industrial applications of PN-based control methods are reported in [7], [11], and [12]. Other related methods, including state-transition diagrams, state charts, rule-based methods, and GRAFCET or sequential function charts (SFC's), are discussed in [5], [8], [10], and [21].

We can develop a control strategy by using a PN model that captures the discrete event dynamics of the process being controlled. This should result in a control design that is easier to understand, troubleshoot, modify, and evaluate. Some previous studies have illustrated certain strengths of PN's over RLL via small-scale automated systems [1], [15]. None, however, use an industrial-scale system design example to illustrate both methods and discuss thoroughly the understandability and flexibility. The need to present more convincing results of both methods to both engineers and researchers motivates this present work. After a brief discussion of PLC's and real-time Petri nets, this paper presents the functional specification for a sequential industrial control problem. Then both RLL and PN are used to design control programs. They are compared to the two most important criteria, i.e., understandability, which relates to the ability to evaluate the programmed logic, verify its correctness, and maintain the control system as well as flexibility, which relates to the easy modification of logic when the specification changes.

II. PLC's

A. Brief History

In the late 1960's, electromechanical devices were the order of the day as far as industrial automated control was

Manuscript received March 17, 1996; revised October 1, 1996 and August 31, 1997. This work was supported by NSF under Grant DMI-940386. A portion of this paper was presented at *The 4th IEEE Conference on Control Applications*, Albany, NY, Sept. 1995, pp. 748–753.

M. C. Zhou is with Discrete Event Systems Laboratory, Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102-1982 USA (e-mail: zhou@njit.edu).

E. Twiss is with International Paper Company, Tuxedo, NY 10987 USA.

Publisher Item Identifier S 1094-6977(98)01533-8.

TABLE I
METHODS FOR DEVELOPING RLL PROGRAMS

METHOD	ADVANTAGES	DISADVANTAGES
Direct RLL Implementation	1. Easily used by engineers with hard-wired relay experience.	1. Fails to capture the system's discrete event dynamics, and difficult to determine initial design specification from RLL programs. 2. Unable to simulate without hardware. 3. Difficult to implement a hierarchical programming structure.
ISA Logic Diagrams S5.2-1976	1. A portable form of logic used to generate RLL code for different manufacturers' PLC's. 2. Graphical symbols familiar to most engineers. 3. Help to eliminate repetitive interlocks.	1-3. Same as direct RLL implementation.
Timing/ Sequence Diagrams	1. Capture the system's discrete event dynamics. 2. Easy to understand for a simple system. 3. Involve little learning curve. 4. Need only a complete understanding of the sequence of operation to develop such diagrams	1. For a system with many subsystems it is necessary to develop a sequence diagram for each one. It is thus difficult to determine interrelations among these different diagrams. 2. Difficult to determine the underlying logic.
State Diagrams	1. Offer a set of rules for developing RLL. 2. Capture the system's discrete event dynamics. 3. Familiar to most engineers developing sequential logic circuits.	1. Difficult to implement when more than four state variables are involved. 2. Difficult to model concurrent events.

concerned. These devices, relays, electromechanical timers, counters, and sequencers were being used by the thousands to control many sequential industrial processes and stand-alone machines. Installed in control cabinets, they used hundreds of wires and their interconnection to effect a control solution [3]. These control systems were able to sequence and synchronize the events required to manufacture a single product. However, if a change was made to the product, or a new product was introduced, the control solution had to be changed by physically rewiring the panel. This required downtime to make the needed wiring changes. If the amount of wiring changes was excessive, it was not uncommon to discard the entire panel in favor of a new one.

In the late 1960's, the Hydromatic division of General Motors Corp. wrote a specification for a new type of programmable controllers [3]. These controllers would eventually be programmed using RLL. The RLL programming language was developed to smooth the transition from relay control systems to PLC's. The initial intent of RLL was to allow plant maintenance personnel to troubleshoot, maintain, and make minor modifications to the system. PLC's greatly improved the flexibility of industrial control systems and reduced the downtime required to make modifications. However, the larger the control system, the more difficult it is to determine the initial design specifications (how the system operates) by examining the control logic. This makes troubleshooting these systems difficult.

Short RLL programs (less than 100 rungs) are not difficult to evaluate. However, as a system becomes more complex, the RLL program can easily grow to 1000 rungs or more. Adverse results of large RLL programs are as follows.

1) Complexity to validate and evaluate the ladder logic grows exponentially (in general).
2) It becomes more difficult to make program modifications to reflect changes in the system functional specification.
3) It becomes more difficult to troubleshoot and maintain the program.

B. Methods for Developing RLL

Several methods exist for developing RLL programs. However, no method has been universally accepted by industry. Table I is a comparison of different methods for developing RLL programs. For the detailed comparison on them see [21].

III. REAL-TIME PN (RTPN) CONTROL METHOD

A. Basic PN's

PN's have proven to be a useful tool for modeling, control, and performance evaluation of manufacturing systems. Their fundamental knowledge can be seen in [5], [9], and [22] and applications in manufacturing automation in [4] and [16]. A marked PN is a five tuple PN $= (\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{m_o})$ where

1) $\boldsymbol{P}$ is a finite set of places pictured by circles;
2) $\boldsymbol{T}$ is a finite set of transitions pictured by bars or boxes, with $\boldsymbol{P} \cup \boldsymbol{T} \neq \emptyset \; and \; \boldsymbol{P} \cap \boldsymbol{T} = \emptyset$;
3) $\boldsymbol{I}$: $\boldsymbol{P} \times \boldsymbol{T} \rightarrow \boldsymbol{N}$ is an input function that specifies arcs directed from places to transitions, where $\boldsymbol{N}$ is the set of all natural numbers;
4) $\boldsymbol{P} \times \boldsymbol{T} \rightarrow \boldsymbol{N}$ are output function that specifies arcs directed from transitions to places;
5) $\boldsymbol{m_o}$: $\boldsymbol{P} \rightarrow \boldsymbol{N}$ is an initial marking whose ith component represents the number of tokens pictured by dots in the ith place.

Places are used to represent conditions (true/false), resource availability, or process status, e.g., a machine is processing a part. Transitions are used for events, the start and end of activities. Places, transitions, input, and output functions that define arcs from places to transitions or transitions to places constitute the structure of a PN. Its state is defined by its marking. A marking is a $|\boldsymbol{P}|$-dimensional vector whose ith element represents the number of tokens in the ith place. Tokens travel from place to place along directed arcs. The flow of tokens is determined by the "firing" of transitions. In order to simulate the dynamic behavior of a system, a marking in a PN is changed according to defined "firing" rules. These firing rules are also referred to as the token player game [9]. They are defined as follows.

1) Transition t is said to be enabled if each input place p of t is marked with at least $\boldsymbol{I}(p, t)$ tokens, where $\boldsymbol{I}(p, t)$ is the weight of the arc from p to t. Mathematically, $t \in \boldsymbol{T}$ is *enabled* if $m(p) \geq \boldsymbol{I}(p, t), \forall p \in \boldsymbol{P}$.
2) Enabled transition may or may not fire (depending on whether the event takes place).
3) Firing of an enabled transition t removes $\boldsymbol{I}(p, t)$ tokens from each input place p of t and adds $\boldsymbol{O}(p, t)$ tokens to each output place p of t, where $\boldsymbol{O}(p, t)$ is the weight of the arc from t to p. Mathematically, *t fires* at marking m', yielding the new marking $m(p) = m'(p) + \boldsymbol{O}(p, t) - \boldsymbol{I}(p, t), \ \forall p \in \boldsymbol{P}$. Marking m is said to be *reachable* from m'.

The *reachability set* is the set of all markings reachable from $\boldsymbol{m_o}$ by a sequence of transition firings, denoted by $\boldsymbol{R}$. A place $p \in \boldsymbol{P}$ is *k-bounded* if $\exists k > 0, \ni m(p) \leq k, \ \forall m \in \boldsymbol{R}$. A PN is *k-bounded* if each place is *k-bounded*. It is *safe* if it is *one-bounded*. It is *live* if $\exists$ a fireable sequence whose firing results in a marking that enables $t \, \forall t \in \boldsymbol{T}$, and $m \in \boldsymbol{R}$. The significance of the PN properties in manufacturing is discussed in [18]. Briefly, boundedness and liveness guarantee the stability and freedom from deadlock of a DES, respectively.

Deterministic time delays are often associated with transitions and/or places in PN's that model industrial automated systems. Random time variables can also be associated to a PN. These models can be used to evaluate system performance, such as throughput and resource utilization [4], [9], [19].

B. RTPN-Based Controller

An RTPN-based controller can be developed by assigning physical output functions to places and assigning physical input and timing variables to transitions of the PN model. Formally, the RTPN could be defined as follows [15]: an RTPN is eight tuple and defined as RTPN $= \{\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{m_o}, \boldsymbol{D}, \boldsymbol{Y}, \boldsymbol{Z}\}$, where

- $\{\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{I}, \boldsymbol{O}, \boldsymbol{m_o}\}$ is as defined before;
- $\boldsymbol{D}$: $\boldsymbol{P} \rightarrow \mathbb{R}^+$ is a firing time-delay function, where $\mathbb{R}^+$ is the set of nonnegative real numbers;
- $\boldsymbol{Y}$: $\boldsymbol{P} \rightarrow \mathbb{B}$ is an input signal function, where $\mathbb{B}$ is the set of Boolean expressions of input addresses;
- $\boldsymbol{Z}$: $\boldsymbol{P} \rightarrow \{0, 1\}$ is a physical output signal function.

1) Timing vector $\boldsymbol{D}$ assigns time delays to transitions. It models the delays and synchronization of activities in the system.
2) Vector $\boldsymbol{Y}$ is used as an enable signal and determines when a transition is fired. It can be mapped to a single input address or Boolean expressions of input addresses. When the function associated with $\boldsymbol{Y}(i)$ is true and all input places are marked, the firing rule is executed. $\boldsymbol{Y}$ is the firing conditions of all transitions in an RTPN. Two basic types of transitions are as follows.
   a) Immediate—Represented as solid bars, these would always have a firing attribute of one and zero time delay.
   b) Input—Represented by hollow bars, these transitions would fire when all input places are marked, the firing condition is true, and the time delay d_i has expired. Transition t_i can have the form: $\boldsymbol{Y}(i) = \boldsymbol{I}\#$ or a Boolean expression, e.g., $\boldsymbol{I}1 * \boldsymbol{I}2 + \boldsymbol{I}3$ and $\boldsymbol{D}(i) = x$ s or min, $\boldsymbol{I}\#$ is an input address often represented by a tag name, and x is a preset time delay.

   $\boldsymbol{Y}(i)$ and $\boldsymbol{D}(i)$ together are the firing attribute of t_i.
3) Vector $\boldsymbol{Z}$ sends commands to the digital output interface. When p_i is marked by a token, some output function occurs (e.g., start motor). Place p_i would only be allowed to write to a single element z_i of vector $\boldsymbol{Z}$, thus, p_i would only be allowed to set or reset a single output.

It should be noted that in [15], input signal functions are associated with places and output signals with transitions. Our practice of PN modeling of industrial systems suggests that it is easier to associate input signal functions with transitions and output signals with places. This latter approach was also adopted in [11] and [17]. Other approaches and CAD tools to design logic controllers by using PN are presented in [6], [7], and [12].

An emergency stop situation normally requires the process to be halted (all outputs reset) and for the system to return to an initial or last state upon the alarm condition being manually or automatically reset. To handle different emergency stop schemes and simplify modeling, special e-stop places are used. These places have an additional attribute that disables all designated outputs when the e-stop place is marked. Once the e-stop place is marked, all tokens are removed from other places. When it loses its token, either the last or initial marking is restored, based on the type of restart required. This place can also be used to execute a fault routine, i.e., a safe shutdown or alarm annunciation routine.

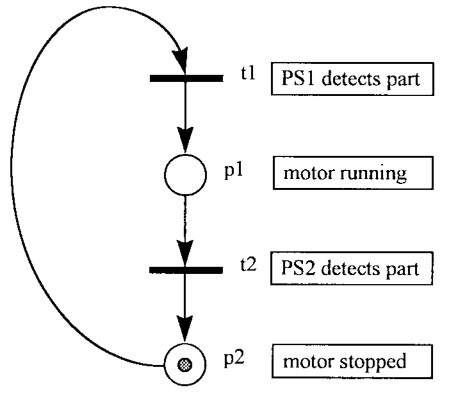
C. RTPN Model of Conveyor Start/Stop

Following is an example of how an RTPN can be used to model a conveyor system. When parts are detected by photo switch (PS1) in the conveyor receiving area, the conveyor motor is commanded to start. When the delivery area PS2 detects a part, the conveyor is commanded to stop. This DES can be broken down into the following events.

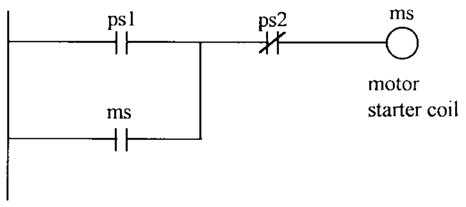
1) Parts are detected by the receiving area PS.
2) The motor is commanded to start.
3) Parts are detected by the delivery area PS.
4) The motor is commanded to stop.

This system can be modeled using two places and two transitions. A place is required for each system state, i.e., conveyor running/conveyor stopped.

- p_1 sets the motor run output, i.e., $z_1 = 1$.
- p_2 resets the motor run output, i.e., $z_1 = 0$.
- t_1 has a firing attribute of PS1 and delay $= 0$, i.e., $\boldsymbol{Y}(1) =$ PS1, and $\boldsymbol{D}(1) = 0$.



(a)



(b)

Fig. 1. (a) RTPN model of conveyor system and (b) RLL model of conveyor system.

- t_2 has a firing attribute of PS2 and delay $= 0$, i.e., $\boldsymbol{Y}(2) =$ PS2 and $\boldsymbol{D}(2) = 0$.

An RTPN model of the system is shown in Fig. 1(a). The initial marking is $(0, 1)^{\boldsymbol{T}}$, i.e., p_1 is not marked and p_2 is marked. Transition t_1 fires when p_2 is marked and PS1 detects a part. Place p_1 starts the conveyor motor by setting output signal z_1 on the digital output interface. Transition t_2 fires when the motor is running (p_1 marked) and a part is detected by the delivery PS2. When the token is removed from p_1, the motor is stopped by resetting output signal z_1. The system is now at its initial state and waits for another part to become available. The RLL equivalent to the RTPN appears in Fig. 1(b).

IV. RLL and PN Design for an Industrial System

A. Functional Specification of an Industrial System

This functional description is based on a process used in water pollution treatment facilities. The control system is for a secondary clarifier scum removal system that is manufactured by Envirex Inc., Waukeshaw, WI and installed in the Deer Island Water Pollution Treatment Facility near Boston, MA. The system, as shown in Figs. 2 and 3, consists of nine clarifier channels, nine skimmer pipes (one for each channel), and two scum boxes. Wastewater flows through the clarifiers toward the skimmer pipe. As shown in Fig. 3, the skimmer pipes are sloped in groups of three toward a scum box. The skimmers are rotated on a sequential basis to allow the froth on the surface of the water to be collected in a scum box.

Skimmer Pipes: Each skimmer pipe has four positions: reverse, shallow skim, deep skim, and rest in terms of water flow through clarifiers. Fig. 3 shows detail about the pipes and clarifiers #4–6. Water flows from #6 to #4. Each pipe is equipped with a motorized actuator and position limit switches. A local control panel is provided with a local/off/remote selector switch. In the local mode, a pipe is operated by using a switch located on its control panel. The local controls are provided for maintenance and emergency handling purposes. In remote mode, the pipes are controlled by a PLC. Mode selections (shallow/deep skim or timer/continuous) in remote operation are determined by inputs to the PLC from a plant-wide Distributed Control System.

Spray Water Valve: Each skimmer pipe is equipped with a spray water solenoid valve. Each valve has a local/off/remote selector switch at the local control panel. In local mode, the valve is operated by local open/close switches. In remote mode, it is controlled by the PLC.

Scum Flushing Valve: A scum flushing valve is provided to flush the scum boxes at the completion of a skimming cycle. It is equipped with open/close limit switches and a torque switch. It is also provided with a local/off/remote selector switch at a local control panel. Local/remote operation is similar to that of a skimmer pipe.

Remote Sequence of Operation: The skimmer pipes are to be provided with a sequential controller to rotate each skimmer on a timed or continuous basis. In the timed mode, the sequence completes one cycle every 6–24 h (adjustable). In the continuous mode, the sequence starts and repeats until it is no longer selected. In both modes of operation, the skimming sequence is the same. It starts with skimmer #9 and finishes with #1 in a decreasing numerical order. The sequencing within each skimmer is as follows.

1) Rest position to reverse position for 10–60 s (adjustable).
2) Reverse to shallow or deep skim position for 10–120 s (adjustable).
3) Step to the next skimmer.
4) Spray water valve opens during the skimming sequence of the associated skimmer pipe.
5) Scum flushing valve opens for 60 s (adjustable) at the completion of the skimming sequence.

The following interlocks affect the skimming cycle.

1) It is interrupted on a high-wet-well level. The cycle is reinitialized at the interrupted skimmer on low-wet-well level.
2) It is halted and an alarm is generated on a skimmer malfunction; it is resumed at the next skimmer on activation of resume push button. A skimmer malfunction is defined as a failure to reach an operational mode (rest, reverse, shallow, or deep) within a specified time.
3) If the local/off/remote selector switch of a particular skimmer is not in the remote position, then resume skimming at the next skimmer.

B. RLL Design

The RLL program was developed with the assistance of ISA logic diagrams whose samples appear in [21]. The RLL program was created by entering one RLL file for skimmer #9. This file was tested with a test stand physically connected to the PLC I/O modules. The test stand consisted of maintained and momentary switches to simulate input

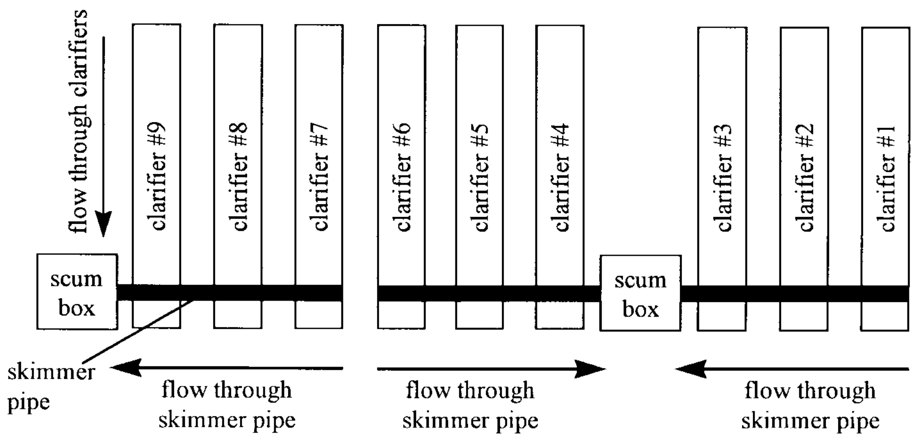


Fig. 2. Top view of clarifiers.

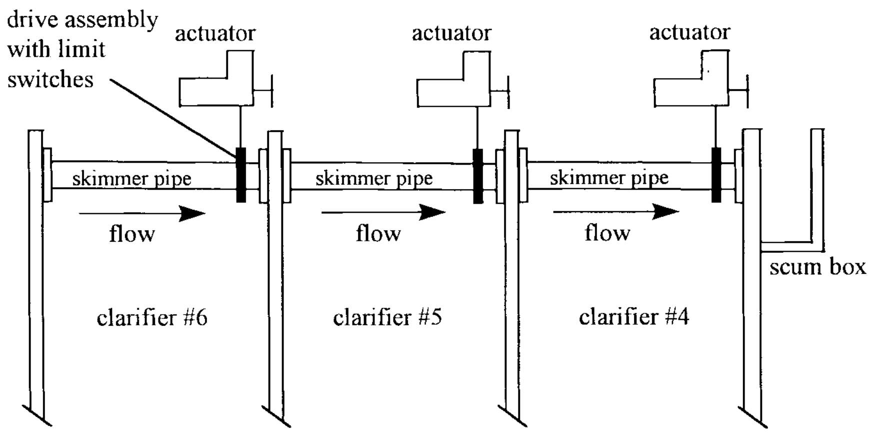


Fig. 3. Detail of skimmers 4–6.

conditions and lights to indicate output signals. After the first program file was completely tested, it was copied and stored in a library file. The addresses I/O and internal status bits of the library were then *searched* and *replaced* with the addresses for skimmers 8–1. When *search* and *replace* were completed, the files were inserted into the RLL program with the logic for skimmer #9. To index from one skimmer subroutine to another, a sequential function chart (SFC) similar to PN was used [5], [8]. SFC's consist of steps (similar to PN places) and transitions. An SFC repeatedly scans a step (program file subroutine) until its output transition(s) become true. Once the transition(s) become true, all output conditions in the step are reset and the next enabled step (program file subroutine) is scanned. After the SFC was entered, the system was again tested by using the test stand. The logic for the scum valves and spray water valves was entered, and a final test was then performed. A portion of the ladder logic as shown in Fig. 4(a)–(d) is described as follows.

6–24-h Timer: It is defined by Rungs #1–3. The minute timer on rung #1 is a 1-min self-resetting timer, whose done-bit pulses and increments the minute counter on rung #2. The counter has a maximum preset value of 32 767 min. It resets when rung #3 is executed.

Timer Mode: Rung #4 output instruction latches when the minute counter done bit is set. The auto-start timer latch bit remains set until skimmer #1 completes its skim cycle or timer mode is no longer selected. This instruction is used to remember the system being initiated in timer mode and to automatically restart skimming after a high-to-low wet-well transition on rung #5. Rung #5 (auto-start timer) is a "one shot" output instruction. It only enables the post-conditions for one program scan while the preconditions are set. This bit is set by either the minute counter done bit "or" a high-to-low wet-well transition when the system is in timer mode. The logic for the high-wet-well level latch and low-wet-well reset bit is not shown.

Continuous Mode: Rung #6 (auto-start continuous) is also a "one shot." This bit is initially set when the resume push button is depressed. It is then repeatedly set by the previous cycle complete bit. It is also set on high-to-low wet-well transitions, while continuous mode is selected. The logic in dotted boxes will be explained later.

Step to Next Skimmer: Rung #7 enables the next skimmer when

1) skimmer fault occurs and the "resume" push button is hit;
2) skimmer is in rest mode and the rest limit switch is activated;
3) remote mode is not selected and the skimmer is in any cycle.

Auto Start: Rung #8 (auto start) is a "one shot" instruction. The bit is set when either the timer or continuous auto-start bits are set and remote mode is selected.

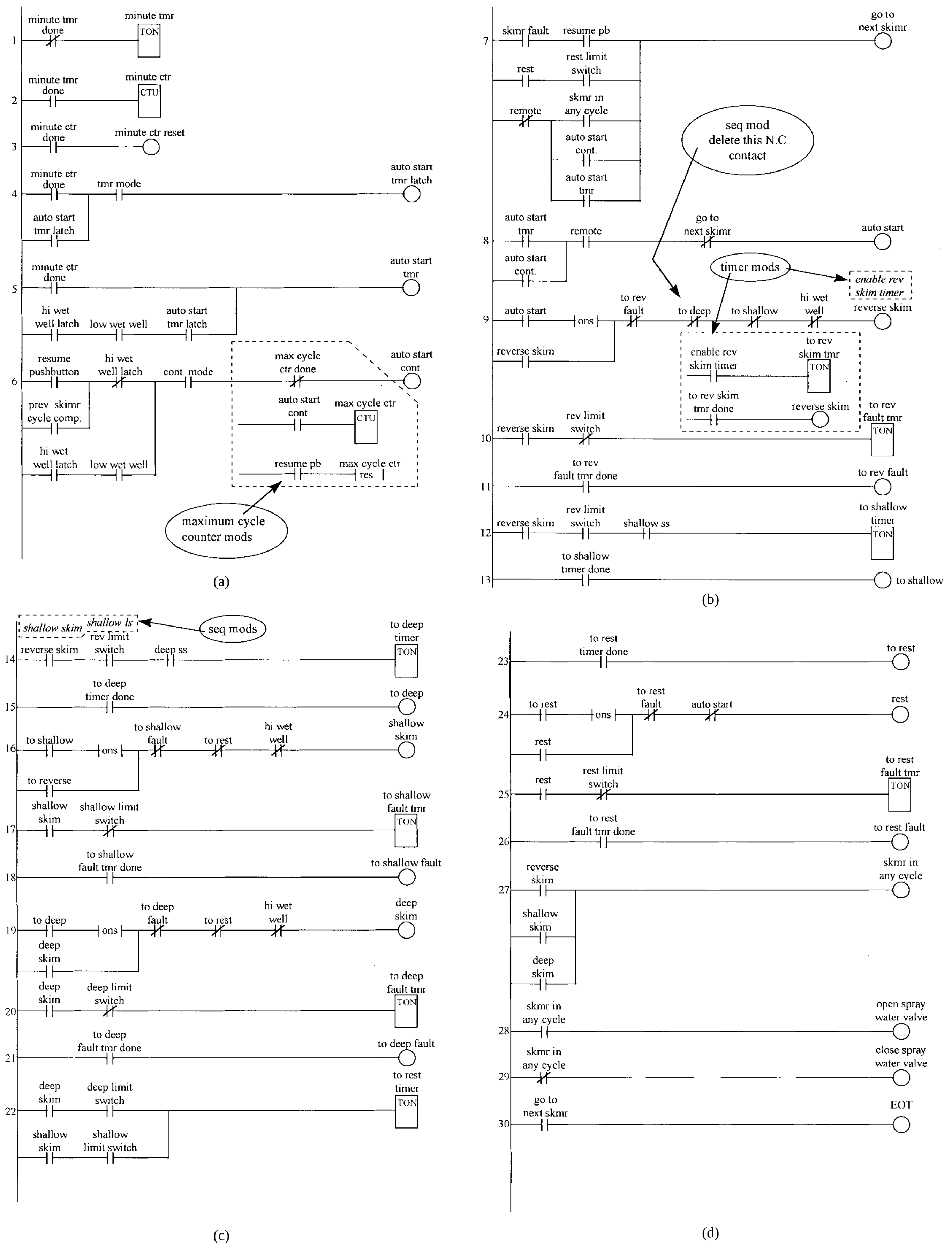


Fig. 4. (a)–(d) Typical skimmer RLL's.

Reverse Skim: Rung #9 (reverse skim) is set and latched when the auto-start bit is set. This bit remains set until any of the following occurs

1) actuator fault goes to the reverse position;
2) deep skim mode is initiated;
3) shallow skim mode is initiated;
4) high-wet-well level is reached.

Once in reverse skim mode, an actuator fault timer is started (rung #10). If the input for the reverse limit switch does not become true within a preset time, an actuator fault is initiated (rung #11). In this skim mode, the PLC selects to start one of two timers, depending on the position of the deep/shallow selector switch. When the selected timer expires, the reverse skim mode is terminated and shallow or deep skim is initiated.

Deep or Shallow Skim: Deep or shallow skim cycles are identical to the reverse skim cycle, as shown on rungs #12, 13, and 16–18, and rungs #14, 15, and 19–21, respectively. When the respective timer expires, the deep or shallow skim mode is terminated and the skimmer returns to the rest position.

Rest Position: Once the skimmer is commanded to go to the rest position and the rest limit switch is actuated, the next skimmer is enabled (rung #7). If the rest limit switch is not actuated within a preset time, an actuator fault occurs.

Spray Water Valve: Rung #27 determines when the skimmer is in any cycle. A normally open (NO) contact on rung #27 opens the spray water valve (rung #28). A normally closed (NC) contact closes it (rung #29).

End of Transition: Rung #30 disables this program file and steps to the next skimmer file in the SFC.

Summary of Ladder Logic Listing: As can be seen from Fig. 4(a)–(d), it is fairly simple to determine the output of an individual rung. However, to determine the complete sequence of operation from the ladder logic takes a concentrated effort. The difficulty in determining the sequence of operation is due to the fact that the ladder logic listing fails to capture the discrete event dynamics of the system.

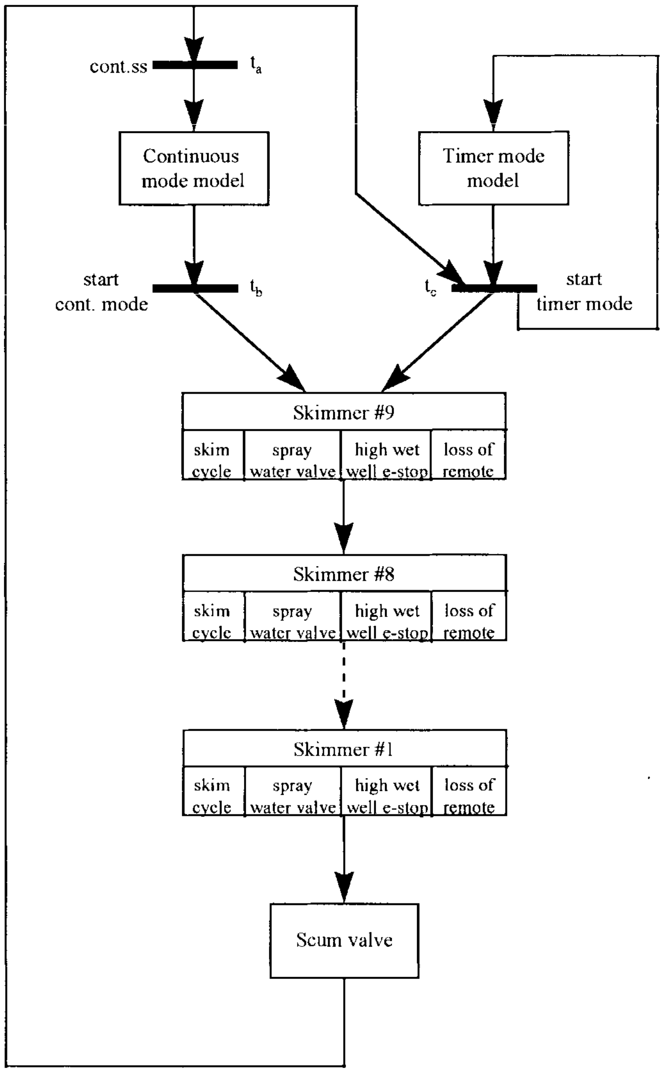


Fig. 5. First-level PN model.

C. RTPN Design

1) Top-Level Design: A top-down approach is used in the RTPN design, whose process is briefly described. The theory and simple examples on top-down PN synthesis are discussed in [13], [14], and [18], and a laboratory example can be seen in [19]. The first level of this design is shown in Fig. 5, which is a cross between a PN and a flow chart. It models the sequencing of major elements of the system, each of which is represented by a block. They are models of continuous and timer mode of operation, nine skimmers, and a scum valve. The model of a skimmer is further broken down into four submodels, skimmer cycle, spray water valve, high-wet-well e-stop, and loss of remote.

The system can be initiated in either continuous or timer mode. When the former is selected, transition t_a fires and enables the continuous mode model. t_b of the continuous mode model enables skimmers #9–1 in a decreasing numerical order. t_a has a firing attribute ($\boldsymbol{Y}(a) =$ continuous: $\boldsymbol{D}(a) = 0$) and t_b is an immediate one, which fires once all the conditions are met. Once skimmer #1 completes its operation, the scum valve is enabled. At the completion of the scum flushing valve operation, the cycle repeats until the continuous mode is no longer selected. When timer mode is selected, t_c fires once every 6–12 h after completion of the scum valve operation until it is no longer selected. $\boldsymbol{Y}(c) =$ timer mode: $\boldsymbol{D}(c) =$ 6–12 h.

2) Second-Level Design: In the second-level design (Fig. 6), the model for skimmer #9 is expanded and details of continuous and timer models are shown. All places and transitions are summarized in the Appendix.

Skimmer #9 Model Expansion: The model for skimmer #9 is expanded to show the relationship among its four different components. Three of them (skimmer sequence/high-wet-well/spray water valve) are synchronized by t_1 (initiate skim cycle) and t_{16} (step to next skimmer). When p_1 (Skimmer #9 start request) is marked by either continuous or timer mode of operation, immediate transition t_1 fires, passing tokens to the models of skimmer sequence, high-wet-well e-stop, and spray water valve. If the remote input from a skimmer in cycle is lost, the skimmer stops and the next available skimmer

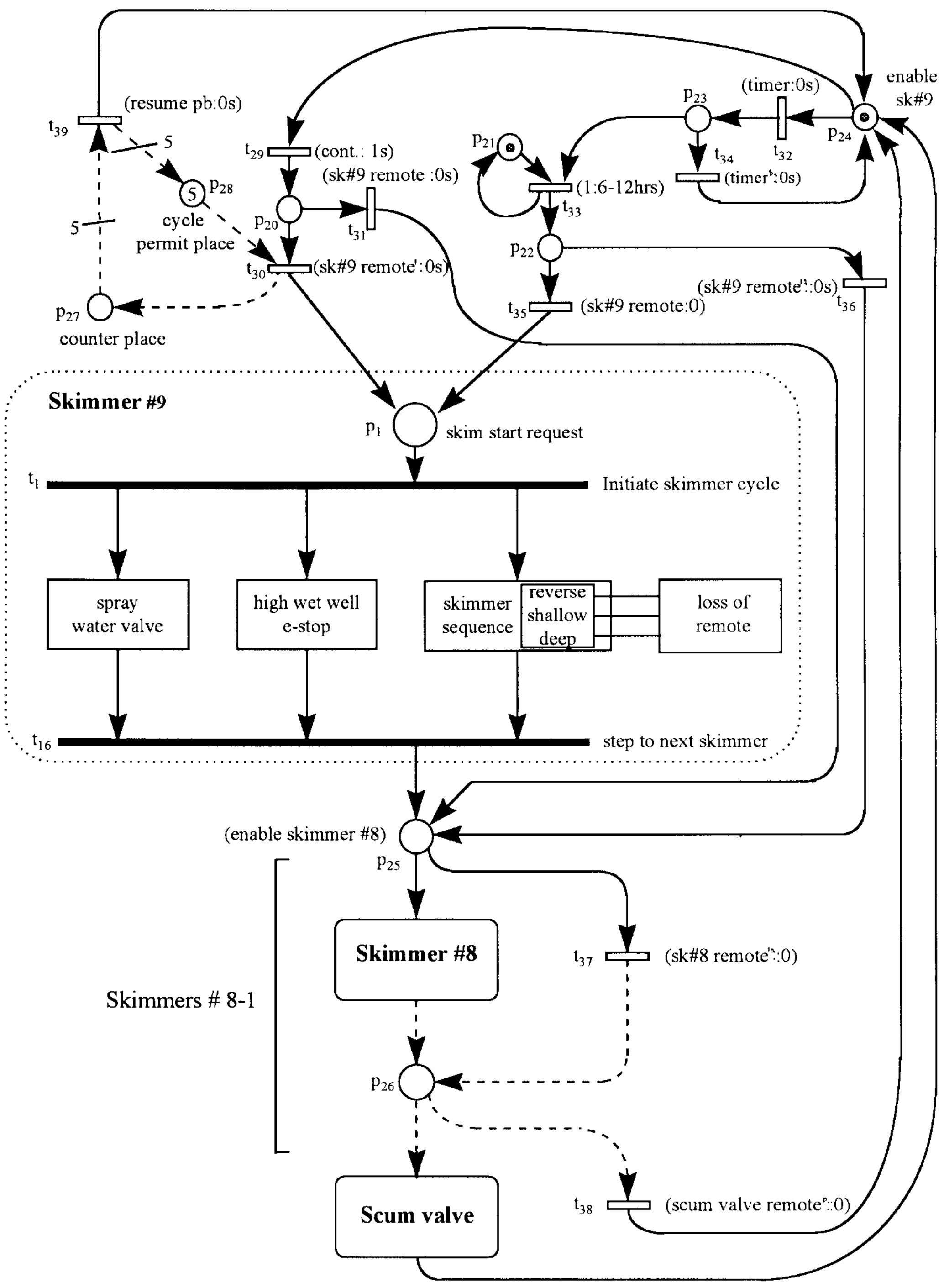


Fig. 6. Second-level PN model (with details of timer and continuous models).

(skimmer with remote selected) is enabled. Thus, the loss of remote model is shown with connections to the box of reverse, shallow, and deep skims.

Continuous Mode Modeling: When the continuous mode is selected, t_{29} (initiate continuous operation) fires. When $\boldsymbol{Y}(29)$ = skimmer #9 continuous mode is selected: $\boldsymbol{D}(29) = 0$. Its firing removes a token from p_{24} (enable skimmer #9) and deposits one to p_{20} (continuous mode start request), where p_{24} is initially marked. Then t_{30} (continuous mode enable skimmer #9) and t_{31} (continuous mode bypass skimmer #9) are enabled. When $\boldsymbol{Y}(30)$ = skimmer #9 remote mode is selected: $\boldsymbol{Y}(31)$ = skimmer #9 "not" in remote and $\boldsymbol{D}(30) = \boldsymbol{D}(31) = 0$. If skimmer #9 is selected for remote operation, t_{30} fires and passes a token to p_1. Otherwise, skimmer #9 is bypassed and the skimming sequence starts at skimmer #8 through t_{31}'s firing. When skimmer #9 has completed its cycle, skimmer #8 is enabled by marking p_{25} (enable skimmer #8). If selected for remote operation, skimmer #8 starts its sequence. Otherwise,

t_{37} (continuous mode bypass skimmer #8) fires to bypass it. Once skimmers #8–1 and the scum valve have completed their sequences, or have been bypassed, p_{24} (enable skimmer #9) is again marked. The cycle repeats until the continuous mode is no longer selected. The dotted portion in the upper corner of Fig. 6 is for design modifications and will be explained later.

Timer Mode Modeling: When the timer mode is selected, t_{32} (initiate timer operation) fires. Its firing removes a token from p_{24} and deposits one into p_{23} (timer mode selected check). Then t_{33} (initiate timer mode delay) and t_{34} (end timer mode) are enabled. When $\boldsymbol{Y}(33) = 1, \boldsymbol{D}(33) = 612h, \text{and} \boldsymbol{Y}(34) =$ skimmer #9 timer mode are "not" selected: $\boldsymbol{D}(34) = 0$. Thus, if p_{23} is marked for 6–12 h, t_{33} fires and passes a token to p_{22} (timer mode start request). If timer mode is deselected while p_{23} is marked, t_{34} fires and p_{24} is remarked. When p_{22} is marked, t_{35} (timer mode enable skimmer #9) and t_{36} (timer mode bypass skimmer #9) are enabled. Therefore, if skimmer #9 is selected for remote operation, t_{35} fires and passes a token to p_1 of skimmer #9. Otherwise, it is bypassed and the skimming sequence starts at skimmer #8. The remaining is similar to the continuous mode of operation.

3) RTPN Model of Skimmer: The RTPN model of a skimmer is composed of four components: the skimmer sequence, the high-wet-well e-stop, spray water valve, and loss of remote. t_1 is used to initialize the first three subnets, and t_{16} to synchronize them at the end of the cycle. In Fig. 7, p_1 receives a token after the continuous or timer mode is selected, as shown in Fig. 6, and then t_1 is enabled. Firing t_1 marks p_2 (moving to reverse position), p_{12} (enable spray water valve), and p_{16} (enable high-wet-well e-stop).

PN of Skimmer Sequence: Marking p_2 sets the output address that drives the skimmer to the reverse position, i.e., $\boldsymbol{Z}(2) =$ skimmer to reverse. Both t_2 (enter reverse skim mode) and t_{14} (enter skimmer-to-reverse fault mode) are enabled. $\boldsymbol{Y}(2) =$ the reverse limit switch, and $\boldsymbol{D}(2) = 0$; and $\boldsymbol{Y}(14) =$ "not" reverse skim limit switch, and $\boldsymbol{D}(14) = 45$ s. If t_2 fires, a token is removed from p_2 and deposited into p_3 (skimmer #9 in reverse skim mode), resetting $\boldsymbol{Z}(2)$. If the reverse limit switch is not actuated before t_{14}'s time delay expires, t_{14} fires and marks p_{10} (skimmer #9 fault). This fault loop is typical for deep, shallow, and rest actuator faults. Marking p_3 sets the output for the reverse skim indicator [$\boldsymbol{Z}(3) =$ reverse skim indicator]. Then t_3 (reverse skim duration 1) and t_6 (reverse skim duration 2) are enabled. $\boldsymbol{Y}(3) = \boldsymbol{Y}(6) = 1, \boldsymbol{D}(3) = 1030$ s, and $\boldsymbol{D}(6) = 1060$ s. Depending on the mode selected (shallow or deep), the skimmer stays in reverse until the delay associated with t_3 or t_6 expires. It then enters shallow or deep skim mode. When p_4 (skimmer #9 moving to shallow position) or p_6 (skimmer #9 moving to deep position) is marked, its corresponding output address that drives the skimmer to the deep or shallow position is set, i.e., $\boldsymbol{Z}(4) =$ skimmer to shallow or $\boldsymbol{Z}(6) =$ skimmer to deep. Then t_4 (enter shallow skim mode) and t_{12} (enter skimmer to shallow fault mode) or t_7 (enter deep skim mode) and t_{13} (enter skimmer to deep fault mode) are enabled. When $t_4 (t_7)$ fires, a token is removed from p_4 (p_6) and deposited into p_5 p_7 [skimmer #9 in shallow (deep) skim mode], thus setting $\boldsymbol{Z}(5)$ [$\boldsymbol{Z}(7)$], i.e., shallow (deep) skim indicator. Similar to the reverse skim mode, if the skimmer does not reach the desired position within a preset delay of t_{12} or t_{13}, t_{12} or t_{13} fires and p_{10} is marked. When p_5 or p_7 is marked, the system stays in the deep or shallow skim mode until the delay associated with t_5 (shallow skim duration, $\boldsymbol{Y}(5) = 1$, and $\boldsymbol{D}(5) = 10120$ s.) or t_8 (deep skim duration $\boldsymbol{Y}(8) = 1$ and $\boldsymbol{D}(8) = 10120$ s.) expires. Two paths (shallow and deep) converge at p_8 (skimming complete). Transition t_9 (end skimming) is an immediate one and fires to mark p_9 (skimmer to rest) once p_8 is marked. Thus, the output address that drives the skimmer to the rest position is set, i.e., $\boldsymbol{Z}(9) =$ skimmer to rest. Both t_{10} (enter rest mode) and t_{11} (enter skimmer to rest fault mode) are enabled. Firing t_{10} removes a token from p_9 and deposits one to p_{11} (skimmer #9 cycle complete). If the skimmer does not reach the desired position within a preset delay of t_{11}, t_{11} fires and p_{10} is marked. Hence, if any actuator fault occurs during the above skim cycle, p_{10} is marked. To reset the fault and continue skimming at the next skimmer, an operator must hit the resume push button. In the RTPN, t_{15} (resume normal operation) fires when the resume push button is hit, thereby removing a token from p_{10} and depositing one to p_{11}.

Spray Water Valve Modeling: When p_{12} (enable spray water valve) is marked, t_{17} (enter spray water valve open status) and t_{20} (enter spray water valve loss of remote status) are enabled. Note that p_{14} (spray water valve closed) is initially marked. If the valve is not selected for remote operation, then t_{20} fires to mark p_{15} (spray water valve loss of remote). Otherwise, t_{17} fires and p_{13} (spray water valve open) is marked, which sets the spray water valve open output. Both t_{18} (close spray water valve) and t_{19} (enter spray water valve loss of remote status) are then enabled. Immediate transition t_{18} fires when p_{11} is also marked. When t_{18} fires, a token is removed and redeposited into p_{11} by the bidirectional arc, removed from p_{13}, and deposited to p_{14}, respectively. Marking p_{14} sets the spray water valve close output. If the remote mode of operation is deselected, then t_{19} fires, removing a token from p_{13} and depositing one into p_{14} and p_{15}, respectively. This allows an operator to take local control of the valve while the automatic skimming cycle continues.

Modeling of High-Wet-Well E-Stop: When p_{17} (high-wet-well level e-stop place) is marked, due to the firing of t_{21} (initiate high-wet-well e-stop), skimmer operation is disabled by removing all tokens from all places. t_{21} fires when a particular skimmer is in any cycle and a high-wet-well condition occurs. Places p_{18} and p_{19} represent high- and low-wet-well levels, respectively. The condition is automatically reset on a low-wet-well level. t_{23} (low-wet-well level switch debounce) and t_{24} (high-wet-well level switch debounce) are used to debounce the high- and low-level switches. $\boldsymbol{Y}(23) =$ low-wet-well level switch, $\boldsymbol{Y}(24) =$ high-wet-well level switch, and $\boldsymbol{D}(23) = \boldsymbol{D}(24) = 3$ s. When the high-wet-well condition is reset, t_{22} (reset high-wet-well e-stop) fires and the token is removed from the e-stop place p_{17}. When it loses its token, then the initial marking is returned to the particular skimmer where the high-wet-well condition occurred. The

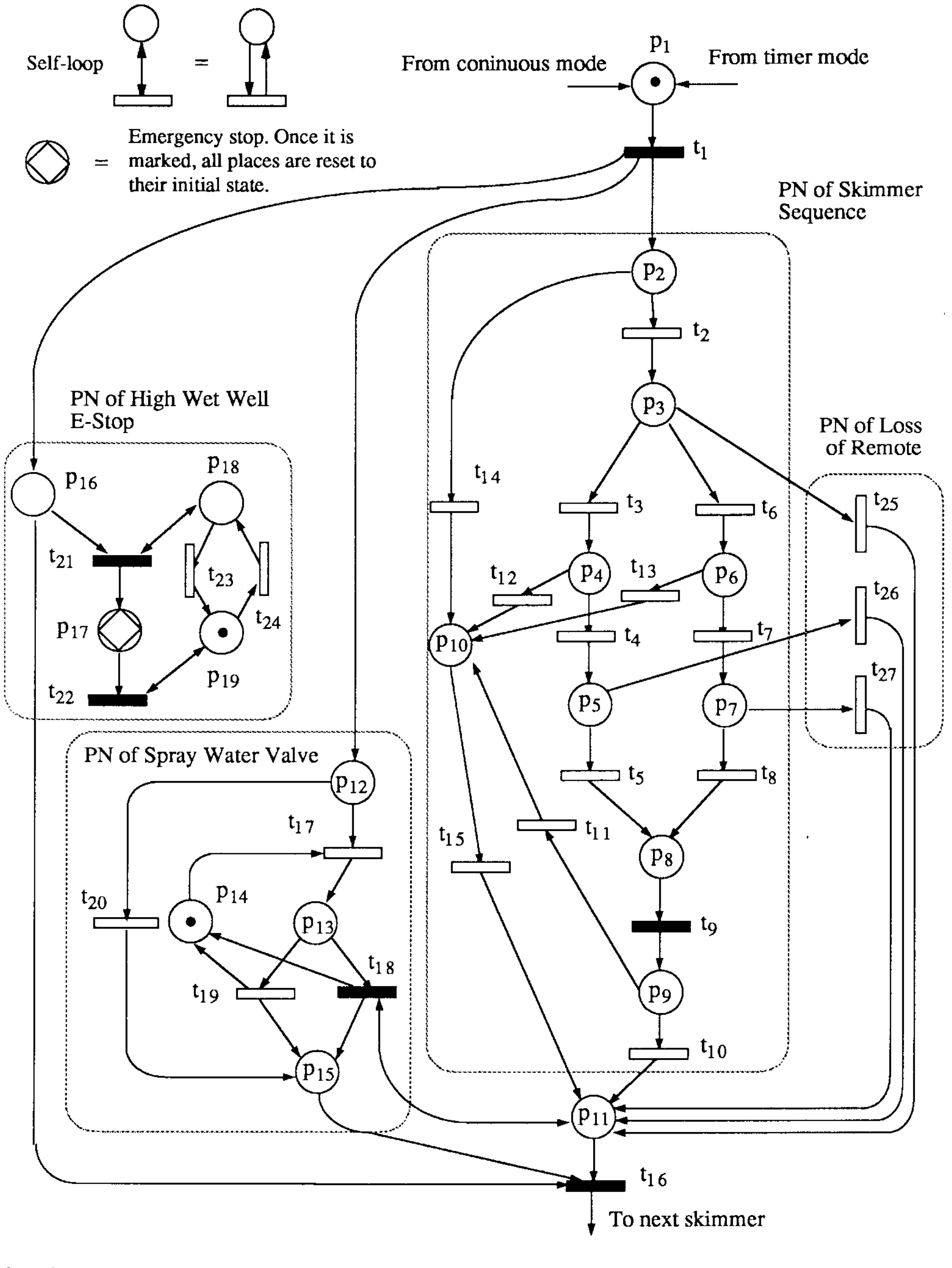


Fig. 7. PN model of a skimmer.

initial marking for skimmer #9 marks p_1, p_{14}, and p_{19} each with one token. The skimming cycle is then reinitialized and continues from this point.

Modeling of Loss of Remote: If the skimmer is in a cycle (reverse/shallow/deep) and whenever the remote input is lost, p_{11} (skimmer #9 cycle complete) is marked by firing one of t_{25}, t_{26}, and t_{27}.

4) Scum Valve Model: The RTPN model of the scum valve is shown in Fig. 8. When skimmer #1 completes its skim cycle, p_{31} (open scum valve) receives a token and sets the output that opens the scum valve, i.e., $\boldsymbol{Z}(31) =$ open scum valve. Both t_{41} (enter scum valve open status) and t_{44} (enter scum valve open fault mode) are enabled. When the scum valve open limit switch is actuated, t_{41} fires, thus marking p_{32} (scum valve open). Place p_{32} indicates that the scum valve is open. It remains marked until the delay associated with t_{42} (scum valve open duration) expires. $\boldsymbol{Y}(42) = 1$ and $\boldsymbol{D}(42) = 30120$ s. Firing t_{42} marks p_{33} (close scum valve), which sets the output that closes the scum valve. When the scum valve closes, t_{43} (enter scum valve closed status) fires and passes a token to p_{24}. Scum valve actuator fault is initiated when the valve is commanded to open or close and does not reach its full open or closed position within a predetermined time or one of t_{44} (enter scum valve open fault mode) and t_{45} (enter scum valve close fault mode) fires. To reset the fault condition, the resume push button must be depressed, and t_{46} fires.

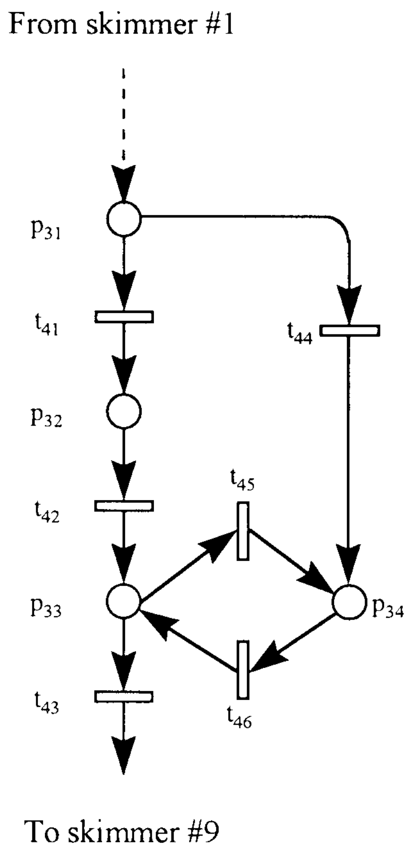


Fig. 8. PN model of scum valve.

V. Design Comparison

A. Understandability and Correctness Verification

The RLL program listing produces approximately 450 rungs of code, or 90 pages [21], while the RTPN program produces four PN designs (Figs. 5–8) plus a table to interpret the meaning of places and transitions. RTPN's make it easier to implement a hierarchical structure. This results in a more compact design algorithm. We could argue that the ladder logic program consists of nine subroutines. The only difference between these subroutines is the addresses and tag numbers. Therefore, once one subroutine is understood, all are understood. Each RLL subroutine is approximately 30 rungs, or seven pages, roughly equivalent to the RTPN in Fig. 7. As mentioned earlier, the RLL program was written with the assistance of a supervisory SFC. The SFC simplifies the sequencing of each RLL subroutine. However, when many subroutines are used in a pure RLL file, it becomes difficult to determine how and in what order they are executed. This is because RLL is similar to a line-number-oriented programming language. Every rung is scanned in sequential order whether the conditions for the rung output instruction are true or false (jump to subroutine is considered a rung output instruction). In an RTPN, only sections of logic, which are enabled, are executed. The order of execution is determined by the structure of the RTPN itself. A simple way to implement RTPN is to select an enabled transition to fire at one time. Then the set of enabled transitions are updated whenever there is a transition firing, an operation finished, or a marking changed. In conclusion, RLL fails to capture the time/event dependency of the different events taking place in the system. RTPN, however, explicitly models the skimmer sequencing and related concurrent operations. It exhibits much better understandability than RLL.

The correctness of the RLL program is checked, even though the simulation that requires connection to the PLC I/O modules takes a lot of time. Even though, it is almost impossible to exhaust all the input combinations and states of this industrial system. In other words, the developed RLL program is not guaranteed for its correctness. The properties of the PN can be guaranteed by using the top-down synthesis approaches, whose theory has been established in [13], [14], and [16] and applied to several small-scale manufacturing systems in [18] and [19]. At each refinement step, the properties, such as boundedness and liveness, are preserved by verifying the correctness of the newly added detail and its connection with the existing one. These properties, in turn, are used to guarantee the correctness of the developed program. Furthermore, the augmentation of the net model can also be conveniently checked for the modifications to accommodate the specification change [16]. Due to the space limitation, this verification process for the correct sequences, properties, and performance by using PN's is not included in this paper. This feature, however, is very important in designing discrete event controllers for many applications.

B. Flexibility

To determine the flexibility of each algorithm, the following changes are made to the original design specifications for the wastewater treatment system described below.

- Time delay is to be inserted before a skimmer begins the reverse skim operation.
- Continuous mode is to be modified to allow a maximum of five complete skimming cycles. A resume push button has to be pressed before another run starts.
- Typical skimmer sequence is to be modified to eliminate the choice structure of going to deep or shallow skim after reverse skim. The new cycle shall be: rest to reverse, reverse to shallow, shallow to deep, deep to rest, all with adjustable time delays as before.

Time Delay: The insertion of a time delay before the reverse skim operation begins has no effect on the structure of the RTPN. Transition t_1 in Fig. 7 is changed from an immediate to a timed one. The insertion of a time delay in the RLL program requires an additional rung to include the timer function, an additional rung to output instruction, and modification to existing rungs of ladder logic. The original output instruction is replaced by the new output instruction. NO contact of the new output instruction is used to enable the new timer. The done bit of the timer is then used to set the original output instruction that starts the reverse skim mode operation. The RLL modifications are shown in Fig. 4(b).

Maximum Cycle Counter: The insertion of a counter requires modification to the basic structures of both designs.

The RTPN requires two additional places p_{27} (counter) and p_{28} (limit place with five tokens initially), one additional transition t_{39} with [$\boldsymbol{Y}(39) =$ resume push button: $\boldsymbol{D}(39) = 0$], and related dashed arcs, as shown in Fig. 6. The "counter place" has an output arc of weight 5 to the new transition and a single input arc from t_{30}. When the system is in continuous mode and each time t_{30} fires, p_{27} is loaded with a token and p_{28} loses one. After t_{30} fires five times, p_{28} has no token, t_{30} is disabled, p_{27} has five tokens, and t_{39} is enabled if the resume push button is hit. Its firing removes five tokens from p_{27}, deposits five to p_{28}, and one to p_{24}. The cycle is reinitiated.

The insertion of a cycle counter in the RLL program requires modifications to existing rungs and the addition of two new rungs. One rung to include the counter function and another to reset the counter. The modifications and additions in the dotted box in Fig. 4(a) are as follows.

- NO contact of the original rung output instruction to start the system in continuous mode is used on a new rung to increment the counter each time a new cycle starts.
- Reset the counter; the resume push-button input is added.
- Prevent more than five cycles from occurring at once; the "not" (NC contact) of the counter done bit is placed in series with the original output instruction that starts the system when continuous mode is selected.

Sequence Modification: Modification of the sequence to eliminate the choice structure of the system requires modifications to the structures of both algorithms. The elimination of the choice structure simplifies the RTPN structure by eliminating one of the two parallel paths after p_3. The elimination of the choice structure in the RLL program does not simplify this algorithm. See [20] for a comparison of the old and new RTPN's and RLL.

In summary, RTPN exhibits better robustness responding to the specification changes than RLL. It requires fewer changes than its RLL program does. Therefore, PN's are a preferred approach to designing a controller whose specification requires frequent change to meet flexible and agile automation needs.

VI. Concluding Remarks

PN's hold a promise as a solution to modern industrial control problems. They display greater flexibility and understandability of the process than today's standard RLL programming techniques. To speed up their applications to industrial discrete event control problems, more standard software tools have to be developed [7], [12]. A number of companies, e.g., Hitachi, Kobe Steel, and Microsoft, have invested in this area and achieved a significant savings in system development time compared to traditional RLL and general-purpose programming approaches [9], [20]. They have also developed their PN-based computer programming tools for their system development, as shown in [10]–[12]. A benchmark study on a variety of methodologies should be helpful in convincing engineers to accept new approaches, such as PN's. This paper presents for the first time a convincing comparison between PN and RLL for an industrial automated system. The detailed PN design process is illustrated. Unfortunately, its real-time implementation was not performed, due to the resource constraints. Future work includes benchmark studies among sequential function charts, PN, structured text, state machine, and state chart methods. Comparison and evaluation of the existing CAD tools for discrete event control designs are also interesting.

Appendix

A summary of places and transitions in Figs. 6–8.

Place	Description	Attributes
p_1	Skimmer #9 start request	status only
p_2	Skimmer #9 moving to reverse position	($\boldsymbol{Z}(2) =$ skimmer to reverse)
p_3	Skimmer #9 in reverse skim mode	($\boldsymbol{Z}(3) =$ reverse skim indicator)
p_4	Skimmer #9 moving to shallow position	($\boldsymbol{Z}(4) =$ skimmer to shallow)
p_5	Skimmer #9 in shallow skim mode	($\boldsymbol{Z}(5) =$ shallow skim indicator)
p_6	Skimmer #9 moving to deep position	($\boldsymbol{Z}(6) =$ skimmer to deep)
p_7	Skimmer #9 in deep skim mode	($\boldsymbol{Z}(7) =$ deep skim indicator)
p_8	Skimming complete	status only
p_9	Skimmer #9 moving to rest position	($\boldsymbol{Z}(9) =$ skimmer to rest)
p_{10}	Skimmer #9 fault	status only
p_{11}	Skimmer #9 cycle complete	status only
p_{12}	Enable spray water valve	status only
p_{13}	Spwater valve open	($\boldsymbol{Z}(13) =$ spray water valve open)
p_{14}	Spray water valve closed	($\boldsymbol{Z}(14) =$ spray water valve closed)
p_{15}	Spray water valve loss of remote	status only
p_{16}	Enable high-wet-well e-stop	status only
p_{17}	High-wet-well e-stop	Disables all outputs when marked/resets RTPN by placing a single token in p_1 when t_{22} fires.
p_{18}	High-wet-well level	status only
p_{19}	Low-wet-well level	status only
p_{20}	Continuous mode start request	status only

Place	Description	Attributes
p_{21}	6-12-h timer enable	status only
p_{22}	Timer mode start request	status only
p_{23}	Timer mode select check	status only
p_{24}	Enable skimmer #9	status only
p_{25}	Enable skimmer #8	status only
p_{31}	Open scum valve	($\boldsymbol{Z}(31) =$ open scum valve)
p_{32}	Scum valve open	($\boldsymbol{Z}(32) =$ scum valve open indicator)
p_{33}	Close scum valve	($\boldsymbol{Z}(33) =$ close scum valve)
p_{34}	Scum valve fault	status only

Transition	Description	Attributes
t_1	Initiate skim cycle	Immediate transition
t_2	Enter reverse skim mode	($\boldsymbol{Y}(2)$ reverse skim limit switch; $\boldsymbol{D}(2) = 0$)
t_3	Reverse skim duration 1	($\boldsymbol{Y}(3) = 1; \boldsymbol{D}(3) = 1060$ s)
t_4	Enter shallow skim mode	($\boldsymbol{Y}(4) =$ shallow skim limit switch; $\boldsymbol{D}(4) = 0$)
t_5	Shallow skim duration	($\boldsymbol{Y}(5) = 1; = \boldsymbol{D}(5) =$ 10120 s)
t_6	Reverse skim duration 2	($\boldsymbol{Y}(6) = 1; \boldsymbol{D}(6) = 1060$ s)
t_7	Enter deep skim mode	($\boldsymbol{Y}(7) =$ deep skim limit switch; $\boldsymbol{D}(7) = 0$)
t_8	Deep skim duration	($\boldsymbol{Y}(8) = 1; \boldsymbol{D}(8) = 10120$ s)
t_9	End skimming	Immediate transition
t_{10}	Rest position limit switch	($\boldsymbol{Y}(10) =$ rest position limit switch; $\boldsymbol{D}(10) = 0$)
t_{11}	Enter skimmer to rest fault mode	($\boldsymbol{Y}(11) =$ “not” rest position limit switch; $\boldsymbol{D}(11) = 45$ s)
t_{12}	Enter skimmer to shallow fault mode	($\boldsymbol{Y}(12) =$ “not” shallow skim limit switch; $\boldsymbol{D}(12) = 45$ s)
t_{13}	Enter skimmer to deep fault mode	($\boldsymbol{Y}(13) =$ “not” deep skim limit switch; $\boldsymbol{D}(13) = 45$ s)
t_{14}	Enter skimmer to reverse fault mode	($\boldsymbol{Y}(14) =$ "not” reverse skim limit switch; $\boldsymbol{D}(14) = 45$ s)
t_{15}	Resume normal operation	$\boldsymbol{Y}(15) =$ resume push button; $\boldsymbol{D}(15) = 0$)
t_{16}	Step to next skimmer	Immediate transition
t_{17}	Enter spray water valve open status	($\boldsymbol{Y}(17) =$ spray water valve remote; $\boldsymbol{D}(17) = 0$)
t_{18}	Initiate to close spray water valve	Immediate transition
t_{19}	Enter spray water valve loss of remote status	($\boldsymbol{Y}(19) =$ spray water valve “not” remote; $\boldsymbol{D}(19) = 0$)
t_{20}	Enter spray water valve loss of remote status	($\boldsymbol{Y}(20) =$ spray water valve “not” remote; $\boldsymbol{D}(20) = 0$)
t_{21}	Initiate high-wet-well e-stop	Immediate transition
t_{22}	Reset high-wet-well e-stop	Immediate transition
t_{23}	Low-wet-well level switch debounce	($\boldsymbol{Y}(23) =$ wet well low level switch; $\boldsymbol{D}(23) = 3$ s)
t_{24}	High-wet-well level switch debounce	($\boldsymbol{Y}(24) =$ wet well high level switch; $\boldsymbol{D}(24) = 3$ s)
t_{25}	Skimmer #9 ends remote mode	($\boldsymbol{Y}(25) =$ skimmer#9 “not” in remote; $\boldsymbol{D}(25) = 1$ s)
t_{26}	Skimmer #9 ends remote mode	($\boldsymbol{Y}(26) =$skimmer#9 “not” in remote; \mbi{D}(26) = 1 s)
t_{27}	Skimmer #9 ends remote mode	($\boldsymbol{Y}(27) =$ skimmer#9 “not” in remote; $\boldsymbol{D}(27) = 1$ s)
t_{29}	Initiate continuous operation	($\boldsymbol{Y}(29) =$ skimmer#9 continuous mode selected; $\boldsymbol{D}(29) = 0$)
t_{30}	Continuous mode enable skimmer #9	($\boldsymbol{Y}(30) =$ skimmer#9 remote mode selected; $\boldsymbol{D}(30) = 0$)
t_{31}	Continuous mode bypass skimmer #9	($\boldsymbol{Y}(31) =$ skimmer#9 “not” in remote mode; $\boldsymbol{D}(31) = 0$)
t_{32}	Initiate timer operation	($\boldsymbol{Y}(32) =$ skimmer#9 timer mode selected; $\boldsymbol{D}(32) = 0$)
t_{33}	Initiate timer mode delay	($\boldsymbol{Y}(33) = 1; \boldsymbol{D}(33) = 612$ h)
t_{34}	End timer mode	($\boldsymbol{Y}(34) =$ skimmer#9 timer mode “not” selected; $\boldsymbol{D}(34) = 0$)
t_{35}	Timer mode enable skimmer #9	($\boldsymbol{Y}(35) =$ skimmer#9 remote mode selected; $\boldsymbol{D}(35) = 0$)
t_{36}	Timer mode bypass skimmer #9	($\boldsymbol{Y}(36) =$ skimmer#9 “not” in remote mode; $\boldsymbol{D}(36) = 0$)
t_{37}	Remote mode bypass skimmer #8	($\boldsymbol{Y}(37) =$ skimmer#8 “not” in remote mode; $\boldsymbol{D}(37) = 0$)
t_{38}	Remote mode bypass scum valve	($\boldsymbol{Y}(38) =$ scum valve “not” in remote mode; $\boldsymbol{D}(38) = 0$)
t_{41}	Enter scum valve open status	($\boldsymbol{Y}(41) =$ scum valve open limit switch; $\boldsymbol{D}(41) = 0$)
t_{42}	Scum valve open duration	($\boldsymbol{Y}(42) = 1; \boldsymbol{D}(42) =$ 30120 s)
t_{43}	Enter scum valve closed status	($\boldsymbol{Y}(43) =$ scum valve closed limit switch; $\boldsymbol{D}(43) = 0$)
t_{44}	Enter scum valve open fault mode	($\boldsymbol{Y}(44) =$ “not” scum valve open limit switch; $\boldsymbol{D}(44) = 45$ s)
t_{45}	Enter scum valve close fault mode	($\boldsymbol{Y}(45) =$ “not” scum valve closed limit switch; $\boldsymbol{D}(45) = 45$ s)
t_{46}	Reset scum valve fault	($\boldsymbol{Y}(46) =$ resume push button; $\boldsymbol{D}(46) = 0$)

References

[1] T. O. Boucher, M. A. Jafari, and A. G. Meredith, "Petri net control of an automated manufacturing cell," *Comput. Ind. Eng.,* vol. 17, no. 1–4, pp. 459–463, 1989.

[2] D. Crockett, A. Desrochers, F. DiCesare, and T. Ward, "Implementation of a Petri net controller for a machining workstation," in *Proc. IEEE Int. Conf. Robot. Automat.,* Raleigh, NC, 1987, pp. 1861–1867.

[3] L. A. Bryan, *Programmable Controllers—Theory and Implementation.* New York: Industrial Text, 1988.

[4] A. A. Desrochers and R. Y. Al-Jaar, *Applications of Petri Nets in Manufacturing Systems.* Piscataway, NJ: IEEE Press, 1995.

[5] R. David and H. Alla, *Petri Nets and Grafcet.* New York: Prentice-Hall, 1992.

[6] L. Ferrarini, "An incremental approach to logic controller design with Petri nets," *IEEE Trans. Syst., Man, Cybern.,* vol. 22, pp. 461–473, Mar. 1992.

[7] ——, "Computer aided design of logic controllers with Petri nets," in *Petri Nets in Flexible and Agile Automation,* M. C. Zhou, Ed. Norwell, MA: Kluwer, 1995, pp. 71–92.

[8] G. Michel, *Programmable Logic Controllers, Architecture and Applications.* New York: Wiley, 1991.

[9] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE,* vol. 77, pp. 541–580, Apr. 1989.

[10] ——, "Applications of Petri nets to sequence control programming," in *Petri Nets in Flexible and Agile Automation,* M. C. Zhou, Ed. Norwell, MA: Kluwer, 1995, pp. 43–69.

[11] T. Murata, N. Komoda, K. Matsumoto, and K. Haruna, "A Petri net-based controller for flexible and maintainable sequence control and its applications in factory automation," *IEEE Trans. Ind. Electron.,* vol. IE-33, pp. 1–8, Jan. 1986.

[12] T. Sato and K. Nose, "Automatic generation of sequence control program via Petri nets and logic tables for industrial applications," in *Petri Nets in Flexible and Agile Automation,* M. C. Zhou, Ed. Norwell, MA: Kluwer, 1995, pp. 93–107.

[13] I. Suzuki and T. Murata, "A method for stepwise refinements and abstractions of Petri nets," *J. Comp. Syst. Sci.,* vol. 27, pp. 51–76, 1983.

[14] R. Valette, "Analysis of Petri nets by stepwise refinements," *J. Comp. Syst. Sci.,* vol. 18, pp. 35–46, 1979.

[15] K. Venkatesh, M. C. Zhou, and R. J. Caudill, "Comparing ladder logic and Petri nets for sequence controller design through a discrete manufacturing system," *IEEE Trans. Ind. Electron.,* vol. 41, pp. 611–619, Dec. 1994.

[16] M. C. Zhou and F. DiCesare, "Adaptive design of Petri net controllers for error recovery in automated manufacturing systems," *IEEE Trans. Syst., Man, Cybern.,* vol. 19, pp. 963–973, May 1989.

[17] M. C. Zhou, F. DiCesare, and D. Rudolph, "Design and implementation of a Petri net based supervisor for a flexible manufacturing system," *Automatica,* vol. 28, no. 6, pp. 1999–2008, 1992.

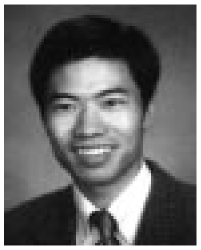
[18] M. C. Zhou and F. DiCesare, *Petri Nets Synthesis for Discrete Event Control of Manufacturing Systems.* Norwell, MA: Kluwer, 1993.

[19] M. C. Zhou, K. McDermott, and P. A. Patel, "Petri net synthesis and analysis of a flexible manufacturing system cell," *IEEE Trans. Syst., Man, Cybern.,* vol. 23, pp. 523–531, Feb. 1993.

[20] M. C. Zhou and E. Twiss, "A comparison of relay ladder logic programming and Petri net synthesis for control of discrete event systems," Discrete Event Syst. Lab., ECE, NJIT, Tech. Rep. 9501, Jan. 1995.

[21] ——, "Discrete event control design methods: A review," in *13th IFAC World Congress, Preprints,* vol. 9. San Francisco, CA, July 1996, pp. 401–409.

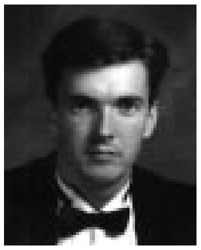
[22] R. Zurawski and M. C. Zhou, "Petri nets and industrial applications: A tutorial," *IEEE Trans. Ind. Electron.,* vol. 41, pp. 567–583, Dec. 1994.



MengChu Zhou (S'88–M'90–SM'93) received the B.S. degree in computer and control engineering from East-China Institute of Technology, Nanjing, China, in 1983, the M.S. degree in automatic control from Beijing Institute of Technology, Beijing, China, in 1986, and the Ph.D. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1990.

He joined the Department of Electrical and Computer Engineering at the New Jersey Institute of Technology (NJIT), Newark, in 1990 and is currently an Associate Professor and Director of Discrete Event Systems Laboratory. He is also affiliated with the Center for Manufacturing Systems at NJIT. He was Assistant Engineer at the Institute for Computer Applications, Beijing, China, from 1986 to 1987. His research interests include computer-integrated manufacturing and networking, embedded control, discrete event systems, Petri nets and applications, and intelligent automation. He coauthored *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems* with F. DiCesare (Norwell, MA: Kluwer, 1993) and edited *Petri Nets in Flexible and Agile Automation* (Norwell, MA: Kluwer, 1995). In addition, he has published more than 100 international journal articles, book chapters, and conference proceeding papers in the above areas. He has been invited to deliver seminars and workshops by several universities and companies in Australia, China, France, Germany, Japan, Taiwan, and the United States.

Dr. Zhou has organized and chaired more than 30 technical and tutorial sessions and served on program committees for many international and regional conferences. He served as the Program Chair of the *9th International Conference on CAD/CAM, Robotics, and Factories of the Future,* Newark, NJ, August 1993, Vice-Chair of the International Program Committee of *1996 IEEE International Conference on Systems, Man, and Cybernetics,* Beijing, China, and Program Chair of *1997 IEEE International Conference on Emerging Technologies and Factory Automation,* Los Angeles, CA. He served as the Guest Co-Editor for IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS Special Section on Petri Nets in Manufacturing, is serving as the Guest Co-Editor for IEEE TRANSACTIONS ON SEMICONDUCTOR MANUFACTURING Special Section on Petri Nets in Semiconductor Manufacturing and is on the Editorial Board of the *International Journal of Intelligent Control and Systems*. He is also serving as the Program Chair of the *1998 IEEE International Conference on Systems, Man, and Cybernetics,* San Diego, CA, and he is the Associate Editor for IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS. He was the recipient of NSF's Research Initiation Award and was listed in the 1994 Computer Integrated Manufacturing LEAD Award by the Society of Manufacturing Engineers. He was granted the H. J. Perlis Research Award by NJIT in 1996. He is currently serving as Vice-President of Technical Affairs and Societies, Chinese Association for Science and Technology-USA.



Edward Twiss (M'95) received the B.S. degree in electrical engineering in 1987 and the M.S. degree in manufacturing engineering in 1995, both from the New Jersey Institute of Technology, Newark.

He is currently a Research Associate at the Corporate Research Center of International Paper Company, Tuxedo Park, NY. He worked for Esab North America, Florence, SC, for two years, where he designed discrete event control systems for steel cutting and conditioning equipment. He then worked for two different control systems integrators in New Jersey for a period of six years, where he designed discrete event and continuous process control systems for pharmaceutical, power, and municipal waste water treatment facilities.

Mr. Twiss is a Member of the ISA.