Petri Net Diagnoser for DES Modeled by Finite State Automata

Marcos V. Moreira and Felipe G. Cabral and Oumar Diene

Abstract—Fault detection and isolation is an essential task in automated manufacturing systems and, as such, has received considerable attention in the literature. We propose in this paper a Petri net approach to online diagnosis of a discrete event system (DES) modeled by a finite state automaton. The diagnosis method requires, in general, less memory than other methods proposed in the literature and relies on the computation of a Petri net diagnoser (PND). In addition, a method for the conversion of a PND into a ladder diagram for implementation on a programmable logic controller (PLC) is presented. The conversion method leads to a ladder diagram that preserves the structure and represents the evolution of tokens of the PND.

I. INTRODUCTION

Fault detection and isolation is an essential task in automated manufacturing systems and, as such, has received considerable attention in the literature [1], [2], [3], [4].

In [1], a discrete-event system (DES) approach to fault diagnosis has been presented for systems modeled by finite state automata (FSA). The diagnosis method proposed in [1] consists of the following steps: (i) construction of a labeled automaton $G_{\ell}$ obtained from the automaton model of the system $G$, whose states are given as $(x, \ell)$, where $x$ is a state of $G$ and $\ell \in \{Y, N\}$; (ii) obtention of a diagnoser automaton $G_{\text{diag}}$ by computing the state observer of the labeled automaton $G_{\ell}$; (iii) identification of fault events based on the state of $G_{\text{diag}}$ reached after the observation of a trace executed by the system.

The diagnoser proposed in [1] can be used for online detection and isolation of faults and also for the off-line verification of the diagnosability of the language generated by the system. Although this diagnoser can be straightforwardly implemented on a computer, this is in general avoided, since, in the worst case, the state space of the diagnoser $G_{\text{diag}}$ grows exponentially in the cardinality of the state space of the system model $G$ [1], [2], [5].

In [2], an online diagnosis method that avoids the construction and storage of the complete automaton $G_{\text{diag}}$ is proposed. In order to do so, a nondeterministic automaton $G_{\text{diag}}^{nd}$ is computed by replacing each transition of $G_{\ell}$ associated with an unobservable event with an $\varepsilon$-transition. In this method only the current state of the diagnoser $G_{\text{diag}}$ and the automaton $G_{\text{diag}}^{nd}$ must be stored for online diagnosis. After the occurrence of an observable event, the next state of $G_{\text{diag}}$ can be obtained online from the current state of $G_{\text{diag}}$ and from $G_{\text{diag}}^{nd}$ in polynomial time.

Several other diagnosis techniques, that use FSA or Petri Nets to model both the system and the diagnoser, have been presented (see, for instance, [5], [6]). However, only few works address the problem of implementing an online diagnoser on a programmable logic controller (PLC). PLCs are the most important tool for the discrete control of automated manufacturing systems and can be programmed using five different languages defined in the international standard IEC 61131-3 [7]: (i) ladder diagram (LD); (ii) function block diagram; (iii) structured text; (iv) instruction list; and (v) sequential function chart (SFC). Among the five PLC programming languages, LD is the most used by the industry and is available in almost all PLCs.

A PLC can be used exclusively for diagnosis or, depending on the specifications of the closed loop system, the online diagnoser can be implemented on the same PLC used in the feedback control. The main advantage of the latter implementation scheme is the reduction in the hardware needed for diagnosis. Notice that, in this case, all the command events become observable for the diagnoser without the need for additional sensors or communication busses.

In [8], a particular PLC platform, the softPLC Orchestra, is used for diagnosis. In this case, the diagnoser is a PLC task, written in C language, which samples the PLC global variables and follows the system evolution through the state transitions of the diagnoser automaton. Although this implementation scheme is successfully applied in [8], the extension of this method to other PLC platforms that do not support C language is not a simple task. From the authors’ knowledge there is no work in the literature that addresses the problem of implementing a diagnoser on a PLC by using any of the five languages defined in the standard IEC 61131-3 [7].

Despite the fact that there is almost no literature about PLC implementation of online diagnosers, several methods for the conversion of complex control codes to LDs have been presented [9], [10]. In [11], [12], two important problems related to the implementation of control codes modeled as automata and SFCs are introduced: (i) the avalanche effect; (ii) the need for a choice between conflicting transitions. In [13] a method for the conversion of a control interpreted Petri net used for the description of the control logic, into an LD for implementation, is presented. The LD is organized in such a way to avoid the avalanche effect.

In this paper, we present a Petri net approach to online diagnosis that allows the obtention of an LD code. The
definition of diagnosability considered here is the same as in [1]. We also assume that the system is described by a finite state automaton $G$ and that the set of fault events $\Sigma_f$ can be partitioned into different fault event sets $\Sigma_{f_k}$, $k = 1, \ldots, r$, where $r$ denotes the number of fault types. The method relies on the computation of an automaton $G_C$, obtained from $G$ and automata $G_{N_k}$, for $k = 1, \ldots, r$, where automaton $G_{N_k}$ models the nonfaulty behavior of $G$ with respect to the fault event set $\Sigma_{f_k}$. In general, $G_{N_k}$ has a smaller number of states than $G$, leading to a reduction in the computational complexity of the online diagnosis in comparison with the method proposed in [2] that uses the normal and faulty behaviors of the system described by $G_{t_k} = G[A_{t_k}]$, where $A_{t_k}$ denotes the label automaton with respect to the fault type $F_k$.

The diagnosis technique consists of finding the reachable states of $G_C$ after the observation of a trace and, based on the set of reachable states of $G_C$, verifying if a fault has occurred. In order to do so, a Petri net diagnoser (PND) is proposed. This Petri net is obtained from a binary Petri net that is able to estimate the reachable states of $G_C$ after the observation of a trace. In addition, a method for the conversion of a PND, that describes the online diagnoser, into an LD is presented. The conversion method, based on [13], avoids the avalanche effect and provides a well structured ladder code.

This paper is organized as follows. In Sections II and III we present the preliminary concepts and the definition of language diagnosability, respectively; in Section IV we show how to obtain automaton $G_C$; in Section V we present the Petri net diagnoser (PND); the conversion method of PNDs into LDs is presented in Section VI; finally, in Section VII the conclusions are drawn. An example is used throughout the text to illustrate the results.

II. PRELIMINARIES

Let $G = (Q, \Sigma, \Gamma, f, q_0)$ denote the deterministic automaton model of a DES, where $Q$ is the finite state space, $\Sigma$ is the set of events, $\Gamma$ is the feasible event function, $f$ is the transition function and $q_0$ is the initial state of the system. For the sake of simplicity, the feasible event function is omitted unless stated otherwise.

The projection $P_s : \Sigma_i^* \rightarrow \Sigma_s^*$, where $\Sigma_a \subset \Sigma_i$, is defined as $P_s(\varepsilon) = \varepsilon$, $P_s(\sigma) = \sigma$, if $\sigma \in \Sigma_a$ or $P_s(\sigma) = \varepsilon$, if $\sigma \in \Sigma_i \setminus \Sigma_a$, and $P_s(\sigma \sigma') = P_s(\sigma) P_s(\sigma')$, for all $s \in \Sigma_i^*$, and $s \in \Sigma_i$. The projection operation can also be applied to the language generated by $G$, $L(G)$, simply by applying these rules to all traces $s \in L(G)$.

Let $G_1$ and $G_2$ be two automata. Then $G_1 \times G_2$ and $G_1 \parallel G_2$ denote the product and the parallel composition of $G_1$ and $G_2$, respectively [14].

Let us now suppose that the event set of $G$ is partitioned as $\Sigma = \Sigma_o \cup \Sigma_{u_o}$, where $\Sigma_o$ and $\Sigma_{u_o}$ denote the set of observable and unobservable events, respectively. The unobservable reach of a state $q \in Q$ with respect to the set $\Sigma_{u_o}$, denoted by $UR(q)$, is defined as

$$UR(q) = \{y \in Q : (\exists t \in \Sigma_{u_o})[f(q, t) = y]\}. \quad (1)$$

This definition is extended to a subset of states $B \subseteq Q$ as follows:

$$UR(B) = \bigcup_{q \in B} UR(q). \quad (2)$$

A Petri net (PN) is another modeling formalism commonly used to describe a DES [14], [15]. The main advantage of using a PN to describe a DES in comparison with FSA, is the distributed nature of the state of the PN, which allows the obtention of compact graphs to describe DESs.

Definition 1: A Petri net $N$ is a six-tuple $N = (P, T, Pre, Post, w, x_0)$, where $P$ is the finite set of places, $T$ is the finite set of transitions, $Pre \subseteq (P \times T)$ is the set of ordinary arcs from places to transitions, $Post \subseteq (T \times P)$ is the set of ordinary arcs from transitions to places, $w : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ is the weight function on the arcs, $w(p_i, t_j) = 0$ and $w(t_j, p_i) = 0$ if and only if $(p_i, t_j) \notin Pre$ and $(t_j, p_i) \notin Post$, respectively, and $x_0$ is the initial marking of the set of places $P$.

The set of places is represented in this paper as $P = \{p_1, p_2, \ldots, p_n\}$ and the set of transitions as $T = \{t_1, t_2, \ldots, t_m\}$. Thus, $|P| = n$ and $|T| = m$, where $|\cdot|$ denotes cardinality.

A transition $t_j$ in a Petri net is said to be enabled when the number of tokens in each one of its input places is greater than or equal to the weight of the arcs connecting the places to transition $t_j$, i.e., $t_j$ is enabled if and only if

$$x(p_i) \geq w(p_i, t_j), \quad \text{for all } p_i \in I(t_j), \quad (3)$$

where $I(t_j)$ denotes the set of input places of $t_j$. If transition $t_j$ is enabled for a marking $\overline{x}$ and the event associated with $t_j$ occurs, then transition $t_j$ fires and a new marking $\overline{x}$ is achieved. The evolution of the markings is given by:

$$\overline{x}(p_i) = x(p_i) - w(p_i, t_j) + w(t_j, p_i), \quad i = 1, \ldots, n. \quad (4)$$

A particular class of Petri nets is the so called state machine Petri net (SMPN). An SMPN is a Petri net where each transition has exactly one input place and one output place. If this Petri net has also only one token, then the SMPN behaves exactly as an automaton, where each place is associated with a state in the corresponding automaton.

III. DIAGNOSABILITY OF DES

Let $G$ denote the automaton model of the system and $\Sigma_f \subseteq \Sigma_{u_o}$ denote the set of fault events. In addition, assume that the set of fault events can be partitioned as

$$\Sigma_f = \bigcup_{k=1}^{r} \Sigma_{f_k}, \quad (5)$$

where $\Sigma_{f_k}$ represents a set of fault events of the same type. Let $\Pi_f$ denote this partition.

Let $L(G) = L$ denote the language generated by $G$ and $G_{N_k}$ be the subautomaton of $G$ that represents the nonfaulty behavior of the system with respect to the fault event set $\Sigma_{f_k}$, i.e., assuming that $L(G_{N_k}) = L_{N_k}$, then $L_{N_k}$ is a prefix-closed language formed with all traces of $L$ that do
not contain any fault event from the set $\Sigma f_k$. The following definition of language diagnosability can be stated [1].

**Definition 2.** Let $L$ and $L_{N_k} \subseteq L$ be the prefix-closed languages generated by $G$ and $G_{N_k}$, respectively, and define the projection operation $P_o : \Sigma^* \rightarrow \Sigma^o$. Then $L$ is said to be diagnosable with respect to the projection $P_o$ and with respect to the partition $\Pi_f$ on $\Sigma_f$ if

$$(\forall k \in \Pi_f)(\exists n_k \in \mathbb{N})(\forall s \in L \setminus L_{N_k})(\forall t \in L \setminus L_{N_k})$$

$$||t|| \geq n_k \Rightarrow (\forall \omega \in P^{-1}_o(P_o(st)) \cap L, \omega \in L \setminus L_{N_k}),$$

where $||.||$ denotes the length of a trace.

According to Definition 2, $L$ is diagnosable with respect to $P_o$ and $\Pi_f$ if and only if for all traces $st$ of arbitrarily long length after the occurrence of a fault event from the set $\Sigma f_k$, there do not exist traces $s_{N_k} \in L_{N_k}$, such that $P_o(s_{N_k}) = P_o(st)$, $\forall k \in I_r = \{1, 2, \ldots, r\}$. Therefore, if $L$ is diagnosable then it is always possible to uniquely identify the type of the fault that has occurred within a bounded delay.

The first step to implement an online diagnoser is to verify if all fault types in a system can be diagnosed after the occurrence of the fault in a bounded number of observations. In [16] a polynomial time algorithm to verify if a language $L$ is diagnosable is presented. The algorithm proposed in [16] is based on the construction of a deterministic verifier automaton and the verification has smaller computational complexity than other methods proposed in the literature. In this paper, we suppose that the language generated by the system is diagnosable with respect to $P_o$ and $\Pi_f$.

**IV. COMPUTATION OF AUTOMATON $G_C$**

In this section, an algorithm for the computation of automaton $G_C$ is presented. Differently from the diagnoser automata introduced in [2], that uses the faulty and nonfaulty behaviors of the system described by $G_{f_k}$, automaton $G_C$ is constructed by using only the nonfaulty behavior of the system, with respect to fault type $F_k$, modeled by $G_{N_k}$, which has in general a smaller number of states and transitions than $G_{f_k}$. The case of multiple fault types is presented in Algorithm 1.

**Algorithm 1**:

- **Step 1:** Compute automaton $G_{N_k}$, for each $k \in \Pi_f$, that models the normal behavior of $G$ with respect to the fault event set $\Sigma f_k$, as follows:
  - **Step 1.1:** Define $\Sigma_{N_k} = S \setminus \Sigma_{f_k}$.
  - **Step 1.2:** Build automaton $A_{N_k}$ composed of a single state $N_k$ (also its initial state) with a self-loop labeled with all events in $\Sigma_{N_k}$.
  - **Step 1.3:** Construct the nonfaulty automaton $G_{N_k} = G \times A_{N_k} = (Q_{N_k}, \Sigma, \delta_{N_k}, q_{N_k}, \Gamma_{N_k})$.

- **Step 2:** Construct the augmented automaton $G^a_{N_k}$, for each $k \in \Pi_f$, by adding a new state $F_k$, that indicates that a fault event from the set $\Sigma f_k$ has occurred. A new transition labeled with an event $\sigma f_k \in \Sigma f_k$ is added, connecting state $(q, N_k)$ of $G_{N_k}$ to the faulty state $F_k$, if $\sigma f_k \in \Gamma(q)$. Add a self-loop transition labeled with all events $\sigma \in \Sigma$ to the faulty state $F_k$.

- **Step 3:** Compute $G_C = (Q_C, \Sigma, f_C, \Gamma_C, q_{0,C}) = G^a_{N_1}||G^a_{N_2}|| \ldots ||G^a_{N_r}$.

Notice that, for each $G^a_{N_k}$, the faulty behavior of the system with respect to the fault event set $\Sigma f_k$ is represented by the faulty state $F_k$, with a self-loop labeled with all events in the set $\Sigma$, that simply indicates that a fault from the set $\Sigma f_k$ has occurred. It is important to remark that this representation does not preserve the language generated by the system after the occurrence of the fault event. However, since the diagnoser is a passive device, this representation does not alter the observation of the system events and, thus, does not interfere in the fault diagnosis.

In order to show how automaton $G_C$ can be used for online diagnosis, we will first define a function that provides the possible current states of $G_C$ after the occurrence of an observable event. This estimate is denoted in this paper as $Reach(\nu)$, where $\nu = \nu o = P_o(s)$ is the trace observed by the diagnoser after the execution of a trace $s \in L$ whose last observable event is $o$, and can be computed recursively as

$$Reach(\varepsilon) = UR(q_{0,C}),$$

$$Reach(\sigma o) = UR(\delta(Reach(\nu), o)), \quad (7)$$

where $\delta(Reach(\nu), o) = \bigcup_{i=1}^{\kappa} \delta_C(q_{C_i}, o)$, with $q_{C_i} \in Reach(\nu)$, $\kappa = |Reach(\nu)|$, and $\delta_C(q_{C_i}, o) = f_C(q_{C_i}, o)$ if $f_C(q_{C_i}, o)$ is defined and $\delta_C(q_{C_i}, o) = \emptyset$, otherwise.

After the observation of trace $\nu$, the set of possible current states of $G_C$, $Reach(\nu)$, can be computed and its states can be used to identify the occurrence of a fault event. The following theorem provides the basis for the diagnosis method proposed in this paper.

**Theorem 1:** Let $L$ be the language generated by $G$ and assume that $L$ is diagnosable with respect to $P_o$ and $\Pi_f$. Let $s \in L \setminus L_{N_k}$ be such that $\forall \omega \in L$, with $P_o(\omega) = P_o(s)$, $\omega \in L \setminus L_{N_k}$. Then, the $k$-th coordinate of the possible current states of $G_C$, reached after the occurrence of $s$, given by $Reach(P_o(s))$, is equal to $F_k$.

**Proof:** The proof follows directly by the construction of automaton $G_C$ and will be omitted.

According to Theorem 1, if $L$ is diagnosable with respect to $P_o$ and $\Pi_f$, then it is always possible to identify the occurrence of a fault of type $F_k$ within a bounded number of observations by verifying the possible current states of $G_C$. If, after the occurrence of a trace $s$ that contains a fault event $\sigma f_k \in \Sigma f_k$, all states of $Reach(\nu)$, where $\nu = P_o(s)$, do not have an $(q, N_k)$ coordinate, then it is not possible that a normal trace with respect to the fault event set $\Sigma f_k$, with the same projection as $\nu$, has been executed, which implies that a fault of type $F_k$ has occurred. Thus, the diagnosis of a fault of type $F_k$ can be performed by verifying if a state of the normal behavior described by $G_{N_k}$ is a coordinate of a possible current state of $G_C$.

**Remark 1:** The number of states of $G_C$ is, in the worst case, equal to $[(2^r - 1) \times |Q|] + 1$, where $r$ is the number of system fault types. Thus, the computational complexity of constructing automaton $G_C$ is $O(2^r \times |Q| \times |\Sigma|)$, which shows that the complexity is linear in the number of states.
and events of the automaton model of the system and exponential in the number of fault types. The computational complexity can be made linear in the number of fault types if each normal behavior with respect to a single fault type is considered separately. In this case, instead of a single automaton \( G_C \), we have \( r \) automata \( G_{N_k}^a \), where each one takes into account only fault type \( F_k \), and the computational complexity is \( O(r \times (|Q| \times |\Sigma|)) \). Although the worst case analysis suggests that it is advantageous to consider automata \( G_{N_k}^a \), \( k = 1, \ldots, r \), instead of the single automaton \( G_C \), it is important to remark that the number of states of \( G_C \) can be smaller than the number of the sum of states of \( G_{N_k}^a \) for \( k = 1, \ldots, r \), leading to a smaller programming code for implementation on a PLC.

**Example 1:** Consider the system modeled by automaton \( G \) shown in Figure 1, where \( \Sigma = \{a, b, c, \sigma_a, \sigma_f, \sigma_f'\} \), \( \Sigma_a = \{a, b, c\} \), \( \Sigma_{uo} = \{\sigma_a, \sigma_f, \sigma_f'\} \), and \( \Sigma_f = \{\sigma_f, \sigma_f'\} \). Assume that the fault event set can be partitioned as \( \Sigma_f = \Sigma_{f_1} \cup \Sigma_{f_2} \) with \( \Sigma_{f_1} = \{\sigma_f\} \) and \( \Sigma_{f_2} = \{\sigma_f'\} \), and suppose that we want to compute \( G_C \). According to Algorithm 1, the first step is to obtain the single state automata \( A_{N_k} \), \( k = 1, 2 \), and the normal automata \( G_{N_k} \) for \( k = 1, 2 \). The next step is the construction of the augmented automata \( G_{N_k}^a \) and \( G_{N_2}^a \), shown in Figures 2 and 3, respectively, by adding the faulty states \( F_1 \) and \( F_2 \) to automata \( G_{N_1} \) and \( G_{N_2} \). The final step of Algorithm 1 is the computation of automaton \( G_C = G_{N_1}^a \parallel G_{N_2}^a \), depicted in Figure 4.

Let us now show how automaton \( G_C \) can be used for online diagnosis. Suppose that the faulty trace \( s = aaa \) has been executed by the system. Then, the observed trace is \( \nu = P_o(s) = aaaa \). According to theorem 1, if there do not exist a trace \( \omega \in L_{N_1} \) such that \( P_o(\omega) = \nu \) then all states in the reachable set \( \text{Reach}(\nu) \) must have the first coordinate equal to \( F_1 \). The reachable set \( \text{Reach}(\nu) \) can be recursively obtained according to Equations (6) and (7), as follows: \( \text{Reach}(\epsilon) = \{(0N_1, 0N_2)\} \), \( \text{Reach}(a) = \{(1N_1, 1N_2), (2N_1, 2N_2), (F_1, 5N_2), (7N_1, F_2), (8N_1, F_2)\} \), \( \text{Reach}(aa) = \{(F_1, 6N_2), (9N_1, F_2)\} \), and \( \text{Reach}(aaa) = \{(F_1, 8N_2)\} \).

Since the unique state reached after the observation of \( \nu = aaaa \) has the first coordinate equal to \( F_1 \), then it is possible to guarantee that the fault event \( \sigma_{f_1} \) has occurred.

Regarding the computational complexity of constructing \( G_C \), notice that \( G_C \) has 12 states and \( G_{N_1}^a \) and \( G_{N_2}^a \) have 9 and 10 states, respectively. Thus \( G_C \) has a smaller number of states than the sum of the states of \( G_{N_1}^a \) and \( G_{N_2}^a \). Therefore, as shown in remark 1, the online diagnosis can be performed in this case with a smaller computational cost by using \( G_C \) instead of \( G_{N_k}^a \), \( k = 1, 2 \).

V. PETRI NET DIAGNOSER

Consider now the problem of finding the possible current states of \( G_C \) after the observation of a trace \( \nu \in \Sigma_o^* \). In order to solve this problem, an online observer that stores the estimate of the current states of \( G_C \) after the occurrence of an observable trace must be computed. This online state observer can be constructed by using the Petri net formalism, exploring the distributed nature of the state of a Petri net, leading to a Petri net state observer.

The first step in the construction of a Petri net state observer is the obtention of an SMPN, \( N_C \), from automaton \( G_C \). This can be easily done by associating to each state \( q_{Ci} \) of \( G_C \) a place \( p_{Ci} \) in \( N_C \) and by associating to each directed arc in \( G_C \), labeled with \( \sigma \in \Gamma_C(q_{Ci}) \), a transition \( t_{Ci} \), labeled with \( \sigma \), in \( N_C \) [14]. The initial state of \( N_C \) is defined assigning a token to the place of \( N_C \) associated with the initial state of \( G_C \) and setting to zero the number of tokens of the other places.

Once \( N_C \) has been obtained, the next step for the computation of the Petri net state observer for \( G_C \) is the creation of new arcs, connecting each transition labeled with an observable event, to specific places that correspond to the
unobservable reach of places after the firing of an observable transition. In order to do so, let $T_{C_o}$ denote the set of all transitions of $N_{C_o}$ labeled with observable events and define function $Reach_T: T_{C_o} \to 2^{T_{C_o}}$, where $P_{C_o}$ is the finite set of places of $N_{C_o}$. The set of places $Reach_T(t_{C_j})$, where $t_{C_j} \in T_{C_o}$, can be computed as follows.

**Algorithm 2:** Let $O(t)$ and $O(p)$ denote the set of all output places of $t$ and the set of all output transitions of $p$, respectively. Let also $O(P) = \bigcup_{p \in P} O(p)$ and $O(T) = \bigcup_{t \in T} O(t)$.

- **Step 1:** Define $p_{out} = O(t_{C_j})$, $P'_t = \{ p_{out} \}$ and $P_t = P_t' \setminus P_{C_o}$.
- **Step 2:** Form the set $T'_o$ with all transitions of $O(P'_t)$ associated with unobservable events. If $T'_o = \emptyset$, $Reach_T(t_{C_j}) = P_t$ and stops.
- **Step 3:** Do $P'_t = O(T'_o)$, $P_t = P_t \cup P'_t$, and return to Step 2.

To implement the unobservable reach after the firing of each observable transition, an arc of weight one, connecting each transition $t_{C_j} \in T_{C_o}$ to each place $p_{C_i} \in Reach_T(t_{C_j})$, must be added to $N_{C_o}$, generating a new Petri net $N'_{C_o}$. After that, all transitions labeled with unobservable events of $N'_{C_o}$, and their related arcs, must be removed generating a new Petri net $N_{C_o}'$, whose transitions are all labeled with observable events of $\Sigma_o$.

Notice that for the obtention of the estimate of the states of $G_{C_o}$, only the places that are associated with the possible current states of $G_{C_o}$ must have tokens and, after the occurrence of a new observable event, the number of tokens in the places that are no longer possible must be set to zero. This implies that the number of tokens in each place of the Petri net state observer must be equal to one or zero, even if the firing of a transition $t_{C_j} \in T_{C_o}$ results, according to equation (4), in a marking with two or more tokens. Therefore, places are forced to have binary markings and equation (4) is no longer valid. This requirement can be satisfied by using binary Petri nets [17]. A binary Petri net can be defined as a Petri net with a different evolution rule for the places markings reached after the firing of a transition $t_j$ given by:

$$\bar{\Sigma}(p_i) = \begin{cases} 0, & \text{if } x(p_i) - w(p_i, t_j) + w(t_j, p_i) = 0 \\ 1, & \text{if } x(p_i) - w(p_i, t_j) + w(t_j, p_i) > 0, \end{cases} \quad i = 1, \ldots, n.$$  \hspace{1cm} (8)

It is important to remark that defining $N_{C_o}$ as a binary Petri net is not a sufficient condition to guarantee that it can be used as a state observer. For instance, suppose that $p_{C_i}$ is a place of $N_{C_o}$ that has a token and does not have an output transition labeled with an observable event $\sigma_o \in \Sigma_o$. Assume also that $p_{C_i}$ does not have an enabled input transition labeled with $\sigma_o$. Then, if $\sigma_o$ occurs, $p_{C_i}$ remains with one token. Considering that a place $p_{C_i}$, with one token represents a possible current state $q_{C_i}$ of $G_{C_o}$, it can be verified that, in this example, $p_{C_i}$ should have finished with no tokens, which shows that the state of the binary Petri net $N_{C_o}$ does not correspond to the possible current states of $G_{C_o}$ after the occurrence of $\sigma_o$.

In order to obtain the Petri net state observer, $N_{SO}$, it is necessary to add an arc connecting each place $p_{C_i}$ of $N_{C_o}$ to a new transition, labeled with the observable events of $\Sigma_o$ that are not in the active event set of the state $q_{C_i}$ of $G_{C_o}$ associated with $p_{C_i}$. This modification and the fact that the Petri net state observer is a binary Petri net, guarantee that if place $p_{C_i}$ is not associated with a possible current state of $G_{C_o}$ after the firing of an observable transition, then the number of tokens of $p_{C_i}$ will be equal to zero. Finally, to define the initial state of $N_{SO}$, we assign a token to each place associated with a state of $UR(q_{0,C})$, and set the number of tokens of the other places to zero. This definition guarantees that the set of all places of $N_{SO}$, that has initially one token, corresponds to the set of all possible initial states of $G_{C_o}$, given by $UR(q_{0,C})$.

Once $N_{SO}$ has been obtained, the Petri net diagnoser $N_{Diag}$ can be computed by adding to $N_{SO}$ transitions $t_{f_k}$, for $k = 1, \ldots, r$, where we also add to each transition $t_{f_k}$ an input place $p_{N_k}$, with initially one token, and an output place $p_{F_k}$ without tokens, both connected to $t_{f_k}$ by ordinary arcs. Each transition $t_{f_k}$ is associated with the verification of the occurrence of a fault type $F_k$. Inhibitor arcs [15] of weight one are used to connect each place, associated with a state of $G_{C_o}$ that has a coordinate $(q, N_k)$ to transition $t_{f_k}$. Since the inhibitor arc enables a transition only when the number of tokens of the input place is equal to zero, then $t_{f_k}$ will be enabled only when the normal behavior with respect to a fault of type $F_k$ is not possible, which implies that a fault from the set $\Sigma_{f_k}$ has occurred. An inhibitor arc will be represented by an arc whose end is marked by a small circle. Transition $t_{f_k}$ will be labeled with the always occurring event $e$ [15], to represent that $t_{f_k}$ fires immediately after being enabled, removing the token of place $p_{N_k}$ and adding a token to place $p_{F_k}$, which indicates that a fault of type $F_k$ has occurred.

Algorithm 3 summarizes the steps that are necessary for obtaining the Petri net diagnoser $N_{Diag}$ from automaton $G_{C_o}$.

**Algorithm 3:**

- **Step 1:** Compute the SMPN $N_{C_o} = (P_{C_o}, T_{C_o}, Pre_{C_o}, Post_{C_o}, w_{C_o}, x_{0,C})$ from $G_{C_o}$.
- **Step 2:** Add to $N_{C_o}$ arcs connecting each observable transition $t_{C_j} \in T_{C_o}$ to the places in $Reach_T(t_{C_j})$, generating the Petri net $N'_{C_o} = (P_{C_o}, T_{C_o}, Pre_{C_o}, Post_{C_o}, w'_{C_o}, x_{0,C})$.
- **Step 3:** Eliminate all transitions of $N'_{C_o}$ labeled with unobservable events and their related arcs, generating the binary Petri net $N_{C_o} = (P_{C_o}, T_{C_o}, Pre_{C_o}, Post_{C_o}, w_{C_o}, x_{0,C})$.
- **Step 4:** Compute $N_{SO} = (P_{SO}, T_{SO}, Pre_{SO}, Post_{SO}, w_{SO}, x_{SO})$ as follows:
  - Step 4.1: Add to $N_{C_o}$ transitions labeled with the observable events of $\Sigma_o$ that are not in the active event set of state $q_{C_i}$ of $G_{C_o}$ associated with $p_{C_i}$.
  - Step 4.2: Define the initial state of $N_{SO}$ by assigning a token to each place associated with a state of $UR(q_{0,C})$ and zero to the other places.
- **Step 5:** Compute the Petri net diagnoser $N_{Diag} = (P_{Diag}, T_{Diag} \cup T_f, Pre_{Diag}, Post_{Diag}, In_{Diag}, w_{Diag})$. 

6746
Fig. 5. Binary Petri net $N_{Co}$ of Example 2.

Fig. 6. Petri net state observer $NSO$ of Example 2.

(x,Diag), where $T_f = \bigcup_{k=1}^r \{ t_{f_k} \}$ and $In_{Diag} \subseteq P_{Diag} \times T_f$ denotes the set of inhibitor arcs, as follows:

- Step 5.1: Add to $NSO$ transitions $t_{f_k}$, $k = 1, \ldots, r$, labeled with the always occurring event $e$. We also suppose in this paper that two or more input events cannot occur simultaneously.

- Step 5.2: Add to each transition $t_{f_k}$ an input place $p_{f_k}$, with one token, and an output place $p_{f_k}$ without tokens, both connected to $t_{f_k}$ by ordinary arcs.

- Step 5.3: Connect each place associated with a state of $G_C$ that has a coordinate $(q, N_k)$ to transition $t_{f_k}$ with an inhibitor arc.

- Step 5.4: The initial state of the places $p_{N_k}$ is one and of the places $p_{f_k}$ is zero. The other places have the same initial condition defined by $x_{0,SO}$.

Example 2: Consider the diagnoser automaton $G_C$ shown in Figure 4, and consider the problem of computing the Petri net diagnoser $N_{Diag}$ for $G_C$. Following Steps 1, 2, and 3 of Algorithm 3, the binary Petri net $N_{Co}$, presented in Figure 5, is obtained. Then, according to Step 4, the Petri net state observer $NSO$, shown in Figure 6, is obtained from $N_{Co}$. Finally, following Step 5 of Algorithm 3, the Petri net diagnoser $N_{Diag}$, presented in Figure 7, is computed.

Let us now show the diagnosis method using $N_{Diag}$. Suppose that the faulty trace $s = a a \sigma_{f_1} a a \in L \setminus L_{N_1}$ has been executed by the system. Then, the observed trace is $\nu = P_o(s) = a a a$. Since the initial state of $N_{Diag}$ has a token only in place $(0N_1, 0N_2)$, associated with the initial state of $G_C$, then after the occurrence of the first event $a$, transition $t_{SO_1}$ fires and the set of places associated with the possible states of $G_C$ that have one token is given by $\{(1N_1, 1N_2), (2N_1, 2N_2), (7N_1, F_2), (8N_1, F_2), (F_1, 5N_2)\}$. Then, when the second event $a$ is observed, transitions $t_{SO_2}, t_{SO_3}, t_{SO_6}, t_{SO_{18}}$ fire and the set of places with one token is given by $\{(F_1, 6N_2), (9N_1, F_2)\}$. Notice that transitions $t_{SO_2}, t_{SO_3}, t_{SO_{18}}$ have been created according to Step 4.1 of Algorithm 3 to remove the token of the places that are not associated with a possible state of $G_C$. After the occurrence of the third event $a$, transitions $t_{SO_{11}}, t_{SO_{19}}$ fire and the unique place of $N_{Diag}$ that remains with one token is $(F_1, 8N_2)$. Since all places associated with a state of $G_C$ with a coordinate $(q, N_1)$ do not have a token, then transition $t_{f_1}$, labeled with event $e$, is enabled and fires, removing the token of place $p_{N_1}$ and adding a token to $p_{F_1}$, indicating the occurrence of the fault event $\sigma_{f_1}$.

VI. CONVERSION OF PNDs INTO LDs

A PLC operates executing scan cycles that consist of three main steps: (i) read and store the inputs of the PLC; (ii) execute the user programming code and; (iii) update the outputs. In general, input events are associated with the rising edge or the falling edge of sensor signals and the outputs are commands sent by the controller to the plant in response to changes in the values of the sensor signals.

In order to implement the online diagnoser in the same PLC used to control the system, the diagnoser code cannot be inserted after the control code, otherwise events associated with changes in the sensor signals, and the command events associated with the response of the controller to these changes, would be seen by the diagnoser as occurring at the same time. Thus, with a view to mimic the actual system behavior, the diagnoser code must be implemented before the control code. We also suppose in this paper that two or more input events cannot occur simultaneously.

In [11], an important problem related to the implementation of controllers in LD, called the avalanche effect, is discussed and a procedure to overcome this problem is proposed. The avalanche effect happens in the ladder code when two or more consecutive transitions are labeled with the same event and, when the event occurs, a transition that was not enabled is transposed. Although the solution proposed in [11] has been successfully applied in some examples, it is not easily implemented when the Petri net contains loops.

A conversion method that establishes transformation rules from interpreted Petri nets to LDs, that preserves the structure of the Petri net and also avoids the avalanche effect, is
presented in [13]. In this paper, the same method is applied to obtain the ladder code for the Petri net diagnoser $N_{Diag}$ of Figure 7. Parts of the corresponding ladder code are presented in Figure 8 where it can be seen that the ladder code has been divided in four modules. The first module is associated with the initialization of the Petri net, i.e., it defines the initial marking. The second module is associated with the observation of events from changes in sensor signals - an one shot rising (OSR) contact is used to guarantee that the enabled transitions will fire only when the sensor signal $S_y$, associated with event $\sigma$, changes its logical value. The third module describes the conditions for the firing of the transitions. When the input places are connected to a transition $t_j$ with ordinary arcs, then a normally open (NO) contact is added to the rung of $t_j$. On the other hand, if the input place is connected to transition $t_j$ with an inhibitor arc a normally closed (NC) contact is added to the rung of $t_j$. Finally, the fourth module describes the evolution of the tokens in the binary Petri net $N_{Diag}$, and can also be used to define actions to be taken after the fault detection.

It is important to remark that when the output transition $t_j$ of a place $p_i \in P$ and its input transitions fire simultaneously, $p_i$ is activated and deactivated simultaneously. In this case, $p_i$ must remain activated. In order to implement this behavior in the ladder code two rungs must be added in the Petri net dynamics module associated with the firing of $t_j$. In the first rung, NC contacts associated with the input transitions of place $p_i$ are added in series with the NO contact associated with the enabling of $t_j$. A reset coil is used to indicate that $p_i$ looses a token when $t_j$ fires. In the second rung, parallel NO contacts associated with the input transitions of $p_i$ are added in series with the NO contact associated with the enabling of $t_j$. The reset coil associated with $p_i$ is no longer used since at least one of the input transitions of $p_i$ will also fire.

VII. CONCLUSIONS

In this paper we present a Petri net diagnoser that can be used for online detection and isolation of systems faults. In general, the online diagnosis procedure requires less memory than other methods proposed in the literature. In addition, a method for the conversion of the Petri net diagnoser to LD for implementation on a PLC is proposed. The conversion technique leads to an LD that mimics the Petri net behavior and avoids the avalanche effect.

REFERENCES


Fig. 8. Ladder diagram for Petri net $N_{Diag}$.