Discovering Data-Aware Mode Switching Constraints to Monitor Mode Switching Decisions in Supervisory Control

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3120020, IEEE Transactions on Industrial Informatics

Discovering Data-Aware Mode Switching Constraints to Monitor Mode Switching Decisions in Supervisory Control

Mukhtar Hussain, Colin Fidge, Ernest Foo, and Zahra Jadidi

***Abstract*—In a multimode industrial control system (ICS), mode switching decisions have to follow standard operating procedures which are set for the safety of the system based on the operating limitations of equipment. A rich literature can be found on monitoring multimode systems. However, that work is mainly focused on mode identification and monitoring anomalies in the process running under each mode. Instead, we present a data-driven method for monitoring the modes' switching constraints. This work is based on state-transition matrix and decision-tree methods to discover data-driven mode switching conditions. Moreover, our approach is not limited to only threshold based condition learning. To capture data trajectory based conditions we adopt a functional data descriptors method. In practical experiments, we showed that our approach can discover anomalous mode switching decisions which can't be discovered by previous multimode process monitoring methods.**

***Index Terms*—industrial control systems, multimode processing, mode switching constraints**

I. Introduction

An Industrial Control System (ICS) can be described as a multilayer architecture of cyber and physical components to automate an industrial process [1]. The automatic control layer is responsible for the automation of the physical layer process (e.g., water treatment, or power generation and distribution) by reading sensors' data and sending commands to actuators. Meanwhile, the supervisory control (SC) layer is responsible for monitoring the ICS's operation by interacting with the control layer, i.e., gathering data and switching into different modes (e.g., start-up, shutdown, fail-safe) [1]. However, mode switching decisions are restricted based on equipment constraints, or working conditions [2]. For example, a nuclear power plant can't be switched directly to *refuelling* mode from *operation* mode, since switching to *refuelling* mode must be done through the *cold shutdown* mode. Moreover, switching from the *power operation* mode to either *cold shutdown* or the *hot shutdown* mode is conditional based on the coolant temperature [3]. Monitoring these mode switching constraints is essential because human error or a cyber-attack at the SC layer may switch the system to a constrained mode which can have serious negative consequences. For instance, the BlackEnergy malware penetrated into the SC layer of a Ukrainian power system and switched it to the shutdown mode [4].

In this paper, we present a method to discover and monitor the mode switching *conditions* at the SC layer of a multimode system. Over the years, tremendous attention has been paid to assist plant operators to monitor for any anomalous behaviour [5]. The anomaly detection methods for multimode systems can be broadly divided into two categories, data-based and knowledge-based. Data-based multimode monitoring methods are mainly concentrated on mode identification or finding anomalous behaviour in a process which consider the sequence of mode changes but does not take into consideration the mode switching constraints [6]. On the other hand building a model solely on experts' knowledge is labour intensive, time consuming, and prone to error [7]. The major bottleneck in building knowledge based monitoring methods is that experts don't explain their approach objectively; instead they provide a "why a specific decision was right" explanation [7].

Afzal *et al*. [2] were the first who have considered the problem of discovering a multimode process model with mode-reachability constraints and they proposed a method to incorporate those constraints in discovering a Hidden Markov Model. Recently, Saez *et al*. [8] proposed a method to model multimode processing systems as hybrid *automata* in which states represents a mode of operation. Both of the above mentioned methods consider mode-reachability constraints, i.e., switching to the *refuelling* mode is not allowed from the *power operation* mode in a nuclear power plant. However, these methods ignore the external environmental conditions under which a system can switch to a reachable mode, e.g., the *cold shutdown* or the *hot shutdown* modes are both reachable via the *power operation* mode but switching to either mode is conditioned based on the coolant temperature [3].

Learning mode switching conditions from data is an inductive inference problem, which includes identifying what decisions have been taken (reachable modes) and what data is used in making the mode switching decisions. The decision mining method [9] addresses a similar problem for workflow management systems. However, it is assumed there that the data recorded in logs contains complete information about events/transitions and data attributes which lead to a specific choice.

Our work applies a *decision mining* approach [9] for SC layer process monitoring with the following contributions.

- The first contribution is to annotate the ICS device logs for decision mining. We employed a functional data

M. Hussain, C. Fidge, and Z. Jadidi are with the Computer Science Department, Queensland University of Technology, QLD 4000, Australia. e-mail: {*m5.hussain, c.fidge, z.jadidi*}*@qut.edu.au*

E. Foo is with School of Information and Communication Technology, Griffith University e-mail: *e.foo@griffith.edu.au*

Manuscript received MMM DD, YYYY; revised MMM DD, YYYY.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3120020, IEEE Transactions on Industrial Informatics

description (FDD) approach [10] to incorporate a human factor in decision making based on data trajectories.
- The second contribution is to show how we can identify anomalies in mode switching decisions using the discovered model that cannot be discovered by other approaches.

II. Background

This section provides a brief discussion on the selected formalism used in this paper and related work. The assumptions we made in the context of available literature on multimode process systems are also explained.

A. Modelling Formalism

A modelling formalism can be seen as the *semantics* of a system's operation. ICSs have been commonly modelled using diverse classes of finite state automata and Petri Nets (PNs) [11]. The selection of a specific formalism depends on the application such as simulation or validation and verification of control logic. For our approach it does not matter fundamentally if either a finite state automaton or PN is adopted because mode logic assumes the system is only in one mode at a time.

In this paper, we use a Data-interpreted Petri Net (DPN) formalism which is an extended PN formalism also referred to as high level PN or predicate/transition PN [12]. A DPN can be defined as a tuple $H = (P, T, F, Y, G)$, where:

- $P = \{p_1, p_2, \ldots, p_{|P|}\}$ is a finite set of places;
- $T = \{t_1, t_2, \ldots, t_{|T|}\}$ is a finite set of transitions;
- $F \subseteq (P \times T) \cup (T \times P)$ is the mapping describing the arcs from places to transitions and vice versa;
- $Y = \{y_1, y_2, \ldots, y_k\}$ is a finite set of k state variables such that $y \in \mathbb{R}$;
- $G : T \rightarrow \mathcal{G}_Y$ is a *guard* function that assigns a *guard* $g \in \mathcal{G}_Y$ to each transition $t \in T$.

In our definition of DPN, a *guard* is $g \in \mathcal{G}_Y$ a logical expression of variables $y \in Y$ (e.g., $y \geq 10$) such that assigning values to variables in the expression, the expression either evaluates to be either `true` or `false`. The state of DPN is reflected by tokens in a place $p \in P$. A token is expressed as a pair (M, A), where M is a marking function $M : P \rightarrow N$ for PN (P, T, F) which represents the number of tokens residing inside each place and A represent values assigned the state variables Y. As stated earlier, that a system can only be in one mode at a time. Therefore, the PN structure is 1-bounded or safe, i.e., $M : P \rightarrow \{0, 1\}$. Moreover, a transition firing is valid if the transition's input place contains a token, and the assigned guard $G(t)$ evaluates `true` based on the values assigned (A) to variables Y.

The DPN structure discussed above does not contain greater expressive power over PNs [11]. The reason for the selection of this extended PN formalism (DPN) is it provides modelling convenience such that the underlying place/transition net (P, T, F) can capture the mode reachability constraints such that each state represents a mode. Meanwhile, a *guard* $G(t)$ adds an extra condition for firing a transition $t \in T$ which captures the mode switching constraints based on continuous state variables. Moreover, existing methods from the *system identification* field can be employed for model discovery.

B. Assumptions

Based on the processing type, multimode systems can be described as either continuous processing or batch processing [6]. The SC process in continuous processing systems such as a nuclear power plant can be described as a sequence of transitions from one mode to another. There is no transition back to the same mode, i.e., a process under the same mode is not repeated again until the whole process is restarted. On the other hand, states of a process executing under the same mode can be repeated in a batch processing system. For example, in an automated paint shop multiple modes are defined for different paint applications, such as base-coat paint and clear-coat paint modes. A painting process under the same painting mode can be repeated for multiple batches.

Batch information is recorded with the logs for quality control purposes [13]. Mode information can also be recorded with the ICS device log using an appropriate data acquisition method [14]. Moreover, extensive research in the literature can be found on mode identification [6]. Therefore, to limit the scope of this paper to discover conditions for switching modes in multimode systems (continuous processing or batch processing) we assumed that the *mode* and *batch* information is available within the recorded logs as shown in Table I.

III. Model Discovery Method

This section explains our approach for discovering mode switching constraints in an ICS. First, basic discrete structure notations are provided. Then we explain our ICS log processing method such that the annotated logs can be used for *decision mining*. Finally, the method for mining mode switching constraints is explained.

A. Preliminaries

In this section, formal notations are provided that are used to explain our device log annotation method for discovering a model of mode switching constraints.

For a set X, Let $\mathbb{P}X$ be the powerset of X, i.e., the set of all the subsets of X. Let $H = \langle \mathrm{h}_1, \ldots, \mathrm{h}_\mathrm{n} \rangle$ be a sequence of n values. The notation $\#H$ denotes the length of sequence H, function $head(H)$ returns the first element of sequence H, $tail(H)$ returns sequence H less its first element, and $last(H)$ returns the last element of sequence H. The operation $S \upharpoonright H$ restricts sequence H to those items whose index is in the set S, where $S \in \mathbb{P}(\mathbb{N} \setminus \{0\})$ is a finite, non-empty set of positive integers. Let $\langle\rangle$ and $\{\}$ represent an empty sequence and an empty set. Let $H_1 \frown H_2$ denote concatenation of sequences H_1 and H_2. Let $A \cup B$ denote union of sets A and B.

B. Pre-processing

This section explains our method of processing ICS logs to enable their use for discovering a PN model and its transition enabling conditions (*guards*). Let the unprocessed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3120020, IEEE Transactions on Industrial Informatics

TABLE I: A snippet of ICS device logs

Time	I_1	I_2	I_3	O_1	O_2	O_3	Batch	Mode
8/07/2020 12:01:37 PM	35.405	0.245	0.041	1	0.170	0	0	1
8/07/2020 12:01:38 PM	35.169	0.106	-0.098	1	0.213	0	0	1
8/07/2020 12:01:39 PM	35.294	60.983	34.748	0	0.206	0	1	2
8/07/2020 12:01:40 PM	35.326	60.804	34.644	0	0.192	0	1	2
⋮								
8/07/2020 12:21:37 PM	35.239	66.700	27.770	0	0.000	0	1	2
8/07/2020 12:21:38 PM	35.219	67.025	27.753	0	0.000	0	1	2
8/07/2020 12:21:39 PM	35.070	67.246	27.629	0	0.215	0	2	2
8/07/2020 12:21:40 PM	35.080	67.376	27.637	0	0.202	0	2	2
⋮								
8/07/2020 1:44:12 PM	35.140	61.673	13.823	0	0.203	0	6	2
8/07/2020 1:44:13 PM	34.967	61.493	13.679	0	0.207	0	6	2
8/07/2020 1:44:14 PM	35.091	78.303	13.783	0	0.216	0	6	3
8/07/2020 1:44:15 PM	35.120	76.346	13.807	0	0.212	0	6	3
⋮								
8/07/2020 1:44:31 PM	35.035	58.864	13.889	0	0.000	0	6	3
8/07/2020 1:44:32 PM	35.223	57.490	13.626	0	0.000	0	6	2
8/07/2020 1:44:33 PM	35.027	62.449	13.679	0	0.000	0	6	2
⋮								
8/07/2020 4:32:18 PM	21.784	61.720	26.800	0	0.283	0	14	2
8/07/2020 4:32:19 PM	21.900	61.629	26.788	0	0.271	0	14	2
8/07/2020 4:32:20 PM	21.899	61.433	26.777	0	0.300	0	14	3
8/07/2020 4:32:21 PM	21.793	61.370	26.759	0	0.269	0	14	3
⋮								
8/07/2020 4:35:11 PM	21.621	67.445	26.766	0	0.305	0	14	3
8/07/2020 4:35:12 PM	21.585	67.625	26.499	0	0.293	0	14	2
8/07/2020 4:35:13 PM	21.707	67.931	26.347	0	0.311	0	14	2
⋮								
8/07/2020 6:21:37 PM	10.676	65.228	21.280	0	0.301	0	20	2
8/07/2020 6:21:38 PM	10.549	65.360	21.174	0	0.340	0	20	2
8/07/2020 6:21:39 PM	10.511	0.836	-0.126	0	0.323	1	21	4
8/07/2020 6:21:40 PM	10.719	0.887	0.031	0	0.353	1	21	4

device log as shown in Table I be a series of finite sequences $\mathcal{L} = \langle D_1, \ldots, D_m \rangle$, where each $D_i \mid 1 \leq i \leq m$ equal $\langle d_1, \ldots, d_n \rangle$ represents a sequence (column in Table I) of n values. Let the first sequence D_1 represent timestamps for each record, then each sequence $D_2, \ldots, D_{m-2}$ represents an ICS physical device's (sensors and actuators) status records, D_{m-1} represents the batch/process instance information, and D_m represents mode information. Let us suppose that sequences $D_2, \ldots, D_{l+1}$ represent the status of input devices (sensors) and $D_{l+2}, \ldots, D_{m-2}$ represent the status of output devices (actuators). In the following paragraphs we explain two custom functions which are used in Algorithm 1 for ICS device logs annotation.

1) Extract Mode Transition Information): The device logs as shown in Table I contains system's mode information at any time instance. The first step is to identify the change in *mode* or *batch process* from the given sequences in the device logs, we define a custom function $\mathcal{F}(V, \kappa, i)$ as per Definition 1 below. Here V denote a sequence of discrete numbered data and κ denote the initial information. This function returns the set of indices i where a change of value occurred in V.

The second step is to extract mode transition event from the 'mode' sequence of device logs (Table I) as event pair as shown in Table. II such that its first element of a pair represents previous mode and the second element of pair represents current mode at an index i. A formal procedure for the extraction of mode transition information is provided in Algorithm 1.

2) Extract Data Attributes: Decision at the SC of an ICS varies, it can be based on the certain threshold value or trend

$$\mathcal{F}(V, \kappa, i) \stackrel{\text{def}}{=} \begin{cases} \{\} & \text{if } V = \langle\rangle \\ \mathcal{F}(tail(V), head(V), i+1) & \text{if } \kappa = head(V) \\ \{i\} \cup \mathcal{F}(tail(V), head(V), i+1) & \text{otherwise} \end{cases} \tag{1}$$

$$\texttt{FDD}(V, U, \delta) \stackrel{\text{def}}{=} \begin{cases} \langle\rangle & \text{if } U = \langle\rangle \\ \langle \texttt{NAN} \rangle \frown \texttt{FDD}(V, tail(U), \delta) & \text{if } head(U) - \delta \leq 0 \\ \langle (V(head(U)) - V(head(U) - \delta))/\delta \rangle \frown \texttt{FDD}(V, tail(U), \delta) & \text{otherwise} \end{cases} \tag{2}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3120020, IEEE Transactions on Industrial Informatics

TABLE II: An example event log derived from ICS logs for discovering mode switching conditions

Time	Events	Data Attributes								
		I_1	I_2	I_3	O_1	O_2	O_3	FDD_I_1	FDD_I_2	FDD_I_3
8/07/2020 12:01:39 PM	(1,2)	35.294	60.983	34.748	0	0.206	0	0.001	0.000	-0.260
8/07/2020 12:21:39 PM	(2,2)	35.070	67.246	27.629	0	0.215	0	-0.005	0.002	-0.270
				⋮						
8/07/2020 1:44:14 PM	(2,3)	35.091	78.303	13.783	0	0.216	0	-0.017	1.623	-0.278
8/07/2020 1:44:32 PM	(3,2)	35.223	57.490	13.626	0	0.000	0	0.021	-1.283	-0.279
				⋮						
8/07/2020 4:32:20 PM	(2,3)	21.899	61.433	26.777	0	0.300	0	0.012	0.001	-0.131
8/07/2020 4:35:12 PM	(3,2)	21.585	67.625	26.499	0	0.293	0	0.018	0.000	-0.254
				⋮						
8/07/2020 6:21:39 PM	(2,4)	10.511	0.836	-0.126	0	0.323	1	-0.019	-6.525	-0.269

in sensors' data [15]. ICS devices' status at the mode transition instances are only useful to mine threshold based conditions using decision mining approach. Therefore, a FDD approach [10] is used to capture the data trajectory or trend as a discrete observation for decision mining. The selection of FDD method is a challenge because of the presence of noise in the data. The aim is to neither overfit nor underfit the data for the classification method. We use the 'slope' FDD method in our approach. However, another challenge is to estimate the time taken by an operator to decide on an action which can vary from a few seconds to a few minutes [16]. Thus, the related information can be extracted in the data recorded few seconds prior to decision making moment. We selected a 10 seconds interval of information prior to each mode transition instance in the device log as the window for processing.

Let V be a sensor's data sequence from the device logs and U denote a sequence of mode transition indices (from Function 1). We define a custom function $\texttt{FDD}(V, U, \delta)$ as per Definition 2 on page 3, which returns a sequence of sensor's FDD values prior to mode transition instances. Here, δ is the window size (time duration $\times$ sampling rate) for processing information prior to the mode transition.

Algorithm 1 (Transform Device Logs into Event Logs): In Algorithm 1, variable $\mathbf{E}$ and $\mathbf{R}$ is initialised on line 1 to contain sequence of events and series of processed data attribute sequences. The processing starts from line 2 by getting the size of device logs. The 'Mode' and 'Batch' sequences are extracted from device logs on lines 3 and 4 respectively. On line 5, Definition 1 is applied on the 'Mode' sequence V_1 to extract set of indices U_1 that represents changes in mode, for instance, row numbers 3, 11, and 14 in Table I (see page 3). Similarly, on line 6, a set of indices U_2 that represents processing of new batch process is extracted using Definition 1, for instance, row numbers 3 and 7 in Table I. On line 7, a set of indices U is formed which is the union of U_1 and U_2, which combines indices represents change in batch and mode such that no index is repeated ($U = \{3, 7, 11, 14, \ldots\}$).

On line 9, a tuple of mode transition events is formed such that its first element represents the previous mode and the second element represents the current mode at index $u \in U$. For example, event (1,2) in Table II refers mode change at row number 3 of the 'Mode' sequence in Table I. This procedure is repeated for all indices of set U using a 'for' loop on line 8 of Algorithm 1 to make a sequence of events as shown in Table II. On line 12, a device's data record at the mode transition event is extracted by restricting the data sequence D_i from device logs to those items whose indices are in U and a series of sequences is formed by concatenating the sequences. The process in line 12 is repeated for each device using a 'for' loop in line 11. On line 15, FDD are evaluated using Definition 2. The process in line 15 is repeated for each sensor/input device using a 'for' loop in line 14 (columns 2 to 4 in Table. I). On line 17, a 'Time' sequence for the event logs is extracted by restricting the 'Time' sequence from device logs to those items whose indices are in U. Finally, an event log $\mathcal{L}_E$ as shown in Table II is formed in line 18 by concatenating the extracted information, i.e., the timestamp sequence, events sequence and series of data attributes.

Algorithm 1: Processing Device Logs for PN and Guards Discovery

Input: Device Logs $\mathcal{L}$
Output: Event Logs $\mathcal{L}_E$
// initialisation
1 $\mathbf{E}, \mathbf{R} \leftarrow \langle\rangle$
// processing
2 $m \leftarrow \#\mathcal{L}$
3 $V_1 \leftarrow \mathcal{L}(m)$
4 $V_2 \leftarrow \mathcal{L}(m-1)$
5 $U_1 \leftarrow \mathcal{F}(V_1, head(V_1), 0)$
6 $U_2 \leftarrow \mathcal{F}(V_2, head(V_2), 0)$
7 $U \leftarrow U_1 \cup U_2$
8 **foreach** $u \in U$ **do**
9 $\quad \mathbf{E} \leftarrow \mathbf{E} \frown \langle (V_1(u-1), V_1(u)) \rangle$
10 **end**
11 **foreach** $D_i \in \mathcal{L} \mid i \in \{2, \ldots, m-2\}$ **do**
12 $\quad \mathbf{R} \leftarrow \mathbf{R} \frown \langle U \restriction D_i \rangle$
13 **end**
14 **foreach** $D_x \in \mathcal{L} \mid x \in \{2, \ldots, k+1\}$ **do**
15 $\quad \mathbf{R} \leftarrow \mathbf{R} \frown \langle \texttt{FDD}(D_x, U, \delta) \rangle$
16 **end**
17 $T \leftarrow U \restriction head(\mathcal{L})$
18 $\mathcal{L}_E \leftarrow \langle \mathbf{T} \rangle \frown \langle \mathbf{E} \rangle \frown \mathbf{R}$
19 **return**$(\mathcal{L}_E)$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3120020, IEEE Transactions on Industrial Informatics

In the event log as shown in Table II, the first column represents a timestamp for each mode transition event in the second column. The remaining columns represent status of ICS physical devices at mode transition instances. The first event (1,2) in Table II refers mode transition observed in the third and fourth row of Table I. Here columns I_1, I_2, and I_3 represent the inputs, i.e., sensor readings and columns O_1, O_2, and O_3 represent outputs from the controller to the actuators. Columns FDD_I_1, FDD_I_2, and FDD_I_3 represent sensors' data trajectories at the mode transition instant. The data attribute columns of Table II or $\mathbf{R}$ can be referred to as the system's state variables represented by Y in our DPN model such that each row of Table II of data attribute columns represent the values assigned (A) to system's state variables at a particular state-transition instance.

Having created the event logs in this way, the next step is to discover the conditions for switching modes, which is described in the following section.

C. Model Discovery

This section explains our approach for discovering data-aware mode switching constraints.

1) Mode Transition Model: The aim of discovering mode/state transition model is two fold. The first aim is to capture mode-reachability, i.e., whether a transition from one mode to another mode is possible or not. The second aim is to capture mutually exclusive modes, i.e., choices of transitions from a certain mode to mine data-driven conditions using decision trees. An existing PN model discovery method [17] from the system identification field can be adopted to discover a PN model from the sequence of events derived in the previous section. In this paper, we use a simple approach to discover the PN model which is based on a state-transition matrix [18]. A brief overview of state transition matrix method is provided as follows.

$$S = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3}$$

A state transition matrix S is an $M \times M$ matrix where M represents the number of modes. Each element s_{ij} of matrix S can either be 0 or 1. If s_{ij} is 0, it represents no transition is allowed from mode i to mode j and if s_{ij} is 1, the transition is allowed. Let us suppose that a given system has four operating modes. Based on the recorded behaviour in Table II following mode transitions are allowed, a) from mode 1 (start-up) to mode 2 (processing), b) from mode 2 to mode 3 (fail-safe) and 4 (shutdown), c) from mode 3 to mode 2, and d) a transition back to the same mode is allowed only for mode 2, which represents that the production process is restarted for a new batch. The state transition matrix which explains the above mode transition constraints can be expressed in equation 3, and a corresponding PN model is shown in Fig. 1.

The algorithm for extracting such a state transition matrix from event logs (Table II) is provided in Sec. III-C3.

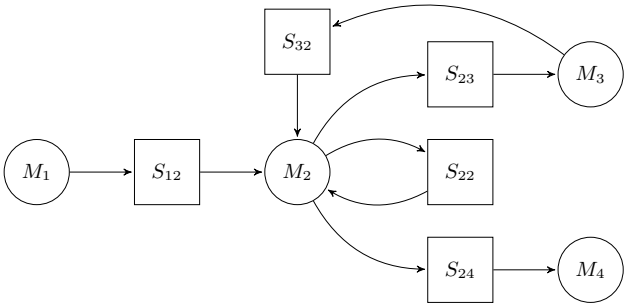

Fig. 1: A Petri Net mode transition model

2) Mode Switching Conditions: Provided the mode-reachability model (discussed in Sec. III-C1), learning mode switching conditions is similar to the decision mining approach [9] discussed in the *process mining* field. Therefore, we abstracted such a function as `generateTransitionGuards`$(Mdl, \mathcal{L}_E)$ which takes a PN model and event logs, and returns a *guard* function G. Let us assume that the guard function G is an $M \times M$ matix of *guards*, such that an element $G(i,j)$ (also denoted by g_{ij}) is a *guard* for transition s_{ij}. We summarise the *guards* discovery method performed by the function `generateTransitionGuards` in the following paragraphs.

For Mutually Exclusive Transitions: The *decision mining* approach employs the C4.5 algorithm [19] to discover conditions for transitions linked to a decision instance in a PN model. For example, if there is a token in place M_2 in Fig. 1 then any one of the transitions S_{22}, S_{23} or S_{24} can fire. The C4.5 algorithm classifies the categorical response variable (events) based on the predictor variables (data attributes) as shown in Fig. 2. A subset of event logs which contains only required events (S_{22}, S_{23} and S_{24}) information and corresponding data attributes is fed to the decision tree algorithm. The discovered guards conditions for each transition are mutually exclusive, thus only one of the reachable transitions (S_{22}, S_{23}, and S_{24}) can be fired. This procedure is repeated for all the decision instances in a PN model in the function `generateTransitionGuards`.

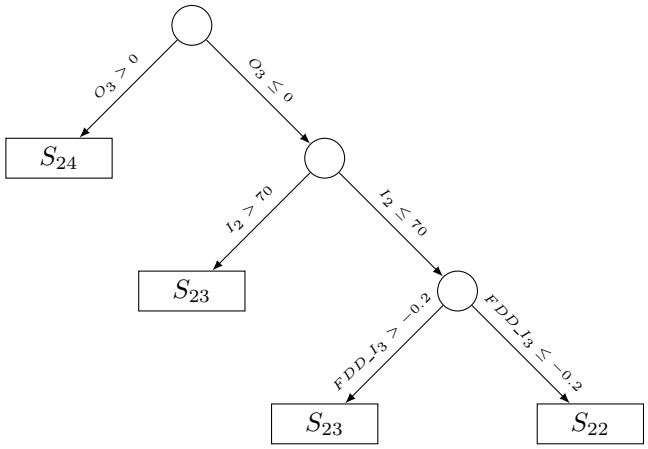

Fig. 2: An example of a decision tree for mutually exclusive transitions

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3120020, IEEE Transactions on Industrial Informatics

A decision tree such as shown in Fig. 2 represents the conditions for a possible outcome, being a disjunction of the conjunction of expressions in general. The conjunction of expressions along the path descending from the root to a leaf node must evaluate to `true` for an outcome to occur. For example, all the conditions for the occurrence of transition S_{22} must be satisfied, i.e., $g_{22} \implies (O_3 \leq 0 \wedge I_2 \leq 70 \wedge FDD\_I_3 \leq -0.2)$. A disjunction of expressions is possible for an outcome on the alternative path. For example, switching to mode 3, i.e., firing transition S_{23} is possible if conditions on either path are satisfied which is disjunction of the conjunction of expressions, i.e., $g_{23} \implies (O_3 \leq 0 \wedge (I_2 > 70 \vee (I_2 \leq 70 \wedge FDD\_I_3 > -0.2)))$. Whereas, the condition for the occurrence of transition S_{24} is simple, i.e., $g_{24} \implies (O_3 > 0)$.

For Independent Transitions: The decision tree algorithm is only useful for mining conditions of mutually exclusive transitions. Hence, the *guard* condition for a valid independent transition is set to be `true`. For example, if there is a token in place M_1 in Fig. 1 then only S_{12} can be fired. Therefore, enabling condition of transition S_{12} is set to be `true`, i.e., $g_{12} \implies$ `true`

Our algorithm for discovering a model of modes' switching conditions is given as follows.

Algorithm 2: Formulation of mode switching conditions

Input: Event Logs ($\mathcal{L}_E$)
Output: Petri Net Model (*Mdl*), Guard Function (G)
`// initialisation`
1 $S \leftarrow 0_{M \times M}$
`// processing`
2 $L \leftarrow \mathcal{L}_E(2)$
3 $\widetilde{L} \leftarrow$ `unique`(L)
4 **foreach** $l \in \widetilde{L}$ **do**
5 $\quad S(l) \leftarrow 1$
6 **end**
7 *Mdl* $\leftarrow$ `buildPNmodel`(S)
8 $G \leftarrow$ `generateTransitionGuards`$(Mdl, \mathcal{L}_E)$
9 **return**(Mdl, G)

3) Algorithm 2 (Discovering Mode Switching Constrains): In Algorithm 2, an $M \times M$ matrix is generated whose elements all contain '0' on line 1. Here M represents the number of modes. On line 2, the event/mode transition information sequence from the series of event logs is extracted, i.e., column 2 of Table. II. Each pair in the mode transition sequence represents the initial and subsequent mode. On line 3, the set of unique pairs are extracted. This information is used to generate a state transition matrix from line 4 to 6. For example, the first pair (1,2) of event sequence in Table II represents the transition from mode 1 to mode 2. The element $S(1,2)$ (also represented as s_{12}) of the state transition matrix S that represents a transition from mode 1 to mode 2 is set as 1. This procedure on line 5 is repeated for each element of the event sequence using a 'for' loop in line 4. On line 7, a corresponding PN model is generated and passed to function `generateTransitionGuards`$(Mdl, \mathcal{L}_E)$ with event logs to discover *guard* conditions on line 8.

IV. Monitoring Mode Switching Decisions

In this section we present our method for identifying violations in mode switching decisions as an illustration of one possible application of the discovered models.

Let an event log as shown in Table II be a series of finite sequences $\mathcal{L}_E = \langle F_1, \ldots, F_p \rangle$, where $F_i \mid 1 \leq i \leq p$ equal $\langle f_1, \ldots, f_q \rangle$, represents a sequence (column) of q values. The first sequence F_1 represents the timestamp for each event (mode transition) in the second sequence F_2, then the remaining sequences $F_3, \ldots, F_p$ represent data attributes. Let j, where $1 \leq j \leq q$, be the index of a specific value in each column F_i of an event log, such that the corresponding value is denoted F_{ij}. Then we define the sequence all such values from the event logs as a "row" which represents the value of system's state variables at that transition's instant:

$$R_j = \langle v : F_{ij} | 3 \leq i \leq p \rangle.$$

Algorithm 3: Conformance analysis to identify mode switching violations

Input: Event Logs ($\mathcal{L}_E$), Transition Matrix (S), Guards Function (G)
Output: Violations C
`// initialisation`
1 $C \leftarrow \langle\rangle$
`// processing`
2 $E \leftarrow \mathcal{L}_E(2)$
3 **foreach** $R_j \in \mathcal{L}_E \mid j \in \{1, \ldots, \#E\}$ **do**
4 $\quad$ **if** $S(E(j)) = 0$ **then**
5 $\quad\quad C \leftarrow C \frown \langle 2 \rangle$
6 $\quad$ **else**
7 $\quad\quad$ **if** $g_{E(j)}(R_j)$ **then**
8 $\quad\quad\quad C \leftarrow C \frown \langle 0 \rangle$
9 $\quad\quad$ **else**
10 $\quad\quad\quad C \leftarrow C \frown \langle 1 \rangle$
11 $\quad\quad$ **end**
12 $\quad$ **end**
13 **end**

Algorithm 3 (Conformance Analysis): Algorithm 3 describes our procedure for monitoring mode switching conditions by labelling them as one of the following cases. Label '0' represents that a transition is valid based on the discovered PN model and the *guard* for the transition is also evaluated to be `true`, '1' represents that a transition from one mode to another mode is possible based on the PN model, however, the *guard* condition is evaluated to `false`, and '2' represents that transition is not allowed from previous mode to new mode.

Algorithm 3 takes ICS logs processed using Algorithm 1, and the transition matrix and the *guards* discovered in Algorithm 2 as the basis for deciding correctness of new logs. On line 1, a labelling sequence C is initialised to monitor the violations. The procedure starts by extracting the information of mode switching events in line 2. In line 4, it is evaluated if

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3120020, IEEE Transactions on Industrial Informatics

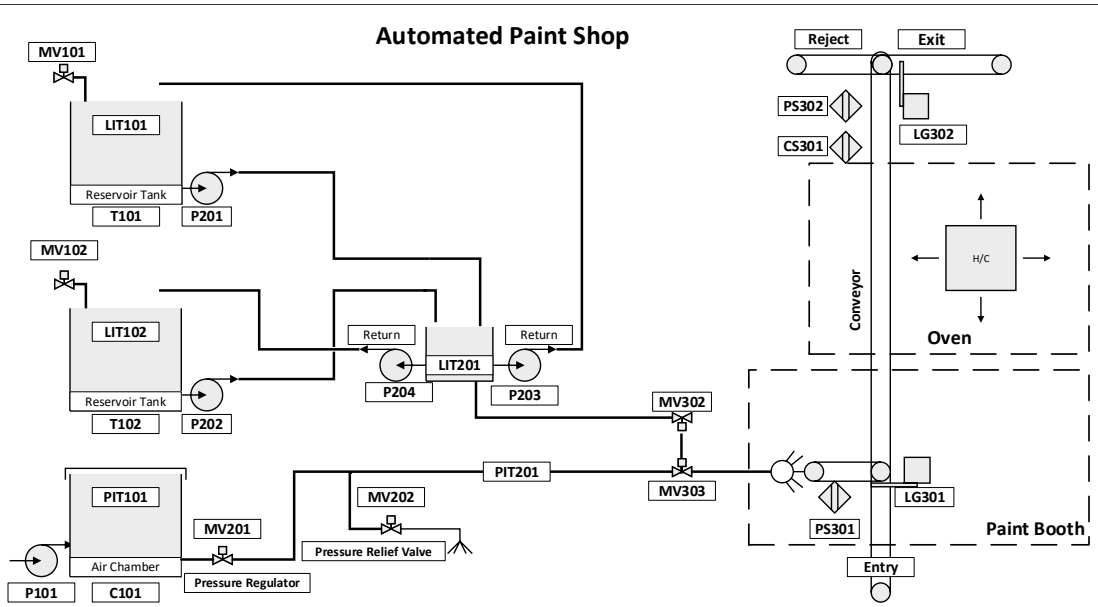

Fig. 3: Automated Paint Shop Schematic Diagram

the transition to that mode is allowed using an 'if' condition based on the mode transition matrix. If the transition is allowed, then on line 7 the *guard* expression is evaluated for that mode transition using the processed data against that transition in the event log. In the case of no violation line 8 will be executed, otherwise line 10 will be executed to report a violation of the *guard* expression. This process is repeated for all the mode transitions using the 'for' loop in line 3. The result is a list of "violations" C noting transitions recorded in the event log which occurred when the corresponding Boolean *guard* in our discovered model was false. Although such transitions would be considered acceptable in a simple model reachable model [2], [8], our analysis is thus shows when such transitions have occurred with data values outside their usual ranges.

V. Case Study

This section shows how our method can be used to create DPN model of an ICS. The industrial process considered for evaluation is explained first. The discovered model is then evaluated based on its ability to identify anomalous behaviour.

A. Experimental Setup

Our case study is based on an automated paint shop (APS). We simulated the functionality of the paint shop based on the features outlined by Hlupic and Paul [20] in SIMULINK using the Simscape tool. An APS is a part of many manufacturing industries, such as, the automotive and furniture industries. Components for each product are produced in flexible manufacturing cells, and after paint coating are assembled to create a final product.

The overall schematic of the system is shown in Fig. 3. A large overhead conveyor chain is used to transport parts to different processing areas. The first stage is the entry where parts are loaded to be painted. Parts are first loaded on the conveyor belt which leads them to the paint booth. The paint-coated parts are then transported to the oven, where they are baked in order to preserve the coating. After this, the parts go to the unloading area, where they are first checked for quality control and sorted accordingly.

There are two paint reservoir tanks T101 and T102 that contain white and black coloured paint. The air chamber C101 holds compressed air to atomise the paint. The paint booth has a small container T201 to hold paint for processing. A pressure regulator valve maintains the pressure in the pipeline for smooth paint distribution. A component can be either painted white or black. Two processing modes M_3 and M_4 are defined based on the colour to be painted. Before switching from one processing mode to another processing mode, or the 'off' mode M_1, the system undergoes temporary halt or emergency mode M_5 for cleaning paint residue in the spraying gun. Production is also halted if the air pressure in the pipeline increases above the normal limit or the spraying nozzle is broken. A 'standby' mode M_2 represents that sensors are calibrated and processing can be started.

For simulation it is assumed that the company produces 20 components in a day. Normally, the first 10 components are painted white and the other 10 are painted black. However, all the produced components can be painted either white or black based on demand. Each component is treated as a different batch and the information is logged in a variable named NoC (number of component). An operator can override the normal routine and this information is recorded in a variable named 'override' in the logs. The simulation starts with the system in *off* mode M_1. It takes 50 seconds to transition from *off* mode to *standby* mode. A transition from *standby* mode M_2 to one of the processing modes, i.e., M_3 and M_4, is allowed. In normal processing the first batch of components must be processed under mode M_3. However, transitions from mode M_3 to M_4 and vice versa are not allowed because the system can't change paint colour without first cleaning the spraying

nozzle. Therefore, a transition from M_3 to M_4 is possible through temporary halt mode M_5 to clean any residue paint in the spray gun or tank T201. M_5 is a *fail-safe* mode to halt production if anything goes wrong. For simulation we set two conditions to switch from production modes M_3 and M_4 to *fail-safe* mode. The first condition was if the pressure in the pipelines (PIT201) is higher than a certain range in which case it should be switched to the emergency mode and halt the production until the pressure is released using a pressure relief valve. The second condition for switching to halt mode was set for if the nozzle of the spraying gun is broken, which is based on the rate of flow out from tank T201 based on the trend in the level sensor's reading, i.e., FDD_LIT201.

B. Dataset

The data recorded from the SIMULINK model of the APS imitates device logs recorded in a real ICS. The sampling rate for recording the dataset was set to be 1 sample/sec, a common criterion used in most of the publicly available datasets [21]. The physical system consists of sensors and actuators as shown in Fig. 3. Here sensors "LIT", "PIT", "PS", and "CS" represent "level indicator", "pressure indicator", "proximity sensor", and "colour sensor" respectively. Moreover, actuators "MV", "P" and "LG" represent "motorised valve", "pump", and "lever gate" respectively. We generated two dataset files from the SIMULINK model. The first file contains the normal behaviour of the system and the second file contains anomalous mode switching.

In the normal training dataset file simulation a separate simulation was conducted 30 times. In these 30 simulations, 10 instances included both processing modes M_3 and M_4 and 20 instances included for only one processing mode, half for each mode M_3 and M_4. Moreover, the probability of the spraying gun nozzle being broken and pressure PIT201 in the pipes exceeding the set limit during a simulation process was derived from a Beroulli distribution such that the probability was set to be '0.5'. On the other hand, *six* anomalous mode switching instances were recorded during the simulation of the anomalous (test) dataset. The detail of simulated anomalous behaviours at the SC layer is provided in Table III.

TABLE III: Description of anomalous behaviours simulated

No.	Anomalous Decision Behaviour
1.	Switch the system from mode M_3 to M_4.
2.	Switch the system from mode M_3 to M_2.
3.	Turning off the system (switch to mode M_1 from M_5) before processing all the components, i.e., (NoC=15).
4.	Turning off the system (switch to mode M_1 from M_5) without emptying the paint tank T201, i.e., LIT201$\geq$2.
5.	Switching to emergency mode M_5 from M_4 under normal conditions, i.e., PIT201$\leq$60 and FDD_LIT201$\geq$-0.2
6.	Keep processing in M_1 with a broken paint gun nozzle.

C. Model of Expected Behaviour

We developed MATLAB scripts to implement the data processing, model generation, and conformance analysis, i.e., Algorithms 1, 2, and 3. Both dataset files (normal and anomalous) were processed using the same Algorithm 1. The mode switching constraints model was discovered using annotated normal logs which was then used for conformance analysis with the processed anomalous logs. The model discovered from the logs is shown in Fig. 4.

For *guard* discovery, to avoid overfitting and underfitting the model, the tuning parameters of the C4.5 algorithm in the decision mining phase were set as follows. Minimum instances per leaf for a node split were set to be 4 because any transition from one mode to another in our dataset was recorded at least times 10 times. The confidence level was set to be 95%. Moreover, the significance of the predictor variable or data sequence were set as follows. The most significant variables were set to be the inputs to automated control layer, i.e., sensors' readings from the plant and FDD of input data, and the least significant variables were set to be the inputs to the actuators. This was done to reflect the assumption that mode changes in an ICS are primarily triggered by operator actions based on monitoring sensors and production plan [5]. The selection of above mentioned parameters was based on the theoretical construct of estimating satisfactory accuracy in the presence of noise as discussed by Breiman *et al*. [22, Chapter 3].

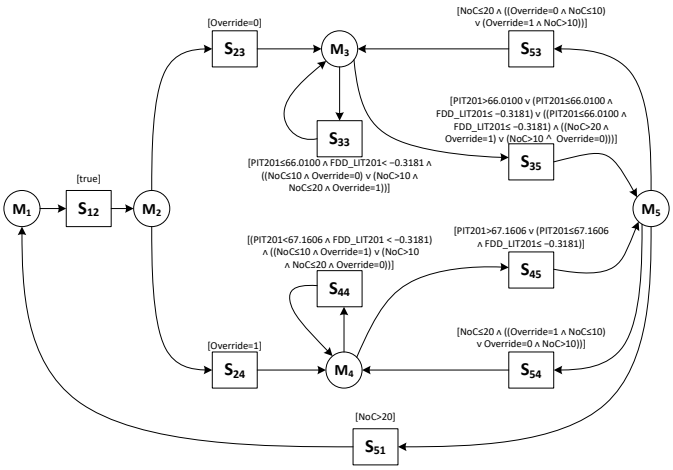

Fig. 4: Discovered DPN mode switching model

D. Conformance Checking Results

We performed conformance analysis using Algorithm 3 to identify which mode switching decision in new logs doesn't align with the discovered model (shown in Fig. 4). Based on the description of anomalous behaviour provided in Table III, first and second mode switching decisions were found to violate the mode-reachability constraints, i.e., transition S_{34} from mode M_3 to mode M_4 transition S_{32} from mode M_3 to mode M_2. Both transition were invalidated based on the place/transition net system of the discovered DPN model as shown in Fig. 4.

The anomalous decision numbers 3, 5, and 6 were identified as anomalous based on the *guard* conditions for the transitions. Anomalous decision number 4 remained unidentified by our discovered model because accurate *guard* conditions were not discovered for the transition S_{51}. Decision trees extract the

TABLE IV: Designed vs Discovered Mode Transition Conditions

Transitions	Guard Conditions	
	Designed	**Discovered**
S_{12}	–	$\texttt{true}$
S_{23}	$\texttt{Override} = 0 \wedge \texttt{NoC} \leq 20$	$\texttt{Override} = 0$
S_{24}	$\texttt{Override} = 1 \wedge \texttt{NoC} \leq 20$	$\texttt{Override} = 1$
S_{33}	$\texttt{PIT201} \leq 70 \wedge \texttt{FDD\_LIT201} > -0.2 \wedge ((\texttt{NoC} \leq 10 \wedge \texttt{Override} = 0) \vee (\texttt{NoC} > 10 \wedge \texttt{NoC} \leq 20 \wedge \texttt{Override} = 1))$	$\texttt{PIT201} \leq 66.0100 \wedge \texttt{FDD\_LIT} > -0.3181 \wedge ((\texttt{NoC} \leq 10 \wedge \texttt{Override} = 0) \vee (\texttt{NoC} > 10 \wedge \texttt{NoC} \leq 20 \wedge \texttt{Override} = 1))$
S_{35}	$\texttt{PIT201} > 70 \vee \texttt{FDD\_LIT201} \leq -0.2 \vee (\texttt{NoC} > 10 \wedge \texttt{Override} = 0) \vee (\texttt{NoC} > 20 \wedge \texttt{Override} = 1))$	$\texttt{PIT201} > 66.0100 \vee (\texttt{PIT201} \leq 66.0100 \wedge \texttt{FDD\_LIT201} \leq -0.3181) \vee((\texttt{PIT201} \leq 66.0100 \wedge \texttt{FDD\_LIT201} > -0.3181) \wedge ((\texttt{NoC} > 10 \wedge \texttt{Override} = 0) \vee (\texttt{NoC} > 20 \wedge \texttt{Override} = 1)))$
S_{44}	$\texttt{PIT201} \leq 70 \wedge \texttt{FDD\_LIT} > -0.2((\texttt{NoC} \leq 10 \wedge \texttt{Override} = 1) \vee (\texttt{NoC} \geq 10 \wedge \texttt{NoC} \leq 20 \wedge \texttt{Override} = 0))$	$\texttt{PIT201} \leq 67.1606 \wedge \texttt{FDD\_LIT201} > -0.3181 \wedge ((\texttt{NoC} \leq 10 \wedge \texttt{Override} = 1) \vee (\texttt{NoC} > 10 \wedge \texttt{NoC} \leq 20 \wedge \texttt{Override} = 0))$
S_{45}	$\texttt{PIT201} > 70 \vee \texttt{FDD\_LIT201} \leq -0.2 \vee \texttt{NoC} > 20$	$\texttt{PIT201} > 67.1606 \vee (\texttt{PIT201} \leq 67.1606 \wedge \texttt{FDD\_LIT201} \leq -0.3181) \vee(\texttt{PIT201} \leq 67.1606 \wedge \texttt{FDD\_LIT201} > -0.3181 \wedge \texttt{NoC} > 20)$
S_{51}	$\texttt{NoC} > 21 \wedge \texttt{PIT101} < 0.5 \wedge \texttt{LIT201} < 0.5$	$\texttt{NoC} > 20$
S_{53}	$(\texttt{PIT201} < 70 \wedge \texttt{Override} = 0 \wedge \texttt{NoC} \leq 10) \vee(\texttt{Override} = 1 \wedge \texttt{NoC} > 10 \wedge \texttt{NoC} \leq 20)$	$\texttt{NoC} \leq 20 \wedge ((\texttt{Override} = 0 \wedge \texttt{NoC} \leq 10) \vee(\texttt{Override} = 1 \wedge \texttt{NoC} > 10))$
S_{54}	$(\texttt{PIT201} < 70 \wedge \texttt{Override} = 1 \wedge \texttt{NoC} \leq 10) \vee(\texttt{Override} = 0 \wedge \texttt{NoC} > 10 \wedge \texttt{NoC} \leq 20)$	$\texttt{NoC} \leq 20 \wedge ((\texttt{Override} = 1 \wedge \texttt{NoC} \leq 10) \vee\texttt{Override} = 0 \wedge \texttt{NoC} > 10))$

conditions as a "disjunction of conjunctions of expressions". The limitations of using decision trees for discovering the data-driven conditions is that the approach is not useful for learning conjunctions of multiple expressions. On the other hand other classification methods such as k-nearest neighbour can't be used to infer the data-driven rules. That is why anomalous decision number 4 remained unidentified. This limitation is inherited to our work with the use of decision trees which we acknowledge.

Multimode process monitoring with mode-reachability constraints method [2] can detect only anomalous decisions numbers 1 and 2. On the other hand, dynamic principal component analysis-based multimode process monitoring method [6] was unable to detect any anomalous mode switching conditions.

E. Discussion

This section provides a brief discussion on the precision of our discovered models. The precision of our devised method of discovering mode switching constraints is provided based on a comparison with the designed system which serves as an oracle. The mode reachability constraints (i.e., transition from one mode to another) were discovered accurately (as designed) by the state transition matrix approach as represented by the place/transition net of DPN model shown in Fig. 4. A comparison of the designed and discovered *guards*, i.e., transition enabling conditions is provided in Table IV.

The first transition S_{12} shows the system is turned *on*. There was no condition set for this transition. On the other hand, it can be observed for transitions S_{23}, S_{24}, and S_{51} that only one condition expression is discovered. This is due to the limitation of decision tree algorithms. The decision tree algorithm populates the tree from the root to the leaf node as a disjunction of the conjunction of expressions. Therefore, only one conjunct can be discovered from the data if there is a conjunction of two or more expressions.

Another noticeable difference can be observed in the designed versus discovered *guards* of transitions S_{53} and S_{54}. The designed *guard* include the condition to make sure the pressure in the pipelines (PIT201) is released and now in a allowed range before switching to operating modes M_3 or M_4. However, this condition (PIT201 $<$ 70) was not discovered by our approach. The reason being that a decision tree algorithm splits a continuous variable's range into disjoint subsets based on the discrete outcomes, e.g., in Fig. 2 where S_{24} occurs when $O_3 > 0$ and other outcomes occur (S_{23} and S_{22}) when $O_3 \leq 0$. However, no disjoint subsets identified in the recorded values of PIT201 based on the outcomes (S_{51}, S_{53}, and S_{54}) because all three transitions S_{51}, S_{53}, and S_{54} occurred for PIT201 < 70. Our approach inherits the above mentioned limitations from the decision tree algorithms which we acknowledge.

Other differences in the discovered versus designed *guards* are of the threshold values. For instance, one of the conditions for transition S_{35} from mode M_3 to M_5 was set in the actual system to be if the pressure in the pipeline increases above a certain threshold, i.e., PIT201 > 70. However, the *guard* for transition S_{35} was discovered to be PIT201 > 66.0100 as shown in Table IV. As discussed earlier, the discovery methods are unbiased of any perception. Thus, the discovered rules only reflect what values are recorded in the dataset used to build the model. We found that this difference was caused by sensor noise and the values being logged in the dataset at a rate slower than the conditions evaluated by the controller so that the values in the log were behind those used to make the decisions. This difference could be minimised by logging ICS data at a higher sampling rate.

VI. Conclusion

In this paper we have presented a method to automatically monitor the modes switching conditions at the SC layer process in ICSs. We showed that our approach is able to discover condition rules from the logs generated during a system's operation and to use these for checking the validation of mode switching transitions taken in subsequent executions. The aim of our approach is that the discovered model for mode switching conditions is not biased by an expert's perception but

is extracted from actual observed behaviours. We have showed that our approach can discover mode switching violations which can't be discovered by previously proposed multimode process monitoring methods. However, just like machine learning algorithms, the accuracy of the models produced using our approach depends on the quality of the available dataset.

REFERENCES

[1] V. L. Do, L. Fillatre, I. Nikiforov, and P. Willett, "Feature article: security of SCADA systems against cyber–physical attacks," *IEEE Aerospace and Electronic Systems Magazine*, vol. 32, no. 5, pp. 28–45, 2017.

[2] M. S. Afzal, W. Tan, and T. Chen, "Process monitoring for multimodal processes with mode-reachability constraints," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 5, pp. 4325–4335, 2017.

[3] J. Livingston, "The nuclear electrical engineer, an educational resource for electrical engineers in the nuclear power industry," 2014 (Accessed Jan 31, 2020). http://www.nuclearelectricalengineer.com/nuclear-power-plant-modes-explained-here/.

[4] K. Hemsley and R. Fisher, "A History of Cyber Incidents and Threats Involving Industrial Control Systems," in *Critical Infrastructure Protection XII*, pp. 215–242, Springer, Cham, 2018.

[5] J. S. Lee, M. C. Zhou, and P. L. Hsu, "An application of Petri nets to supervisory control for human-Computer interactive systems," *IEEE Transactions on Industrial Electronics*, vol. 52, no. 5, pp. 1220–1226, 2005.

[6] M. Quiñones-Grueiro, A. Prieto-Moreno, C. Verde, and O. Llanes-Santiago, "Data-driven monitoring of multimode continuous processes: A review," *Chemometrics and Intelligent Laboratory Systems*, vol. 189, no. April, pp. 56–71, 2019.

[7] V. Uraikul, C. W. Chan, and P. Tontiwachwuthikul, "Artificial intelligence for monitoring and supervisory control of process systems," *Engineering Applications of Artificial Intelligence*, vol. 20, no. 2, pp. 115–131, 2007.

[8] M. A. Saez, F. P. Maturana, K. Barton, and D. M. Tilbury, "Context-Sensitive Modeling and Analysis of Cyber-Physical Manufacturing Systems for Anomaly Detection and Diagnosis," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 1, pp. 29–40, 2020.

[9] A. Rozinat and W. M. P. van der Aalst, "Decision Mining in ProM," in *International Conference on Business Process Management, BPM 2006*, pp. 420–425, 2006.

[10] P. Kokoszka and M. Reimherr, *Introduction to Functional Data Analysis*. Chapman and Hall/CRC, sep 2017.

[11] A. Giua and M. Silva, "Petri nets and Automatic Control: A historical perspective," *Annual Reviews in Control*, vol. 45, pp. 223–239, 2018.

[12] H. Genrich and K. Lautenbach, "System modelling with high-level Petri nets," *Theoretical Computer Science*, vol. 13, no. 1, pp. 109–135, 1981.

[13] Q. Jiang, S. Yan, X. Yan, H. Yi, and F. Gao, "Data-Driven Two-Dimensional Deep Correlated Representation Learning for Nonlinear Batch Process Monitoring," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 4, pp. 2839–2848, 2020.

[14] D. Myers, S. Suriadi, K. Radke, and E. Foo, "Anomaly detection for industrial control systems using process mining," *Computers and Security*, vol. 78, pp. 103–125, 2018.

[15] T. B. Sheridan, "Adaptive Automation, Level of Automation, Allocation Authority, Supervisory Control, and Adaptive Control: Distinctions and Modes of Adaptation," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 41, pp. 662–667, jul 2011.

[16] R. E. Barber and H. C. Lucas, "System Response Time Operator Productivity, and Job Satisfaction," *Communications of the ACM*, vol. 26, no. 11, pp. 972–986, 1983.

[17] T. Tapia-Flores, E. Lopez-Mellado, A. P. Estrada-Vargas, and J. J. Lesage, "Discovering Petri Net Models of Discrete-Event Processes by Computing T-Invariants," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 3, pp. 992–1003, 2018.

[18] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[19] W. Loh, "Classification and regression trees," *WIREs Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14–23, 2011.

[20] V. Hlupic and R. J. Paul, "Simulating an automated paint shop in the electronics industry," *Simulation Practice and Theory*, vol. 1, no. 5, pp. 195–205, 1994.

[21] S. Choi, J.-h. Yun, and S.-k. Kim, "A Comparison of ICS Datasets for Security Research Based on Attack Paths," in *Critical Information Infrastructures Security CRITIS 2018. Lecture Notes in Computer Science*, vol. 6712, pp. 154–166, Springer International Publishing, 2019.

[22] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification And Regression Trees*. Routledge, 1984.

Mukhtar Hussain received the BS Electrical Engineering degree from COMSATS University Islamabad, Lahore Campus, and the MS Information and Communication Engineering from Xi'an Jiaotong University, Xi'an, China in 2013 and 2016 respectively. He is currently pursing the PhD degree in the School of Computer Science, Queensland University of Technology with focus on the security of industrial controls systems. In particular, he is looking into process analytics and behavioural modelling of industrial control systems for anomaly detection.

Colin Fidge is a full professor in the School of Computer Science, Queensland University of Technology, where he teaches computing fundamentals and research principles. His research interests include modelling and analysis of complex computer-based systems, and software engineering of enterprise systems. He has led major research projects for a variety of industry partners in defence, electricity generation, and power and water distribution.

Ernest Foo (Member, IEEE) received the Ph.D. degree from the Queensland University of Technology (QUT), Brisbane, Australia, in 2000. Since 2007, he has been a Senior Lecturer and a Researcher with the Information Security Discipline, School of Electrical Engineering and Computer Science, QUT. He is currently an Associate Professor with Griffith University and an Adjunct Associate Professor with QUT. His research interests can be broadly grouped into the field of secure network protocols with an active interest in the security of industrial controls systems, such as supervisory control and data acquisition and the smart grids.

Zahra Jadidi received the Ph.D. degree in Information Technology from the Griffith University, Australia, in 2016. She is currently a research fellow in the School of Computer Science, Queensland University of Technology. Her research interest include image processing, artificial intelligence, and deep learning with an active interest in the application of artificial intelligence in network security and network management.