Identification of Industrial Automation Systems: Building Compact and Expressive Petri Net Models from Observable Behavior

Ana-Paula Estrada-Vargas, Jean-Jacques Lesage, Ernesto López-Mellado

Identification of Industrial Automation Systems: Building Compact and Expressive Petri Net Models from Observable Behavior

Ana Paula Estrada-Vargas, Jean-Jacques Lesage, and Ernesto López-Mellado, *Members, IEEE*

***Abstract*— The paper deals with black-box identification of industrial automated discrete manufacturing systems. The problem of obtaining Petri net (PN) models from the observable behavior, expressed as a sequence of input-output vectors, is addressed. First the problem is stated: important issues to handle in systems automated by Programmable Logic Controllers that cannot be dealt by other methods are detailed. Then a novel method is presented; it focuses on building a compact and expressive representation of the observable part of the model which allows consequently the construction of a reduced complete Interpreted PN describing both observable and unobservable behavior.**

I. Introduction

Identification allows building systematically a mathematical model that describes the behavior of an unknown or ill-known system based on the observation of its evolution. In the case of discrete event systems (DES), observations consist of data revealing the system activity: sequences of operations, events, messages, etc., and the models are abstract machines that reproduce the observed behavior.

A. Related Work

This problem has been addressed in literature in various formulations and from diverse approaches.

The works in [1], [2] obtain a Petri net system from the knowledge of the language it generates, i.e. the set of transition sequences that can be fired from the initial marking. For automated systems identification, such transitions are unknown if a black-box approach is being performed; that is, the only available information about the system is the evolution of input and output signals. Besides, some of the stated hypotheses on these works are not well adapted for real complex DES, particularly the consideration of the entire system language observation. In practice, only part of the language is observed, especially when there are many concurrent tasks in the system.

In [3] several algorithms are introduced to synthesize a Petri net with regard to an event propagation set. However, distinction between input and output signals is not made and obtained models do not express how inputs and outputs of the system are interrelated to produce the observed behavior, although it is the core of a reactive system.

E. López-Mellado is with CINVESTAV Unidad Guadalajara. Av. del Bosque 1145, Col. El Bajío 45019 Zapopan, Mexico. Email: elopez@gdl.cinvestav.mx. J-J. Lesage is with Ecole Normale Supérieure de Cachan. 61, av du Président Wilson, 94235 Cachan Cedex, France. Email: Jean-Jacques.lesage@lurpa.ens-cachan.fr. A. P. Estrada-Vargas is with both institutions. Email: aestrada@gdl.cinvestav.mx.

In [4] it is described a method to incrementally construct an IPN model from a single output vectors sequence. The considered DESs to identify must be event-detectable by the outputs. Applying this method to an I/O sequence would lead to models in which same output changes caused by different input evolutions would not be distinguished, and then incorrect behavior could be introduced.

The method presented in [5] obtains automata models representing a set of cyclic I/O sequences. This method also considers automated systems. However, in the obtained models, structural information such as parallelism cannot be explicitly expressed. An extension of this work has been presented in [6], where splitting the system on concurrent parts is performed. Even if modeled subsystems represent parallelism, the method is strongly adapted for fault detection purposes.

In [7], [8] an event sequence is observed, as well as the corresponding output symbols of a DES to produce an IPN model, in which the sequence and the observed output vectors are reproducible. This methodology requires the definition of an event list, which is not available in the context of black-box identification problem addressed in this work. An alternative to this lack of events list could be the consideration of all the observed input changes. In this case, models with several paths describing input changes would be constructed, in which some input-output relations would not be explicitly observed.

In [9] a technique for constructing a Petri net-like model that describes the relationship between tasks from a sequence of workflow events is presented. This technique allows the discovering of events belonging to certain threads and synchronization points (forks and joins of tasks) through a probabilistic analysis of metrics such as the entropy, number and regularity of task occurrences. It is assumed that all the workflow operations are observable.

B. Input-Output approach

Beyond the theoretical interest of defining model synthesis methods from symbol sequences, the challenges of applying identification methods to actual industrial automated systems are related to the scalability of the algorithms and technological issues: the techniques must be efficient to cope with large and complex systems that handle actual signals.

In our approach we deal with Programmable Logic Controller (PLC) based automated systems. The aim is to discover, from observations of the system behavior expressed as a single sequence of PLC input and output signals, how components of the system are interrelated and

construct a concise model which can explicitly show the discovered behavior, in particular, concurrency, synchronization, resource sharing, etc. Identification of systems in operation involves two important aspects to consider: the system operation and the observation process. Technological issues of both aspects must be considered in the proposed algorithms to construct suitable abstractions.

In a previous work [10] an I/O sequence is considered to compute an IPN including cyclic behavior. Although the proposed methodology is scalable due to the algorithms efficiency, the obtained models are close to finite automata and can be huge, due to the explicit representation of observed input changes that could not be relevant to define the output evolution.

In this paper we address these problems by analyzing the observed sequence to establish a clearer relation between inputs and outputs of the controller. The proposed method allows building a reduced representation of the observable part of the model which yields consequently, a reduced complete IPN. None of the black-box identification approaches in related works allows obtaining such well structured models.

C. Contents

The paper is organized as follows. In section II IPN basic notions are overviewed. Section III states the problem of industrial automated systems identification. Section IV introduces the input-output approach and section V describes the method for building a concise representation of the IPN model observable part.

II. Interpreted Petri Nets

This section contains the basic concepts and notation of PN and IPN used in this paper.

Definition 1: An ordinary Petri Net structure G is a bipartite digraph represented by the 4-tuple $G = (P, T, I, O)$ where: $P = \{p_1, p_2, ..., p_{|P|}\}$ and $T = \{t_1, t_2, ..., t_{|T|}\}$ are finite sets of vertices named places and transitions respectively; $I(O) : P \times T \rightarrow \{0,1\}$ is a function representing the arcs going from places to transitions (from transitions to places).

The incidence matrix of G is $C = C^+ - C^-$, where $C^- = [c_{ij}^-]$; $c_{ij}^- = I(p_i, t_j)$; and $C^+ = [c_{ij}^+]$; $c_{ij}^+ = O(p_i, t_j)$ are the pre-incidence and post-incidence matrices respectively.

A marking function $M : P \rightarrow Z^+$ represents the number of tokens residing inside each place; it is usually expressed as an $|P|$-entry vector. Z^+ is the set of nonnegative integers.

Definition 2: A Petri Net system or Petri Net (PN) is the pair $N = (G,M_0)$, where G is a PN structure and M_0 is an initial marking.

In a PN system, a transition t_j is *enabled* at marking M_k if $\forall p_i \in P$, $M_k(p_i) \geq I(p_i, t_j)$; an enabled transition t_j can be fired reaching a new marking M_{k+1} . This behavior is represented as $M_k \xrightarrow{t_j} M_{k+1}$. The new marking can be computed as $M_{k+1} = M_k + Cu_k$, where $u_k(i) = 0$, i≠j, u_k(j) = 1; this equation is called the PN state equation. The reachability set of a PN is the set of all possible reachable markings from M_0 firing only enabled transitions; this set is denoted by $R(G,M_0)$.

Now it is defined IPN [11], an extension to PN that allows associating input and output signals to PN models.

Definition 3 : An IPN (Q, M_0) is a net structure $Q = (G, V, \Sigma, \Phi, \lambda, \varphi)$ with an initial marking M_0 where: G is a PN structure, $V = \{v_1, v_2, ..., v_r\}$ is the set of variables, $\Sigma = \{\alpha_1, \alpha_2, ..., \alpha_s\}$ is the set of events, and $\Phi = \{\phi_1, \phi_2,..., \phi_q\}$ is the output alphabet. $\lambda : T \rightarrow C \times E$ is a labeling function of transitions, where $C=\{C_1, C_2,\ldots\}$ is the set of variable conditions and $E=\{E_1, E_2,\ldots\}$ is the set of events.

In an *IPN*, a transition t_j will be fired if a) t_j is enabled, ***and*** b) condition $C(T_j)$ is true, ***and*** c) the event in $E(T_j)$ occurs.

$\varphi : R(Q,M_0) \rightarrow (Z^+)^q$ is an output function, that associates to each marking in $R(Q,M_0)$ a q-entry output vector; $q=|\Phi|$ is the number of outputs. φ is represented by a $q \times |P|$ matrix, such that if the output symbol ϕ_i is present (turned on) every time that $M(p_j) \geq 1$, then $\varphi(i, j) = 1$, otherwise $\varphi(i, j) = 0$.

The state equation is completed with the marking projection $Y_k = \varphi M_k$, where $Y_k \in (Z^+)^q$ is the k-th output vector of the *IPN*.

Definition 4: A place $p_i \in P$ is said to be *observable* if the i-th column vector of φ is not null, i.e. $\varphi(\bullet,i) \neq 0$. Otherwise it is *non-observable*. $P = P^o \cup P^u$ where P^o is the set of observable places and P^u is the set of non-observable places.

III. Identification of Industrial Automated Systems

A. {PLC + Plant} identification

In this work we consider the systems composed by a Controller (a PLC) and a Plant denoted as {PLC + Plant} working on a closed loop. The input signals of the PLC (outputs of the Plant) are generated by the sensors of the Plant. The output signals of the PLC (inputs of the Plant) control the actuators of the Plant.

The identification is made from the point of view of the PLC (Fig. 1). A PLC cyclically performs three main steps: a) Input reading, where signals are read from the sensors; b) Program execution, to determine the new outputs values for the actuators; and c) Output writing, where the control signals to the actuators are set. At each end of the Program execution phase, the current value of all Inputs and Outputs (called I/O vector) is captured and recorded in a data base.

Regarding the implementation of the data link between PLC and identification data base, we use the UDP (User Datagram Protocol) connection presented in [12]. Tests performed using a Siemens PLC (CPU 315-2 DP) equipped with a program leading to a PLC-cycle time of 25 to 30ms have shown that this connection is reliable and efficient: no data packets got lost during the transmission and the execution of the PLC program is not delayed by the capture of data. The only available data for the identification procedure is therefore a single I/O vector sequence whose length depends on the observation duration:

$$w = \left[\frac{I(1)}{O(1)}\right], \left[\frac{I(2)}{O(2)}\right], \left[\frac{I(3)}{O(3)}\right], \ldots$$

$I(j)$ and $O(j)$ are respectively the values of the r inputs and q outputs at the j-th PLC cycle.

Our aim is to express the system's behavior extractible from the I/O vector sequence as an IPN. In the next section we explain why this formalism is well adapted for representing {PLC + Plant} behavior in a suitable form.

B. Relation between Petri Nets and DES behavior

Left side of Fig. 2 shows how a DES evolves across the time. At each PLC cycle (denoted by $_K$), state ($X(K)$) and output ($O(K)$) are updated according to functions S and A respectively. S considers previous state and current input signal values. A considers the new reached state and current input signal values. The systems we are dealing with are those which can be described without time or conditional actions, thus the outputs depend only on the state.

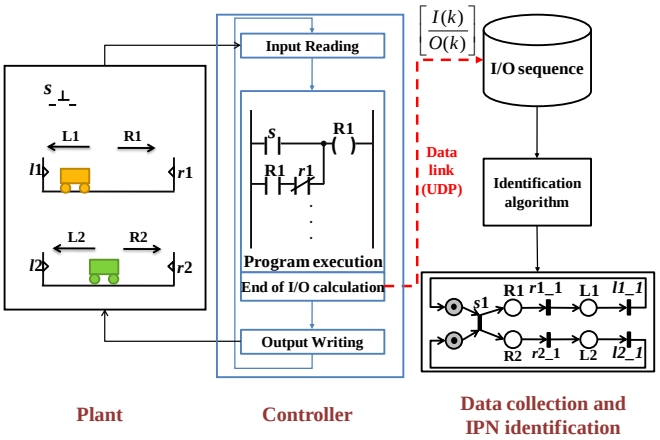


Fig. 1 {PLC + Plant} compound and Identification procedure

Discrete Event System		Interpreted Petri Nets	
$\begin{cases} X(K)=S(X(K-1),I(K)) \\ O(K)=A(X(K),I(K)) \end{cases}$		$\begin{cases} M_K = M_{K-1} + Cu_K \\ Y_K = \varphi M_K \end{cases}$	
$X_K \begin{cases} X_K^o \\ X_K^u \end{cases}$	Observable state evolution Unobservable state evolution	$M_K \begin{cases} M_K^o \\ M_K^u \end{cases}$	Observable marking evolution Unobservable marking evolution

Fig. 2 Relation between DES and IPN

Example 1. In order to explain different type of evolutions that could arrive on a DES, let us introduce an instance of the well known two wagons example. It consists of five input signals $l1$, $l2$, $r1$, $r2$, and s and four output signals R1, R2, L1, and L2 (Fig. 1).

When both wagons are at their leftmost position ($l1$ = 1 and $l2$ = 1) and signal s appears, they start moving to the right (R1 and R2). When one of the wagons arrives to its rightmost position (indicated by $r1$ or $r2$) it is ordered to stop (turn off R1 or R2). Once both wagons are at their rightmost position, they go back to the left (L1 and L2) and wagons are stopped again at its leftmost position in order to start a new cycle. Notice that signal s, which is generated by an operator, can appear at any instant of the cycle.

The sequence of Fig. 3 has been collected from the considered system as described in section III.A.

In order to analyze signals evolution, we compute *event vectors*, i.e., the difference between two consecutive I/O vectors: $E(k) = w(k+1) - w(k)$. Each event vector can be decomposed into input and output event vectors:

$$E(k) = \begin{bmatrix} IE(k) \\ \hline OE(k) \end{bmatrix}$$

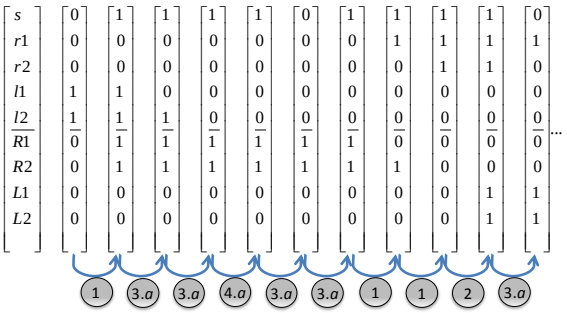


Fig. 3 Different observed types of I/O evolution

Regarding input and output event vectors, there exist four situations (types) that could be observed, which are explained by different occurring phenomena:

Type 1. $IE(k) \neq 0 \wedge OE(k) \neq 0$

An input change has provoked directly an output change, and consequently, a state evolution. The I/O causality is observed at the same PLC cycle.

Type 2. $IE(k) = 0 \wedge OE(k) \neq 0$

The controller has arrived at step k-1 to a state in which, given the inputs, an output (and state) evolution is allowed at step k.

Type 3. $IE(k) \neq 0 \wedge OE(k) = 0$

a) $X(k-1) = X(k)$ *It has occurred an input evolution to which the controller is not sensitive.*

b) $X(k-1) \neq X(k)$ *It has occurred a non-observable state evolution of the controller.*

Type 4. $IE(k) = 0 \wedge OE(k) = 0$

a) $X(k-1) = X(k)$ *The controller remains in a stable state, i.e., no state evolution condition is satisfied.*

b) $X(k-1) \neq X(k)$ *It has occurred a non-observable state evolution of the controller.*

All of these situations can be easily described by the IPN dynamics, as showed at the right side of Fig. 2. The state evolution function can be represented by the IPN state equation and the output function can be easily translated into the marking projection function. However Type 4 situations cannot be detected by external observation of inputs/outputs, and then they will be not dealt by our algorithm. Consequently, the sequence stored in the database will be built by adding a new I/O vector only when it is different to the last one. Observable state evolutions will be represented by arcs entering and leaving observable places. State evolutions non-exhibited by output evolutions will be represented by transitions between non-observable places. This has been done previously in literature [4], [7] [10] [13]. However, existing approaches show only direct event-output evolutions (Type 1). Some adaptations must be done on IPN in order to represent information regarding system evolution restricted to input conditions (instead of input changes). In our approach this is supported by the IPN transition labeling function. The variable set is the input set of the system, and the event set is the set of combinations over rising and

falling edges of input variables, denoted {i_1_1, i_1_0, i_2_1, …, i_r_1, i_r_0}. Then, $E(T_j)$ can be expressed as a conjunction of events, and $C(T_j)$ will be a condition over the input signals of the system. In the next sections we will deeply present how we translate the observed behavior of a controlled DES into an IPN model.

IV. Input-Output Identification Approach

Based on classification of types of evolution described in section III.B, we can divide our identification method in two steps: a) computing the observable part of the model and b) computing the non-observable part of the model.

A. Identification of the {PLC + Plant} observable behavior

Here, we determine directly from observed behavior:

- If certain input changes *always* produce output changes at the same PLC cycle (evolution type 1).
- If there is an input evolution (evolution type 3.*b*) in which during later PLC cycles satisfies conditions to produce output changes (evolution type 2).
- If certain inputs have no influence on the output changes (evolution type 3.*a*).

B. Identification of the non-observable behavior

Some state evolutions (Evolution type 3.*b*, 4.*b*) cannot be observed only through the outputs, and they could be confused with other type of evolutions (type 3.*a*, 4.*a* respectively). Some of these situations cannot be discerned, but some of them can be inferred from the imposed order of events apparition. Owing to lack of space, the technique for determining such relations is not included in this paper.

V. Identification of the Observable Behavior

Now, we describe the methodology to identify the {PLC + Plant} observable behavior. It consists of the steps summarized in Algorithm 1, which will be deeply described on the next sub-sections through a simple case study inspired from a manufacturing example.

Algorithm 1.General description of the method

Input: I/O sequence *w*

Output: Observable incidence matrix φC and labeling transition function λ

Step 1. Analyze sequence *w* in order to

- Compute events vector sequence
- Compute elementary events
- Compute Direct and Indirect Causality Matrices
- Construct Output Event Firing Functions
- Find Input events with differed influence

Step 2. Use computed data in Step 1 to

- Compute transitions of the IPN and their labeling λ
- Compute observable incidence matrix φC

Example 2. The purpose of this system (Fig. 4) is to sort parcels according to their size. It has 9 signal sensors from the system: a_0, a_1, a_2, b_0, b_1, c_0, c_1, k_1, k_2, and 4 signals to the actuators: A+, A-, B, C. This example has been used in other publications [10], [13] and we take it up again to confront previous results to the new ones.

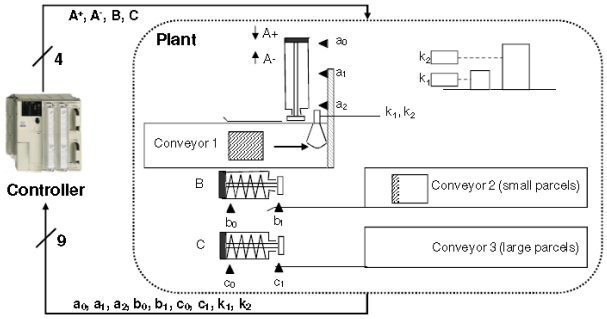
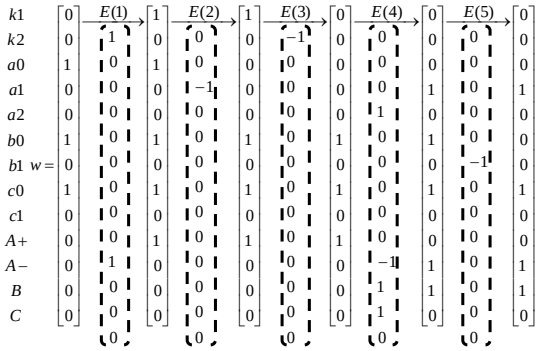


Fig. 4 Layout of the system case study

For illustrative purposes, we present here only the beginning of an I/O vector sequence. However, recall that treated sequences are usually very much longer (thousands of events vectors).

A. Events vector sequence

We have included in the sequence the first step of the algorithm, i.e., the computed event vectors (dotted lines) between each two consecutive I/O vectors (solid lines).



B. Elementary events

In order to analyze the system behavior in a deeper way, event vectors can be decomposed into a set of elementary events (simply called events):

$$IE(k) = \{IE_{k1}, IE_{k2}, ..., IE_{kj}\} = \bigcup IE_{ki} \text{ s.t. } I_i(k+1) - I_i(k) \neq 0$$

$$OE(k) = \{OE_{k1}, OE_{k2}, ..., OE_{ki}\} = \bigcup OE_{ki} \text{ s.t. } O_i(k+1) - O_i(k) \neq 0$$

If no elementary input (output) event occurs in $E(j)$, we denote it as $IE(j)=\{\varepsilon\}$ ($OE(j)=\{\varepsilon\}$). The rising (resp. falling) edge event of variable v_i is denoted as v_i_1 (resp. v_i_0).

In Table 1 the elementary events computed for the example sequence are showed.

Event vector	Elementary input events	Elementary output events
***E*(1)**	***IE*(1) = {*k*1_1}**	***OE*(1) = {*A*+_1}**
***E*(2)**	***IE*(2) = {*a*0_0}**	***OE*(2) = {*ε*}**
***E*(3)**	***IE*(3) = {*k*1_0}**	***OE*(3) = {*ε*}**
***E*(4)**	***IE*(4) = {*a*1_1}**	***OE*(4) = {*A*+_0, *A*−_1, *B*_1}**
***E*(5)**	***IE*(5) = {*b*0_0}**	***OE*(5) = {*ε*}**

Table 1. Elementary events list for Example 2

C. *Direct and Indirect Causality Matrices*

As stated in section III, the influence of some input signals over the outputs setting is observed at the same PLC cycle. In order to discover such an input-output direct relationship, we analyze the relative frequency of the simultaneous emergence of an input event IE_i and an output event OE_k, with respect to the emergence of OE_k during the whole sequence of events. That relationship can be naturally expressed as the conditional probability of the occurrence of an output event OE_k, given that a certain input event IE_i has occurred at the same PLC cycle:

$$Prob(OE_k \mid IE_i) = \frac{N_{Obs}(OE_k, IE_i)}{N_{Obs}(OE_k)}$$

where $N_{Obs}(.)$ denotes the number of observed occurrences. Using all values $Prob(OE_k|IE_i)$, a matrix can be filled. We call such a matrix the *Direct Causality Matrix* (*DM*), in which every $DM_{ik} = Prob(OE_k|IE_i)$. Table 2 presents the computed *DM* matrix for the Example 2, considering a longer sequence than the presented one.

	A+_1	A+_0	A-_1	A-_0	B_1	B_0	C_1	C_0
k1_1	0.111	0.111	0.111	0.111	0.000	0.200	0.000	0.000
k1_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
k2_1	0.222	0.000	0.000	0.000	0.000	0.000	0.000	0.000
k2_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
a0_1	0.222	0.000	0.000	1.000	0.000	0.000	0.000	0.000
a0_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
a1_1	0.000	0.444	0.444	0.000	1.000	0.000	0.000	0.000
a1_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
a2_1	0.000	0.556	0.556	0.000	0.000	0.000	1.000	0.000
a2_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
b0_1	0.333	0.000	0.000	0.000	0.000	0.000	0.000	0.000
b0_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
b1_1	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
b1_0	0.000	0.000	0.000	0.111	0.000	0.000	0.000	0.000
c0_1	0.111	0.000	0.000	0.000	0.000	0.000	0.000	0.000
c0_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
c1_1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
c1_0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table 2. Direct Causality Matrix for the Example 2

Conditional probabilities have been used in [9] for analyzing the relative occurrence between workflow events; this analysis is done in the first step of the procedure. However the remaining steps and the kind of obtained model differ from that of our method.

With the *DM* matrix, we can find Evolution type 1 simply by looking at each column the values that add up to 1, since this represents the total of occurrences of event OE_k. For example, from Table 2, we can discover that events a1_1 and a2_1 are the input events which *always* provoke the output event A+_0. The general case where several input events can provoke an output event is formalized on the next section. Also evolution type 3.*a* can be found at *DM* matrix, under the form of null rows. For example, the second row indicates that the controller is not sensitive to the input event k1_0.

Similarly, to discover input-output non direct relationship, we look at the present input values when a certain output event occurs. We compute the occurrence probability of an output event OE_k, given that certain input has a given value IL_i at the same PLC cycle:

$$Prob(OE_k \mid IL_i) = \frac{N_{Obs}(OE_k, IL_i)}{N_{Obs}(OE_k)}$$

We construct the *Indirect Context Matrix* (*IM*) in which every $IM_{ik} = Prob(OE_k|IL_i)$. The *IM* matrix for Example 2 is showed in Table 3.

	A+_1	A+_0	A-_1	A-_0	B_1	B_0	C_1	C_0
k1=1	0.444	0.111	0.111	0.333	0.000	0.250	0.200	0.200
k1=0	0.556	0.889	0.889	0.667	1.000	0.750	0.800	0.800
k2=1	0.556	0.000	0.000	0.333	0.000	0.250	0.000	0.200
k2=0	0.444	1.000	1.000	0.667	1.000	0.750	1.000	0.800
a0=1	1.000	0.000	0.000	1.000	0.000	0.500	0.000	0.000
a0=0	0.000	1.000	1.000	0.000	1.000	0.500	1.000	1.000
a1=1	0.000	0.444	0.444	0.000	1.000	0.000	0.000	0.000
a1=0	1.000	0.556	0.556	1.000	0.000	1.000	1.000	1.000
a2=1	0.000	0.556	0.556	0.000	0.000	0.000	1.000	0.000
a2=0	1.000	0.444	0.444	1.000	1.000	1.000	0.000	1.000
b0=1	1.000	1.000	1.000	0.556	0.000	0.000	1.000	1.000
b0=0	0.000	0.000	0.000	0.444	1.000	1.000	0.000	0.000
b1=1	0.000	0.000	0.000	0.111	1.000	1.000	0.000	0.000
b1=0	1.000	1.000	1.000	0.889	0.000	0.000	1.000	1.000
c0=1	1.000	1.000	1.000	0.889	0.000	1.000	1.000	0.000
c0=0	0.000	0.000	0.000	0.111	1.000	0.000	0.000	1.000
c1=1	0.000	0.000	0.000	0.000	1.000	0.000	0.000	1.000
c1=0	1.000	1.000	1.000	1.000	0.000	1.000	1.000	0.000

Table 3. Indirect Context matrix for the Example 2.

Using the *IM* matrix we can discover evolution types 3.*b* and 2 by inspecting in every column the values that add up to 1 which are not zero in the *DM* matrix. In the example 2, k1=1 and k2=1 are input values which *can* provoke A+_1 output event, even if they were *not always* observed at the same PLC cycle.

Now we will present how these relations can be automatically discovered from the *DM* and *IM* matrices.

D. *Computing Output Event Firing Functions*

It can be noticed that the occurrence of every output event OE_k is caused by one or several input events occurring at the same PLC cycle and by a condition on the input values. This can be expressed by the Output Event Firing Function:

$$\chi(OE_k) = G(OE_k) \bullet F(OE_k)$$

where $G(OE_k)$ is a function of input events and $F(OE_k)$ is a function of inputs levels which allow the triggering of the output event OE_k.

We compute $G(OE_k)$ as a conjunction of disjunctions:

$$G(OE_k) = \Pi DisjE_j$$

where each disjunction $DisjE_j = (IE_x \oplus ... \oplus IE_z)$ involves those variables corresponding to non-zero column values of the *DM* matrix, which add up to 1, i.e. those satisfying conditions:

1. $DM_{xj} \neq 0, DM_{yj} \neq 0, ... DM_{zj} \neq 0$
2. $DM_{xj} + DM_{yj} + ... + DM_{zj} = 1$

$F(OE_k)$ is computed in a similar way: $F(OE_k) = \Pi DisjL_j$ with $DisjL_j = (IL_x \oplus IL_y \oplus ... \oplus IL_z)$ such that

1. $IM_{xj} \neq 0, IM_{yj} \neq 0, ... IM_{yj} \neq 0$
2. $IM_{xj} + IM_{yj} + ... + IM_{zj} = 1$
3. $DM_{xj} \neq 0, DM_{yj} \neq 0, ... DM_{yj} \neq 0$

Then, for every output signal O_i, we will have the input events and input conditions to produce its rising and falling edges O_i_1 and O_i_0 respectively. This can be easily translated into *IPN* fragments, as showed in Fig. 5.

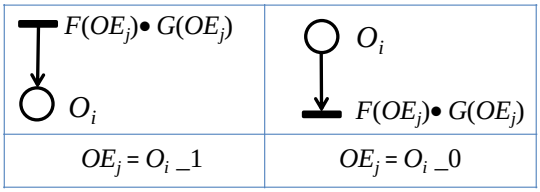


Fig. 5 Rising and falling transitions of output O_i

E. Input events with differed influence on the outputs

As stated below, when an input change is observed, it could or not produce a state evolution. Since we cannot always distinguish between these two situations through the firing functions construction, some input events could be being ignored if they are not included in any function, i.e., if they have never provoked directly an output change. In order to avoid this, we keep the set of inputs $D = \{ I_i \}$ such that *DM* lines corresponding to I_i_0 and I_i_1 are both zero. We call it the set of *inputs with differed influence* over the outputs. For the Example 2, $D = \{\}$; the computed PN fragments showed in Fig. 6.

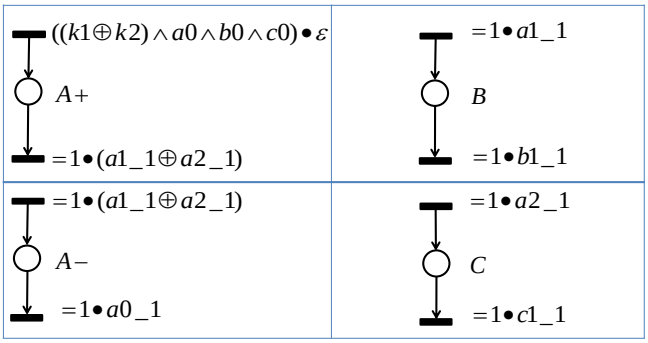


Fig. 6 IPN fragments for Example 2

F. Fusion of IPN fragments

As stated below, at each PLC cycle, several input and state conditions could lead to the simultaneous occurrence of several output events. This behavior is reproduced by merging such conditions into a unique transition, which is labeled by a firing function computed from individual firing functions of each output event. This is captured in the model as a fusion of IPN fragments as showed in Fig. 7.

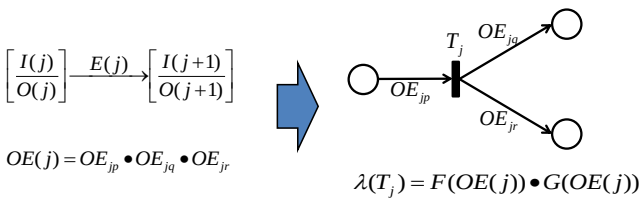


Fig. 7 IPN representation of several output events at the same cycle

This can be systematically done through the next procedure, which can be executed in polynomial time:

Algorithm 2.

Input: I/O sequence *w*, I/O events sequence *E*, Matrices *DM* and *IM*, Differed input set *D*

Output: Observable incidence matrix φC and labeling transition function λ

1. $P \leftarrow \{p_1, p_2, \ldots, p_q\}$ *//Create q observable places, one for every output of the system*
2. $\forall j=1,\ldots,|E|$ *//We will consider all the I/O sequence values w(j) and I/O events sequence values E(j):*
   - 2.1. If $OE(j) = 0$ and $\exists\ IE_s,\ldots,IE_u \in IE(j) \cap D$ *//There is not an output change, but IE(j) contains elementary input events* $IE_s,\ldots,IE_u$ *belonging to D*
     - $T \leftarrow T \cup \{t_j\}$, $\lambda(t_j) = IE_s \bullet \ldots \bullet IE_u$, $\forall p_i \in P\ C(t_j,p_i) \leftarrow 0$ *//If it has not been created before, create a new zero transition* t_j *(a zero column in the incidence matrix) representing input changes* $IE_s,\ldots,IE_u$
   - 2.2. If $OE(j) \neq 0$ *//There is an output change*
     - 2.2.1. $\forall OE_{jk} \in OE(j)$ *//Consider all the elementary output events in OE(j) in order to compute G(OE(j)) and F(OE(j))*
       - 2.2.1.1. $\forall DisjE_i \in G(OE_{jk})$, $DisjE_i' \leftarrow DisjE_i \cap IE_{jk}$ *// Look into IE(j) the input event* IE_{jk} *which has satisfied* $DisjE_i$ *and assign it to auxiliary variable* $DisjE_i'$
       - 2.2.1.2. $G'(OE_{jk}) \leftarrow \Pi DisjE_i'$ *//Combine into* $G'(OE_{jk})$ *all the conditions* $DisjE_i'$ *which have satisfied* $G(OE_{jk})$
       - 2.2.1.3. $G(OE(j)) \leftarrow \Pi G'(OE_{jk})$ *//Combine into G(OE(j)) all the input event conditions* $G'(OE_{jk})$ *which have satisfied all the events* OE_{jk}
       - 2.2.1.4. $\forall DisjL_i \in F(OE_{jk})$, $DisjL_i' \leftarrow DisjL_i \cap I(j+1)$ *// Looking the I(j+1) vector as a set of Boolean variables, save into* $DisjL_i'$ *the input value* IL_{ik} *which has satisfied* $DisjL_i$
       - 2.2.1.5. $F'(OE_{jk}) \leftarrow \Pi DisjL_i'$ *//Combine into* $F'(OE_{jk})$ *all the conditions which have made true* $F(OE_{jk})$
       - 2.2.1.6. $F(OE(j)) \leftarrow \Pi F'(OE_{jk})$ *//Combine into F(OE(j)) all the conditions which have produced all the* OE_{jk}
     - 2.2.2. $T \leftarrow T \cup \{t_j\}$, $\lambda(t_j) = F(OE(j)) \bullet G(OE(j))$ *If it has not been created before, create a new transition* t_j *and label it with the computed F(OE(j)) and G(OE(j))*
     - 2.2.3. $\forall p_i \in P$, If O_q_1 $\in OE(j)$ $C(t_j,p_q) \leftarrow 1$, else If O_q_0 $\in OE(j)$ $C(t_j,p_q) \leftarrow -1$, else $C(t_j,p_{jq}) \leftarrow 0$ *//for all elementary output events in OE(j) =* $OE_{jp} \bullet OE_{jq} \bullet \ldots \bullet OE_{jr}$*, put a 1 into the line corresponding to* OE_{jk} *if it is a rising event, and a -1 if it is a falling event; for the rest of the lines, put a 0.*

Fig. 8 shows how events $E(1)$ and $E(4)$ are treated by the algorithm. For $E(1)$ the elementary output event A+_1 in $OE(1)$ is analyzed and function $\lambda(t_1) = (k1 \bullet a0 \bullet b0 \bullet c0) \bullet (\varepsilon)$ is extracted considering that $k1=1$, $a0=1$, $b0=1$, and $c0=1$ are the input values which have satisfied $\chi(A+\_1)$. For $E(4)$ all elementary output events A+_0, A-_1 and B_1 in $OE(4)$ are considered; then their Firing Functions $\chi(A+\_0) = (=1) \bullet (a1\_1 \oplus a2\_1)$, $\chi(A-\_1) = (=1) \bullet (a1\_1 \oplus a2\_1)$, and $\chi(B\_1) = (=1) \bullet (a1\_1)$ are combined into $\lambda(t_2) = (=1) \bullet (a1\_1)$.

Notice that interesting labeling functions have been computed. For example, the output event A+_1 is provoked by the presence of a piece (k1=1 or k2=1) and it occurs only

when the three components corresponding to outputs A+, B, and C are on its initial position (a0=1, b0=1 and c0=1).

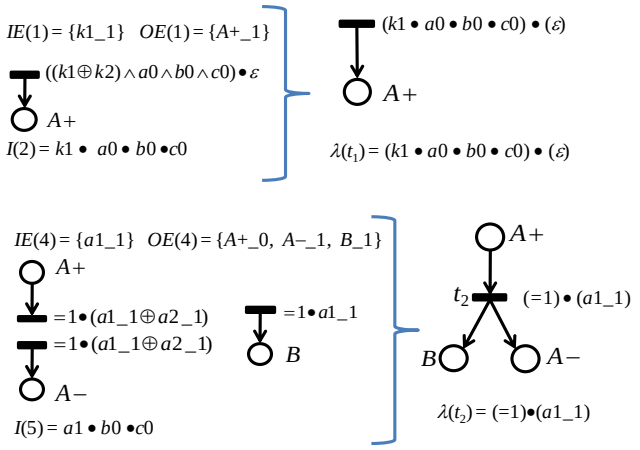


Fig. 8 Treatment of *E*(1) and *E*(4) by the Algorithm 2.

At the end of this procedure, the following observable incidence matrix φC and labeling functions are obtained.

$$\begin{array}{c} \\ A+ \\ A- \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 \end{array} \\ \begin{bmatrix} 1 & -1 & 0 & 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \end{array}$$

$$\lambda(t_1) = (k1 \wedge a0 \wedge b0 \wedge c0) \bullet (\varepsilon)$$
$$\lambda(t_2) = (=1) \bullet (a1\_1)$$
$$\lambda(t_3) = (=1) \bullet (b1\_1)$$
$$\lambda(t_4) = (=1) \bullet (a0\_1)$$
$$\lambda(t_5) = (k2 \wedge a0 \wedge b0 \wedge c0) \bullet (\varepsilon)$$
$$\lambda(t_6) = (=1) \bullet (a2\_1)$$
$$\lambda(t_7) = (=1) \bullet (c1\_1)$$

The corresponding partial model, depicted in solid lines is showed in Fig. 9. For illustrative purposes the complete model including non-observable places (depicted in gray) is showed; the inferring procedure which allows discovering the non-observable behavior is not described in this paper. As it can be noticed, firing transitions allow creating a compact abstraction of the observable behavior by exploiting IPN semantics potential.

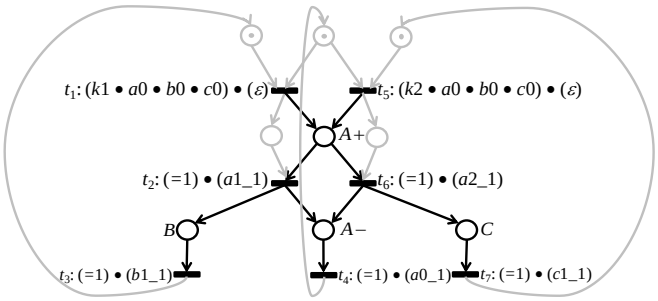


Fig. 9 Observable IPN model

VI. CONCLUSIONS

A method to discover the actual input-output relation of PLC controlled discrete event systems has been presented. The method allows building a concise representation of the observable part of an IPN model in which the transitions are labeled with sufficient conditions on the inputs which represent both the input changed and the inputs execution context.

The obtained structure is remarkably more clear and expressive than that synthesized with the previous method, because it is directly expressed in the structure of the IPN. Neither our previous methodology nor the approaches considered in the related work allow discovering such kind of input conditions. Additionally, those methods yield more complex and less expressive models.

Current research deals with the building of the non-observable part of the model and further simplifications regarding the representation of concurrent behavior.

ACKNOWLEDGEMENTS

A.P. Estrada-Vargas has been sponsored by CONACYT (Mexico) Grant No. 50312, and by Région Île-de-France.

REFERENCES

[1] M.P. Cabasino, A. Giua and C. Seatzu, "Identification of Petri Nets from Knowledge of Their Language", Discrete Event Dynamic Systems, 17(4), pp. 447-474, 2007

[2] M. P. Cabasino, A. Giua, C. Seatzu, "Linear Programming Techniques for the Identification of Place/Transition Nets", IEEE Int. Conf. on Decision & Control, Cancun, Mexico, pp. 514-520, Dec. 2008

[3] S. Ould El Medhi, E. Leclercq, D. Lefebvre, "Petri nets design and identification for the diagnosis of discrete event systems", 2006 IAR Annual Meeting, Nancy, France, Nov. 2006

[4] M. Meda-Campaña, E. López-Mellado, "Identification of Concurrent Discrete Event Systems Using Petri Nets", IMACS 2005 World Congress, Paris, France, pp.1-7, Jul. 2005

[5] S. Klein, L. Litz, J.-J. Lesage, "Fault detection of Discrete Event Systems using an identification approach", 16th IFAC World Congress, paper N°02643, 6 pages, Praha, Czech Republic, Jul. 2005

[6] M. Roth, J.-J. Lesage, L. Litz, "Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems", American Control Conf. (ACC 2010), Baltimore, Maryland, USA, pp. 2601-2606, Jun. 2010

[7] M. Dotoli, M. P. Fanti, A. M. Mangini, "Real time identification of discrete event systems using Petri nets", Automatica, 44(5), pp. 1209-1219, May. 2008

[8] M. Dotoli, M. P. Fanti, A. M. Mangini, W. Ukovich, "Identification of the unobservable behaviour of industrial automation systems by Petri nets", Control Engineering Practice, Vol. 19, Issue 9, Sep. 2011, pp. 958-966

[9] J.E. Cook, Z. Du, C. Liu, A. L. Wolf, "Discovering models of behavior for concurrent workflows", Computers in Industry, 53(3), pp. 297 – 319, 2004

[10] A. P. Estrada-Vargas, J.-J. Lesage, E. López-Mellado, "Stepwise Identification of Automated Discrete Manufacturing Systems", 16th IEEE Int. Conf. on Emerging Technologies and Factory Automation, Toulouse, France, paper N°4-6, 8 pages, Sep. 2011

[11] R. David and H. Alla, "Petri Nets for Modeling of Dynamic Systems–A Survey", Automatica, 30(2), pp. 175-202, 1994

[12] M. Roth, J.-J. Lesage, L. Litz, "Identification of Discrete Event Systems, implementation issues and model completeness", 7th Int. Conf. on Informatics in Control Automation and Robotics, (ICINCO 2010), pp. 73-80, Funchal, Portugal, Jun. 2010

[13] A.P. Estrada-Vargas, E. Lopez-Mellado, J.-J. Lesage, "An Identification Method for PLC-based Automated Discrete Event Systems". Proc. of the IEEE Int. Conf. on Decision and Control (CDC 2010), pp.6740-6746. Atlanta, USA. Dec. 2010