Open access • Journal Article • DOI:10.1109/TASE.2015.2445332

A Black-Box Identification Method for Automated Discrete-Event Systems

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

Institutions: CINVESTAV, École normale supérieure de Cachan

Published on: 01 Jul 2017 - IEEE Transactions on Automation Science and Engineering (IEEE)

Topics: Input/output, Petri net, Programmable logic controller, Stochastic Petri net and Black box

A Black-box Identification Method for Automated Discrete Event Systems

Ana Paula Estrada-Vargas, E. López-Mellado, Jean-Jacques Lesage

A Black-box Identification Method for Automated Discrete Event Systems

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

***Abstract*— This paper deals with the identification of discrete event manufacturing systems that are automated using a programmable logic controller (PLC). The behavior of the closed loop system (PLC and Plant) is observed during its operation and is represented by a single long sequence of observed input/output (I/O) signals vectors. The proposed method follows a black-box and passive identification approach that allows addressing large and complex industrial DES and yields compact and expressive interpreted Petri net (IPN) models. It consists of two complementary stages; the first one obtains, from the I/O sequence, the reactive part of the model composed by observable places and transitions. The I/O sequence is also mapped into a sequence of the created transitions, from which the second stage builds the non observable part of the model including places that ensure the reproduction of the observed input output sequence. This method, based on polynomial-time algorithms on the size of the input data, has been implemented as a software tool that generates and draws the IPN model; it has been tested with input/output sequences obtained from real systems in operation. The tool is described and its application is illustrated through a case study.**

***Note to practitioners*— Automated modeling of controlled discrete manufacturing systems can be achieved by efficient identification algorithms that cope with large and complex plants performing concurrent and repetitive tasks a priori unknown. The black-box identification procedure processes an input/output sequence recorded during the system functioning for a long period of time, and then yields a comprehensive model of the closed-loop controlled system; this model approximates closely the actual behavior of the compound system controller-plant. A tool based on identification algorithms constitutes an excellent resource for computer-aided reverse engineering of controlled manufacturing systems. The method proposed herein allows processing sequences composed by thousands of I/O vectors in few seconds.**

***Index Terms*— Discrete Event Systems, Black-box Identification, Interpreted Petri Nets.**

I. Introduction

IDENTIFICATION of discrete event systems (DES) allows building systematically a mathematical model (Petri nets, automata) that describes the behavior of an unknown or ill-known system based on the observation of its evolution. Observations consist of data revealing the system activity: sequences of operations, events, messages, signals etc., and the models allow reproducing the observed behavior.

A. Related works

DES identification has been first addressed as a problem of grammatical inference [1], [2] for obtaining finite automata (FA) that represents a given language. Afterwards, Petri net (PN) models have been proposed for coping with more complex systems exhibiting concurrent behavior; in [3] an algorithm for constructing PN models is presented.

Several approaches for identification of DES have been proposed in literature in various formulations and from diverse approaches. Below there is an overview of the main approaches; other works can be found in detailed surveys on identification methods [4] and [5].

In [6], [7] methods based on Integer Linear Programming (ILP) are proposed; they allow obtaining accurate Petri nets from a set of transition sequences that can be fired from the initial marking. These methods require the *a priori* knowledge of the set of transitions and of the number of places, what makes difficult their application to identify real DES as black boxes, since the only available information after observation is the evolution of input and outputs signals exchanged between the control system and the plant.

In [8], [9] it is described an efficient 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 [10] is dedicated to fault detection and isolation (FDI). It allows obtaining a finite automaton representing precisely a set of cyclic I/O sequences. An extension to distributed identification and distributed FDI has been presented in [11]. However, due to the usage of finite automata, structural information such as parallelism cannot be explicitly expressed into the models, what makes this approach inefficient for applications such reverse engineering. Other proposals related with FDI in the PN framework are presented in [12] and [13].

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 LURPA, ENS Cachan, Univ Paris-Sud, F-94235 Cachan, France. Email: Jean-Jacques.lesage@lurpa.ens-cachan.fr. A. P. Estrada-Vargas was with both institutions; she has been sponsored by CONACYT (Mexico) under Grant No. 50312, and by Région Île-de-France. Email: aestrada@gdl.cinvestav.mx.

In [14], [15] 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 method requires the definition of an event list, which is not available *a priori* 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. This work has been extended in [16] towards the determination of stochastic transitions for FDI purposes.

In [17] 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.

In [18] the modeling of a workflow is also considered. The input of the algorithm is a workflow log of several workflow instances composed by several tasks. Workflow instances are recorded sequentially, even if tasks may be executed in parallel. Based on the information in the workflow log and by making some assumptions about completeness of the log, a process model in the form of a workflow net is deduced.

B. Black-box approach

Beyond the theoretical interest of defining model synthesis methods from event 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 to 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 previous works [19], [20] 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.

C. Contribution

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. It consists of two complementary stages; the first one obtains, from the I/O sequence, the reactive part of the model composed by observable places and transitions. A first version of this stage has been presented in [21]. The I/O sequence is mapped into a sequence of the created transitions, from which the second stage builds the non-observable part of the model including places that ensure the reproduction of the observed input output sequence. This method, based on polynomial-time algorithms on the size of the input data, has been implemented as a software tool that generates and draws the IPN model [22]. None of the black-box identification approaches in related works allows obtaining such well structured models. The present article gathers both stages of the method; it includes a detailed presentation of the revised results, additional illustrative examples, all the proofs omitted in the conference papers, and a case study regarding the identification of a real process.

D. 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 and overviews the two steps method. Such steps are explained in Section IV and Section V. Finally, the implementation details and a case study are presented in Section VI.

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 $G = (P, T, Pre, Post)$ 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; $Pre(Post) : P \times T \rightarrow \{0,1\}$ is a function representing the arcs going from places to transitions (from transitions to places).

The incidence matrix of G is $W = W^+ - W^-$, where $W^- = [w_{ij}^-]$; $w_{ij}^- = Pre(p_i, t_j)$; and $W^+ = [w_{ij}^+]$; $w_{ij}^+ = Post(p_i, t_j)$ 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 a $|P|$-entry vector. Z^+ is the set of nonnegative integers. In particular, in this paper $M : P \rightarrow \{0,1\}$; the PN is referred as 1-bounded or *safe*.

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 $\forall p_i \in P$, $M_k(p_i) \geq Pre(p_i, t_j)$; 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 $M_{k+1} = M_k + Wu_k$, 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 $R(G,M_0)$.

Definition 3. A Petri net *circuit* is a path of vertices linked by arcs starting and ending in the same node. A circuit is said to be *simple* if it does not use the same transition more than once, and *elementary* if it does not use the same place more than once.

Now it is defined IPN, an extension to PN that allows associating input and output signals to PN models. This definition is adapted from [23].

Definition 4 *:* An interpreted Petri net (IPN) (Q, M_0) is a labeled net structure $Q = (G, \Sigma, \Phi, \lambda, \varphi)$ with an initial marking M_0 where:

- G is a PN structure,
- $\Sigma = \{I_1, I_2, ..., I_r\}$ is the input alphabet,
- $\Phi = \{O_1, O_2, ..., O_q\}$ 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 input conditions in which every C_i is a Boolean function on Σ; when a Ci is always true it is denoted as "=1", and
  - $E=\{E_1, E_2, \ldots\}$ is the set of input events conditions; every E_i is a Boolean function of input events, build on Σ; events are denoted as Ii_0 and Ii_1 for representing that the input value changes from 1 to 0, or from 0 to 1 respectively. A condition E_i may not exist; this is denoted as "ε".

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 with each marking in $R(G,M_0)$ a q-entry output vector, where $q=|\Phi|$ is the number of outputs. φ is represented by a $q \times |P|$ matrix, such that if the output symbol O_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 of PN 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 5: A place $p_i \in P$ is said to be *observable* if the i-th column vector of φ (denoted as $\varphi(\bullet,i)$) is not null. Otherwise it is *non-observable*. $P = P^{obs} \cup P^{nobs}$, and $P^{obs} \cap P^{nobs} = \varnothing$; where P^{obs} is the set of observable places and P^{nobs} the set of non-observable places.

III. Identification of Industrial Automated Systems

A. The process PLC+Plant

In this work we consider 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 with respect to the inputs-outputs of the PLC (Fig. 1). A PLC cyclically performs three main steps: i) Input reading, where signals are read from the sensors; ii) Program execution, to determine the new outputs values for the actuators; and iii) Output writing, where the control signals to the actuators are set. At each end of the Program execution phase, the current value of all r inputs and q 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 [24]. 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 sequence of I/O vectors whose length depends on the observation duration:

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

$I(k)$ and $O(k)$ are vectors whose entries are respectively the values of the r inputs $I_1, I_2, \ldots I_r$ and q outputs $O_1, O_2, \ldots O_q$ at the k-th PLC cycle. Furthermore we denote $I_i(k)$ and $O_i(k)$ the values of input I_i and output O_i respectively at the k-th cycle.

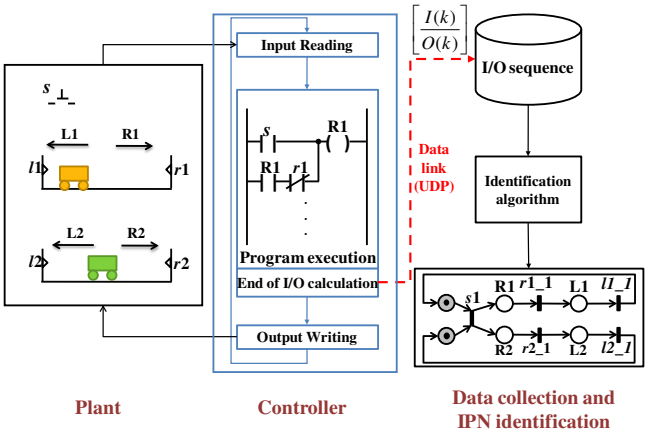

Fig. 1. {PLC + Plant} compound and identification procedure.

B. Event types

In order to analyze signals evolution, we compute *event vectors*, i.e., the difference between two consecutive I/O vectors. Each event vector can be decomposed into input and output event vectors:

$$E(k) = \left[\frac{IE(k)}{OE(k)}\right] = \left[\frac{I(k+1)}{O(k+1)}\right] - \left[\frac{I(k)}{O(k)}\right],$$

$$\text{where } IE(k) = \begin{bmatrix} IE_1(k) \\ IE_2(k) \\ \vdots \\ IE_r(k) \end{bmatrix} \text{ and } OE(k) = \begin{bmatrix} OE_1(k) \\ OE_1(k) \\ \vdots \\ OE_q(k) \end{bmatrix} \quad (2)$$

Regarding input and output event vectors and the PLC cycle described in the previous subsection, there only exist four

situations (behavior types) between consecutive I/O vectors that could be observed, which are explained by different occurring phenomena:

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

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

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

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

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

Let X(k) be the internal current state of the controller,

a) $X(k-1) \neq X(k)$ *An input evolution has provoked a non-observable state evolution of the controller.*

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

Type 4. $IE(k) = 0$ and $OE(k) = 0$

a) $X(k-1) \neq X(k)$ *It has occurred a non-observable state evolution of the controller which is not exhibited by any input nor output change.*

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

All of these situations should be taken into account to represent the system dynamics. Our aim in this work is to express the system's behavior extractible from the I/O vector sequence as an IPN.

C. Input-Output identification approach

C.1 Overview of the method

The purpose in this research is not only to compute an IPN model in which the observed sequence is reproducible, but also to achieve expressivity and compactness in the identified model allowing representing causal relationship and concurrency of the involved operations.

The method processes off-line the I/O-sequence *w* captured during the process operation and delivers an IPN model that reproduces the observed behavior (*w*).

The method is outlined here with the help of a simple example. It regards a controller handling 3 inputs (s, x, y) and 3 outputs (A, B, C), from which the following I/O sequence is obtained:

$$w = \begin{array}{c} \\ s \\ x \\ y \\ A \\ B \\ C \end{array} \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

The method consists of two main steps which are outlined below.

Step1. Discovering the reactive input/output behavior. In this step is determined the observable part of the IPN consisting of subnets, named fragments, composed by observable places labeled with output symbols, and transitions labeled with algebraic expressions of input symbols (Fig. 2). From the sequence *w,* a corresponding sequence of transitions $S=t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_5\ t_6\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_5$ is obtained.

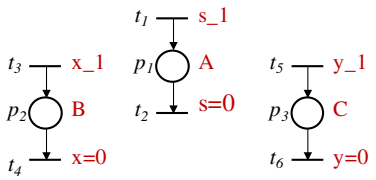

Fig 2. First step of the identification method: IPN fragments.

Step2. Determining the non-observable part of the IPN and the initial marking M_0. The sequence *S* is processed for obtaining causal and concurrency relationships useful for determining the non-observable places that relate the fragments such that *S* (thus *w*) can be executed from M_0 (Fig. 3).

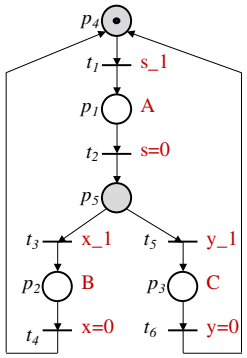

Fig 3. Second step: assembled IPN fragments

C.2 Dealing with event types

Since situations *Type 1* and *Type 2* (cf. section III.B) are directly observable by an output change, they can be straightforwardly modeled in an IPN. Such a modeling is performed by the first step of our method.

The *Type 1* situation represents a direct input/output reactive behavior, and thus the modeling is quite easy: the input change is associated with the label of a transition and the output change is represented as arcs relating such a transition with the observable places representing outputs involved.

In the *Type 2* situation the input values which lead to the output evolution are not observed at the same PLC cycle (i.e. at the same event vector). In order to represent such a behavior, the context (the values of the inputs) in which the output changes occur is analyzed; in this case, the output change is modeled such as in the *Type 1* situation, but the label of the corresponding transition contains only a condition on inputs levels (the input change is ε).

The *Type 3* situation is divided in two, depending on whether or not there is an internal state evolution of the controller. Situation *Type 3.a* is the case of the input events which provoke internal state evolutions and eventually lead to an output event of *Type 2*. Such internal evolutions cannot be directly computed, but can be inferred. By looking in the sequence built in Step 1, the order in which transitions appear can be determined. Such internal state inference will be performed by the second step of our method and will be

modeled by the addition of non observable places assuring the order of the transition firings, such as in Fig. 3.

In the situation *Type 3.b* there is no internal state evolution, and thus there is nothing to be inferred, as well as the situation *Type 4*, where there are neither input nor output events occurring in a PLC cycle. 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. Notice that in this work we can only infer internal state evolutions by means of transition firing order. Other type of internal evolutions, such as timers or counters, is out of the scope of this work. We can now make the description of the two identification steps.

IV. Identification of the Observable Behavior

In this section the first step of the method is presented. The introduced concepts and algorithms are illustrated through a simple case study inspired from a manufacturing example.

A. Overview and case study description

Algorithm 1 summarizes the steps of the procedure to identify the {PLC + Plant} observable behavior; the steps will be described in detail in the next sub-sections.

Algorithm 1.Computing the observable IPN components

Input: I/O sequence w

Output: Observable incidence matrix φW and labeling transition function λ

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

2) Use computed data in the previous step to
- Compute transitions of the IPN and their labeling λ
- Compute observable incidence matrix φW

Example 1. 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 [19], [20] and we describe it to confront this work against previous results.

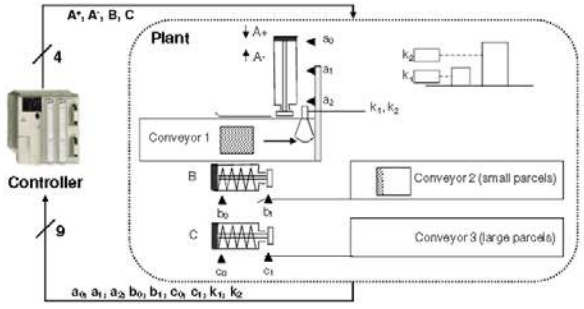

Fig. 4. Layout of the system case study.

B. Events vector sequence

In Fig. 5 the beginning of an I/O vector sequence is shown for illustrative purposes; however, recall that treated sequences are usually very much longer (thousands of vectors). We have included in the sequence the result of first sub-step of the algorithm, i.e., the computed event vectors (below the arrows) between each two consecutive I/O vectors.

$$w = \begin{matrix} k1 \\ k2 \\ a0 \\ a1 \\ a2 \\ b0 \\ b1 \\ c0 \\ c1 \\ A+ \\ A- \\ B \\ C \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{E(1)} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{E(2)} \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{E(3)} \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{E(4)} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{E(5)} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{E(6)} \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{E(7)} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Fig. 5. Beginning of I/O vector sequence.

C. 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):

$$IE(k) = \{IE_{k1}, IE_{k2}, ..., IE_{kr}\} = \bigcup_{i=1}^{i=r} IE_{ki} \text{ s.t. } I_i(k+1) - I_i(k) \neq 0 \quad (3)$$

$$OE(k) = \{OE_{k1}, OE_{k2}, ..., OE_{kq}\} = \bigcup_{i=1}^{i=q} OE_{ki} \text{ s.t. } O_i(k+1) - O_i(k) \neq 0 \quad (4)$$

If no elementary input (output) event occurs in $E(k)$, we denote it as $IE(j)=\{\varepsilon\}$ ($OE(j)=\{\varepsilon\}$). The rising edge event of input I_i (output O_i) is denoted as I_i_1 (O_i_1). The falling edge event of input I_i (output O_i) is denoted as I_i_0 (O_i_0).
Table 1 shows the elementary events computed for the example sequence.

TABLE I
Elementary Events List for Example 1

Event vector	Elementary input events	Elementary output events
$E(1)$	$IE(1) = \{k1\_1\}$	$OE(1) = \{A+\_1\}$
$E(2)$	$IE(2) = \{a0\_0\}$	$OE(2) = \{\varepsilon\}$
$E(3)$	$IE(3) = \{k1\_0\}$	$OE(3) = \{\varepsilon\}$
$E(4)$	$IE(4) = \{a1\_1\}$	$OE(4) = \{A+\_0,\ A-\_1,\ B\_1\}$
$E(5)$	$IE(5) = \{b0\_0\}$	$OE(5) = \{\varepsilon\}$
$E(6)$	$IE(6) = \{a1\_0\}$	$OE(6) = \{\varepsilon\}$
$E(7)$	$IE(7) = \{b1\_1\}$	$OE(7) = \{B\_0\}$

D. 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 occurrence of both input events IE_i and output events OE_k, with respect to the occurrence of OE_k along the whole sequence of events. This 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:

$$Prob(OE_k \mid IE_i) = \frac{N_{Obs}(OE_k, IE_i)}{N_{Obs}(OE_k)} \quad (5)$$

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 $DM_{ik} = Prob(OE_k|IE_i)$. Fig. 6 presents the computed *DM* matrix for the *Example 1*, considering a sequence much longer 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

Fig. 6. Direct Causality Matrix for the Example 1

Similarly, conditional probability has been used in [17] for determining the relationship between workflow operations.

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 number of occurrences of event OE_k. For example, from Fig. 6 we can discover that the output event A+_0 is *always* provoked by event a1_1 (in 44.4% of the observed cases) or by event a2_1 (in 55.6% of the observed cases). The general case where several input events can provoke an output event is formalized on the next section.

Similarly, to discover input/output non direct relationship, we look at the 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:

$$Prob(OE_k \mid IL_i) = \frac{N_{Obs}(OE_k, IL_i)}{N_{Obs}(OE_k)} \quad (6)$$

We construct the *Indirect Context Matrix* (*IM*) in which every $IM_{ik} = Prob(OE_k|IL_i)$. The *IM* matrix for *Example 1* is shown in Fig. 7.

Using the *IM* matrix we can discover evolution *Type 2* by inspecting in every column the values that add up to 1 which are not zero in the *DM* matrix. In the *Example 1*, 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.

	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

Fig. 7. Indirect Context Matrix of the Example 1.

E. Computing Firing Functions of Output Events

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. In order to represent such conditions, a firing function $\chi(OE_k)$ has to be defined for every OE_k. It is called the Output Event Firing Function (OEFF):

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

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 of input events E_j:

$$G(OE_k) = \Pi DisjE_j \quad (8)$$

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

$$DM_{xj} \neq 0, DM_{yj} \neq 0, \ldots DM_{zj} \neq 0 \quad (9)$$

$$DM_{xj} + DM_{yj} + \ldots + DM_{zj} = 1 \quad (10)$$

Similarly, $F(OE_k)$ is computed as a conjunction of disjunctions of input levels L_j:

$$F(OE_k) = \Pi DisjL_j \quad (11)$$

with $DisjL_j = (IL_x \oplus \ldots \oplus IL_z)$ such that

$$IM_{xj} \neq 0, IM_{yj} \neq 0, \ldots IM_{yj} \neq 0 \quad (12)$$

$$IM_{xj} + IM_{yj} + \ldots + IM_{zj} = 1 \quad (13)$$

$$DM_{xj} \neq 0, DM_{yj} \neq 0, \ldots DM_{yj} \neq 0 \quad (14)$$

At the end of the computing, 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 shown in Fig. 8.

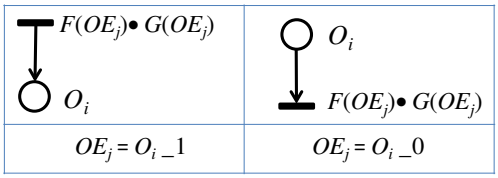

Fig. 8. Rising and falling edges of output O_i

F. Input events with differed influence on the outputs

Notice that condition $DM_{xj} \neq 0$, $DM_{yj} \neq 0,\ldots, DM_{zj} \neq 0$ requires that the inputs related to the output change were observed at least once changing its value at the same PLC cycle that the considered output. This condition may be restrictive if the input-output reaction is not observed in the same event vector. For example, in order to avoid component damages, in the absence of an input sensor to indicate that a pusher has been retracted, there may be some security temporizations which do not allow another actuator reacting at the moment an input condition has been satisfied.

In such cases, the input-output reaction would not be found and thus there may be an output event with empty conditions on its firing function. In order to find the correct *OEFF*, we can relax the condition to consider input events which have been observed in previous event vectors instead of the same event vector. Formally, we can compute:

$$Prob(OE_k \mid IE_i) = \frac{N_{Obs}(OE_k, IE_{i(\text{previous vector})})}{N_{Obs}(OE_k)} \qquad (15)$$

But this time, the computation is done by considering IE_i occurred at the previous event vector than OE_k. A new *OEFF* can be computed using new values instead of those of the *DM* matrix. If the computed *OEFF* has still empty conditions, we can take the previous to the previous event vector and successively while empty conditions are computed. In the *Example 1*, such relaxing condition is not necessary, since, as it can be noticed, no empty conditions have been computed. However, in the experimental case study of Section VI.B, such a technique is applied.

For the *Example 1*, $D = \varnothing$; the computed PN fragments are shown in Fig. 9.

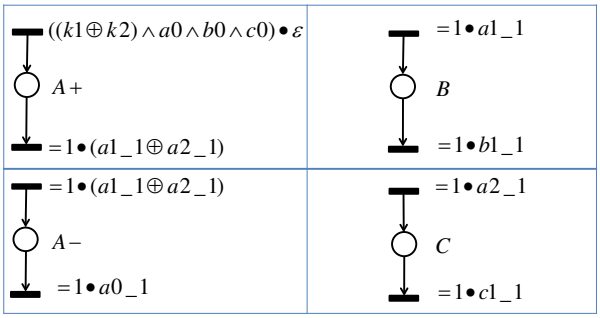

Fig. 9. IPN fragments for Example 1

G. 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 shown in Fig. 10.

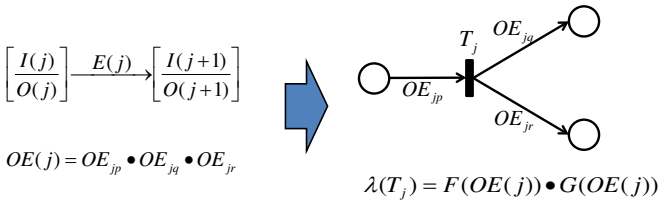

Fig. 10. IPN representation of several output events at the same cycle

The construction of the observable IPN can be systematically done with the next procedure:

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, labeling transition function λ, and sequence of transitions *S*

1. $P \leftarrow \{p_1, p_2, \ldots, p_q\}$ *//Create q observable places, one for every output of the system*
2. $S \leftarrow \varepsilon$ *//Initialize the sequence S*
3. $\forall E(j), j=1,\ldots,|E|$ **do** *//Consider all the computed I/O events in E*
   - 3.1. **If** $OE(j) = 0$ and $\exists\ IE_s,\ldots,IE_u \in IE(j) \cap D$ *//There is not an output change in E(j), but IE(j) contains elementary input events $IE_s,\ldots,IE_u$ belonging to D*
     then
     - 3.1.1. $T \leftarrow T \cup \{t_j\}$; $\lambda(t_j) \leftarrow IE_s \bullet \ldots \bullet IE_u$; $W(t_j,p_i) \leftarrow 0$, $\forall p_i \in P$ *//If it has not been created before, create a new zero transition t_j (a zero column in the incidence matrix) representing input changes $IE_s,\ldots,IE_u$*
     - 3.1.2. $S \leftarrow S \cdot t_j$ *//Concatenate t_j to S*
   - 3.2. **else if** $OE(j) \neq 0$ *//There is an output change in E(j)*
     then
     - 3.2.1. $\forall OE_{jk} \in OE(j)$ *//Consider all the elementary output events in OE(j) in order to compute G(OE(j)) and F(OE(j))*
       - 3.2.1.1. $\forall DisjE_i \in G(OE_{jk})$, do $DisjE_i' \leftarrow DisjE_i \cap IE_{jk}$ *// Look into IE(j) the input event IE_{jk} which has satisfied $DisjE_i$ and assign it to $DisjE_i'$*
       - 3.2.1.2. $G'(OE_{jk}) \leftarrow \Pi DisjE_i'$ *//Combine into $G'(OE_{jk})$ all the conditions $DisjE_i'$ which have satisfied $G(OE_{jk})$*
       - 3.2.1.3. $G(OE(j)) \leftarrow \Pi G'(OE_{jk})$ *//Combine into G(OE(j)) all the input event conditions $G'(OE_{jk})$ which have satisfied all the events OE_{jk}*
       - 3.2.1.4. $\forall DisjL_i \in F(OE_{jk})$, do $DisjL_i' \leftarrow DisjL_i \cap I(j+1)$ *// Looking the I(j+1) vector as a set of Boolean variables, save into $DisjL_i'$ the input value IL_{ik} which has satisfied $DisjL_i$*
       - 3.2.1.5. $F'(OE_{jk}) \leftarrow \Pi DisjL_i'$ *//Combine into $F'(OE_{jk})$ all the conditions which have made true $F(OE_{jk})$*
       - 3.2.1.6. $F(OE(j)) \leftarrow \Pi F'(OE_{jk})$ *//Combine into F(OE(j)) all the conditions which have produced all the OE_{jk}*

3.2.2. $T \leftarrow T \cup \{t_j\}$, $\lambda(t_j) = F(OE(j)) \bullet G(OE(j))$ *//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))*

3.2.3. $\forall p_i \in P$, **do**

If $O_q\_1 \in OE(j)$ **then** $W(t_j,p_q) \leftarrow 1$,

else If $O_q\_0 \in OE(j)$ **then** $W(t_j,p_q) \leftarrow -1$,

else $W(t_j,p_{jq}) \leftarrow 0$ *//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, assign a 0.*

3.2.4. $S \leftarrow S \cdot t_j$ *//Concatenate t_j to S*

Complexity of Algorithm 2. Let $r' \prec r$ and $q' \prec q$ be respectively the maximum number of input and output elementary events appearing simultaneously in an event vector, and $|E|$ be the length of the events sequence. The Algorithm 2 processes each one of the events in E. When a transition should be added to represent one of such events, an appropriate firing function should be computed. If only inputs changed, it is only necessary to include in the firing function the elementary input events with differed influence. This is achieved in $O(r')$. If there is at least an output change, for each one of the output elementary events which have occurred, we need search for each individual firing function the input events and input conditions that produced the evolution. This is performed in $O(q'(r' \log r'))$. Thus, the complexity of the procedure for building the transition sequence and fragments is $O((q'r' \log r')|E|)$. Consequently, the *Algorithm 2* can be executed in polynomial time on the size of the input data.

Property 1. The transitions sequence S is a translation of the I/O sequence w into transition firings of the *PN*-fragments built by Algorithm 2.

Proof. It is easy to see that at the steps 3.1.2 and 3.2.4 of Algorithm 2, S is formed by concatenating the computed transitions from the event sequence produced by w. This allows that the reactive behavior can be reproduced in the created IPN model. ■

Fig. 11 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 and 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).

At the end of the procedure, the following observable incidence matrix φW and labeling functions are obtained, as well as the transition sequence S, which is the projection of w over T: $S = t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_4\ t_3\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_4\ t_3\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_4\ t_3\ t_1\ t_2\ t_3\ t_4\ t_1$.

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

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

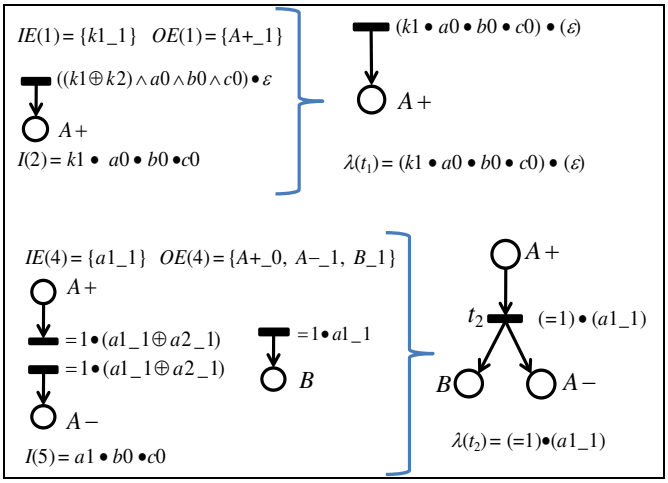

Fig. 11. Treatment of $E(1)$ and $E(4)$ by the Algorithm 2.

The corresponding partial model is shown in Fig. 12. The inferring procedure which allows discovering the non-observable behavior is described in next section.

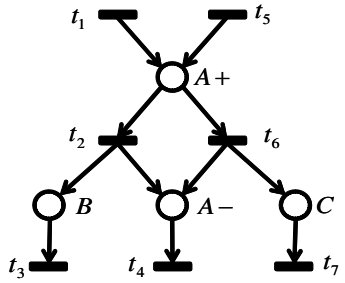

Fig. 12. Observable IPN model

V. IDENTIFICATION OF THE NON-OBSERVABLE BEHAVIOR

A. Problem (re)statement

The previously described procedures allow obtaining an observable structure which represents the reactive behavior of the system. Given that events and transitions of the net are completely defined, we need to add non-observable places to translate an aggregation of the non-observable dynamics of the process in such a way that the global PN will reproduce the whole behavior of the system. By adding non-observable places (depicted as grey circles), we make the inference of situation *Type 3.a* described in Section III.*B*, which is the case of input events provoking internal state evolutions.

The problem of determining the non-observable part of the *IPN* model complementary to that describing the observable (reactive) behavior can be stated as follows.

Given an observable *IPN* model whose structure is $G^{obs}=(P^{obs}, T, Pre^{obs}, Post^{obs})$ and a transitions sequence $S = t_1\ t_2\ \ldots\ t_j\ \ldots\ \in T^*$ reproducing the I/O sequence w, an ordinary *PN* structure $G^{nobs}=(P^{nobs}, T, Pre^{nobs}, Post^{nobs})$ that reproduces S and an initial marking M_0 enabling S must be found; (G^{obs}, M_0) must be safe.

Thus, the *PN* structure of the complete identified model is $G=(P, T, Pre, Post)$ with $P= P^{obs} \cup P^{nobs}$, $Pre= Pre^{obs} \cup Pre^{nobs}$, $Post= Post^{obs} \cup Post^{nobs}$.

Observe that in S there are not consecutive apparitions of the same transition, due to the nature of the considered events (rising and falling edges of binary signals).

In the literature there are many approaches which tackle the identification problem as stated above. However, our problem exceeds the hypothesis held in such works or they are not enough efficient to cope with long sequences. In particular,

a) the system cycles are not know *a priori*,

b) the whole language of the system is not known,

c) the size of S is very large; thus finding efficient algorithms is required,

d) the aim is building *IPN*s that shows structural parallelism.

New places and arcs must be determined such that they join the *IPN* fragments found in the first part of the method. Since the tasks in different processes can occur simultaneously or at some predefined order, each two fragments can be related in two manners only: sequentially or concurrently. Thus, several connecting forms are possible. Some of them are illustrated in Fig. 13, where "clouds" represent the fragments.

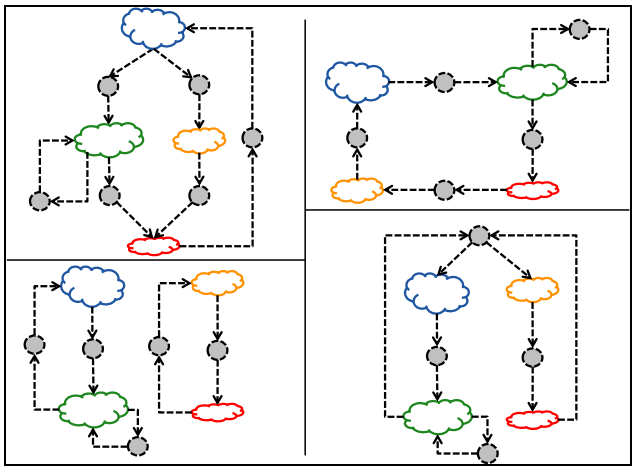

Fig. 13. Some different possibilities for fragments assembling

In this section, we present a procedure to build a non-observable PN structure that is able to reproduce the sequence S of transitions firings. This construction principle is based on the precedence and concurrency relations among transitions, which will determine the final structure of the identified model. Algorithm *3* given below provides an overview of the procedure.

Algorithm 3. **Non-observable behavior construction**

Input: Transitions sequence S

Output: Non-observable model representing S

1. Compute basic structures and relations (*Seq*, *SP*, and *TC*) from S
2. From the information in *Seq*, *SP* and *TC* compute the causal relation between transitions *CausalR*
3. From *Seq* and *CausalR*, compute concurrency relation (*ConcR*) between transitions
4. Build a *PN* model representing *CausalR* and *ConcR*
5. Verify the tokens flow and correct part of the structure if needed

The steps of Algorithm *3* are detailed in the following subsections. First some properties derived from the sequence S are introduced. Afterwards, based on such properties, an analysis technique allowing determining causal and concurrency relationships among the transitions in S is proposed. Then, the steps for building a *PN* structure observing the causal and concurrency relationships are presented.

B. Dynamical properties

Since the construction method is based on the analysis of causal and concurrency relationships, some notions must be defined before introducing the non-observable behavior construction procedure.

Definition 6. The relationship between transitions in S that are observed consecutively is expressed in a relation $Seq \subseteq T \times T$ which is defined as $Seq =\{(t_j, t_{j+1}) |\ 1 \le j < |S|\ \}$. If $(t_a, t_b) \in Seq$, this is denoted by $t_a \prec t_b$.

Example 2. Let us reuse the sequence S of example 1, which is the projection of the observed I/O sequence w over the set of observable transitions T:

$S = t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_4\ t_3\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_4\ t_3\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_4\ t_3\ t_1\ t_2\ t_3\ t_4\ t_1$

We can compute $Seq = \{(t_1, t_2), (t_2, t_3), (t_3, t_4), (t_4, t_1), (t_2, t_4), (t_4, t_3), (t_3, t_5), (t_5, t_6), (t_6, t_7), (t_7, t_4), (t_4, t_5), (t_3, t_1)\}$, which can be expressed also as $\{t_1 \prec t_2, t_2 \prec t_3, t_3 \prec t_4, t_4 \prec t_1, t_2 \prec t_4, t_4 \prec t_3, t_3 \prec t_5, t_5 \prec t_6, t_6 \prec t_7, t_7 \prec t_4, t_4 \prec t_5, t_3 \prec t_1\}$.

In a *PN* model every pair in *Seq* may in fact be represented differently. If t_a, t_b were observed consecutively in S, this behavior could be issued from one of two situations in G^{nobs} described in the following definition.

Definition 7. Every couple of consecutive transitions t_a, t_b in *Seq* can be classified in one of the following situations:

Causal relationship. If the occurrence of t_a enables t_b. In a PN structure, this implies that there must be at least one place from t_a to t_b (Fig. 14a).

Concurrent relationship. If both t_a and t_b are simultaneously enabled, but t_a occurs first and its firing does not disable t_b. In an ordinary *PN* structure, this implies that it is impossible the existence of a place from t_a to t_b. In this case, t_a and t_b are said to be concurrent, denoted as $t_a \| t_b$. (Fig. 14b).

In order to find which is the situation occurring between every pair of transitions in *Seq*, some other definitions are now introduced. The following notion is the *systematical precedence* of a transition t_j with respect to another transition t_k; it establishes a necessary condition for t_j to occur repeatedly.

Definition 8. A transition t_j is *preceded systematically* by t_k, denoted as $t_k \angle t_j$ iff t_k is always observed between two apparitions of t_j in *S*. By convention, we say that $t_j \angle t_j$ if t_j was observed at least twice in *S*. Then the *Systematical Precedence Set* of a transition t_j is given by the function SP: $T \rightarrow 2^T$, that indicates which transitions *must* be fired to re-enable the firing of t_j, i.e. $SP(t_j)=\{t_k \mid t_k \angle t_j\}$. If t_j was observed only once in *S*, then $SP(t_j) = \varnothing$.

In the sequence *S* from *Example 2*, one may compute that $t_1 \angle t_1$, $t_2 \angle t_1$, $t_3 \angle t_1$, and $t_4 \angle t_1$, thus $SP(t_1)=\{t_1, t_2, t_3, t_4\}$. Notice that $SP(t_j)$ is the set of transitions that must invariantly occur to fire t_j repeatedly. The rest of the *SP* sets are : $SP(t_2)=\{t_1, t_2, t_3, t_4\}$, $SP(t_3)=\{t_1, t_2, t_3\}$, $SP(t_4)=\{t_4\}$, $SP(t_5)=\{t_4, t_5, t_6, t_7\}$, $SP(t_6)=\{t_4, t_5, t_6, t_7\}$, $SP(t_7)=\{t_4, t_5, t_6, t_7\}$.

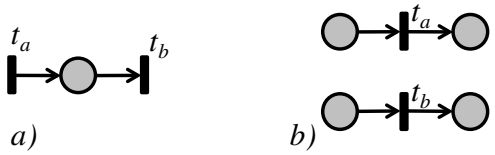

Fig. 14. Structures that represent $t_a \prec t_b$. a) shows a causal relationship from t_a to t_b, whereas b) shows a concurrent relationship between t_a and t_b.

Definition 9. Two transitions t_a, t_b are called transitions in a two length cycle relationship (named two-cycle transitions) if *S* contains the subsequence $t_a t_b t_a$ or the subsequence $t_b t_a t_b$. The two-cycle transitions set *TC* of S is given by $TC=\{(t_a,t_b) \mid t_a, t_b$ are in a two-cycle$\}$.

From the sequence *S* in *Example 2*, we observe that the set of transitions in a two-cycle is $TC= \varnothing$.

Remark. Computing *Seq*, *SP* and *TC* can be executed in polynomial time on the size of *S*.

We will now extract some structural properties regarding *N* from *S*. The previously defined terms will be used to determine which situation between causality and concurrence is the most appropriated for every pair of consecutively observed transitions in *S*.

C. Causal and concurrency relationships

1) Causal relationship

In order to determine that two transitions are causally related as shown in Figure 14.a, several conditions stated below must be fulfilled.

Proposition 1. If $t_a \angle t_b$ ($t_a \in SP(t_b)$) then, there must exist in *N* a simple elementary circuit (SE circuit) to which both t_a and t_b belong.

Proof. Suppose that there is not a SE circuit containing t_a and t_b. Thus, right after the firing of t_b, all the tokens in $t_b^\bullet$ (the output places of t_b) could be displaced by transition firings through some path to ${}^\bullet t_b$ (the input places of t_b), enabling t_b without needing to fire t_a, which implies that $t_a \notin SP(t_b)$. ■

Proposition 2. If $t_a \prec t_b$ and $t_a \angle t_b$, then there must exist in *N* a place from t_a to t_b.

Proof. Suppose that there is not a place from t_a to t_b. In order to allow the observation $t_a \prec t_b$, both t_a and t_b should be enabled simultaneously. By Proposition 1, there is at least one SE circuit containing t_a and t_b and thus, at least one path from t_a to t_b. Thus, if t_a and t_b are enabled simultaneously and t_a is fired, all paths from t_a to t_b contain two tokens. If all transitions in a path from t_a to t_b are fired, then there will be two tokens in one of the input places of t_b, resulting in a non-safe net. Then, at least one of the transitions t_i in each path from t_a to t_b must be conditioned to the previous firing of t_b. But if t_b is fired, all the transitions in paths from t_a to t_b can be fired and all the transitions in paths from t_b to t_b which do not include t_a can be fired; thus t_b will be enabled before t_a fires and as a consequence $t_a \notin SP(t_b)$. ■

Proposition 3. If $t_a \prec t_b$ and $t_b \angle t_a$, then there must exist in *N* a place from t_a to t_b.

Proof. Suppose that there is not a place from t_a to t_b. Then, before the observation of $t_a \prec t_b$, both t_a and t_b must be enabled, and thus the occurrence of $t_b \prec t_a$ is possible. Furthermore, together with $t_b \angle t_a$ and by Proposition 2 implies that there should be a place from t_b to t_a. However, at the firing of t_b there are two tokens in such a place, and thus the net is not safe. ■

Proposition 4. If $(t_a,t_b) \in TC$, then there must exist in *N* a place from t_a to t_b and a place from t_b to t_a.

Proof. The sequence $t_a t_b t_a$ must be reproducible in *N*. Right after the firing of t_a there is a token on its output places, and thus t_b must be at the output of such places; otherwise, there would be two tokens in such places after the second firing of t_a. Similarly, right after the first firing of t_a, there are no tokens on its input places, and thus t_b must be at the input of such places; otherwise, t_a could not be fired again. The same reasoning can be applied to reproduce the sequence $t_b t_a t_b$. ■

Notice that when two transitions are observed consecutively and one is systematically preceded by the other, a causal relationship is found. Also, when two transitions are involved in a two-cycle relation, they are in a causal relationship each other. Observe that all of these relationships are structural, and thus they do not depend of the initial marking of the net.

Definition 10. The causal relationship set *CausalR* keeps track of all the causal relationships in *S*. $CausalR = \{(t_a,t_b) \mid (t_a \prec t_b)$ and ($t_a \angle t_b$ or $t_b \angle t_a$ or $(t_a,t_b) \in TC)\}$.

From the *Seq* set in *Example 2* (see Definition 6), the *SP* sets (see Definition 8) and the *TC* set (see Definition 9) we compute $CausalR=\{(t_1,t_2), (t_2,t_3), (t_4,t_1), (t_2,t_4), (t_5,t_6), (t_6,t_7), (t_7,t_4), (t_4,t_5), (t_3,t_1)\}$.

If a couple of transitions (t_a,t_b) in the *Seq* set, belongs also to *CausalR*, then there must be a place from t_a to t_b in order to

constrain the observed firing order. For the rest of the transition couples in *Seq*, we must decide if a place should exist to relate them. Next, we will discuss some cases where the existence of a place can be discarded.

2) Concurrency relationship

If two transitions t_a and t_b are concurrent, there must not exist a place neither from t_a to t_b nor from t_b to t_a; otherwise, the firing of one would constrain the firing of the other one.

Definition 11. The set of all pairs of concurrent transitions is called $ConcR=\{(t_a,t_b) \mid t_a \| t_b\}$.

If the sequence w is complete, (consequently, S) i.e., if it exhibits all of the possible behavior of the observed system, we can find concurrence between transitions that are not in a causal relation, as shown in the next proposition.

Proposition 5. Let t_a, t_b be two transitions which have been observed consecutively in a complete sequence S in both orders, i.e. $(t_a, t_b) \in Seq$, $(t_b, t_a) \in Seq$. Then $(t_a, t_b) \notin CausalR$ and $(t_b, t_a) \notin CausalR$ if and only if $t_a \| t_b$.

Proof. Suppose that $(t_a,t_b) \notin ConcR$. Without loss of generality, we suppose there is a place p_{ab} from t_a to t_b. Since $(t_b, t_a) \in Seq$, there must also be a place p_{ba} from t_b to t_a; otherwise, t_a could be enabled simultaneously with t_b to allow $t_b < t_a$ and t_a may be fired, yielding to the presence of two tokens in the place p_{ab} and breaking the safeness condition. Since $(t_a,t_b) \notin CausalR$, $t_b \notin SP(t_a)$ and thus there must be at least one path from p_{ab} to p_{ba} which does not contain t_b. Similarly, there must be at least one path from p_{ba} to p_{ab} which does not contain t_a. Since $(t_a,t_b) \notin TC$, $t_a t_b t_a$ should not be enabled and thus, there must be at least one SE circuit to which t_a belongs, but t_b does not belong. The resulting net violates the free-choice conditions (observe Fig. 15).

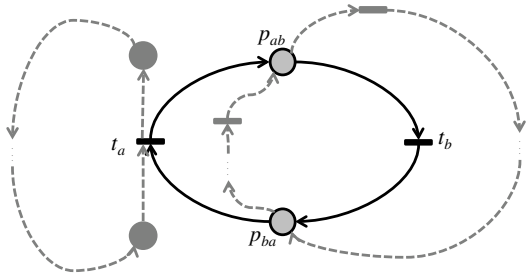

Fig. 15. Structure where $(t_a, t_b) \in Seq$ and $(t_b, t_a) \in Seq$ but $(t_a,t_b) \notin ConcR$

Suppose now that $(t_a,t_b) \in ConcR$. That means that they can be both enabled simultaneously and one can be fired without needing the firing of the other one, and thus $t_a \notin SP(t_b)$ and $t_b \notin SP(t_a)$. Also, since there cannot be any place from t_a to t_b nor from t_b to t_a, neither the subsequence $t_a t_b t_a$, nor the subsequence $t_b t_a t_b$ can be enabled, and thus $(t_a,t_b) \notin CausalR$ and $(t_b, t_a) \notin CausalR$. ■

Notice that our methodology allows computing also non free-choice nets. Only in the case where the system includes a behavior like the one shown in Fig. 15, the transitions t_a and t_b would be wrongly considered as concurrent and the existence of links from t_a to p_{ab} and from t_b to p_{ba} would be missed. However, the obtained model would be still capable to reproduce the sequence S.

It is well known that in practice, the sequence w is not complete, since in the general case, the observed systems do not show all their possible behavior during a finite time of data collection. In fact, it is not possible to assure that the whole behavior of a system has been observed. The consideration of Proposition 5 is then very restrictive, since it demands the observation of all possible behavior; it could lead to the construction of incorrect models in case of incomplete sequences. Then, some less constraining rules to find concurrence must be considered. Next, we present several properties which allow us to identify couples of transitions which must be concurrent in the identified net N.

First, we will introduce the notion of *Sequential Independence*, which is a characteristic of concurrent transitions. Later, the propositions to find concurrency will be introduced.

Definition 12. Two transitions t_a and t_b are *Sequentially Independent* if $t_a \notin SP(t_b)$ and $t_b \notin SP(t_a)$.

From the *SP* sets of *Example 2* (see Definition 8) we compute the set of Sequentially Independent transitions: $\{(t_1,t_5), (t_1,t_6), (t_1,t_7), (t_2,t_5), (t_2,t_6), (t_2,t_7), (t_3,t_4), (t_3,t_5), (t_3,t_6)\}$.

Observe the net in Fig. 16 which is composed by two independent t-components X_1 and X_2 with supports $\langle X_1 \rangle = \{t_a, t_i\}$ and $\langle X_2 \rangle = \{t_b, t_k\}$ respectively. In a sequence belonging to the language of such a net, transitions belonging to different t-components are sequentially independent. In fact, *SP* sets of this net correspond exactly to t-components of the net.

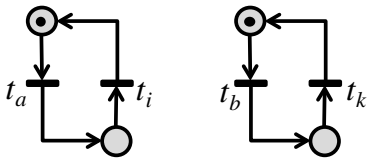

Fig. 16. A net with two t-components

Proposition 6. Let t_a and t_b be two transitions in S which have been observed consecutively in both orders ($t_a < t_b$ and $t_b < t_a$). If:

a) $(t_a,t_b) \notin CausalR$ and $(t_b,t_a) \notin CausalR$,

b) and $|SP(t_a)| > 1$ and $|SP(t_b)| > 1$,

then $t_a \| t_b$.

Proof. Suppose that t_a and t_b are not concurrent. Without loss of generality, we suppose there is a place p_{ab} from t_a to t_b. Since $t_b < t_a$ has been observed, there must be also a place p_{ba} from t_b to t_a (and as consequence N contains a two-transition cycle); otherwise, t_a could be enabled simultaneously with t_b to allow $t_b < t_a$ and t_a may be fired, yielding to the presence of two tokens in the place p_{ab} and breaking the safeness condition. Since $t_b \notin SP(t_a)$, there must be at least one path leading from p_{ab} to p_{ba} not including t_b. Since $|SP(t_a)| > 1$, there must be at least one circuit including t_a and not including p_{ab}, p_{ba} nor t_b. Since $t_a \notin SP(t_b)$, there must be at least one path leading from

p_{ba} to p_{ab} not including t_a. Consider the first transition t_x of this path. The free-choice conditions are not satisfied, since t_x and t_a share p_{ba} as input place, but t_a has at least one different input place. ■

We may observe that for *Example 2,* (t_3, t_4) are sequentially independent (see Definition 12), however, $|SP(t_4)| = 1$ and thus we cannot infer any concurrence.

When $SP(t_j)$ is a singleton, it means that it belongs to several elementary circuits and then Proposition 6 does not allow to find concurrent transitions to t_j. But if t_j is included in the *SP* of other transitions, we may find some concurrence relations, as shown in the next proposition.

Proposition 7. Let t_a and t_b be two transitions in S that have been observed consecutively in both orders ($t_a \prec t_b$ and $t_b \prec t_a$). If t_a and t_b

a) are *Sequentially Independent* and

b) there exists a transition t_k such that $t_a \angle t_k$ ($t_a \in SP(t_k)$) and $t_b \angle t_k$ ($t_b \in SP(t_k)$)

then $t_a \| t_b$.

Proof. Suppose that it does not hold that $t_a \| t_b$. Without loss of generality, we suppose that there is a place from t_a to t_b. Since $t_a \in SP(t_k)$ and $t_b \in SP(t_k)$, after the firing of t_k, both t_a and t_b must be fired before the next firing of t_k. Since $t_b \prec t_a$ may happen, the place from t_a to t_b must be marked. However $t_a \prec t_b$ may occur too, leading to the presence of two tokens in the same place after the firing of t_a, and making the net not safe. ■

Fig. 17 shows an example of the case characterized by Proposition 7. It is the general case of transitions belonging to concurrent threads (t_a, t_c and t_b, t_d, t_e, t_f respectively), which are eventually synchronized by one transition (t_k). If we make several firings to build a transition sequence, eventually the *SP* sets would become: $SP(t_k) = \{t_k, t_a, t_c, t_b, t_d, t_f\}$, $SP(t_a) = SP(t_c) = \{t_k, t_a, t_c\}$, $SP(t_b) = SP(t_f) = \{t_k, t_b, t_d, t_f\}$, $SP(t_e) = \{t_e, t_d\}$, $SP(t_d) = \{t_d\}$. Even if $SP(t_d)$ is a singleton, the synchronization point t_k help us to find by Proposition 7 several concurrent relationships: $t_a \| t_b$, $t_a \| t_d$, $t_a \| t_f$, $t_c \| t_b$, $t_c \| t_d$, and $t_c \| t_f$.

In *Example 2*, (t_3,t_4) are Sequentially Independent (see Definition 12), and we have determined that $t_3 \angle t_1$ and $t_4 \angle t_1$ (see Definition 8), thus we can conclude that $t_3 \| t_4$.

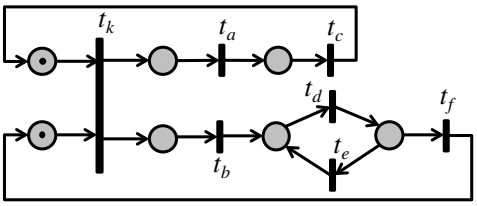

Fig. 17. Concurrent threads synchronized by a transition

If concurrent transitions do not belong to synchronized threads, conditions of the next propositions help us to find a subset of concurrent transitions which do not depend from another transition t_k.

Proposition 8. Let be two transitions t_a and t_b which have been observed consecutively in both orders ($t_a \prec t_b$ and $t_b \prec t_a$). If t_a and t_b are:

a) Sequentially Independent, and

b) $\exists\ t_k$ such that $t_k \in SP(t_b)$, $t_k \notin SP(t_a)$, and

c) $(t_a, t_k) \in Seq$

then $t_a \| t_b$.

Proof. Suppose there is a place p_{ab} from t_a to t_b. Since $t_b \prec t_a$ has also been observed, there must be also a place p_{ba} from t_b to t_a; otherwise, t_a should be enabled simultaneously with t_b to allow $t_b \prec t_a$ and t_a may be fired, yielding to the presence of two tokens in p_{ab}. Since there exists t_k such that $t_k \in SP(t_b)$, then there must be a SE circuit containing both t_b and t_k. If such a circuit contains places p_{ba} or p_{ba}, it is not possible to fire $t_a \prec t_k$ and thus such a circuit must contain another input place p_{kb} of t_b and another output place p_{bk} of t_b. Now, to accomplish that $t_b \notin SP(t_a)$, there must be at least one path leading from p_{ab} to some input place of t_a not including t_b. Consider the first transition t_x of this path. In order to respect the free-choice conditions, p_{kb} should be an input place of t_x, making the occurrence of $t_a \prec t_k$ impossible. ■

Definition 13. The *Inverse Systematical Precedence* set of a transition SP^{-1}: $T \rightarrow 2^T$ contains the transitions which are dependent of a common transition to re-enable their firing:

$$PS^{-1}(t_j) = \{t_k \mid t_k \neq t_j \text{ and } t_j \in PS(t_k)\} \quad (13)$$

Proposition 9. Let t_a and t_b be two transitions which have been observed consecutively in both orders ($t_a \prec t_b$ and $t_b \prec t_a$). If t_a and t_b are: *Sequentially Independent*, and $SP^{-1}(t_a) \neq \emptyset$, $\forall t_j \in SP^{-1}(t_a)$, $t_j \parallel t_b$, then $t_a \| t_b$.

Proof. Suppose there is a place p_{ab} from t_a to t_b. Since $t_b \prec t_a$ has also been observed, there must be also a place p_{ba} from t_b to t_a; otherwise, t_a should be enabled simultaneously with t_b to allow $t_b \prec t_a$ and thus t_a may be fired, yielding to the presence of two tokens in the place from t_a to t_b. Since $t_b \notin SP(t_a)$, there must be at least one path leading from p_{ab} to p_{ba} not including t_b. Similarly, there must be at least one path leading from p_{ba} to p_{ab} not including t_a. Since $SP^{-1}(t_a) \neq \emptyset$, there is at least one transition t_j concurrent to t_b such that $t_j \angle t_a$ and there must be a SE circuit including t_a and t_j. Such a circuit cannot contain p_{ab} nor p_{ba} otherwise t_j may be able to fire without need of firing t_a. Consider the input place p_x of t_a in this path. The free-choice conditions are not satisfied between p_x and p_{ba}: they share t_a as output transition, but p_{ba} has at least another output transition. ■

An example where Proposition 9 can be used is shown in Fig. 18. $SP^{-1}(t_a) = \{t_{j1}, t_{j2}\}$ and $t_{j1} \| t_b$, $t_{j2} \| t_b$ are determined by Proposition 6. Consequently, $t_a \parallel t_b$.

Remark. Computing *CasualR* and *ConcR* can be executed in polynomial time on the size of *S*.

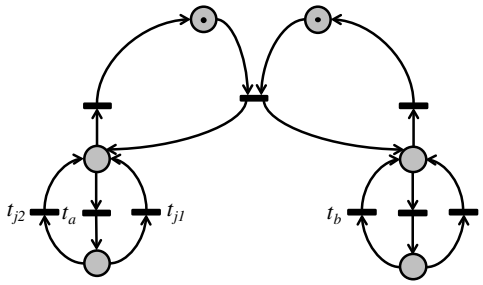

Fig. 18. Concurrence between transitions whose *SP* is a singleton

D. Building the non-observable PN

We will use now the computed data from sequence *S* to infer internal evolutions of the system. We will make an analysis of causal and concurrency relations that have been found between consecutive transitions in order to compute non-observable places of the net.

Definition 14. The set *Seq*'= (*Seq* - *CausalR*) - *ConcR* contains the set of transition pairs (t_a,t_b) which have been observed consecutively, but are not in a causal relation or in a concurrency relation.

Until now, we have computed for *Example 2* that *Seq* = {(t_1, t_2), (t_2, t_3), (t_3, t_4), (t_4, t_1), (t_2, t_4), (t_4, t_3), (t_3, t_5), (t_5, t_6), (t_6, t_7), (t_7, t_4), (t_4, t_5), (t_3, t_1)}, *CausalR*={(t_1, t_2), (t_2, t_3), (t_4, t_1), (t_4, t_5), (t_5, t_6), (t_6, t_7), (t_7, t_4), (t_2, t_4), (t_3, t_1)} and *ConcR* = {(t_3,t_4)(t_4,t_3)}. Thus, *Seq*' = {(t_3, t_5)}. This means that there is a relationship which has not been explained.

If *Seq*' ≠ ∅, then there are two possibilities for the remaining transition pairs (t_a,t_b) in *Seq*':

b) They are both input and output transitions of a place with several input and output transitions
c) They are concurrent, but *w* (thus, *S*) is not complete enough to find such a relationship

Since our goal is to approximate as much as possible the language generated by identified IPN, to the observed sequence *S*, we assume that if we have observed two transitions consecutively (t_a<t_b) but by none of the previous propositions we have determined that they are concurrent, thus the firing of t_a has enabled t_b. This is made in order to preserve in the PN the firing order observed in *S*. Then, a place will be added from t_a to t_b; this denoted by [t_a, t_b].

When it is found that [t_a, t_c] and [t_b, t_c], and the involved transitions are related by a single place, this is represented as [$t_a t_b$, t_c]. In general, a place *p* can be denoted as [t_{a1} $t_{a2...}$ t_{al}, t_{b1} $t_{b2...}$ t_{bh}], where t_{ai} are the input transitions of *p* and t_{bi} are the output transitions of *p*, and l=|•*p*|, h=|*p*•|, as illustrated in Fig. 19.

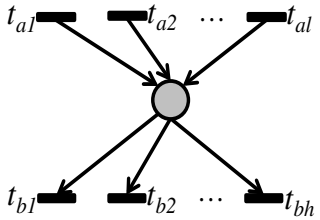

Fig. 19. A PN place *p* = [t_{a1} $t_{a2...}t_{al}$, t_{b1} $t_{b2...}$ t_{bh}]

The same place could be used to relate several consecutive transitions. If a transition t_k has been observed followed by two transitions t_{ai}, t_{aj} in *S* (t_k<t_{ai} and t_k<t_{aj}), there are two cases to represent such observations into the PN model: the case of selection, where they are represented with the same place [t_k, t_{ai} t_{aj}] (Fig. 20a) or the case of concurrence, where they are represented with different places [t_k,t_{ai}] [t_k, t_{aj}] (Fig. 20b).

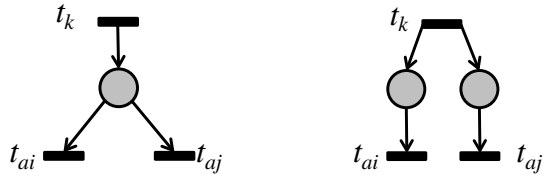

Fig. 20. Selection and parallelism representation. a)Shows the case where t_{ai}, t_{aj} are not concurrent and have not been observed consecutively whereas b) shows the case where t_{ai}, t_{aj} are concurrent or have been observed consecutively.

In a generalized form, for every set t_k<t_{a1},…, t_k<t_{aw} of non-concurrent consecutive transition pairs with the same first transition t_k, we can thus merge all t_k<t_{a1},…, t_k<t_{ax} whose second transitions t_{a1}…t_{aw} are non-concurrent nor consecutive and represent them into a single place [t_k, t_{a1}…t_{aw}], as illustrated in Fig. 21.

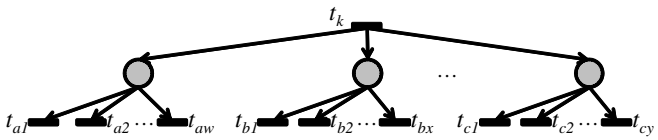

Fig. 21. A PN place *p* = [t_{a1} $t_{a2...}t_{al}$, t_{b1} $t_{b2...}$ t_{bh}]

Once we have made the first merging, all places [t_{k1}, t_{a1}…t_{aw}], [t_{k2}, t_{a1}…t_{aw}],…, [t_{kz}, t_{a1}…t_{aw}] whose input transitions are non-concurrent nor consecutive and whose output transitions are the same, can be merged into a single place as illustrated in Fig. 22.

Remark. Building the non-observable *PN* can be executed in polynomial time on the size of *Seq*.

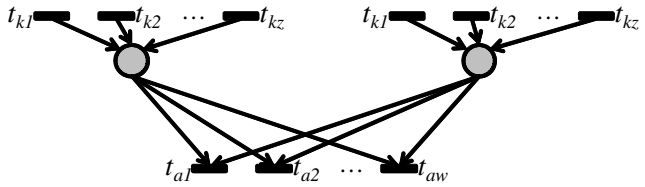

Fig. 22. Selection and concurrence between pre-transitions

E. Initial marking

Once the structure of the net is built, the initial marking can be computed by allowing the firing of *S*. All transitions are processed, from the last transition till the first one. The processing of a transition is as follows:

- If its output places are unmarked, the tokens in such places are retired,
- Tokens are added to its unmarked input places.

Example 2 (Cont.). By considering the couples of consecutive non-concurrent transitions in Seq'(which in this example is only (t_3, t_5) – see Definition 14), the places: [t_1, t_2]

$[t_2, t_3]$ $[t_3, t_1t_5]$ $[t_4, t_1t_5]$ $[t_5, t_6]$ $[t_6, t_7]$ and $[t_2t_7, t_4]$ are computed. The PN structure and the computed initial marking is shown in Fig. 23.

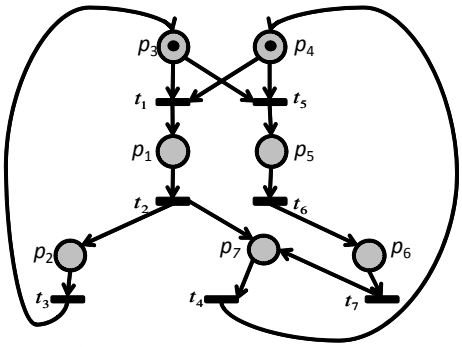

Fig. 23. (G^{nobs}, M_0) the non-observable IPN of Example 1

F. Token flow verification

As stated before, with the proposed mechanisms in last section, the sequence *w* may not have shown enough combinations which allow us to determine concurrence. If the sequence *w* were complete, all the concurrent and sequential behavior could be found and represented, according to Proposition 6. However, since we know that *w* could not be complete, in order to approximate the language of the identified IPN to *S* as much as we can, we have considered that if two transitions have not been declared as concurrent, they must be in a sequential relationship. But if the transitions are actually concurrent, the sequential consideration could lead us to links or places in the built model which restrict too much the behavior of the system and don't allow the firing of *S*. Now, we present some notions that will help us to verify if added places until now do not interfere in the correct reproduction of *S*.

Proposition 10. If the IPN model has been correctly build, every computed non-observable place p in N must fulfill the place input/output flow equation:

$$\sum_{t_i \in {}^\bullet p} Occ(t_i) = \sum_{t_i \in p^\bullet} Occ(t_i) \pm 1 \quad (16)$$

where $Occ(t_k)$ is the number of occurrences of t_k in *S*.

Proof. Equation follows straightforward from the IPN transition enabling and firing conditions and from the fact that (G^{nobs}, M_0) must be safe. ■

Proposition 11. If there exists a place p such that $|{}^\bullet p|=1$, then $\forall t_j \in p^\bullet$, $t_k \in SP(t_j)$, $SP(t_j)\neq\varnothing$ where t_k is the input transition of p. Also, if there exists a place p such that $|p^\bullet|=1$, then $\forall t_j \in {}^\bullet p$, $t_k \in SP(t_j)$, $SP(t_j)\neq\varnothing$ where t_k is the output transition of p.

Proof. If $|{}^\bullet p|=1$, for the re-enabling of t_j, p must be marked and the only way to do so is the firing of t_k, and thus $t_k \in SP(t_j)$. Similarly, if $|p^\bullet|=1$, for the re-enabling of t_j, p must be unmarked and the only way to do so is the firing of t_k, thus $t_k \in SP(t_j)$. ■

Correction rule. If the input/output flow equation or the conditions in *Proposition 11* are not satisfied by some place, the arcs relating transitions which are not in *CausalR* are removed. If there are not *CausalR* represented in such a place, it is deleted.

Example 2 (Cont.). In the model of Figure 23, we verify the input/output flow equation for each place. From *Example 2*, we can compute $Occ(t_1)=12$, $Occ(t_2)=11$, $Occ(t_3)=11$, $Occ(t_4)=20$, $Occ(t_5)=9$, $Occ(t_6)=9$, and $Occ(t_7)=9$. We check also the condition of Proposition 11.

p_1: $Occ(t_1) = Occ(t_2)$ (±1), $t_1 \in SP(t_2)$, $t_2 \in SP(t_1)$
p_2: $Occ(t_2) = Occ(t_3)$ (±1), $t_2 \in SP(t_3)$, $t_3 \in SP(t_2)$
p_3: $Occ(t_3) \neq Occ(t_1) + Occ(t_5)$ (±1), $t_3 \in SP(t_1)$, $t_3 \in SP(t_5)$
p_4: $Occ(t_4) = Occ(t_1) + Occ(t_5)$ (±1), $t_4 \in SP(t_1)$, $t_4 \in SP(t_5)$
p_5: $Occ(t_5) = Occ(t_6)$ (±1), $t_5 \in SP(t_6)$, $t_6 \in SP(t_5)$
p_6: $Occ(t_6) = Occ(t_7)$ (±1), $t_6 \in SP(t_7)$, $t_7 \in SP(t_6)$
p_7: $Occ(t_2) + Occ(t_7) = Occ(t_4)$ (±1), $t_4 \in SP(t_2)$, $t_4 \in SP(t_7)$

As can be observed, p_3 is a wrong place, since $Occ(t_3) \neq Occ(t_1) + Occ(t_5) \pm 1$. Since $(t_3, t_5) \in Seq'$; this means that the sequence is not complete, and thus the causal relationship we assumed between t_3 and t_5 is wrong. In order to fix this, we can delete the arc going from place p_3 to transition t_5. After this correction, all of the conditions from *Proposition 10* and *Proposition 11* are satisfied.

Finally, the identified IPN of the sorting system described in Example 1 is obtained by merging the observable model in Fig. 12 and the non-observable model from Fig. 23 after applying the places correction. We can also delete non-observable implicit places. Then the *IPN* shown in Fig. 24, which reproduces *w*, is the final result of the model merging.

In the supplementary file [26] several additional examples regarding the method for identifying a non observable model from a sequence S are included.

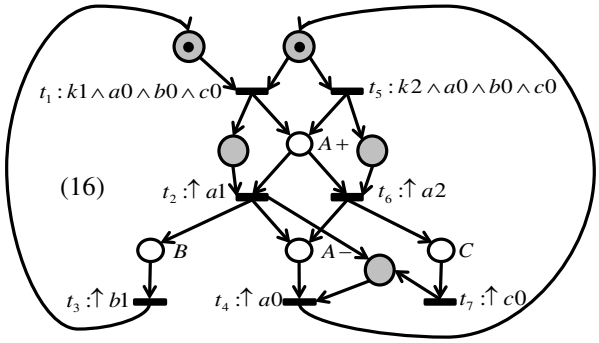

Fig. 24. Final IPN model for the process in *Example 1*

G. Features of the method

G.1 Reproducibility of S

Proposition 12. The *PN* model (G^{nobs}, M_0) built with the previous procedures summarized in Algorithm 3 reproduces the sequence *S*.

Proof. Regard that we have computed the following sets:

- *Seq* containing all the consecutive transition couples in *S*. If we represent into a net all couples in *Seq*, the net will be able to reproduce *S*,
- *CausalR* containing transition couples $(t_a, t_b) \in Seq$ that must be related by a place,

- *ConcR* containing transition couples $(t_a, t_b) \in Seq$, that must not be related by any place.

If the set $Seq' = (Seq - CausalR) - ConcR = \emptyset$, it means that all transition couples $(t_a,t_b) \in Seq$ are correctly represented in N and thus the sequence S is reproducible. If $Seq' \neq \emptyset$, it means that there are still transition couples that cannot be distinguished as concurrent or sequential. Thus, by merging several couples in *Seq*, all couples in *Seq*' are considered as sequential by creating places with several input and output transitions. If they are actually sequential, all the verification rules are satisfied. Otherwise, they are actually concurrent and they are corrected using the described procedure. Once they are corrected, it only remains places relating sequential transitions and thus the sequence S is reproducible.■

G.2 Performance

Given that all of the procedures of *Algorithm 3* are executed in polynomial time on $|S|$, the construction of (G^{nobs}, M_0) is efficiently performed.

Note also that the application of *Algorithm 3* to a sequence S yields always the same PN model, due to that all the constructive steps in the procedures are deterministically performed, i.e. there are not random selections on the input and intermediate data.

VI. Method Implementation and Application

Based on the presented algorithms, a software tool has been developed to automate the IPN model synthesis. The architecture of the tool is shown in Fig. 25.

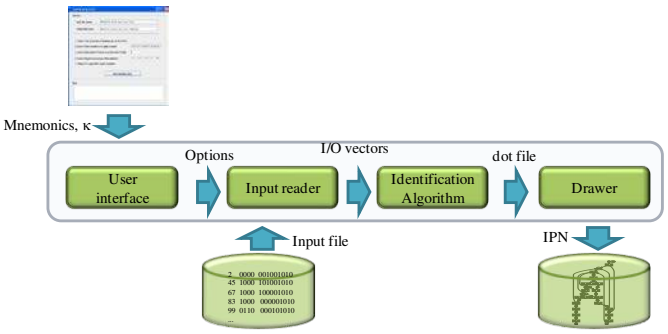

Fig. 25. Software architecture

The user interface allows capturing the input/output sequence and shows the obtained model graphically. Following input data is provided to the tool: the name of a text file containing the I/O sequence (with one line per I/O vector), the names of the input and output signals, and the desired name for the output file. Additionally it is specified the order in which inputs and outputs appear in the txt file (since depending on data collection procedure, order could change) and the index numbers of the signals to take into account if a mask is going to be applied (some inputs or outputs could be ignored like indicator lights or push-buttons).

Later, an input reader component processes the input file and transforms the input/output sequence into a vector sequence. These vectors are delivered to a component called Algorithm in which the identification procedure is implemented. The output of this component is an XML file that can be opened with the Platform Independent Petri net Editor (PIPE [25]), which is an editor for visualization and analysis of Petri nets.

The presented identification tool has been tested on several examples of diverse size and complexity. A small size case study regarding an actual manufacturing system is described in the supplementary files to this article [26] in which the use of such a software tool is illustrated.

VII. Conclusion

The proposed identification method discovers the actual input-output relation of PLC controlled discrete event systems. The technique allows building a concise 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 a previous method.

The technique copes with complex automated DES because it takes into account technological characteristics of actual controlled systems, and because it is based on efficient algorithms. This feature is not still addressed in current literature on the matter, in which several features considered in the current stated problem have not been dealt.

The algorithms issued from the present method have been implemented as a software tool and tested on experimental case studies which are very close to actual industrial discrete event processes. The performed tests reveal the efficiency of the methods when data including thousands of input-output vectors are processed in few seconds.

Due to this is a black-box approach, the obtained models represent the observed behavior; consequently, when the observation has been made for a long time, the identified IPN approximates closely the actual behavior. Afterwards this model can be completed using available knowledge on the process.

References

[1] E.M. Gold, "Language Identification in the Limit", *Information and Control*, vol. 10, pp. 447–474, 1967.

[2] D. Angluin, "Queries and Concept Learning", *Machine Learning*, vol. 2, no. 4, pp. 319–342, 1988.

[3] K. Hiraishi, "Construction of Safe Petri Nets by Presenting Firing Sequences", *Lectures Notes in Computer Sciences*, vol. 616, pp. 244–262, 1992.

[4] A.P. Estrada-Vargas, E. López-Mellado, J.-J. Lesage. "A Comparative Analysis of Recent Identification Approaches for Discrete-Event Systems", *Mathematical Problems in Engineering*, Vol. 2010, 21 pages, 2010.

[5] M. P. Cabasino, P. Darondeau, M. P. Fanti, C. Seatzu, "Model identification and synthesis of discrete-event systems", *Contemporary Issues in System Science and Engineering*, IEEE/Wiley Press Book Series, M. Zhou, H.-X. Lim M. Weijnen (Eds), 2013.

[6] M.P. Cabasino, A. Giua, and C. Seatzu, "Identification of Petri Nets from Knowledge of Their Language", *Discrete Event Dynamic Systems*, vol. 17, no. 4, pp. 447–474, 2007.

[7] M. P. Cabasino, A. Giua, C. Seatzu, “Linear Programming Techniques for the Identification of Place/Transition Nets”, in *Proc. IEEE Int. Conf. on Decision & Control*, pp. 514–520, Cancun, Mexico, Dec. 2008.
[8] M. Meda-Campaña, and E. López-Mellado, “Identification of Concurrent Discrete Event Systems Using Petri Nets”, in *Proc. 17th IMACS World Congress*, pp.1–7, Paris, France, Jul. 2005.
[9] M. Meda, A. Ramírez, E. López, “Asymptotic Identification for DES”, in *Proc. IEEE Conf. on Decision and Control*, Sydney, Australia, pp. 2266–2271, Dec. 2000.
[10] S. Klein, L. Litz, J.-J. Lesage, “Fault detection of Discrete Event Systems using an identification approach”, in *Proc. 16th IFAC World Congress*, pp. 1–6, Praha, Czech Republic, Jul. 2005.
[11] M. Roth, J.-J. Lesage, L. Litz, “Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems”, in *Proc. American Control Conference*, pp. 2601–2606, Baltimore, Maryland, USA, Jun. 2010.
[12] M. P. Cabasino, A. Giua, C. N. Hadjicostis, and C. Seatzu, “Fault Model Identification and Synthesis in Petri Nets,” *Discrete Event Dynamic Systems*, vol. 24, no. 3, pp. 275-307, 2014.
[13] S. Ould El Medhi, E. Leclercq, D. Lefebvre, **“**Petri nets design and identification for the diagnosis of discrete event systems”, in *Proc. 2006 IAR Annual Meeting*, Nancy, France, Nov. 2006.
[14] M. Dotoli, M. P. Fanti, and A. M. Mangini, “Real time identification of discrete event systems using Petri nets”, *Automatica*, vol. 44, no. 5, pp. 1209–1219, May. 2008.
[15] M. Dotoli, M. P. Fanti, A. M. Mangini, and W. Ukovich, “Identification of the unobservable behaviour of industrial automation systems by Petri nets”, *Control Engineering Practice*, vol. 19, no. 9, pp. 958–966, Sep. 2011.
[16] S. Ould El Mehdi, R. Bekrar, N. Messai, E. Leclercq, D. Lefebvre, B. Riera, “Design and Identification of Stochastic and Deterministic Stochastic Petri Nets”, *IEEE Trans. on Systems, Man and Cybernetics, Part A*, vol. 42, no. 4, pp. 931–946.
[17] J. E. Cook, Z. Du, C. Liu, A. L. Wolf, “Discovering models of behavior for concurrent workflows”, *Computers in Industry*, vol. 53, no.3, pp. 297–319, 2004.
[18] W. van der Aalst, T. Weijters, L. Maruster, “Workflow Mining: Discovering Process Models from Event Logs”, *IEEE Trans. on Knowledge and Data Engineering*, vol. 16, no. 9, Sep. 2004.
[19] A. P. Estrada-Vargas, J.-J. Lesage, E. López-Mellado, “A Stepwise Method for Identification of Controlled Discrete Manufacturing Systems”, *Int. Journal of Computer Integrated Manufacturing*, vol. 28, no. 2, pp. 187-199, 2014.
[20] A. P. Estrada-Vargas, E. López-Mellado, J.-J. Lesage, “Input-Output Identification of Controlled Discrete Manufacturing Systems”, *Int. Journal of Systems Science*, vol. 45, no. 3, pp. 456-471, 2014
[21] A. P. Estrada-Vargas, J.-J. Lesage, E. López-Mellado, “Identification of Industrial Automation Systems: Building Compact and Expressive Petri Net Models from Observable Behavior”, in *Proc. American Control Conference*, pp. 6095 – 6101, Montréal, Canada, Jun. 2012.
[22] A. P. Estrada-Vargas, E. López-Mellado, J.-J. Lesage, “Identification of Partially Observable Discrete Event Manufacturing Systems”. in *Proc. IEEE International Conference on Emmerging Technologies and Factory Automation*, pp1-7, Cagliari, Italy, Sept. 2013.
[23] R. David and H. Alla, “Petri Nets for Modeling of Dynamic Systems–A Survey”, *Automatica*, vol. 30, no. 2, pp. 175–202, 1994.
[24] M. Roth, J.-J. Lesage, and L. Litz, “Identification of Discrete Event Systems, implementation issues and model completeness”, in *Proc. 7th Int. Conf. on Informatics in Control Automation and Robotics*, pp. 73–80, Funchal, Portugal, Jun. 2010.
[25] PIPE 2: Platform Independent Petri net Editor 2, http://pipe2.sourceforge.net/
[26] Supplementary file. Examples and Case study: http://www.gdl.cinvestav.mx/Black-box-Ident-Suppl.zip

Biographies

Ana-Paula Estrada-Vargas received the B.Sc. degree in computer engineering from the Universidad de Guadalajara, Guadalajara, Mexico, in 2007, and the M.Sc. degree from CINVESTAV, Guadalajara, Mexico, in 2009. She obtained the Ph.D. degree from both CINVESTAV in Guadalajara and the ENS de Cachan, in Cachan, France in 2013. Currently, she belongs to the Oracle Semantic Technologies team in Mexico Development Center. Her research interests include identification of Discrete Event Systems, formal modelling and analysis using Petri nets, as well as Semantic Web technologies.

Ernesto López-Mellado received the B.Sc. degree in electrical engineering from the Instituto Tecnologico de Cd. Madero, México, in 1977, the M.Sc. degree from the CINVESTAV, México City, México, in 1979, and the Docteur-Ingénieur degree in automation from the University of Toulouse, France, in 1986. Currently, he is Professor of Computer Sciences at CINVESTAV Unidad Guadalajara, Guadalajara, Mexico. His research interests include discrete event systems, and distributed intelligent systems.

Jean-Jacques Lesage received the Ph.D. degree from the Ecole Centrale de Paris and the “Habilitation à diriger des recherches” from the University Nancy 1 in 1989 and 1994 respectively. He is currently Professor of Automatic Control at the Ecole Normale Supérieure de Cachan, France, where he was head of the Automated Production Research Laboratory during eight years. His research interests are in the field of formal methods and models for synthesis, analysis and diagnosis of Discrete Event Systems (DES), and applications to manufacturing systems, network automated systems, energy production, and more recently ambient assisted living.