Refining Discovered Petri Nets by Sequencing Repetitive Components

Ernesto López-Mellado, Tonatiuh Tapia-Flores

CINVESTAV Unidad Guadalajara.
Av. del Bosque 1145. Col. El Bajío 45015 Zapopan, México
{elopez, ttapia}@gdl.cinvestav.mx,

Abstract. The problem of refining a Petri net (PN) discovered from a single sequence S of events T generated by discrete event processes is addressed. The refinement aims to reduce a possible exceeding language in the discovered model. A technique that extends a t-invariant based discovery method is presented. Given a discovered PN and its set of minimal t-invariants Y, the technique analyses the execution of the t-components in S and determines a sequencing pattern S_Y that schedules the execution of t-components in the initial PN. Then, S_Y is used to discover a PN' that uses transitions in T and new places; PN' schedules representative transitions of each t-invariant in S_Y. The refined model is obtained by merging the representative transitions of both PN and PN' if the pattern is discovered. A first result coping with a subclass of safe PN in which each t-component has at least a transition not shared with any other component is reported.

Keywords: Discrete event process discovery; Petri nets refinement; Repetitive components.

1 Introduction

Automated modelling of discrete event processes from external behaviour is nowadays studied by numerous research groups along the two last decades. The main motivation is to discover the current behaviour of unknown or ill known systems, because the documentation is not updated or missing. The aim is to obtain models that express clearly causal and concurrency relationships between the events generated by the process. Petri nets have been mostly used to represent such models.

Variations on this problem have been named differently in the research community. Earlier methods were called language learning techniques, which aimed to build finite automata, or grammars of languages from samples of accepted words [1, 2]. In discrete manufacturing systems the problem is referred as process identification, in which the purpose is to build finite automata or Petri nets from sequences of input-output signals. In this field several approaches and methods have been proposed: the incremental approach [3, 4], and the integer programming based approach [5, 6].

Input-output identification of automated production process is addressed in [7]; an interpreted PN is obtained from a long single observation of input-output vectors. Overviews of these methods are presented in [8] and [9].

In the field of workflow management systems (WMS) the problem is named process mining: discovery. Most of the proposals obtain Petri nets from a set of event traces representing the system behaviour as process cases. A complete review of recent works can be found in [10].

The work presented herein concerns to identification of discrete event processes, in which the available data is a set of sequences of input-output vectors, which represent the exchanged signals between a controller and a plant from the controller point of view. The events, the number of places, and the number of transitions are not known a priori. The proposed method yields a Petri net model including input functions and outputs. The method is based on a two-phase strategy presented in a recent previous work [7], in which the observable components of the labelled PN are first obtained, and then the non-observable part is inferred.

This paper focuses on the second phase in which a PN is obtained from a sequence *S* of events (transitions) from a given alphabet T. More precisely, a method based on the computation of the t-invariants [11] is extended; the proposed technique refines the obtained PN by determining a sequencing pattern, given also as a PN', that schedules the execution of t-components in the initial PN.

The paper contains a) an overview of the t-invariant based discovery method and b) the repetitive component scheduling technique.

2 Context and background

2.1 Input-Output identification

The present paper follows the approach presented in [7, 11] for dealing with input-output identification. The method processes off-line the I/O-sequence *w* captured during the process operation and builds an interpreted PN (IPN) model that reproduces *w*.

Example 1. Consider a process handling 3 inputs (s, x, y) and 3 outputs (A, B, C), from which the following I/O sequence is captured:

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

The construction of an interpreted Petri net (IPN) model from *w* is performed in two stages. The first stage processes the sequence *w* and obtains the observable part of the model consisting of components using observable places and transitions labelled with output symbols and expressions of input symbols respectively (Fig. 1). This determines the set of transitions T. A transition sequence *S* that corresponds to the observed event sequence is also delivered. In the example $S_1=t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_5\ t_6\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_5\ t_6\ t_1$ is obtained. A simpler technique for this stage is proposed later in [12].

The second stage builds a PN model that is able to reproduce *S* with a reduced excessive behaviour; furthermore, the number of nodes is small since only the necessary places to represent the discovered structural constrains are added. The resulting PN corresponds to the process' internal behaviour represented by non observable places. In the example the obtained PN showed in figure 2 reproduces S_1 (thus *w*). Merging this model with the observable model and eliminating implicit places (p_{11}, p_{22}, p_{33}) yields the final IPN model shown in figure 3.

A technique that allows determining more complex behaviours, in particular implicit dependencies, is proposed in [11]; besides discovering the non observable PN the technique delivers the t-invariant supports. In this example, the supports are $<Y_1>=\{t_1, t_2, t_3, t_4\}$ and $<Y_2>=\{t_1, t_2, t_5, t_6\}$; the t-components $\|Y_i\|$ can be clearly distinguished in this example.

In the present paper we are extending the method in [11] for obtaining a more precise non observable IPN with respect to the observed sequence S.

In Example 1, notice that the execution of S exhibit always the occurrence of t-components in alternate way: $S_Y=\tau_1\tau_2\tau_1\tau_2\tau_1\tau_2...$, where τ_i is a sub-sequence of S including transitions in $<Y_i>$. However, in the model sub-sequences of repeated components ($\tau_1\tau_1\tau_1{}_{...}\tau_2\tau_2$) may occur, which is a kind of exceeding behaviour.

A possible refining of the PN N of figure 3 to reduce this behaviour can be done by adding a component that ensures the occurrence of the t-components. Being $\rho(\tau_1)=t_3$ and $\rho(\tau_1)=t_5$ the representative transitions of each component, it is easy to determine that a circuit N' including such transitions and new places can be added to the PN to ensure the order of occurrence, as shown in figure 4. Next section describes this meta-analysis technique.

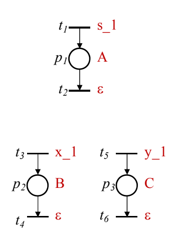

Fig. 1. IPN fragments

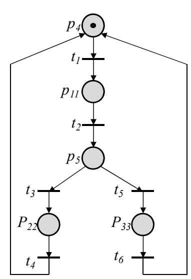

Fig. 2. Non-observable PN

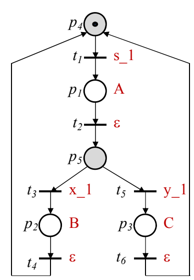

Fig. 3. Complete IPN

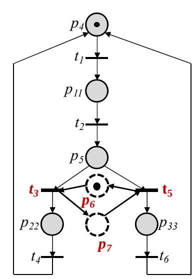

Fig. 4. Scheduled t-components

2.2 Petri nets definitions and notation

This section presents the basic concepts and notations of ordinary PN.

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

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

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

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

In a PN system, a transition t_j is *enabled* at marking M_k if $\forall p_i \in P$, $M_k(p_i) \geq I(p_i, t_j)$; an enabled transition t_j can be fired reaching a new marking M_{k+1}, which can be computed as $M_{k+1} = M_k + Cu_k$, where $u_k(i) = 0$, $i\neq j$, $u_k(j) = 1$; 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 PN system is *1-bounded* or *safe* iff for any $M_i \in R(G,M_0)$ and any $p \in P$, $M_i(p) \leq 1$. A PN system is *live* iff for every reachable marking $M_i \in R(G,M_0)$ and $\forall t \in T$ there is a reachable marking $M_k \in R(G, M_i)$ such that t is enabled in M_k.

Definition 4. A *t-invariant* Y_i *of a PN* is an integer solution to the equation $CY_i=0$ such that $Y_i\geq 0$ *and* $Y_i\neq 0$. The *support* of Y_i denoted as $<Y_i>$ is the set of transitions whose corresponding entries in Y_i are strictly positive. Y is *minimal* if its support is not included in the support of other t-invariant. A *t-component* $G(Y_i)$ is a subnet of *PN* induced by a $<Y_i>$: $G(Y_i)=(P_i, T_i, I_i, O_i)$, where $P_i = {}^{\bullet}<Y_i>\cup<Y_i>^{\bullet}$, $T_i =<Y_i>$, $I_i= P_i\times T_i\cap I$, and $O_i=P_i\times T_i\cap O$. $G(Y_i)$ is usually denoted as $\|Y_i\|$.

In a t-invariant Y_i, if we have initial marking (M_0) that enables a $t_i \in <Y_i>$, when t_i is fired, then M_0 can be reached again by firing only transitions in $<Y_i>$.

3 Sequencing repetitive components

3.1. Problem statement

Consider the t-invariant based discovery method, which treats a single sequence $S\in T^*$ and delivers a safe PN N and a set of minimal t-invariant supports $<Y>=\{<Y_i>\}$ such that PN executes all the t-invariants Y [11]. The refining technique has to find a PN N' using a set of transitions $T'\subseteq T$ and new places, such that the composed PN $N''= N\|N'$ by merging transitions in T' in both nets executes the t-components $\|Y_i\|$ in the order they appear in S. This statement has been briefly introduced through Example 1, where N and N'' are illustrated in figures 2 and 4 respectively.

3.2 Sequence of t-components

A sequence S_Y of t-components execution can be determined by parsing S from the first transition. Every component $\|Y_i\|$ can be identified when a given transition that distinguishes the component is found or, in the worst case, the whole set $<Y_i>$ is found during the tracking of S.

3.2.1 Representative transitions

Definition. The *distinguishable transitions* of a t-component $\iota(\|Y_i\|)\subseteq<Y_i>$ are the transitions that only belong to this support. When such transitions are fired, we are sure that a t-component is executed. $\iota(\|Y_i\|)=<Y_i>\cap(\otimes_j<Y_j>\forall j)$.

Notice that $\iota(\|Y_i\|)=\varnothing$ when every $t_j \in <Y_i>$ belongs also to other t-invariant supports. Eventually, all the $\iota(\|Y_i\|)=\varnothing$.

Definition. The *representative transition* ρ_i of $<Y_i>$ is a $t_j \in <Y_i>$ such that it determines precisely the t-component is being executed is S. If $\iota(\|Y_i\|) \neq \emptyset$ then ρ_i is the first transition in $\iota(\|Y_i\|)$ that occurs in S; else, ρ_i is the last transition in $<Y_i>$ found in the current subsequence tracked in S, then τ_i is determined.

In Example 1, $\iota(\|Y_1\|)=\{t_3, t_4\}$ and $\iota(\|Y_2\|)=\{t_5, t_6\}$; $\rho_1= t_3$ and $\rho_2=t_5$.

Property 1. In a firing sequence S all the transitions of a t-component are fired consecutively as a subsequence τ_i if there is no other nested t-components (τ_j, τ_k ...) sharing transitions with τ_i. In case of nested t-components, part of τ_i is fired, then one or several occurrences of τ_j may appear; afterwards some other transitions of τ_i are fired, then other nested τ_k may occur and so on; finally, the remainder transitions of τ_i are fired.

Proof. In a repetitive execution of a PN if a t-invariant is executed, all the transitions of the support must be fired. When a nested t-component is executed, then this component will be executed (one or several times), and then the remaining transitions of the first t-component will be fired. □

It is clear from *S* of Example 1 that all the transitions of $<Y_1>$ are fired first and then those of $<Y_2>$. Thus, it can be expressed as $S_Y=\tau_1\tau_2\tau_1\tau_2\tau_1\tau_2...$

Example 2. The PN in Figure 5, discovered from S= $t_1\ t_2\ t_4\ t_2\ t_3\ t_1\ t_2\ t_4\ t_2\ t_3\ t_1\ t_2\ t_4\ t_2\ t_{3....}$ has two supports: $<Y_1>=\{t_1, t_2, t_3\}$ and $<Y_2>=\{t_2, t_4\}$. $\iota(\|Y_1\|)=\{t_1, t_3\}$ and $\iota(\|Y_2\|)=\{t_4\}$; $\rho_1= t_1$ and t_3, and $\rho_2=t_4$. Then $S_Y=\tau_{11}\tau_2\tau_{12}\tau_{11}\tau_2\tau_{12}...$, which means that τ_1 is split into two sub-sequences τ_{11} and τ_{12}. Since $\|Y_2\|$ is a nested t-component, $\rho_{11}= t_1$ $\rho_{12}= t_3$

When some $\iota(\|Y_i\|)=\emptyset$, the transitions in $<Y_1>$ must be tracked during the parsing of *S* until the last transition or the support is found; then τ_k is determined.

3.2.2 Obtaining the Y-Sequence

The sequence of t-components S_Y, thus S_ρ can be obtained from *S* by considering the representative transitions and the cardinality of the t-invariant support. A procedure that parses S and delivers S_ρ is decidable and yields always the same S_ρ; the outline of a simple procedure for the case in which all $\iota(\|Y_i\|)\neq\emptyset$, and there are not nested t-invariants is given below.

Algorithm1. Building S_Y

Input: S, $<Y> = \{<Y_1>,<Y_2>,... <Y_{NY}>\}$
Output: S_ρ

1. Determine $\rho=\{\rho_1, \rho_2, ...\rho_{NY}\}$ from S and $<Y>$
2. $S_Y \leftarrow \varepsilon$; $k \leftarrow 1$
3. Repeat
   a. If $S(k)\in \rho$, where $S(k)= \rho_r$
   b. Then $S_\rho \leftarrow S_\rho \bullet \rho_r$; $S_Y \leftarrow S_Y \bullet \tau_r$
   c. Endif
4. $k \leftarrow k+1$
5. Until $k>|S|$
6. Return S_ρ

The procedure for determining ρ must deal with the case in which some $\iota(\|Y_i\|)=\varnothing$; in the worst all the transitions in $<Y_i>$ must be tracked in S to determine uniquely the t-component that is being executed. Furthermore, when there are nested t-components, a stack can be used to parse the components execution, in particular when there is nesting at several levels; in this case, regarding the t-component at higher level ρ_r, several representative transitions ρ_{r1}, ρ_{r2}, ... could be determined to go ahead the execution of the “interrupted” t-component. The structure of the discovered PN can be also exploited.

In Example 1, $S_Y=\tau_1\tau_2\tau_1\tau_2\tau_1\tau_2$... can be straightforward obtained by observing ρ_1 and ρ_2 in S=...t_3...t_5...t_3...t_5.... In Example 2 instead, one must consider two representative transitions $\rho_{11}=t_1$ $\rho_{12}=t_3$ for one component; S = t_1...t_4...t_3...t_1...t_4...t_3..., thus $S_Y=\tau_{11}\tau_2\tau_{12}\tau_{11}\tau_2\tau_{12}$... Finally, the sequence of representative transitions S_ρ is derived. In Examples 1 and 2 the sequences are $S_\rho=t_3\ t_5\ t_3\ t_5$... and $S_\rho=t_1\ t_4\ t_3\ t_1$... respectively.

3.3 Repetitive patterns

Now, the next stage is to determine from S_ρ the repetitive pattern that schedules the execution of the t-components. The problem is similar to that of process discovery but some other facts must be considered.

3.3.1 Simple patterns

It is easy to see in Example 1 that the $S_\rho=t_3\ t_5\ t_3\ t_5$ corresponds to a pattern that can be represented by a PN *N’* that is a cycle $p_6\ t_3\ p_7\ t_5\ p_6$ with p_6 initially marked. Similarly, in Example 2, the pattern *N’* derived from $S_\rho=t_1\ t_4\ t_3\ t_1$ is a cycle including the representative transitions; the merging of *N’* with *N* is shown in Figure 6; when the implicit place (marked) is removed, the simplified PN is shown in Figure 7. Notice that in previous examples, the refined PNs have one t-component.

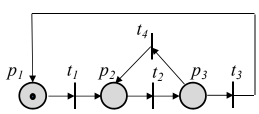

Figure 5. Nested cycles

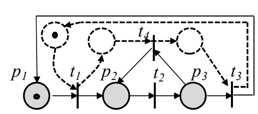

Figure 6. Sequenced t-components

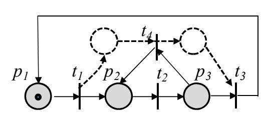

Figure 7. Simplified PN

3.3.2 Other patterns

Consider a sequence $S_1=t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_5\ t_6\ t_1\ t_2\ t_5\ t_6\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_5\ t_6\ t_1\ t_2\ t_5\ t_6\ t_1$... The invariant-based method discovers the same PN of Figure 2 and the same t-invariants. Thus, the representative transitions are the same; however, the sequence of representative transitions is not the same: $S_\rho=t_3\ t_5\ t_5\ t_3\ t_5\ t_3$...; in the pattern the t-component $\|Y_1\|$ is executed once, whilst $\|Y_2\|$ twice. This cyclic pattern, which establishes a production rate 1-2 for the jobs, is designed as a 2-bounded PN *N’* that is shown in Figure 8. For patterns that handle *k* jobs given by $S_Y=(\tau_1)^n(\tau_2)^m...(\tau_k)^r(\tau_1)^n$... with regular executions (fixed n, m, ..., r) of jobs, the PN can be designed similarly.

Now, consider the discovered PN, which describes the behaviour of a process involving three jobs. The t-invariant supports are: $<Y_1>=\{t_1, t_2, t_3\}$, $<Y_2>=\{t_1, t_4, t_5\}$, and $<Y_1>=\{t_1, t_6, t_7\}$; the representative transitions are: $\rho_1=t_2$, $\rho_2=t_4$, and $\rho_3=t_6$. Besides consider that $S_\rho=\boldsymbol{t_2}\ t_4\ \boldsymbol{t_2}\ t_6\ \boldsymbol{t_2}\ t_4\ \boldsymbol{t_2}\ t_6 \ldots$, that is, τ_1 is executed between the alternation of τ_2 and τ_3. The PN *N'* that describes this pattern is shown in Figure 10. Notice that *N'* has two invariants; thus, in a recursive manner, being t_4 and t_6 the representative transitions of these t-components, *N*'' can be obtained (Figure 11). The refined PN is obtained by merging *N* with *N*''; such a PN, shown in Figure 12 has only a t-invariant.

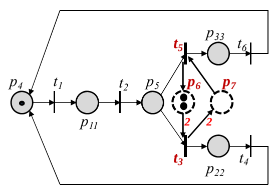

Fig. 8. Sequencing jobs rate

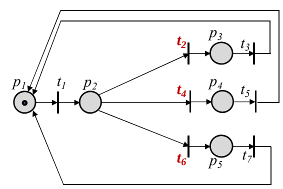

Fig.9. Modelling three jobs: *N*

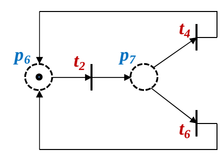

Fig. 10. Sequencing $||Y_i||$

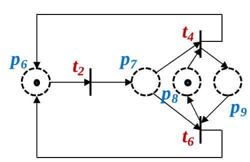

Fig. 11. Refining *N'*: *N''*

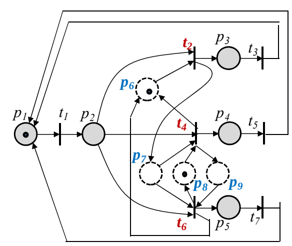

Fig. 12. The refined PN: *N*||*N''*

4 Remarks and challenges

We have sketched a refining technique for PN discovered by a t-invariant method. If the scheduling pattern can be determined, t-components relied by N' become a single t-component; thus, the refined PN has less exceeding behaviour than the first discovered PN. In this meta-analysis of a discovered model, the patterns considered in this short paper are found often in discrete manufacturing systems; however, the strategy to discover more complex patterns is still being studied.

Determining the patterns from S_Y is an interesting problem. In general, the dependency of occurrence among the τ_i must be found; this pose a new discovery problem in which, besides the order of execution of the τ_i in S_Y, the regularity of the repetitions of τ_i must be established.

References

[1] M. E. Gold, "Language identification in the limit", Information and Control, 10(5), pp. 447-474, 1967

[2] D. Angluin, "Queries and Concept Learning", Machine Learning, vol. 2, pp. 319-342, 1988

[3] M. Meda-Campana, A. Ramirez-Treviño, and E. Lopez-Mellado, "Asymptotic identification of discrete event systems", in Proc. of the 39th IEEE Conf. on Decision and Control, pp. 2266-2271, 2000

[4] M. Meda-Campana and E. Lopez-Mellado, “Identification of concurrent discrete event systems using Petri nets”, in Proc. of the 17th IMACS World Congress on Computational and Applied Mathematics, pp. 11-15, 2005

[5] M.P. Cabasino, A. Giua, C. Seatzu, "Identification of Petri nets from knowledge of their language," Discrete Event Dynamic Systems, Vol. 17, No. 4, pp. 447-474, 2007

[6] M. Dotoli, M. Pia Fanti, A. M. Mangini, and W. Ukovich, “Identification of the unobservable behaviour of industrial automation systems by Petri nets”, Control Engineering Practice, 19(9), pp. 958-966, 2011

[7] A. P. Estrada-Vargas, E. Lopez-Mellado, and J.-J. Lesage, “A Black-Box Identification Method for Automated Discrete-Event Systems,” IEEE Transactions on Automation Science and Engineering, Early Access 2015, pp. 1 - 16, DOI: 10.1109/TASE.2015.2445332

[8] A.P. Estrada-Vargas, E. Lopez-Mellado, and J.-J. Lesage, “A comparative analysis of recent identification approaches for discrete event systems”, Mathematical Problems in Engineering, vol. 2010, 2010

[9] M. P. Cabasino, P. Darondeau, M. P. Fanti, and C. Seatzu, “Model identification and synthesis of discrete-event systems”, Contemporary Issues in Systems Science and Engineering, IEEE/Wiley Press Book Series 2013

[10] W. Van der Aalst, Process Mining: Discovery, “Conformance and Enhancement of Business Processes”, Berlin: Springer-Verlag, 2011.

[11] T. Tapia-Flores, E. López-Mellado, A. P. Estrada-Vargas, and J. J. Lesage, “Discovering Petri Net Models of Discrete Event Processes by Computing T-invariants”. IEEE Transactions on Automation Science and Engineering. May, 2017 DOI: 10.1109/TASE.2017.2682060

[12] E. Rodríguez-Pérez, T. Tapia-Flores, E. López-Mellado, “Identification of Timed Discrete Event Processes. Building Input-Output Petri Net Models”. ATAED@Petri Nets/ACSD 2016: 153-167. Torun, Poland, June 2016