Identification of Timed Discrete Event Processes. Building Input-Output Petri Net Models

Edelma Rodríguez-Pérez, Tonatiuh Tapia-Flores, Ernesto López-Mellado

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

Abstract. *This paper deals with the identification of controlled discrete event processes from timed input-output vector sequences; the sampling date of every vector is provided. An efficient method allowing building a transition timed interpreted Petri net (TTIPN) model is presented. The method is based on a three-stage strategy: 1) the observable components of the TTIPN are first obtained, 2) the non-observable part is inferred, and 3) the time parameters associated to transitions are obtained. This paper focuses on the first and third phases of the method; a new method for building the observable model is proposed; additionally, a technique to compute the timing of transitions is presented. The derived algorithms are polynomial-time on the length of the input-output sequences.*

Keywords: I-O identification; Discrete event processes; Timed Petri nets.

1 Introduction

Nowadays, many automated processes in operation do not have enough information about how they work. This is because the updates have not been documented or in many cases the specification does not exist.

Earliest identification methods, named language learning techniques, appear in computer sciences. The aim of such methods was to build fine formal specifications (finite automata, grammars) of languages from samples of accepted words [1, 2]. In DES the problem is referred as process identification; in this field several approaches and methods have been proposed for building abstract machines representing the observed behaviour of automated processes. In [3, 4] an incremental approach allows synthesising safe interpreted Petri net (PN) models from a sequence of system's outputs. Later, in [5], an approach for building PN from a set of sequences of events is presented; it is based on the statement and solution of an integer linear programming problem. Numerous extensions to this method have been presented, for example [6, 7]. In [8] a method for deriving finite automata from sequences of inputs and outputs is presented; it is applied to fault detection of manufacturing processes. An extension to this method that allows building distributed models is presented in [9]. In [10] Input-output identification of automated manufacturing process is addressed; an interpreted PN is obtained from a set of sequences of input-output vectors sampled from the controller during the system cyclic operation. The method is extended for dealing

with a long single observation of input-output vectors [11]. Surveys presented in [12] and [13] provide a detailed presentation on DES identification.

In the field of workflow management systems (WMS) the problem is named workflow mining. The aim of the approach is similar but the problem statement is somehow different. A complete review of recent works can be found in [14].

Most of the proposed identification techniques process sequences of events and obtain models expressed as Petri nets of finite automata (FA). However, few proposals address the identification of timed systems. Relevant works on the matter are [15], [16], [17].

The work presented herein addresses identification of timed discrete event processes, in which the available data is only 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. Additionally, the instants when each vector is observed are considered. 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 and also timing information regarding the durations of operations. Timing is expressed through two parameters associated to transitions: a positive real value and pair of positive real values expressing an interval, which can be used according to the transition timed PN and time PN semantics respectively.

The method is based on a two-phase strategy presented in a recent previous work [18], in which the observable components of the TTIPN are first obtained, and then the non-observable part is inferred. This paper focuses on the first stages of the method and proposes a more efficient technique for determining the observable components and the timing parameters.

The paper is organised as follows. Section II gives an overview of the basic notions on Petri nets. Section III presents the method for building the qualitative observable model. Section IV describes the non observable model synthesis. Section V presents the strategy for computing the time parameters. Finally concluding remarks are given.

2 Petri Nets Background

This section presents the basic concepts and notation of Petri Nets (PN), Interpreted Petri nets (IPN), Timed and Time Petri Nets (TPN) 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 behaviour 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 $u_k(i) = 0$, $i \neq j$, $u_k(j) = 1$; 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 $R(G,M_0)$.

Now it is defined IPN, an extension to PN that allows associating input and output signals to PN models. This definition is adapted from [19].

Definition 3 : An interpreted Petri net (IPN) (Q, M_0) is a labelled net structure $Q = (G, \Sigma, \Phi, \lambda, \varphi)$ with an initial marking M_0 where:

- G is a PN structure,
- $\Sigma = \{\alpha_1, \alpha_2, ..., \alpha_r\}$ is the inputs alphabet,
- $\Phi = \{\beta_1, \beta_2, ..., \beta_q\}$ is the outputs alphabet,
- $\lambda : T \rightarrow Ev \times C$ is a labelling function of transitions, where
  $C=\{C_1, C_2, \ldots, C_{|T|}\}$ is the set of input conditions in which every C_i is a Boolean function on Σ; when a C_i is always true it is denoted as "=1", and
  $Ev=\{Ev_1, Ev_2, \ldots\}$ is the set of input events conditions; every Ev_i is a Boolean function of input events, built on Σ; events are denoted as α_i_0 and α_i_1 for representing that the input value changes from 1 to 0, or from 0 to 1 respectively. A condition Ev_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_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 β_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 4: 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.

Now the definitions of timed and time PN are recalled.

Definition 5: A timed transition PN is the tuple $N_\delta = (G, M_0, \delta)$, where G is an ordinary PN, M_0 is the initial marking, and $\delta: T \rightarrow \Re^{\geq 0}$ is function that assigns a nonnegative value to each $t_j \in T$; such a value represents the firing time of is the average firing of t_j once it is enabled [20].

Definition 6: A time PN is the tuple $N_\iota = (G, M_0, \iota)$, where G is an ordinary PN, M_0 is the initial marking, $\iota: T \rightarrow \Re^{\geq 0} \times \Re^{\geq 0}$ is the firing time interval function that assigns a firing interval $[l_j, u_j]$ to each transition $t_j \in T$. The interval represents a time window restriction; t_j must be fired after l_j or before u_j ($l_j \leq u_j$) time units computed from the instant in which t_j is enabled [21].

In both definitions, when $\delta(t_j)=0$ or $\iota(t_j)=[0, 0]$, t_j is called immediate. Timed and immediate transitions are usually represented by empty and filled boxes, respectively.

3 Input-output identification

3.1. The black-box approach

The process to identify consists of a pair controller-plant interacting in a closed loop (Fig. 1). The exchanged signals between them are sampled every time a signal changes and recorded for building a sequence of vectors representing input and outputs from the controller point of view.

The identification method developed in this paper follows the general approach proposed in [18]; it consists of two stages. The first one obtains the observable model and a transitions sequence *S*. The second stage processes *S* and builds the non-observable model. Then the observable and non-observable models are merged to create the final model, which describes closely the actual behaviour of the controller.

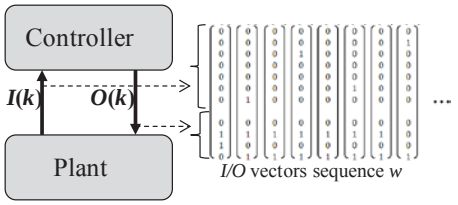

Figure 1. Closed loop controller-plant DES

3.2. Identification of the reactive behaviour

The method presented herein extends the previous one by proposing more efficient algorithms and addressing the temporal behaviour of the process and obtaining the identified model. Thus, besides the input-output vector sequence, the instants when each vector is sampled are considered. In this paper we will focus on the construction of the observable part and the computing of the time parameters associated to each transition.

3.2.1. Problem statement

The input data is a timed I/O vector sequence $w= w(1)w(2) ... w(|w|)$, where $w(k)=[I(k)|O(k)]^T$ such that $I(k)\in\{1,0\}^r$, $O(k)\in\{1,0\}^q$ and $w(k)\neq w(k-1)$, i.e. a new vector is recorded when an input or an output changes. Furthermore each $w(k)$ has associated a date $\tau(w(k))\in\Re^{\geq 0}$, which represents the instant when the k-th I/O vector is observed.

The aim of the method is to obtain an IPN *Q* and a timing function *Tim* that associates to each transition two parameters given by δ and ι, which express the firings duration and interval respectively from the timed and time PN definitions; $Tim: T\rightarrow\{(val_j, int_j)| val_j=\delta(t_j)$ and $int_j=\iota(t_j)\}\forall t\in T\}$. In this paper a technique that builds the observable components given by P^{obs} is presented.

It is assumed that the process formed by the controller and the plant behaves correctly, i.e. bounded, fault free, and deadlock free; besides, the sequence w is observed from the initial state.

3.2.2. Events

Definition 7. An *elementary event* is the difference between two consecutive I/O vectors $E(k\text{-}1)=w(k)-w(k\text{-}1)\neq 0$, $k>1$. Each event vector is composed by two parts $E(k)=[IE(k)|OE(k)]^T$, where $IE(k\text{-}1)=I(k)-I(k\text{-}1)$ and $OE(k\text{-}1)=O(k)-O(k\text{-}1)$ referred as input events and output events respectively. An $E(k)$ is called a *reactive event* iff $OE(k)\neq 0$.

Remark. The event vectors are unknown a priori but the number is bounded by $|3^{r+q}|$ since $E(k)\in\{-1,0,1\}^{r+q}$. The actual number of events is determined from the observed *w*. Another important issue on this definition is that the only events considered are those which are detected by changes in inputs or outputs (or both). Internal events that are not due to input changes or do not cause output changes are not considered.

3.2.3. Events sequences

The timed event sequence is then $E=e(1)e(2)e(3)\ldots e(k\text{-}1)$, where the instants when they are observed are given by $\tau(e(k))=\tau(w(k+1))$. It is assumed that $\tau(w(0))=0$.

In this work we are interested in determining the reactive behaviour of the controller; thus we will focus on events that provoke changes in the outputs, i.e. reactive events.

A new sequence RE is formed by the reactive events $re(h)$ from E by preserving the order in which they appear in E and their associated dates: $\tau(re(h))=\tau(e(k))$ such that $e(k)$ is a reactive event.

Given that between two reactive events in E there could be other non reactive events, RE is usually shorter than E.

Example 1. Consider a controlled process that handles 7 inputs (*m, a, b, c, d, e, f*) and 4 outputs (R_1, L_1, R_2, L_2); the I/O vectors have the format $[m\ a\ b\ c\ d\ e\ f\ |R_1\ L_1\ R_2\ L_2]^T$. Figure 2 shows the first vectors of the I/O sequence *w*, and the derived event sequences *E*, and *R*.

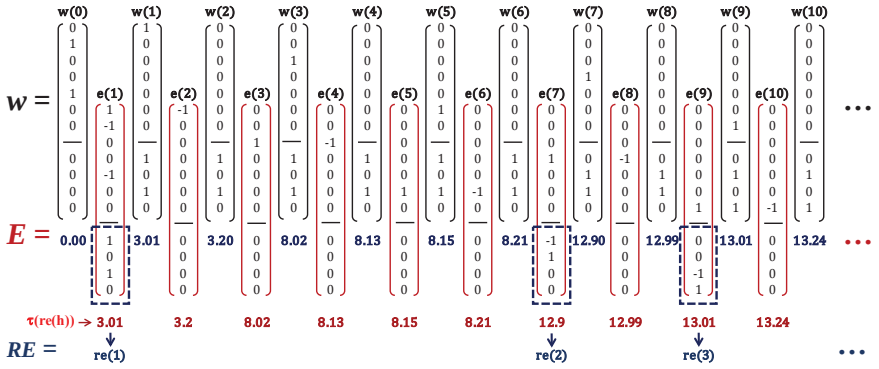

Figure 2. A fragment of I/O sequence *w* and its corresponding sequences of events *E* and *RE*.

3.3. IPN representation of reactive events

Every OE part of a reactive event re(h) represents a change in the output variables β_i of the system whose entries are different of zero; β_i_1 and β_i_0 denote the changes of the variable β_i from 0 to 1 and from1 to 0 respectively. The number of different OE in the observed reactive events is the minimum number of transitions in the IPN. All the different OEs are stored in a set OES.

For every OE(h), the symbolic expression of the event part is $SOE(k)=\prod SOE_i\ (k)\ \forall i$ s.t. $OE_i(k)\neq 0$, where $SOE_i(k)=\beta_i\_1$ if $OE_i(k)=1$ or $SOE_i(k)=\beta_i\_0$ if $OE_i(k)=-1$.

In *Example 1* the symbolic representation of the OE of re(1) and re(2) are $SOE(1) = R_1\_1 \wedge R_2\_1$ *and* $SOE(2) = R_1\_0 \wedge L_1\_1$ respectively.

There are as many observable places as output variables. The corresponding observable places to a re(h) are marked or unmarked when the value is 1 or -1 accordingly.

A substructure relating observable places p_i, such that $\phi(p_i)=\beta_i$, by a transition t_j that represents the event re(h) must be created through arcs (t_j, p_i) and (p_i, t_j) for the values 1 and -1 in the entry corresponding to p_i in OE(h).

In *Example 1* the reactive events re(1)=e(2) and re(2)= e(7) define the structures shown in figure 3.

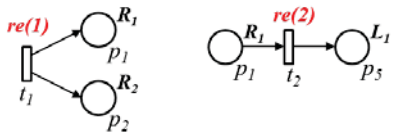

Figure 3. Examples of IPN sub-structures related to reactive events.

3.4. Firing functions

In a very long event sequence, reactive events that have identical OE≠0 may appear repeatedly; such events may have different IE. This means that several IE provoke the same changes in the system outputs. Thus, it is necessary to define a function that embeds all the IE for the same OE; such firing function is associated to the transition of the substructure i.e. that associated by λ of the IPN definition.

3.4.1. Input event functions

For every reactive event re(*h*) one must consider the corresponding IE(*k*) and also the previous IE whose OE=0 to build a function *f* in the form *Ev*×*C*; such functions are named *Event* and *Context* parts respectively of the input event function f_r.

- *Event part.* It represents the changes of the inputs that yield the outputs change. It is often determined from IE(*k*) or IE(*k-1*) if IE(*k*)=0. The symbolic expression of the event part is $SRE(k)= \prod SIE_i(k)\ \forall i$ s.t. $IE_i(k)\neq 0$, where $SIE_i(k)=\alpha_i\_1$ if $IE_i(k)=1$ or $SIE_i(k)=\alpha_i\_0$ if $IE_i(k)=-1$.
- *Context part*. It represents the values of the input variables of *w*(*k*). The symbolic expression of the condition part is $SCRE(k)=\prod SCI_i(k)\ \forall i$, where $SCI_i(k)=\alpha_i$ if $I_i(k)=1$or $SCI_i(k)=\overline{\alpha}_i$ if $I_i(k)=0$. Furthermore, if a literal α_i has changed twice its value during the previous E(k) in which OE(k)=0, α_i must not be included in *SCRE*(k).

For every computed f_r, the symbolic expression of the OE is associated through a function ρ, and the instants $\tau(\text{re}(h))$ are associated in a set of real values; when a f_r is computed again along the RE, then the corresponding date is added to the set. Based in the previous notions the procedure to build the input event functions is given below.

Algorithm 1. Input event functions

Input: Sequences *w* and *RE*
Output: *F* and the sequence *FS*

1.F←Ø; FS←Ø; r←1; OES←Ø
2.∀ *re*(*h*) in RE where h=1, ... , |RE|
Let *IE(k)* be the *k-th* input event corresponding to *re(h)*
2.1. If *IE(k)* $\neq \vec{0}$
then Compute *SRE* (*k*);
If SRE(k)∉OES
then OES←OES∪ *SRE* (*k*)
else Compute *SRE (k-1)*;
If SER(k-1)∉OES
then OES←OES∪ *SRE* (*k*-1)
2.2. Compute *SCRE(k)* from *w(k)*
2.3. If *IE(k)* $\neq \vec{0}$
then $f_r \leftarrow$ (*SRE (k), SCRE(k)*);
else $f_r \leftarrow$ (*SRE (k-1), SCRE(k)*)
2.4. $\rho(f_r)$←SOE(k)
2.5. If $f_r \notin F$
then *F*←*F* ∪ f_r; *FS*←*FS* • f_r; *r*= *r*+*1*; $\tau(f_r)$←τ(*re*(*h*))
else Let $f_s \in F$ the function already computed s.t. $f_s = f_r$; *FS*←*FS* • f_s ;
$\tau(f_S)$← $\tau(f_s)\cup\tau(re(h))$
3. Return *F* and *FS*

Property 1

- The previous algorithm produces a set *F* of functions that represent all the different reactive events in *RE* and their corresponding execution contexts. Consequently *FS* has a correspondence with the sequence of events *E*. Thus, *FS* reproduces the input-output sequence *w*.
- It is easy to see that the complexity of the procedure for building inputs events functions is *O*(|*RE*|).

In *Example 1*, regarding the reactive event re(1) which corresponds to e(1), SIE(1)=m_1∧a_0∧d_0 and SCI(2)=$\bar{b}\bar{c}\bar{e}\bar{f}$; then $f_1(R_1\_1 \wedge R_2\_1)$= (*m*_1 a_0∧d_0, $\bar{b}\bar{c}\bar{e}\bar{f}$). Additionally, $f_2(R_1\_0 \wedge L_1\_1)$= (c_1, $\overline{m}\bar{a}\bar{d}\bar{f}$); notice that *b* changed twice between re(1) and re(2). Also $f_3(R_1\_1 \wedge L_1\_0)$=(f_1, $\overline{m}\bar{a}\overline{b}\overline{c}\overline{d}\bar{e}$).

3.4.2. Merging input event functions

Once the reactive functions f_h are obtained, several input event functions could correspond to a same output event vector. In the sequence *RE* of figure 4 regarding *Ex-*

ample 1, the output event $R_1\_0 \wedge L_1\_1$ is found several times with different f_h, i.e. e(7), e(23), e(39), etc. These events are enhanced with rectangles in the sequence shown in figure 4.

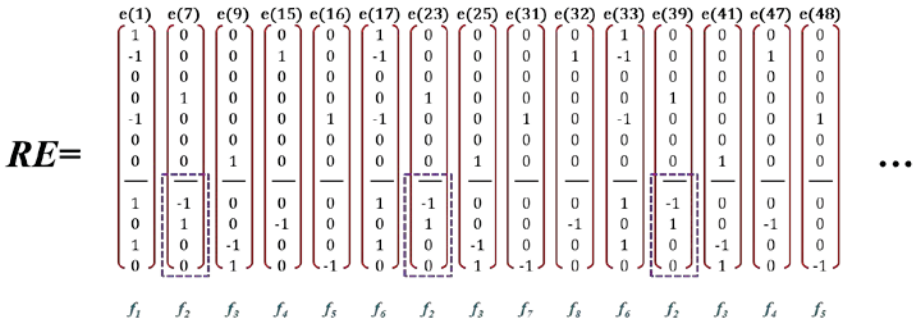

Figure 4. Fragment of a sequence of reactive events.

Such functions could be associated to the transition that yields the marking change in the obtained IPN structures as a *compound event*. The functions that provoke a given oe∈OES are associated through the function Ω: OES→2^F.

Afterwards, functions associated to every oe must be gathered according to the event part of the input function; then the condition part can be expressed as a disjunction of contexts of the corresponding functions, and then compacted by Boolean simplifications.

In the example, the output event oe= $R_1\_0 \wedge L_1\_1$ is provoked by the input events represented by the functions:

$f_2(R_1\_0 \wedge L_1\_1) = (c\_1, \overline{m}a\overline{d}\overline{f})$, and

$f_{10}(R_1\_0 \wedge L_1\_1) = (c\_1, \overline{m}a\overline{b}d\overline{e}\overline{f})$,

which can be embedded by the composed function $g_1 = f_{2,10} = c\_1 \bullet (\overline{m}a\overline{d}\overline{f} + \overline{m}a\overline{b}d\overline{e}\overline{f}) = (c\_1, \overline{m}a\overline{d}\overline{f})$, which is associated to t_2 in figure 3.

3.4.3. Transitions sequence

Once the observable components are obtained, the sequence of transitions S that reproduces the reactive events is computed. This is straightforward performed by tracking the firing functions sequence *FS* and applying a mapping $\lambda: T \rightarrow \{g_i\}$.

The technique above described is summarised in Algorithm 2 given below.

Algorithm 2. Compound functions

Input: *F, FS,* and *OES*

Output: *T*, λ, *S, Timed* sequence S_τ

1.$S \leftarrow \emptyset$; $S_\tau \leftarrow \emptyset$; $T \leftarrow \emptyset$;

2.// Finding functions with the same output event (oe)

$\forall$oe $\in$ OES: Ω(oe)$\leftarrow\varnothing$;

$\forall f_r \in$ F:

If $\rho(f_r)$=oe then

Ω(oe)$\leftarrow$ Ω(oe) $\cup$ f_r

3.// Gathering the functions in Ω(oe) that have the same SRE
∀oe ∈ OES
i← *1*; $\omega_i \leftarrow \emptyset$; ωs←∅
While Ω(oe) ≠∅
$f_f \leftarrow$ first(Ω(oe))
$\forall f_r \in \Omega(oe)$
If SRE(f_f) =SRE(f_r)
then $ge_i \leftarrow$SRE(f_r); $\omega_i \leftarrow \omega_i \cup f_r$;
Ω(oe) ← Ω(oe)\ f_r;
Endif
T ← T∪ t_i // determining the transitions set
$\forall f_s \in \omega_i$: $\lambda'(f_s) \leftarrow t_i$; ωs ←ωs ∪ ω_i; i=i+1
Endwhile
4.//Building compound functions
$\forall \omega_i \in$ ωs
$gc_i \leftarrow$BooleanRedContext(ω_i)
$\lambda(t_i) \leftarrow (ge_i, gc_i)$
5.// Transitions sequence formation
$\forall f_r \in$ FS: $S \leftarrow S \bullet \lambda'(f_r)$; $S_\tau \leftarrow S_\tau \bullet (\lambda'(f_r), \tau(f_r))$
6. Return T, λ, S, S_τ

Property 2

- Algorithm 2 returns the set of Transitions *T* and the labelling λ, which associates the input firing functions. Furthermore, the procedure builds both sequences *S* and S_t. As defined in the Step 5, *S* is formed by transitions t_i that correspond to f_r in *FS*; then by Property 1, *S* correspond to event sequence *E*, consequently to *w*.
- Step 2 of Algorithm 2 is performed in $O((|OES|)(|F|))$. Step 3 is executed in $O((|OES|)(|\Omega(oe)|)^2)$. Step 4 is completed in $O(|\omega s|)$. Step 5 is performed in $O(|FS|)$. The length of diverse structures is as follows: $|\Omega(oe)| < |\omega s| \leq |OES| < |F| < |FS|$. Then, the approximated complexity of the procedure O(|FS|). Given that |w|=κ|RE| and |FS|=|RE|, this complexity is linear on the length of *w*.

In *Example 1*, 12 different input functions for 6 output events were computed. They are summarised in Table 1. The observable sub-model is depicted in figure 4.

Table 1. Output events and their associated compound functions

Transition	Output event	Input functions	Compound function
t_1	$R_1\_1 \wedge R_2\_1$	f_1	g_1=(m_1∧a_0∧d_0, $\bar{b}\bar{c}\bar{e}\bar{f}$)
t_2	$R_1\_0 \wedge I_1\_1$	f_2, f_9	g_2=(c_1, $\overline{m}\overline{a}\bar{d}\bar{f}$)
t_3	$R_2\_0 \wedge L_2\_1$	f_3, f_8	g_4=(f_1, $\overline{m}\bar{a}\bar{c}\bar{d}$)
t_4	L_1_0	f_4, f_7	g_5=(a_1, $\bar{c}\bar{f}(m + \bar{b}\bar{e})$)
t_5	L_2_0	f_5, f_6, f_{10}	g_6=(d_1, $\bar{c}\bar{f}(\bar{b}\bar{e} + \bar{m})$)

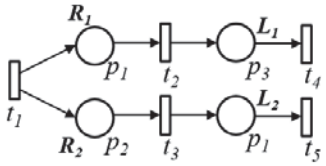

Figure 4. Observable component of a IPN model.

The transitions set is T={t_1, t_2, t_3, t_4, t_5} and obtained sequences S and S_τ are:

S= t_1 t_2 t_3 t_4 t_5 t_1 t_2 t_3 t_5 t_4 t_1 t_2 t_3 t_4 t_5 t_1 t_2 t_3 t_5 t_4 t_1 t_3 t_2 t_4 t_5 t_1 t_3 t_2 t_5 t_4 t_1 t_3 t_2 t_4 t_5 t_1 t_3 t_2 t_5 t_4 t_1 t_2 t_3 t_4 t_5 t_1 t_2 t_3 t_5 t_4 t_1 t_3 t_2 t_4 t_5 t_1 t_3 t_2 t_5 t_4 t_1 t_3 t_2 t_4 t_5 t_1 t_3 t_2 t_5 t_4 t_1.

S_τ = (t_1, 3.01) (t_2, 12.90) (t_3, 13.01) (t_4, 22.91) (t_5, 23.10) (t_1, 25.85) (t_2, 35.83) (t_3, 36.03) (t_5, 45.84) (t_4, 46.02) (t_1, 49.07) (t_2, 58.89) (t_3, 59.19) (t_4, 68.99) (t_5, 69.00) (t_1, 71.99) (t_2, 81.75) (t_3, 82.12) (t_5, 92.01) (t_4, 92.05) (t_1, 94.89) (t_3, 104.97) (t_2, 105.03) (t_4, 115.01) (t_5, 115.33) (t_1, 117.97) (t_3, 127.87) (t_2, 128.01) (t_5, 137.69) (t_4, 137.88) (t_1, 140.99) (t_3, 150.85) (t_2, 151.35) (t_4, 161.03) (t_5, 161.10) (t_1, 163.84) (t_3, 173.81) (t_2, 173.99) (t_5, 183.99) (t_4, 184.03) (t_1, 187.02) (t_2, 196.85) (t_3, 197.01) (t_4, 207.01) (t_5, 207.10) (t_1, 210.00) (t_2, 219.98) (t_3, 220.01) (t_5, 229.94) (t_4, 229.99) (t_1, 232.01) (t_2, 241.91) (t_3, 242.01) (t_4, 252.03) (t_5, 252.09) (t_1, 255.10) (t_2, 264.78) (t_3, 264.99) (t_5, 274.69) (t_4, 274.99) (t_1, 278.10) (t_3, 287.79) (t_2, 288.12) (t_4, 298.01) (t_5, 298.04) (t_1, 301.04) (t_3, 310.79) (t_2, 311.01) (t_4, 321.02) (t_5, 321.04) (t_1, 324.30) (t_3, 333.90) (t_2, 334.01) (t_5, 344.10) (t_4, 344.12) (t_1, 347.01) (t_3, 356.61) (t_2, 357.06) (t_4, 367.10) (t_5, 367.12) (t_1, 370.01) (t_3, 379.94) (t_2, 380.41) (t_5, 389.99) (t_4, 390.01) (t_1, 393.04).

Proposition 1. The observable model reproduces the input-output sequence w and behaves 1-bounded with respect to the execution of such a sequence.

Proof. The non-observable model is built from the output part of the reactive events, which mark and unmark the observable places following the reactive events sequence RE. In turn RE reproduces w by property 1; then, the model is able to execute w. Since by assumption the process behaves correctly, there is not two successive requests to activate an operation before deactivate it and then the places have zero or one tokens during the execution of w. □

4 Determining the non observable model

In order to complete the model, a technique that finds a PN model, which reproduces the transitions sequence S has to be applied. This model, called the non observable PN, is merged with the observable model to obtain the IPN that reproduces the input/output sequence w.

The method used for computing the non observable model is presented in [22], which builds from S, a safe PN and its initial marking using only the transitions in T. The method computes the causal and concurrent relations between the transitions in the sequence S. This is achieved by determining the t-invariants, which are used to determine substructures in the discovered model. In the last stage of the method the t-invariants are used for reducing the possible exceeding language by determining cau-

sality between events not observed consecutively. Further details on the method can be consulted in [22]. The application of this method to the sequence S of the example yields the model shown in figure 5.

Both the non observable and the observable models are combined by merging the transitions that have the same name. In the example the merging of models in figures 4 and 5 yields the IPN shown in figure 6. Afterwards, non observable implicit places are removed, and then the model yield is shown in figure 7.

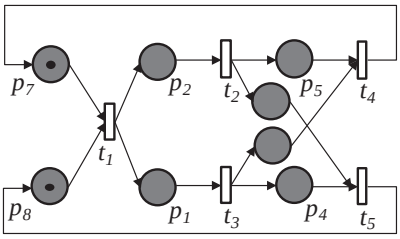

Figure 5. Non observable IPN

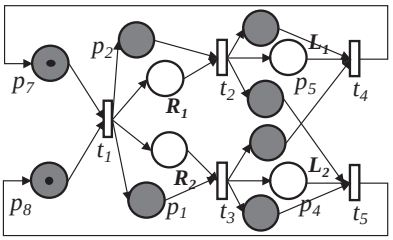

Figure 6.Merging observable and non-observable models

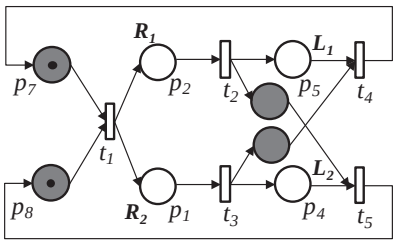

Figure 7. The reduced IPN model that reproduces *w*

Proposition 2. The partially observable IPN model (Q, M_0) obtained through the proposed method is able to reproduce the input-output sequence w. Such a model behaves 1-bounded when *w* is fired.

Proof. By property 2, the observable model reproduces *w*, hence the transitions sequence *S*. The method for computing the non observable models also guarantees the reproduction of *S*; therefore, the compound model reproduces the sequence w. In this model, by Proposition 1 and by the 1-boundedness of the non observable model, the final model is also 1-bounded. □

5 Computing the timing parameters

The last stage of the identification method is to obtain the timing parameters of the IPN models. For this purpose, the *Tim* function is determined from the timed sequence S_τ and the structure of the synthesized PN.

The strategy consists in parsing S_τ by comparing the transition t_k in S_τ(k) with one or several previous transitions in the sequence, and then determining the time elapse between $\tau(f_r)$ of S_τ(k) and the corresponding $\tau(f_r)$ in the upstream S_τ(k-i). This reasoning is based on the semantics of transition-timed Petri nets in which the time elapse associated to a transition t_k represents the maximum stay duration of the marking that

enables t_k. Thus, every time t_k appears in the sequence one must analyze the occurrence of ${}^\bullet({}^\bullet t_k)$ in the subsequence preceding $S_\tau(k)$, up to the previous appearance of t_k or $S_\tau(1)$. Figure 8 illustrates the general structure of the PN fragment to analyze.

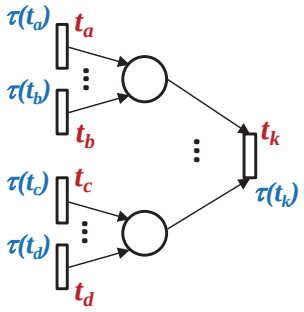

Figure 7. Possible transitions occurred preceding to t_k in S.

For every place in ${}^\bullet t_k$, one must detect one of the input transitions (since the PN is 1-bounded) in the subsequence preceding t_k in S_τ; among these transitions, let t_r be that whose date is the latest, then the maximum residence time of tokens in the marking that enables t_k is $\delta = \tau(t_k) - \tau(t_r)$, which is indeed the time elapse associated to t_k for this subsequence. The above strategy is summarized and structured in Algorithm 3 given below.

Algorithm 3. Determining timing function

Input: S_τ
Output: *function Tim*

1. $\forall t_j \in T: \gamma(t_j) \leftarrow \varnothing$
2. $\forall t_j \in T$:
   For $k=1$ *to* $|S_\tau|$ // *analysing the occurrences of* t_j *in* S_τ
   If t_j= GetTrans($S_\tau(k)$)
   then
   { $\tau(t_j) \leftarrow$ GetTime($S_\tau(k)$);
   r ← k - 1
   While (GetTrans($S_\tau(r)$) $\notin {}^\bullet({}^\bullet t_j)$) do
   r ← k – 1
   Endwhile
   $\tau(t_r) \leftarrow$ GetTime($S_\tau(r)$);
   $\gamma(t_k) \leftarrow \gamma(t_k) \cup (\tau(t_k) - \tau(t_r))$}
   Endfor
3. $\forall t_j \in T$
   $\delta(t_j) \leftarrow$ average($\gamma(t_j)$);
   $\iota(t_j) \leftarrow$ [min($\gamma(t_j)$), max($\gamma(t_j)$)]
   $Tim(t_j) \leftarrow (\delta(t_j), \iota(t_j))$
4. Return *Tim*

Property 3

- The algorithm obtains for every transition t_j, both the average of the duration of the making enabling t_j, and their minimum and maximum values. It is easy to see the S_τ is executed witin the instants given by the computed interval.
- The first and the third steps are performed in $O(|T|)$. The second step is more time consuming; inside the two *for* iterations (executed in $O(|T||S_\tau|)$), one must explore in S a subsequence preceding t_j, which contains ${}^{\bullet\bullet}t_j$. Since the maximum value of the $|{}^{\bullet\bullet}t_j|$ is small compared to $|S_\tau|$, the complexity is $O(|T||S_\tau|)$.

In *Example 1*, the timing parameters obtained from S_τ are given in Table 2.

Table 2. Timing parameters for the IPN model.

Transition	Tim(t_j)	$\gamma(t_j)$
t_1	(2.90, [2.02, 3.26])	{3.01, 2.75, 3.05, 2.99, 2.84, 2.64, 3.11, 2.74, 2.99, 2.02, 3.01, 3.11, 3.00, 3.26, 2.89, 2.89, 3.03}
t_2	(9.98, [9.68, 10.40])	{9.89, 9.98, 9.82, 9.76, 10.14, 10.04, 10.36, 10.15, 9.83, 9.98, 9.90, 9.68, 10.02, 9.97, 9.71, 10.05, 10.40}
t_3	(9.92, [9.60, 10.18])	{10.00, 10.18, 10.12, 10.13, 10.08, 9.90, 9.86, 9.97, 9.99, 10.01, 10.00, 9.89, 9.69, 9.75, 9.60, 9.60, 9.93}
t_4	(9.93, [9.60, 10.11])	{9.90, 9.99, 9.80, 9.93, 9.98, 9.87, 9.68, 10.04, 10.00, 9.98, 10.02, 10.00, 9.89, 10.01, 10.11, 10.04, 9.60}
t_5	(9.93, [9.58, 10.30])	{10.09, 9.81, 9.81, 9.89, 10.30, 9.68, 9.75, 10.00, 10.09, 9.93, 10.08, 9.70, 9.92, 10.03, 10.09, 10.06, 9.58}

6 Conclusions

In this paper the problem of identifying timed discrete event processes is addressed. A novel method that obtains the observable components of an IPN model and determines the timing parameters is proposed.

In this problem the only available input data is a sequence of input-output vectors and the instants when they are recorded. Using the method that discover the non observable model, the final obtained model is an IPN in which the process outputs are associated to some places, and the transitions are labelled with functions of inputs that express the reactive behaviour of the process. The time parameters are given as a pair (δ,ι) corresponding to the parameters of timed and time PN respectively.

The proposed method for computing the observable components is simpler than a previous work [18] since it focuses on reactive events and the processing is less complex; besides the technique for determining the transitions timing is simple. These features lead to polynomial time algorithms on the size of the input-output sequence, which are able to handle long sequences efficiently.

Current research studies the obtained time parameters; the time elapsed can be highly dispersed and then a refinement of the untimed model could be required.

7 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. P. Cabasino, A. Giua, and C. Seatzu, “Linear programming techniques for the identification of place/transition nets”, in *Proc. of the 47th IEEE Conf. on Decision and Control*, pp. 514-520, 2008

[7] 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

[8] S. Klein, L. Litz, J.-J. Lesage, “Fault detection of discrete event systems using an identification approach”, in *Proc. of the 16th IFAC world Congress*, 6 pages, 2005

[9] M. Roth, S. Schneider, J.-J. Lesage, and L. Litz, “Fault detection and isolation in manufacturing systems with an identified discrete event model”, *International Journal of Systems Science*, 43(10), pp. 1826-1841, 2012

[10] A. P. Estrada-Vargas, E. López-Mellado, J.-J. Lesage ”Input-Output Identification of Controlled Discrete Manufacturing Systems”. *International Journal of Systems Science*. Volume 45, Issue 3, 2014, pp. 456-471

[11] 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, 187, 2015

[12] 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

[13] 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

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

[15] M. E. Meda-Campaña and S. Medina-Vazquez, “Synthesis of timed Petri net models for on-line identification of Discrete Event Systems,” in *2011 9th IEEE*

International Conference on Control and Automation (ICCA), Santiago, 2011.

[16] D. M. Muñoz, A. Correcher, E. García and F. Morant, "Identification of Stochastic Timed Discrete Event Systems with st-IPN," *Mathematical Problems in Engineering,* vol. 2014, no. 835312, p. 21, 2014.

[17] F. Basile , P. Chiacchio and J. Coppola, "An approach for the identification of time Petri net systems," in *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, Cagliari, 2013.

[18] A. Estrada-Vargas, E. López-Mellado and J. J. Lesage, "A Black-Box Identification Method for Automated Discrete-Event Systems," *IEEE Trans. on Automation Science and Engineering,* vol. PP, no. 90, pp. 1-16, 2015.

[19] R. David and A. Hassane, Discrete, Continuous, and Hybrid Petri Nets, Berlin Heidelberg: Springer-Verlag, 2010.

[20] C. Ramchandani, *Analysis of asynchronous concurrent systems by Timed Petri Nets,* Massachusetts: Massachusetts Institute of Technology Cambridge, 1974.

[21] P. M. Merlin, *A study of the recoverability of computing systems,* California: University of California, Irvine, 1974.

[22] T. Tapia, E. López-Mellado, A. P. Estrada-Vargas and J. J. Lesage, "Petri net discovery of discrete event processes by computing t-invariants," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, Barcelona, 2014.