



**HAL**  
open science

## **Petri Net Discovery of Discrete Event Processes by Computing T-invariants**

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

### ► **To cite this version:**

Tonatiuh Tapia-Flores, Ernesto López-Mellado, Ana-Paula Estrada-Vargas, Jean-Jacques Lesage. Petri Net Discovery of Discrete Event Processes by Computing T-invariants. 19th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA'14), Sep 2014, Barcelone, Spain. paper 345, 8 p. <hal-01067790>

**HAL Id: hal-01067790**

**<https://hal.science/hal-01067790v1>**

Submitted on 24 Sep 2014

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization

# Petri Net Discovery of Discrete Event Processes by Computing T-invariants

Tonatiuh Tapia-Flores<sup>1</sup>, Ernesto López-Mellado<sup>1</sup>, Ana Paula Estrada-Vargas<sup>2</sup>, Jean-Jacques Lesage<sup>3</sup>

<sup>1</sup>CINVESTAV Unidad Guadalajara. Av. del Bosque 1145, Col. El Bajío. 45019 Zapopan, Mexico

<sup>2</sup>Oracle de México S. A. de C. V. Av. de los Empresarios 135, Col. Puerta de Hierro. 45110 Zapopan, Mexico

<sup>3</sup>LURPA, Ecole Normale Supérieure de Cachan. 61, Av. du Président Wilson. 94235 Cachan Cedex, France  
{ttapia, elopez}@gdl.cinvestav.mx, ana.estrada@oracle.com, Jean-Jacques.lesage@lurpa.ens-cachan.fr

**Abstract**— *In this paper the problem of discovering a Petri net (PN) from sampled events sequences representing the execution of industrial or business processes is addressed. A method for building a 1-bounded PN from a single event sequence S composed of numerous execution traces is presented; it is based on determining causal and concurrency relations between tasks. A technique for computing the t-invariants of the PN from S is proposed; the obtained invariants allow determining the structure of a PN that executes S. The algorithms derived from the method have been implemented and tested on numerous examples of diverse complexity.*

**Keywords**— *Model discovery; Petri Nets; t-invariants*

## I. INTRODUCTION

The synthesis of formal models from external observation of systems behaviour is an interesting and challenging approach for reverse engineering purposes in discrete event systems. Although the problem is relatively recent, it deserves the attention of several research groups in the fields of discrete event systems (DES) and workflow management systems (WMS).

Pioneer works on the matter, named language learning techniques, appear in computer sciences. The aim was to build fine representations (finite automata, grammars) of languages from samples of accepted words [1, 2].

In the field of DES, where the problem is named identification, several approaches have been proposed for building models representing the observed behaviour of automated processes. The incremental approach proposed in [3, 4] allows building safe interpreted Petri net (PN) models from a continuous stream of system's outputs. In [5], a method based on the statement and solution of an integer linear programming problem is proposed; it allows building PN from a set of sequences of events. Extensions of this method are proposed in [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 obtaining distributed system 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 collected from the controller during the system cyclic operation. The method is extended for dealing with a long single observation of input-output vectors [11]. More

complete reviews on DES identification can be found in [12] and [13].

In WMS the analogous problem is named workflow mining; the system observation is given as a set of sequences from a finite alphabet of tasks, representing execution logs of business processes. A first proposal is reported in [14], in which a finite automaton, called conformal graph is obtained. In [15] it is proposed a probabilistic approach to find the concurrent and direct relations between tasks. The input of the method is a sequence of events that represent the activities that have occurred in a workflow management system; the obtained model is graph similar to a PN. In [16] a mining method called algorithm alpha is presented. In this method a workflow tasks log composed by several traces is recorded sequentially and processed yielding a subclass of PN called workflow net. Numerous publications present extensions of this algorithm namely [17, 18, and 19]. In particular in this last work a strong hypothesis is held: the workflow engine provides, for every task in the log, the next tasks to be executed even if they are not consecutive; this means that all the causal relationships are a priori known. More related works can be found in [20].

In the present paper a new method for building a safe Petri net (PN) from a single sequence of tasks S, composed by numerous processes execution traces, is proposed. It follows the approach presented in [10] and proposes new results allowing addressing more complex behaviours such as implicit dependencies between tasks that are not observed consecutively. The method is based on determining, from S, causal and concurrency relations between tasks and the computing of the t-invariants of the PN to discover. The obtained invariants allow, first, determining the initial structure of a PN, and later, adjusting the model when the computed t-invariants do not coincide with those of the initial model. The paper is organized as follows. In Section 2, the basic notions on PN are recalled. Section 3 states the addressed problem. In Section 4, basic relations, computed from the tasks sequence, are introduced. Section 5 presents a technique for determining the t-invariants. In Section 6, the PN synthesis method is described. Section 7 outlines implementation and tests.

## II. BACKGROUND

This section presents the basic concepts and notations of ordinary PN used in this paper.

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

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

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

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

In a PN system, a transition  $t_j$  is *enabled* at marking  $M_k$  if  $\forall p_i \in P, M_k(p_i) \geq I(p_i, t_j)$ ; an enabled transition  $t_j$  can be fired reaching a new marking  $M_{k+1}$ , 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 *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  $\langle Y_i \rangle$  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  $\langle Y_i \rangle$ :  $G(Y_i) = (P_i, T_i, I_i, O_i)$ , where  $P_i = \bullet \langle Y_i \rangle \cup \langle Y_i \rangle \bullet$ ,  $T_i = \langle Y_i \rangle$ ,  $I_i = P_i \times T_i \cap I$ , and  $O_i = P_i \times T_i \cap O$ ; where  $\bullet \Theta$  ( $\Theta \bullet$ ) is the set formed by the input (output) nodes to (from) nodes in  $\Theta$ .

### III. PROBLEM STATEMENT AND PROPOSED APPROACH

#### A. Model discovery

First, we formulate the problem of model discovering in a general way; afterwards this technique is placed in the contexts of automated manufacturing processes and workflow management systems.

**Definition 4.** Given a finite alphabet of events or tasks  $T = \{t_1, t_2, \dots, t_n\}$  and a set of finite sequences  $S_i = t_1 t_2 \dots t_j \in T^*$ , we define the PN discovery problem as the synthesis of a 1-bounded PN structure using only transitions in  $T$  and the discovery of an initial marking, which allows firing every  $S_i$ . The number of places of the PN is not known *a priori*.

In the context of automated manufacturing systems,  $S_i$  represents the observation of relevant input-output events sampled from the controller during a long execution period of time, for example a complete production process performing repetitive jobs [10].

In the context of workflow mining, the observed behaviour is a log composed by traces  $\sigma_i \in T^*$ , which are sampled from the beginning to the end of execution traces (cases). In the current problem formulation a  $S$  can be formed by the

concatenation of tasks traces  $S = \sigma_1 \sigma_2 \dots \sigma_r$  regardless the order of  $\sigma_i$  in  $S$ . The knowledge of cases delimiting, i.e., the beginning and ending of traces, is no longer required.

**Assumptions.** In both contexts it is assumed that processes are well behaved, i.e. there are no faults, deadlocks, or overflows during the observation period. This is a realistic assumption since the processes whose models have to be discovered are supposed to be in operation, although the model is currently unknown or ill known. Thus we can consider that the event stream  $S \in T^*$  is generated by a deadlock-free 1-bounded PN to be discovered.

#### B. Overview of the method

The proposed method synthesises an ordinary PN structure and finds an initial marking from which  $S$  can be fired. It focuses on the computation of the causal and concurrent relations between the tasks in the sequence  $S$ . This is achieved by determining the t-invariants (that are supposed to exist since most systems exhibit repetitive behaviour), which also are used to find causal implicit relations between events that are not observed consecutively.

In a first stage several binary relations between transitions are determined from  $S$ ; based on these relations the t-invariants are computed. Afterwards, causal and concurrent relations are determined, and together with the discovered t-invariants, the structure of a PN model is built. Finally, again the t-invariants are used for reducing the possible exceeding language by determining causality between events not observed consecutively. The method is presented for dealing with a single sequence  $S$  since the extension to deal with several  $S_i$  is straightforward.

### IV. BASIC CONCEPTS AND RELATIONS

First we introduce several relations derived directly from  $S$ . Some of the following definitions have been taken and adapted from [10].

**Definition 5.** The relationship between transitions that are observed consecutively in  $S$  is expressed in the relation  $Seq \subseteq T \times T$  which is defined as  $Seq = \{(t_j, t_{j+1}) \mid 1 \leq j < |S|\}$ ;  $t_a Seq t_b$  will be frequently denoted as  $t_a < t_b$ . The relation between transitions that never occur consecutively in  $S$  is  $T \times T \setminus Seq$ ; pairs in this relation are denoted as  $t_a >< t_b$ .

**Definition 6.** Every couple of consecutive transitions  $(t_a, t_b) \in Seq$  can be classed into one of the following situations: i) *Causal relationship*. The occurrence of  $t_a$  enables  $t_b$ , denoted as  $[t_a, t_b]$ . In a PN structure, this implies that there must be at least one place from  $t_a$  to  $t_b$ . ii) *Concurrent relationship*. If both  $t_a$  and  $t_b$  are simultaneously enabled, and  $t_a$  occurs first, its firing does not disable  $t_b$ . In a 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  $t_a || t_b$ .

Now a relation that establishes a key property named repetitive dependency is introduced.

**Definition 7.** A transition  $t_j$  is *repetitively dependent* of  $t_k$ , denoted as  $t_j < t_k$  iff  $t_k$  is always observed between two apparitions of  $t_j$  in  $S$ . If  $t_j$  has been observed at least twice in  $S$ ,



$T_i$	$Seq$ ( $\bullet < t_j$ )	$CausalR$ ( $\bullet, t_j$ )	$ConcR$ ( $\bullet    t_j$ )	$Seq'$ ( $\bullet > t_j$ )	$T \times T \setminus Seq$ ( $\bullet > t_j$ )
$t_0$	$t_2, t_5, t_7$	$t_5$		$t_2, t_7$	$t_1, t_3, t_4, t_6$
$t_1$	$t_2, t_7$			$t_2, t_7$	$t_0, t_1, t_3, t_4, t_5, t_6$
$t_2$	$t_3, t_5, t_4$		$t_5$	$t_3, t_4$	$t_1, t_0, t_7, t_6$
$t_3$	$t_6$	$t_6$			$t_0, t_1, t_2, t_4, t_5, t_7$
$t_4$	$t_6$	$t_6$			$t_0, t_1, t_2, t_3, t_5, t_7$
$t_5$	$t_2, t_4, t_7$	$t_4$	$t_2, t_7$		$t_1, t_0, t_3, t_4, t_6$
$t_6$	$t_0, t_1$	$t_1, t_0$			$t_7, t_2, t_3, t_4, t_5$
$t_7$	$t_3, t_5, t_4$		$t_5$	$t_3, t_4$	$t_1, t_0, t_2, t_6$

Table1. Relations between tasks in *Example 2*.

The knowledge of transitions that belong only to one  $RdM_i$  will be useful for determining the invariants.

**Definition 11.** The set of transitions that belong to only one  $RdM_i$  is  $T_{RdM} = \bigcup_{i=1}^r (RdM_i \setminus \bigcup_{j=1, j \neq i}^r RdM_j)$  where  $r=|RdM|$ .

Now it is possible to enlarge these sets by merging  $RdM_i$  that share common transitions. This can be done when the  $RdM_i$  fulfils several conditions stated below.

**Proposition 3.** All the transitions in a  $RdM_{x,y} = RdM_x \cup RdM_y$  are included in the support of a t-invariant if there exist  $t_i \in RdM_x$  and  $t_j \in RdM_y$  such that i)  $(t_i, t_j) \in ConcR$ , and ii)  $Rd(t_i) \cap Rd(t_j) \neq \emptyset$ .

**Proof.** Let be  $t_k \in Rd(t_i) \cap Rd(t_j)$ . Since  $(t_i, t_j) \in ConcR$ , the subsequence  $t_i t_j \dots t_k \dots t_j t_i \dots t_k \dots t_i t_j \dots t_k$  is found in  $S$ ; that is, both transitions  $t_i$  and  $t_j$  appear between the occurrences of  $t_k$ . Therefore  $t_i$  and  $t_j$  belong to a largest repetitive dependence  $RdM_{x,y} = RdM_x \cup RdM_y$ , which is part of the support of a t-invariant.  $\square$

The next procedure obtains  $RdM^+$ , the set of extensions of  $RdM_i$  by performing the union operation between members of  $RdM$ .

#### Algorithm 1. Merging $RdMs$

Input:  $RdM = \{RdM_1, RdM_2, \dots, RdM_r\}$

Output:  $RdM^+$

1.  $RdM^+ \leftarrow RdM$
2.  $\forall (t_i, t_j) \in ConcR$   
    If  $Rd(t_i) \cap Rd(t_j) \neq \emptyset$  then  
         $RdM_{x,y} \leftarrow RdM_x \cup RdM_y$   
         $RdM^+ \leftarrow RdM^+ \cup \{RdM_{x,y}\}$

After applying this procedure to  $RdM$  obtained in *Example 2*, given that  $(t_5 || t_7)$ ,  $(t_2 || t_5)$ , and  $Rd(t_5) \cap Rd(t_7) \neq \emptyset$  and  $Rd(t_5) \cap Rd(t_2) \neq \emptyset$ , two new maximal sets are obtained:  $RdM_{1,4} = RdM_1 \cup RdM_4$ ,  $RdM_{1,5} = RdM_1 \cup RdM_5$ . Then  $RdM^+ = \{RdM_1, RdM_2, RdM_3, RdM_4, RdM_5, RdM_{1,4}, RdM_{1,5}\}$ .

#### B. Finding the repetitive behaviour

A t-invariant induces a sub-graph of the  $PN$  model, called repetitive component or t-component. In the case of a deadlock-free and 1-bounded  $PN$  the t-component is strongly connected ( $Sc$ ). We will analyse the extended  $RMS_i$  through a graph representation of  $CausalR$  and the transition pairs in  $Seq'$ .

**Definition 12.** The Graph of causality relations between tasks, named causality graph of a  $RdM_i$ , is a digraph denoted  $G_i$ , defined as follows.

$$G_i = (V_i, E_i); V_i = \{t_k \mid t_k \in RdM_i\}$$

$$E_i = \{(t_k, t_l) \in V_i \times V_i \mid (t_k, t_l) \in CausalR \cup Seq'\}$$

The set of causality graphs corresponding to  $RdM^+$  is denoted  $CG = \{G_1, G_2, \dots, G_q\}$ , where  $G_i$  is the causality graph of a  $RdM_i$ . A  $G_i$  is maximal iff there is not a  $G_k \in CG$  such that  $G_i \subset G_k$ .

The set  $CG$  corresponding to the  $RdM^+$  computed before for *Example 2* is shown in Figure 1.

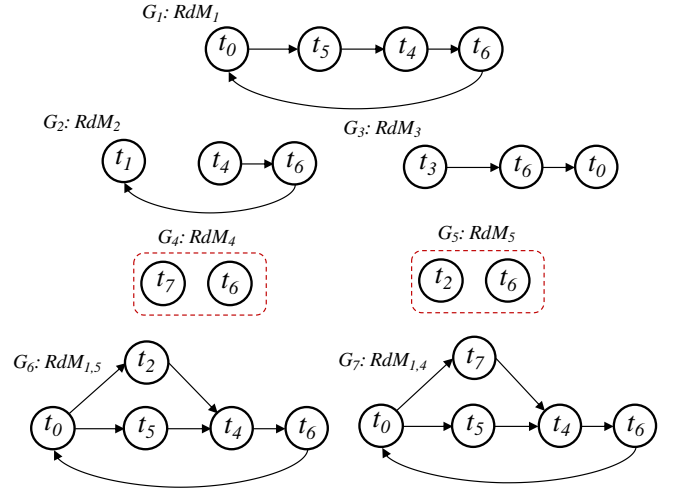


Figure 1. CG corresponding to  $RdM^+$  of *Example 2*.

**Theorem 1.** Let  $G_i$  be a causality graph in  $CG$ . If a maximal  $G_i$  is  $Sc$ , then its nodes are the support of some minimum t-invariant of the  $PN$  that reproduces  $S$ .

**Proof.** The vertices of  $G_i$  correspond to a  $RdM_i$  whose transitions are included in the support of a t-invariant  $Y_i$  (*Proposition 3*). Suppose that the transitions in  $V_i$  are not the support of a t-invariant; then there exists at least a  $t_k \notin Y_i$  such that  $t_k \in \langle Y_i \rangle$  that must fire to allow the repetitive firing of transitions in  $V_i$  together with  $t_k$ ; thus there are not cycles containing  $t_k$  in  $G_i$ , consequently it is not  $Sc$ .  $\square$

If the connectivity test is applied to the graphs in  $CG$ , it may occur that some  $G_i$  are not  $Sc$ . Then it is possible to obtain larger graphs by merging  $G_i$  with common vertices, through a merging operation of graphs defined below.

**Definition 13.** The merging operation ( $\bigcup_G$ ) of two causality graphs  $G_i \bigcup_G G_j$  produces a new graph  $G_{i,j}$ .

$$G_{i,j} = (V_{i,j}, E); V_{i,j} = V_i \cup V_j$$

$$E = \{(t_k, t_l) \in V_{i,j} \times V_{i,j} \mid (t_k, t_l) \in CausalR \cup Seq'\}$$

Figure 2 shows the merging of the graphs  $G_2$  and  $G_5$ . The idea is to merge iteratively graphs  $G_i, G_j \in GC$  such that  $V_i \cap V_j \neq \emptyset$ . In each iteration every  $G_{i,j}$  produced must not include other  $Sc$  graphs. Based on this strategy, a procedure for computing all the  $Sc$  graphs from  $CG$  is presented below.

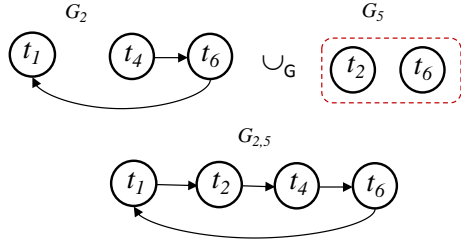


Figure 2.  $G_2 \cup_G G_5$ , where  $(t_1, t_2), (t_2, t_4) \in \text{CausalR} \cup \text{Seq}$

**Algorithm 2.** Getting the t-invariants from S

Input:  $CG = \{G_1, G_2 \dots G_q\}$

Output:  $\langle Y(S) \rangle$ : Supports of t-invariants

1.  $G_{Sc} \leftarrow$  all maximal Sc  $G_i \in CG$
2.  $G_{NSc} \leftarrow$  all non Sc  $G_i \in CG$
3.  $l_{NSc} \leftarrow |G_{NSc}|$
4. For 1 to  $l_{NSc}$ 
  - 4.1  $\forall G_i \in G_{NSc}$ 
 $\forall G_j \in G_{NSc} \mid G_i \neq G_j \text{ and } G_i \cap G_j \neq \emptyset$ 
    - a)  $G_{i,j} \leftarrow G_i \cup_G G_j$
    - b) If  $G_{i,j}$  is Sc and  $\forall G_k \in G_{Sc}, G_k \not\subseteq G_{i,j}$  then  $G_{Sc} \leftarrow G_{Sc} \cup G_{i,j}$  else  $NewNSc \leftarrow NewNSc \cup \{G_{i,j}\}$
  - 4.2  $G_{NSc} \leftarrow NewNSc; NewNSc \leftarrow \emptyset$
5. Return  $G_{Sc}$

The above algorithm ensures that the nodes of each  $G_i \in G_{Sc}$  correspond to the support of minimal t-invariants.

**Remark 2.** The computational complexity of finding the supports of T-invariants when no  $G_i \in CG$  is strongly connected (worst case) is  $O(|G_{NSc}|^3)$ . However, the worst case is unlikely since when  $G_{i,j}$  is built (step 4.1.a), the  $G_i$  that are Sc are discarded. Furthermore if  $G_{i,j}$  is Sc but if it contains other  $G_k$  that is Sc (step 4.1.b), then  $G_{i,j}$  is also discarded.

**Theorem 2.** Algorithm 2 obtains all the supports of the minimal t-invariants of a PN model that reproduces the task sequence S.

**Proof.** This procedure performs exhaustively the union of graphs which are not Sc and have common vertices. In every iteration, the formed Sc graphs are no longer considered in the union operations; this reduces progressively the number of non Sc graphs. Since it is avoided using the already obtained Sc graphs; this guarantees finding minimal Sc graphs and then the support of minimal invariants. When it is not possible to generate new Sc graphs the procedure stops. Every  $V_i$  of  $G_i$  in  $G_{Sc}$  is the support of a t-invariant.  $\square$

The set of obtained t-invariants is  $Y(S) = \{Y_i \mid Y_i \text{ is the vector corresponding to } V_i\}$

When Algorithm 2 is applied to CG of Example 2, the resulting supports of t-invariants are  $\langle Y_1 \rangle = \{t_0, t_4, t_5, t_6, t_7\}$ ,  $\langle Y_2 \rangle = \{t_0, t_4, t_5, t_6, t_2\}$ ,  $\langle Y_3 \rangle = \{t_1, t_4, t_6, t_2\}$ ,  $\langle Y_4 \rangle = \{t_1, t_4, t_6, t_7\}$ ,  $\langle Y_5 \rangle = \{t_0, t_3, t_6, t_2\}$ , and  $\langle Y_6 \rangle = \{t_0, t_3, t_6, t_7\}$ .

## VI. BUILDING THE PN MODEL

Causal relations  $[t_i, t_j]$  determine the existence of a place between transitions. Using this basic structure, named dependency, and the knowledge of t-invariants, a technique for building a PN model is now presented.

### A. Merging transitions of dependencies

All the transitions named  $t_i$  within several dependencies must be merged into a single one.

**Rule 1.** Two dependencies in the form  $[t_i, t_j]$  and  $[t_j, t_k]$  produce, straightforward, a sequential sub-structure including two places, which allows the firing of the sequence  $t_i t_j t_k$ , as illustrated in Fig 3.a).

**Rule 2.** When the first transitions in two dependencies are the same ( $[t_i, t_j]$  and  $[t_i, t_k]$ ), two possible substructures can be created (Fig. 3.b):

- a) The places of the dependencies are merged into a single one iff  $t_j$  and  $t_k$  belong to different t-invariants. This is denoted as  $[t_i, t_j+t_k]$ . This rule applies most of the time, but a special situation could appear when  $t_j // t_k$ ; in this case the dependency  $[t_i, t_j+t_k]$  is not created.
- b) The places of the dependencies are not merged iff  $t_j$  and  $t_k$  belong to a same t-invariant. This is denoted as  $[t_i, t_j // t_k]$ .

Similarly, for dependencies having a common second transition ( $[t_i, t_k]$  and  $[t_j, t_k]$ ), the substructure created will be either  $[t_i+t_j, t_k]$  or  $[t_i // t_j, t_k]$  (Fig. 3.b). In both cases the observations  $(t_i, t_j), (t_i, t_k), (t_j, t_k) \in \text{Seq}$ , deriving the dependencies, are preserved. This merging rule is illustrated in Figure 3. In general, a set of dependencies in the form  $\{[t_i, t_j], [t_i, t_k], \dots [t_i, t_r]\}$  may produce either  $[t_i, t_j+t_k+\dots+t_r]$  or  $[t_i, t_j // t_k // \dots // t_r]$  according to the relations between transitions  $t_j, t_k, \dots, t_r$ .

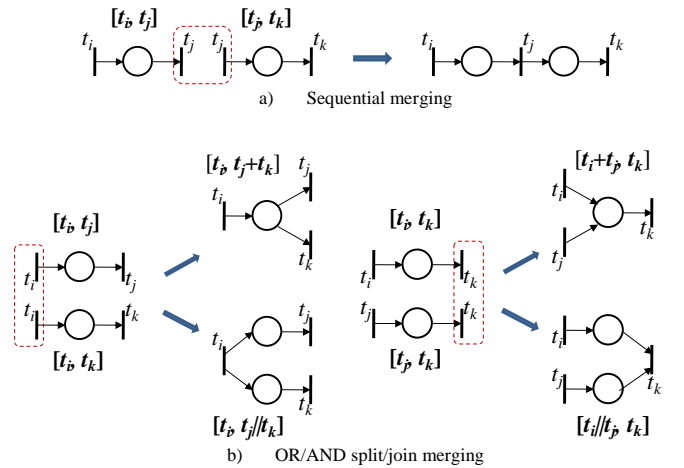


Figure 3. Rules for merging dependencies

Consequently, the merging can be applied to composed dependencies that coincide with one expression of transitions of type  $t_i+t_j$  or  $t_i // t_j$ ; for example  $[t_i+t_j, t_k]$  and  $[t_i+t_j, t_r]$  leads to  $[t_i+t_j, t_k+t_r]$  if both  $t_k$  and  $t_r$  do not belong to the same invariant.

The application of these merging rules to the dependencies derived from the pairs in  $CausalR \cup Seq'$ , leads to a PN model  $N_I$  including all the transitions.

In *Example 2*, the application of rules 1 and 2 to the obtained relations in  $CausalR \cup Seq'$  of *Table 1* yields the set of composed dependencies:  $[t_5, t_4]$ ,  $[t_0, t_2//t_5]$ ,  $[t_0, t_5//t_7]$ ,  $[t_0, t_2+t_7]$ ,  $[t_1, t_2+t_7]$ ,  $[t_2, t_3+t_4]$ ,  $[t_7, t_3+t_4]$ ,  $[t_4+t_3, t_6]$ ,  $[t_6, t_0+t_1]$ ,  $[t_0+t_1, t_2]$ ,  $[t_0+t_1, t_7]$ ,  $[t_2//t_5, t_4]$ ,  $[t_7//t_5, t_4]$ ,  $[t_7+t_2, t_3]$ ,  $[t_7+t_2, t_4]$ . Afterwards the obtained dependencies are  $p_0:[t_6, t_1+t_0]$ ,  $p_1:[t_0+t_1, t_2+t_7]$ ,  $p_1, p_2:[t_0, (t_2+t_7)//t_5]$ ,  $p_3:[t_5, t_4]$ ,  $p_4:[t_2+t_7, t_4+t_3]$ ,  $p_5:[t_4+t_3, t_6]$ . The sequential merging of substructures of the dependencies yields the PN model  $N_I$  shown in *Figure 4*.

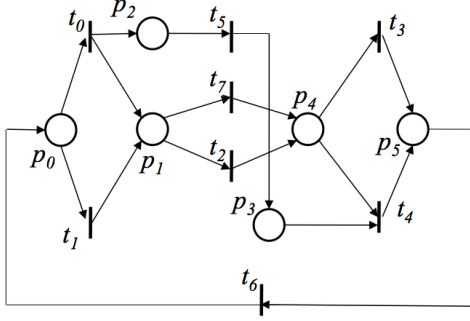


Figure 4.  $N_I$  built from  $S$  of *example 2*.

### B. Model adjustment

Although  $S$  may be fired in  $N_I$  most of the times, the obtained model could not fire  $S$ , or could fire  $S$  but also exceeding sequences. The PN in *Figure 4* does not reproduce  $S$  of *Example 2* in particular the subsequences  $t_1 t_2 t_4$  and  $t_1 t_7 t_4$  cannot be fired in  $N_I$ . This is because the computed t-invariants  $Y(S)$  differ from those of  $N_I$  ( $J(N_I)$ ). If  $Y(S)$  coincide with  $J(N_I)$ , then  $N_I$  is the correct model; otherwise it must be adjusted.

The mismatching between  $Y(S)$  and  $J(N_I)$  is due to the fact that the computed model does not include PN elements (places and arcs) which assure implicit behaviours not exhibited in  $S$ , named implicit dependencies.

**Definition 14.** In a 1-bounded PN,  $[t_i, t_j]$  is called an *implicit dependency*, if although there is a place between the transitions, the firing of  $t_i$  does not produce a marking that enables  $t_j$ . It is necessary the firing of at least one transition before  $t_j$ .

In a PN model, implicit dependencies represent the record of the occurrence of a  $t_i$ , which is used as condition to enable a future event  $t_j$ . In general, an implicit dependency represents a constraint in the flow of tokens in the net by assuring that  $t_j$  is fired only when  $t_i$  is fired before; otherwise the absence of such a dependency will allow the firing of exceeding sequences in the remainder model.

When  $Y(S) \neq J(N_I)$ ,  $N_I$  must therefore be adjusted by finding the pertinent implicit dependencies that extend  $N_I$  into  $N_2$ , whose t-invariants agree with  $Y(S)$ . In order to amend  $N_I$ , two cases of mismatching are considered: 1)  $Y(S) \subset J(N_I)$ , or 2)  $Y(S) \neq J(N_I)$  and  $Y(S) \not\subset J(N_I)$ , i.e.  $\exists Y_i \in Y(S)$  such that  $Y_i \notin J(N_I)$ . The handling of each case is described below.

### Case 1

In this case  $N_I$  has more invariants than those computed from  $S$ ; thus it represents an exceeding behaviour. A new place between two transitions  $t_i$  and  $t_j$  has to be added to  $N_I$  in order to constrain the differed firing of  $t_j$  after the firing of  $t_i$ .

**Proposition 4.** A dependency  $[t_i, t_j]$  must be added to  $N_I$  if the following condition holds:  $(t_i < t_j) \wedge (t_i, t_j \in \langle Y_k \rangle) \wedge (t_i, t_j \in T_{RDM})$ .

**Proof.** If  $[t_i, t_j]$  must not be added, it is because i)  $t_i$  and  $t_j$  have been observed consecutively ( $t_i < t_j$ ), or ii) each transition belongs to a different t-component, or iii) at least one of  $t_i, t_j$  does not belong to a  $T_{RDM}$ .  $\square$

### Case 2

Let  $J(N_I) = \{J_1, J_2, \dots, J_r\}$  be the set of t-invariants of  $N_I$ , such that  $CJ_j = 0$ , where  $C$  is the incidence matrix of  $N_I$ . Consider a  $Y_r \notin J(N_I)$ . Let  $p_k$  be the place corresponding to the row in which  $CY_r \neq 0$  (i.e.  $C(p_k)Y_r \neq 0$ ). In order to obtain the dependency  $[t_i, t_j]$ , other transition in  $N_I$  must be linked through  $p_k$  to one of the transitions in  $\bullet p_k$  or  $p_k \bullet$  according to the following rule.

**Proposition 5.** A dependency  $[t_i, t_j]$  must be added to  $N_I$  if  $t_i > t_j$ , and  $t_i, t_j \in \langle Y_r \rangle$ , and if one of the following conditions holds: i)  $t_i \in \bullet p_k$  and  $t_j \in T_{RDM}$ , when  $|\bullet p_k| < |p_k \bullet|$ , or ii)  $t_j \in p_k \bullet$  and  $t_i \in T_{RDM}$ , when  $|p_k \bullet| > |\bullet p_k|$ . This dependency ensures that  $C(p_k)Y_r = 0$ .

**Proof.** The two first conditions are the same than those of *Case 1*. We will analyse the conditions regarding  $\bullet p_k$  and  $p_k \bullet$ . In both situations  $|\bullet p_k|$  and  $|p_k \bullet|$  are unbalanced and one of  $t_i$  or  $t_j$  has to be related to one of  $\bullet p_k$  and  $p_k \bullet$  accordingly, to enforce  $J_r$  as t-invariant of  $N_I$ . Furthermore  $|\bullet p_k| = |p_k \bullet|$  yielding  $C(p_k)Y_r = 0$ .  $\square$

When all the corrections to  $N_I$  are done, it is possible that  $Y(S) \subset J(N_I)$ , then the rule of *Case 1* is applied and the new model  $N_2$  fulfils  $Y(S) = J(N_2)$ . *Algorithm 3* summarises the procedure to obtain the implicit dependencies.

### Algorithm 3. Finding implicit dependencies

Input:  $N_I, J(N_I), Y(S)$

Output:  $N_2$

1. If  $Y(S) \subset J(N_I)$ 
  - a)  $\forall (t_i, t_j) \mid t_i > t_j \wedge t_i, t_j \in T_{RDM} \wedge t_i, t_j \in y_i$   
add a place between  $(t_i, t_j)$
2. If  $\exists y_i \in Y(S) \mid y_i \notin J(N_I)$ 
  - a) Find a  $p_k / C(p_k)y_i \neq 0$
  - b) Add  $[t_i, t_j]$  through  $p_k$  relations that fulfil
$$t_i > t_j \wedge t_i, t_j \in y_i \wedge (t_i \in \bullet p_k, t_j \in T_{RDM})$$
or
$$t_i > t_j \wedge t_i, t_j \in y_i \wedge (t_j \in p_k \bullet, t_i \in T_{RDM})$$

**Remark 3.** The complexity of computing the implicit dependencies is  $O(|P| \times |T|)$ ; it is related to the matrix-vector product operation  $C(p_k)y_i$ .

Let us analyze  $N_I$  in *Figure 4*, obtained from  $S$  in *Example 2*. First it is computed  $J(N_I) = \{\langle J_1 \rangle, \langle J_2 \rangle, \langle J_3 \rangle, \langle J_4 \rangle\}$ ;  $\langle J_1 \rangle =$

$\{t_0, t_4, t_5, t_6, t_7\}$ ,  $\langle J_2 \rangle = \{t_0, t_4, t_5, t_6, t_2\}$ ,  $\langle J_3 \rangle = \{t_1, t_2, t_3, t_6\}$ ,  $\langle J_4 \rangle = \{t_1, t_7, t_3, t_6\}$ . There is a mismatch between both sets and since  $Y(S) \not\subseteq J(N_1)$ , the problem is handled as in *Case 2*. It can be noticed that  $Y_3, Y_4, Y_5, Y_6 \notin J(N_1)$ . In the analysis of  $Y_3$ ,  $p_k = p_3$  because it fulfils the condition  $C_{N_1}(p_3)Y_i \neq 0$ , as show in the next equation.

$$\begin{array}{c} \begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & -1 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 & 0 \end{bmatrix} \\ C_{N_1} \end{array} \cdot \begin{array}{c} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \\ Y_3 \end{array} = \begin{array}{c} \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \end{array}$$

The transition in  $t_4 \in p_k^*$  is chosen to find the implicit dependency  $[t_i, t_4]$ . The transition that fulfils the conditions  $t_i > t_4$ ,  $t_i, t_4 \in \langle Y_3 \rangle$ ,  $t_i \in T_{RDM}$ , is  $t_1$ ; therefore the implicit dependency  $[t_1, t_4]$  is added to  $N_1$  by the corresponding arc  $(t_1, p_3)$ . Similarly  $Y_4, Y_5, Y_6$  are treated and the implicit dependency  $[t_0, t_3]$  in  $p_2$  is found. Finally the resulting PN model  $N_2$ , which exactly reproduces  $S$  is shown in Figure 5.

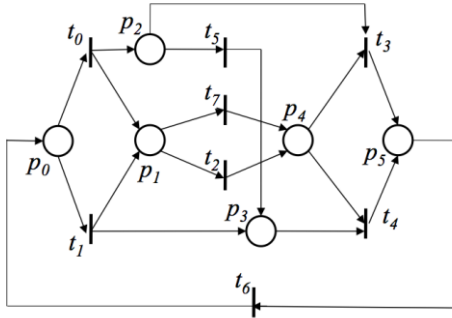


Figure 5. Resulting PN  $N_2$  after model adjustment.

**Theorem 3.** Given a sequence of transitions  $S \in T^*$ , a 1-bounded PN model  $N_2$  that reproduces  $S$  can be obtained by applying the rules 1 and 2, and performing the adjustments of *Algorithm 3*.

**Proof.** Causality between transitions, established by the pairs in  $CausalR \cup Seq'$  represents the precedence relationship between consecutive transitions in  $S$  that are not in  $ConcR$ . The substructure associated to a dependency  $[t_i, t_j]$  guarantees the consecutive firing of these transitions; thus by applying *Rule 1* the flow expressed in  $CausalR \cup Seq'$  is fulfilled by  $N_1$ . Furthermore, *Rule 2* determines, by the knowledge of the t-invariants, whether the flow is split or joint in choice or parallel structures. Dependencies involving transitions included in  $Sc$  causality graphs assure the construction of repetitive components in  $N_1$ . Furthermore, adjustments to  $N_1$  provided by *Propositions 4* and *5* allow fitting the invariants computed from the observed behaviour with those of the discovered model.  $\square$

### C. Initial marking

The Initial marking must enable  $S$ ; thus the procedure for determining  $M_0$  is simple; it suffices a) to place tokens in the

input places of the first transition in  $S$ , and b) executing the remainder  $t_j$  in  $S$  and eventually adding tokens in some places of  $\bullet t_j$  when the reached marking is not enough for firing  $t_j$ . In the case of example 2, the only place initially marked in the PN Fig. 5 is  $p_5$ .

### D. Processing several event sequences

This synthesis method may process  $r$  event traces  $S_i$  corresponding to the observed behaviour of the same discrete event process. The only constraint is that all the sequences must be sampled from the starting of the process. All the observed precedence relationships in  $Seq_i$  of every  $S_i$  are gathered into the  $Seq$  relation at the beginning of the discovery procedure. The initial marking is determined for enabling every  $S_i$ .

## VII. IMPLEMENTATION AND TESTS

Algorithms derived from the proposed method have been implemented as a software tool and tested on numerous examples of diverse complexity. The tests were performed using the following scheme: first, a PN model is designed, and with the help of the PN editor/simulator PIPE [21], a long sequence  $S$  is produced. Then the tool processes  $S$  and the obtained model, coded in XML, is displayed using PIPE again.

Below we provide an example regarding a less simple PN model that can be discovered using the proposed PN discovery method. The model in Figure 6 has been obtained by processing the task log  $S = T16 T14 T2 T4 T3 T5 T9 T7 T3 T5 T9 T3 T5 T8 T17 T2 T3 T5 T9 T3 T4 T7 T5 T8 T11 T13 T15 T16 T1 T2 T4 T3 T5 T8 T6 T10 T17 T2 T3 T4 T5 T6 T9 T3 T5 T10 T9 T3 T5 T9 T3 T5 T9 T3 T5 T9 T3 T5 T9 T3 T5 T8 T17 T2 T4 T3 T7 T5 T9 T3 T5 T9 T3 T5 T8 T11 T12 T15 T16 T1 T2 T4 T7 T3 T5 T8 T11 T12 T15 T16 T14 T2 T3 T5 T8 T4 T6 T10 T17 T2 T3 T4 T6 T10 T5 T9 T3 T5 T8 T17 T2 T4 T6 T3 T10 T5 T9 T3 T5 T8 T17 T2 T3 T5 T8 T4 T6 T10 T11 T13 T15 T16 T14 T2 T4 T7 T3 T5 T9 T3 T5 T8 T11 T13 T15 T16 T1 T2 T3 T5 T9 T4 T6 T3 T10 T5 T9 T3 T5 T9 T3 T5 T9 T3 T5 T8 T17 T2 T3 T5 T9 T4 T3 T6 T5 T9 T3 T10 T5 T8 T11 T12 T15 T16 T14 T2 \dots$ , where  $|S| = 1500$ .

This model includes diverse structures (nested t-component evolving concurrently) which are more complex than others published in literature. As a sign of performance, the processing time for  $S$  in a laptop computer (2.4GHz dual-core, Intel Core i5 processor, 4GB of 1333MHz DDR3 memory) was about 3.6 s.

Thanks to the software tool we developed it has been possible to test models of diverse structures, which include cycles nested into t-components, concurrency, and implicit dependencies. Special models such as two independent PNs concurrently evolving, and concurrent components related by mailbox places (message exchange) have been successfully built. This reveals the power of the method for dealing with black-box model discovery.

## VIII. CONCLUSION

The proposed method for PN discovery handles long sequences  $S_i$  representing the observed behaviour of a process from their initial states. No a priori knowledge about the number of places nor the start and end of tasks in traces  $\sigma_j$  in  $S_i$  is required.

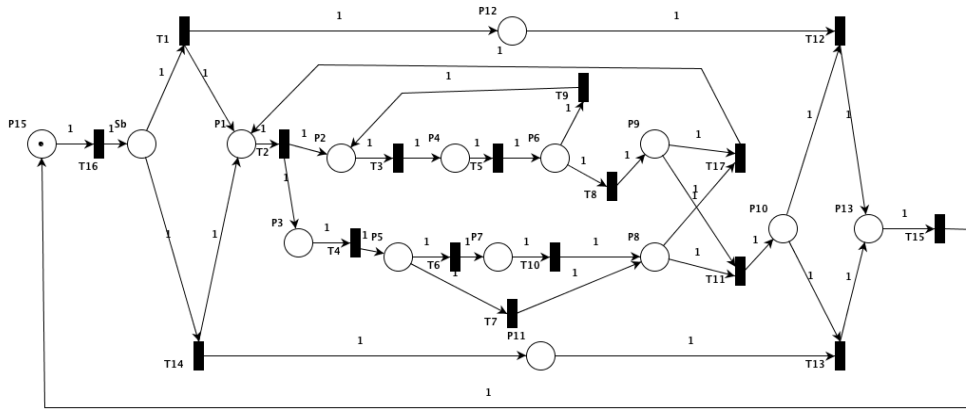


Figure 6. A non trivial discovered PN model from  $S_3$

This approach allows addressing efficiently discrete event processes exhibiting more complex behaviours than the approaches proposed in the fields of identification and process mining. This method is based mainly on searching the supports of t-invariants from the observed sequences  $S_i$ , and allows building an initial model which is adjusted later with the help of the computed t-invariants; the final model includes implicit causal relationships between transitions that have not been observed consecutively. The discovered PN fires exactly the sequences  $S_i$  from  $M_0$  and may eventually accept exceeding iterative sub-sequences, which correspond to the behaviour inherent to PN with repetitive components.

Implementation and tests revealed accuracy and efficiency of the method when complex PN structures were addressed. Current research addresses the problem of PN discovery from incomplete observed sequences.

## IX. 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 39<sup>th</sup> IEEE Conf. on Decision and Control*, pp. 2266-2271, 2000
- [4] M. Meda-Campana and E. López-Mellado, "Identification of concurrent discrete event systems using Petri nets", in *Proc. of the 17<sup>th</sup> 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 47<sup>th</sup> 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 16<sup>th</sup> 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. Lopez-Mellado, and J.-J. Lesage, "Identification of partially observable discrete event manufacturing systems", in *Proc. of the 18<sup>th</sup> IEEE Conf. on Emerging Technologies & Factory Automation*, pp. 1-7, 2013
- [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*. Pub. on-line: Jan, 27, 2014. ISSN: 0951-192X pp.1-13
- [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] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining Process Models from Workflow Logs", *Lecture Notes in Computer Science*, Vol. 1377, pp. 469-483, 1998
- [15] J. E. Cook, Z. Du, C. Liu, and A. L. Wolf, "Discovering models of behavior for concurrent workflows", *Computers in industry*, 53(3), pp. 297-319, 2004
- [16] W. Van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs", *IEEE Trans. On Knowledge and Data Engineering*, 16(9), pp. 1128-1142, 2004
- [17] L. Wen, J. Wang, and J. Sun, "Detecting implicit dependencies between tasks from event logs", *Lecture Notes in Computer Science*, Vol. 3841, pp 591-603, 2006
- [18] D. Wang, J. Ge, H. Hu, and B. Luo, "A new process mining algorithm based on event type", in *Proc. of the 9<sup>th</sup> IEEE Conf. on Dependable Autonomic and Secure Computing*, pp. 1144-1151, 2011
- [19] D. Wang, J. Ge, H. Hu, B. Luo, and L. Huang, "Discovering process models from event multiset", *Expert Systems with Applications*, 39(15), pp. 11970-11978, 2012.
- [20] W. M. Van der Aalst, "Discovery, Conformance and Enhancement of Business Processes", 368 pages, Springer, 2011
- [21] PIPE 2: Platform Independent Petri net Editor 2, <http://pipe2.sourceforge.net/>