IdentifyTPN: a tool for the identification of Time Petri nets

Basile F. [*] **Chiacchio P.** [*] **Coppola J.** [*]

[*] *DIEM, Università di Salerno, Italy (e-mail: fbasile@ unisa.it).*

Abstract: Despite of the fact that the field of system identification of discrete event processes/systems has attracted the attention of many researchers in the last decade, there is a lack of tools that allow the use of the existing algorithms, alone or combined, in a unified environment. In this paper IdentifyTPN is presented, a tool developed by the Automatic Control Group of the University of Salerno, to solve a number of identification problems using Time Petri net models and algorithms, alone or combined, presented in the literature by the authors. The tool is available free of charge for interested readers.

Keywords: Discrete-event systems, System identification, Model repair, Time Petri nets.

1. INTRODUCTION

The field of system identification of discrete event processes/systems from external observation has attracted the attention of many researchers in the last decade. The methods presented in the literature for the identification of Discrete Event Systems (DESs) produce a mathematical model expressed as a Petri Net (PN) or a finite state automaton model of the system behavior from sequences observed during the system operation (Estrada-Vargas et al., 2010; Cabasino et al., 2015; Fanti and Seatzu, 2008). If the resulting model is a logical PN system, the net structure (places, transitions and arcs) and its initial marking must be identified, while if it is a timed net systems, also the timing structure should be identified.

Nowadays, real-time systems are ubiquitous, so the explicit consideration of time is crucial; to ensure the correct specification, design, control and verification of systems such as transportation systems (Basile et al., 2004), communication protocols, circuits, implemented in a centralized or decentralized control architecture (Basile et al., 2015), it is also crucial to rely on the mathematical framework provided by formal methods . In this paper Time PNs (TPNs) (Merlin, 1974) are used, for which the firing duration of a transition t can assume any value of a given interval $I(t) = [l\ ,\ u]$.

The interest for the identification of Discrete Event Systems (DESs) usually comes from different contexts, as for example reverse engineering for (partially) unknown systems, process mining, model repair, fault diagnosis, system verification. Each context exhibits identification problems with different level of complexity; this has caused the proliferation of many approaches in the literature with few differences between each other. For some applications benefits can be obtained combing these approaches: a process can be identified from a set of observations as long as a certain index is satisfied, then a model repair approach, that is less computational expensive, can be used to adjust the model using the subsequent observations. In our opinion, it is now necessary to make available as many as possible of these approaches in a unified environment.

In this paper a tool, based on the results proposed by the authors in Basile et al. (2016c) and Basile et al. (2016a), is presented to solve the following problems for labeled TPNs (LTPNs):

(1) identify a LTPN from a set of observations;
(2) repair a nominal LTPN making it able to generate a set of observations;
(3) identify a LTPN from a set of observations and a partial knowledge of nominal systems.

The tool is developed in MATLAB© and requires a Mixed Integer Linear Programming Problem (MILPP) solver (actually, it is ready to be interfaced with CPLEX©).

In (1) given a set of observed event sequences (i.e., sequences of event occurrences with their occurrence time instants), a number of places m and a set of events E, the structure of a $LTPN$, i.e., the pre and post incidence matrices **Pre** and **Post**, its initial marking $\boldsymbol{m}_0$, the timing structure $I(t)$, i.e., the firing interval of each transition $t \in T$, and the labeling function ϕ, is identified so that the set of timed sequences generated by this net system contains all the observed ones.

In (2), using the concepts of residuals, introduced in Roth et al. (2011) and extended to timed DESs context in Basile et al. (2016c) and Basile et al. (2016a), a nominal LTPN model is repaired by adding a subnet represented by a pre and post incidence matrices $\mathbf{Pre}_f$ and $\mathbf{Post}_f$ and/or enlarging of Δl and Δu the firing interval of each transition. The repaired model is able to generate the faulty observed behavior. Residuals allow to formalize two very generic fault symptoms (discrepancies) of discrete event systems, *unexpected* and *missed* behavior, leading to observed but unexpected events and missed event observations respectively.

In (3) the approach (1) and (2) are mixed to obtain a net model starting from a partial knowledge of nominal model.

As usual in DES identification, the identified system can also produce timed sequences that do not belong to the unknown system behavior, as well as, it can reproduce sequences of the original system that have not been observed.

2. NOTATIONS AND PRELIMINARY ASSUMPTIONS

It is assumed that the reader is familiar with the theory of PNs. For a complete review about them, the reader is remanded to Murata (1989).

Definition 1. (TPN system, Seatzu et al. (2013)). Let $\mathcal{I}$ be the set of closed intervals with a lower bound in the set of positive rational numbers $\mathcal{Q}^+$ and an upper bound in $\mathcal{Q}^+ \bigcup \infty$. A *Time* Petri net (TPN) system is the triple $\mathcal{T} = \langle N, \boldsymbol{m}_0, I\rangle$, where N is a standard P/T net, $\boldsymbol{m}_0$ is the initial marking, and $I : T \to \mathcal{I}$ is the *statical firing time interval function* which assigns a firing interval $[l_j, u_j]$ to each transition $t_j \in T$.

A transition t_j can be fired at time τ if the time elapsed from the enabling belongs to the interval $I(t_j)$; moreover, an enabled transition must fire if the upper bound of $I(t_j)$ is reached, thus enforcing urgency. A clock measuring the time elapsed from the enabling is implicitly associated to any transition.

It is assumed that there is a start-up transition that fires only once at time zero producing tokens considered by the initial marking and setting to zero the value of clocks. $\diamond$

Definition 2. (Labeling function). Given a Petri net N with set of transitions T, a labeling function $\phi : T \to E \bigcup \{\epsilon\}$ assigns to each transition $t \in T$ a symbol, from a given alphabet E, or assigns to it the empty string ϵ. A labeled Petri net system is a 3-tuple $\langle N, \boldsymbol{m}_0, \phi\rangle$ where N is a P/T net, $\boldsymbol{m}_0$ is its initial marking, and $\phi : T \to E \bigcup \{\epsilon\}$ is the labeling function. $\diamond$

Let E^* denote the set of all infinite strings of elements of alphabet E including the empty string ϵ.

The labeling function can be extended to define the projection operator $\phi : T^* \to E^*$ recursively as follows:

(i) if $t_j \in T$ then $\phi(t_j) = e_i$ for some $e_i \in E$;
(ii) if $\sigma \in T^* \wedge t_j \in T$ then $\phi(\sigma t_j) = \phi(\sigma)\phi(t_j)$;

Moreover, $\phi(\lambda) = \epsilon$ where λ is the empty sequence.

Definition 3. (Labeled Time Petri Net System). A Labeled Time Petri Net (LTPN) system is a couple $S = \langle \mathcal{T}, \phi\rangle$ where $\mathcal{T}$ is a TPN and ϕ is the labeling function. $\diamond$

Given the set of events w, $\phi^{-1}(w)$ is the system of sets $\{t_j \in T \mid \phi(t_j) = e_i \in w\}$, and each set is hereinafter called *interpretation* of w.

Example 1. Assume to have the alphabet $E = \{e_1, e_2, e_3\}$, the set of transitions $T = \{t_1, t_2, t_3, t_4\}$ and the labeling function ϕ such that $\phi(t_j) = \begin{cases} e_j & \text{if } j = \{1,2\} \\ e_3 & \text{if } j = \{3,4\} \end{cases}$.

Consider the set $w = \{e_1, e_3\}$, then $\phi^{-1}(w) = \big\{\{t_1, t_3\}, \{t_1, t_4\}\big\}$. $\diamond$

Definition 4. (Timed firing sequence). A sequence

$$\mathfrak{S} = (T_1, \tau_1) \dots (T_q, \tau_q) \dots (T_L, \tau_L)\,,$$

where T_q is the set of transitions fired at time τ_q and $\tau_1 < \tau_2 \cdots < \tau_L$ denote firing time instants, is called *timed firing sequence.* The position q that the couple (T_q, τ_q) occupies in the sequence is called *time step*, so (T_1, τ_1) is associated with step 1, (T_2, τ_2) is associated with step 2 and so on; the number of couples (T_q, τ_q) in $\mathfrak{S}$ is called length $L = |\mathfrak{S}|$ of the timed firing sequence.

The notation $\boldsymbol{m}[\mathfrak{S}\rangle\boldsymbol{m}'$ is used to denote that marking $\boldsymbol{m}'$ is reached from marking $\boldsymbol{m}$ by firing $\mathfrak{S}$. $\diamond$

The labeling function is extended also to timed firing sequences $\mathfrak{S}$ as follows:

(i) $\phi((T_q, \tau_q)) = (E_q, \tau_q)$, where $E_q = \{e_i \in E \mid \phi(t_j) = e_i,\ t_j \in T_q\}$
(ii) $\phi(\mathfrak{S}(T_q, \tau_q)) = \phi(\mathfrak{S})\phi((T_q, \tau_q))$.

The set T_q is made up of $n_q = |T_q|$ transitions whose firing is observed at the same instant τ_q. The firings of these transitions are enabled either by a marking $\boldsymbol{m}_k$ reached at a time $\tau_k < \tau_q$ or by the firing of another transition fired at τ_q with null firing duration.

Definition 5. (Firing Duration). Given a timed transition t_j, fired at the q-th step, enabled at the k-th step, so that $\boldsymbol{m}_k[t_j\rangle$, let $\boldsymbol{m}_k$ be the first marking that enables t_j since its previous firing, the function $\delta(t_j, k, q) : T \times \mathbb{N} \times \mathbb{N} \to \mathcal{Q}^+$ returns the time elapsed from the enabling of t_j at τ_k until its firing at τ_q, i.e., $\delta(t_j, k, q) = \tau_q - \tau_k$. $\diamond$

From now on, it is referred to $\delta(t_j, k, q)$ as the firing duration of transition $t_j \in T_q$ from the marking $\boldsymbol{m}_k$. When $\delta(t_j, k, q) = 0$ the firing of t_j at τ_q is called *immediate*, otherwise, when $\delta(t_j, k, q) > 0$, the firing of t_j is called *timed.*

Let $\boldsymbol{m}_0$ be the initial marking of the system, the set of candidate markings for the enabling of a transition $t_j \in T_q$ can be formally defined as $\mathcal{M}(t_j, q) = \big\{\boldsymbol{m}_k \mid \exists \mathfrak{S}_T^{'}, \mathfrak{S}_T^{''}, \mathfrak{S} = \mathfrak{S}_T^{'}\mathfrak{S}_T^{''},\ \boldsymbol{m}_0[\mathfrak{S}_T^{'}\rangle\boldsymbol{m}_k[\mathfrak{S}_T^{''}\rangle\boldsymbol{m}_q$, with $t_j \in \mathfrak{S}_T^{''},\ k < q\ :\ \tau_k + l_j \leq \tau_q \leq \tau_k + u_j\big\}$, having cardinality $|\mathcal{M}(t_j, q)|$.

The set T_q can be partitioned into the couple of sets (T_q^t, T_q^{im}): $T_q^t = \{t_j \in T_q | \ \exists k, \boldsymbol{m}_k \in \mathcal{M}(t_j, q)\}$ is the set of transitions fired at τ_q with timed firing, with cardinality $n_q^t = |T_q^t|$, $T_q^{im} = T_q \setminus T_q^t$, with cardinality n_q^{im}, is the set of transitions fired at τ_q with immediate firing.

Immediate firings always follow the timed ones, even if they are observed at the same time τ_q. Indeed an immediate firing occurs at the same time it has been enabled, while a timed firing occurs in a subsequent time with respect to the one at which it has been enabled. Consequently, a timed firing enabled by an immediate firing occurred at time τ_q, surely fires in a time greater than τ_q.

The firing of transitions in the set T_q^t is concurrent, however, each firing can have been enabled at a different marking. On the other hand, the firing of transitions in T_q^{im} may be sequential. Given the set of transitions T_q^{im}, these transitions can fire in any order, which, anyway, can include concurrent transition firings.

Denote $\boldsymbol{m}_{q_1}$ the marking reached by firing transitions belonging to T_q^t, Denote $\boldsymbol{m}_{q_s}$, with $s \geq 2$, the markings reached after the immediate firings of transitions.

Given the firing sequence associated to the set T_q^{im}, it can be considered made up of the union of n_q^{im} disjoint subsets of concurrent transition firings. Hence, firing of transitions in T_q^{im} can be considered occurred in n_q^{im} substeps; each substep is denoted by q_s, with $s \in [2, n_q^{im}+1]$. Finally, it holds that $T_q^{im} = \bigcup_{s=2}^{n_q^{im}+1} T_{q_s}^{im}$.

This paper focuses on the context of automated manufacturing systems, where a control architecture interacts with a plant according to a scan time faster than the time evolution of the system. In this context, the multiple firing of a transition in the same time instant has no sense. This motivates the next assumption.

Assumption 1. A transition can fire only once in the same time instant.

However, the results presented in this paper are still valid removing this assumption, introducing some technicalities.

Definition 6. (Timed event sequence). A sequence

$$\gamma = (E_1, \tau_1) \dots (E_q, \tau_q) \dots (E_L, \tau_L),$$

where E_q is the set of events occurred at time τ_q, $\tau_1 < \tau_2 \dots < \tau_L$ denote firing time instants, is called *timed event sequence*. The position q the couple (E_q, τ_q) occupies in the sequence is called *time step*, so (E_1, τ_1) is associated with step 1, (E_2, τ_2) is associated with step 2 and so on; the number of couples (E_q, τ_q) in γ is called length $L = |\gamma|$ of the timed event sequence. ◇

The operator $\phi^{-1}(\cdot)$ can be extended to the timed event sequence γ, as shown in the follow:

$\phi^{-1}(\gamma)$ is the set of sequences $\{\mathfrak{S}_r = (T_{r,1}, \tau_1) \dots (T_{r,q}, \tau_q) \dots (T_{r,L}, \tau_L) : \phi(\mathfrak{S}_r) = \gamma,\ T_{r,q} \in \phi^{-1}(E_q)\}$, and each sequence $\mathfrak{S}_r$ is hereinafter called *interpretations* of γ.

The set of events is partitioned into the set E^c of controllable events, assumed known, and the set E^{uc} of uncontrollable events. Consequently, the transition set T is partitioned into the set T^c of controllable transitions, with cardinality n_c, and the set T^{uc} of uncontrollable transitions, with cardinality n_{uc}.

Assumption 2. (Controllable transitions). It is assumed that:

(1) All controllable transitions are known and immediate since they are associated to events managed by the controller.
(2) All transitions that make up a choice – i.e., all transitions $t \in p^\bullet$ with $|p^\bullet| > 1$ – must be controllable. Hence if $|p^\bullet| > 1 \quad \Rightarrow \quad I(t) = [0, 0]\ \forall\ t \in p^\bullet$, for all the places in P. This assumption is motivated by the fact that, when a timed activity is associated with conflicting transitions, a conflict resolution policy may be a race between conflicting transitions, which is pointless in the context of manufacturing systems. Then, conflicts only include controllable transitions which are immediate and known, so a set of constraints, which must be fulfilled in order to guarantee this assumption holds, can be devised. ◇

The approach used in this paper does not perform any controlling action on the system by means of controllable events, but only assumes that the set of controllable events is known. This is realistic in the context of manufacturing systems, where controllable events are the outputs of the controlling agent, which is usually accessible.

Assumption 3. A maximal upper bound $u_{max}(e_i)$ and a minimal lower bound $l_{min}(e_i)$ are available for each event $e_i \in E$.

Given the step k, reached at time τ_k, if an event e_i occurs later than $\tau_k + u_{max}(e_i)$ then the activity associated to the event does not start at τ_k. When $u_{max}(e_i)$ is not explicitly defined, then it is assumed $u_{max}(e_i) = \infty$ and consequently the occurrence of e_i at step q can be associate to any activity started at time $\tau_k < \tau_q$. Given an activity, started at time τ_k, an event e_i occurs in a time greater than or equal to $l_{min}(e_i)$. When $l_{min}(e_i)$ is not explicitly defined, then it is assumed $l_{min}(e_i) = 0$. ◇

Knowledge of such bounds helps to devise counterexamples, that are sequences that cannot be generated by the LTPN.

Assumption 4. (Properties of the observed system). The observed system is modeled by a LTPN system with the following assumptions

(1) *λ-free labeled nets*, i.e., there can be multiple transitions with the same label but there are no transitions labeled with the empty string.
(2) *k-bounded nets*, i.e., the number of tokens in each place of the net is never greater than k.
(3) *Single-server* firing semantic (more details in Seatzu et al., 2013), i.e., no concurrent firings of the same transition are possible.
(4) *Enabling memory* policy of timed transitions, i.e., when a new marking is reached and a timed transition is not enabled, the elapsed time is reset. ◇

3. TOOL OVERVIEW

IdentifyTPN tool allows two different modes of operations: a) the interactive execution and b) the execution from file. In the former, the user interacts with the tool by means of menus and guided procedures. In the latter, the user runs the program implementing the sequence of operations to be executed from MATLAB© command window. IdentifyTPN tool behavior is summarized in Fig. 1.

3.1 Interactive execution

`StartMenu()` starts the execution of the tool in the interactive execution mode: a "menu" is printed in the MATLAB© command window, allowing the user to choose between 5 options:

(1) *Add a new LTPN system*: function `modelFromCommandWindows()` creates a `.m` file, where a LTPN system is saved. Such a file can be used successively as nominal (partial) model of the system during the Repaired Model or Partial Model Identification.

LTPN system is acquired by means of a guided procedure, that allows the user to enter the structure of the net (number of places and transitions, incidence matrices), the timed structure (firing interval for each transition),

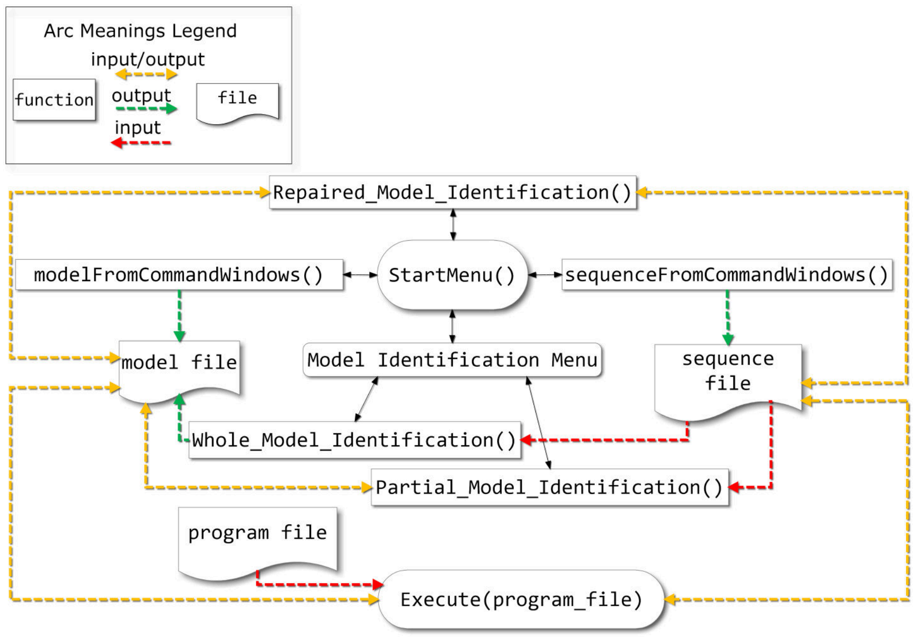
Arc Meanings Legend
input/output
function
output
file
input
Repaired_Model_Identification()
modelFromCommandWindows()
StartMenu()
sequenceFromCommandWindows()
model file
Model Identification Menu
sequence file
Whole_Model_Identification()
Partial_Model_Identification()
program file
Execute(program_file)

Fig. 1. Scheme of the tool behaviour.

the initial marking $\boldsymbol{m}_0$ and the labeling function ϕ of the system.

At the end of the procedure it is asked to the user to select (an existing or a new) `.m` file where, automatically, the function `[m0,Pre,Post,l,u,phi]=namefile()` is created, where `namefile` is the name (without extension) of the file selected by the user.

(2) *Add a new sequence*: function `sequenceFromCommandWindows()` is executed to create a `.m` file, where an observed timed event sequence is saved. Such a file can be used successively during the (Repaired) Model Identification.

The sequence is acquired by means of a guided procedure, that allows the user to enter the length of the sequence and the couple (E_q, τ_q) for each step q. At the end of the procedure it is asked to the user to select (an existing or a new) `.m` file where, automatically, the function `[sequence]=namefile()` is created, where `namefile` is the name (without extension) of the file selected by the user.

Sequence (model) files can be also created manually by the user along the same line of the ones created automatically, as long as the keyword `%<MODEL>` (`%<SEQUENCE>`) is added at the first line of the file. Such a keyword is used during the (repaired) model identification to check that a correct file is selected by the user, when he/she is invited to do it. Notice that once checked the keyword coherence, the correctness of the file content is assumed.

(3) *Start identification*: a *sub-menu* is printed in the command windows, allowing the user to choose between two kinds of identification:

- whole model identification;
- partial model identification.

In the first case, function `Whole_Model_Identification()` is executed and it is required to the user to select a file where the observed sequence γ is stored and to enter some additional information about the system to identify (e.g., the set E^c of controllable events, the event lower (upper) bound l_{min} (u_{max}). These information can be entered both by means of a guided procedure and loading an existing file, reporting the keyword `%<INFO>` at the first line. Then, the identification procedure starts.

In the second case, function `Partial_Model_Identification()` is executed, and the file where the partial model of the system is stored must be selected by the user, as well.

In both cases, first an MILPP model is built according to Basile et al. (2016c) and CPLEX$^{©}$ syntax, then CPLEX$^{©}$ is invoked; at the end of the identification, the function returns the identified model $S = \langle N, \boldsymbol{m}_0, I, \phi \rangle$, that can be also saved in `.m` file to be used for successive analysis.

(4) *Start Repaired Model Identification*: function `Repaired_Model_Identification()` is executed to identify a repaired model. To proceed with the identification, the user must select the files where the nominal LTPN system S_0 and the observed event sequence γ are stored. Moreover the user has the possibility to add additional constraints, limiting the set of places that can belong to the pre(post)-set of fault transitions as well as the set of transitions the firing interval lower (upper) bound l (u) can be extend for. Then, an MILPP model is built according to Basile et al. (2016a) and CPLEX$^{©}$ syntax and CPLEX$^{©}$ is invoked.

The function returns the repaired identified model S and the correct *interpretation* of the timed event sequence γ, the timed firing sequence $\mathfrak{S}_r$, including fault transitions firings: they can be both saved in `.m` file to be used for successive analysis.

(5) *Exit*: the execution of the tool is ended.

3.2 Execution from file

In addition to the interactive execution, it exists an *off-line* mode of operation, running a program the user has previously written in a file, by means of the function `Execute(program_file)`.

The program must be written as in the follow:

```
%<PROGRAM>
instr_1 FROM file_1a, file_1b;
instr_2 FROM file_2a, file_2b;
⋮
instr_n FROM file_na, file_nb;
%<END_PROGRAM>
```

and saved as a `.m` file.

The following set of high level instructions are available:

(1) `identify FROM seqFile` starts the model identification of the system based on the timed event sequence γ, stored in the file `seqFile`, and returns the identified model, S, the additional information used during the identification, and the correct *interpretation* of the timed event sequence γ, the timed firing sequence $\mathfrak{S}_r$, as well; additional information must be entered by the user in the same way described in Section 3.1.

(2) `p_identify FROM seqFile, modelFile` starts the model identification of the system based on the timed event sequence γ, stored in the file `seqFile`, and on the partial model stored in the file `modelFile`, and returns the identified model S, the additional information used during the identification, and the correct *interpretation* of the timed event sequence γ, the timed firing sequence $\mathfrak{S}_r$, as well; additional information must be entered by the user in the same way described in Section 3.1.

(3) `repair_model FROM seqFile, modelFile` starts the identification of the repaired model of the system based on the timed event sequence γ stored in the file `seqFile` and on the nominal model stored in the file `modelFile`; at the end it returns the identified repaired model S and the correct *interpretation* of the timed event sequence γ, the timed firing sequence $\mathfrak{S}_r$, including firings of added fault transitions. Additional constraints can be entered by the user in the same way described in Section 3.1.

A crucial functionality of the tool is the possibility to combine different identification approaches. In our opinion, this can be extremely useful because the computational complexity of available algorithms is very different. Hence, it would be efficient to use a complex algorithm until the knowledge of the systems is not sufficient, then one can use the identified model as input for a less complex algorithm, as for example a model repair algorithm, dealing as reparations the next discrepancies with observed sequences. To switch between two different identification approaches the following syntax must be used

```
instr_1 FROM file1, file2 UNTIL step THEN {
instr_2 UNTIL end;
}
```

It allows to execute `instr_1` until the step `step` of the observed sequence and to execute `instr_2` for the remaining steps. The second instruction can be either `p_identify` or `repair_model`: in the former case the model obtained as result of the first instruction is assumed as the partial known model of the system while in the latter it is assumed to be the nominal one. For sure, `step` will be replaced by a logical condition evaluating the right degree of satisfaction of the model identified using `instr_1`.

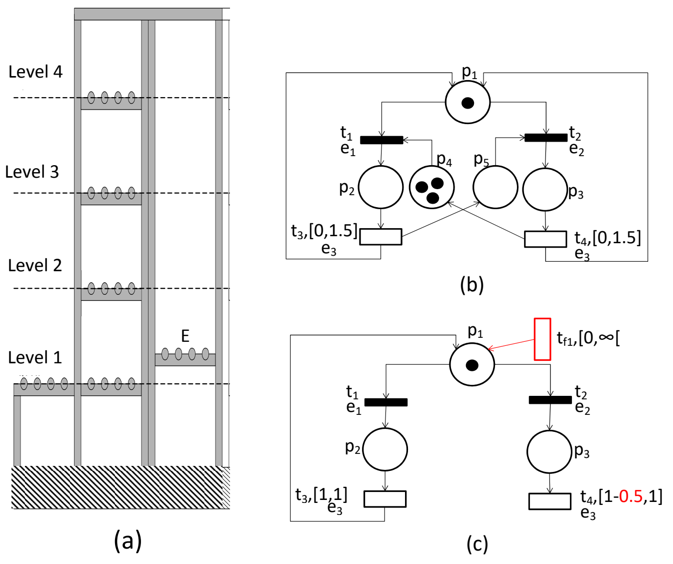

Fig. 2. (a) One zone of the prototype of an automated handling system installed at the University of Salerno; (b) LTPN model of the example and (c) identified model.

4. EXAMPLE

This example focuses on the combination of two identification approaches for obtaining the model of a system, starting from an off-line observed timed sequence. In detail, the example presented in Basile et al. (2016b) is considered again to show how it is possible to enrich the identified model using a less expensive approach (in terms of computation complexity and resolution time).

The scheme of the observed system is shown in Fig. 2(a): it is one zone of an automatic material handling system, installed at the University of Salerno, made up of four levels and an elevator, named E, moving stock units in vertical way though the four levels. The available events are e_1 (start to move to one level up), e_2 (start to move to one level down), and e_3 (elevator is aligned at a level); e_1 and e_2 are controllable.

The LTPN of Fig. 2(b) models the behavior of the elevator. It is assumed that E is initially stopped to Level 1. The places have the following meaning: p_1 - E at a level; p_2 - E moving up; p_3 - E moving down; p_4 - number of levels currently over E; p_5 - number of levels currently under E.

During the system observation, the timed event sequence $\gamma = (\{e_1\}, 0)\ (\{e_1, e_3\}, 1)\ (\{e_2, e_3\}, 2)\ (\{e_3\}, 3)\ (\{e_2\}, 3.4)\ (\{e_3\}, 3.9)$ (of length $L = 6$) has been saved into the file `seq_demo.m`.

To identify the system model, two different approaches are executed sequentially: first a whole model identification is executed, then the identified model is assumed as the nominal model of the system and starting from the step 5 of sequence γ, a repaired model identification is executed.

With this aim, the following program has been written and saved in the file `demo.m`

```
%<PROGRAM>
identify FROM seq_demo_WI.m UNTIL 4 THEN {
repair_model UNTIL end;
}
%<END_PROGRAM>
```

When function
`identify FROM seq_demo_WI.m UNTIL 4`
ends, the identified model and the additional information about the system are automatically converted in the right format to be used by function `repair_model UNTIL end`: no user action is needed.

Model identified at the end of the first instruction is the one shown in Fig. 2(c) (discarding red parts). Such a model generates the timed event sequence γ until the step 4 but not the observation of events e_2 and e_3 at the two successive steps. Indeed, when the Repaired Model Identification starts, both occurrences are associated to an unexpected firing of transitions; in details, the occurrence of event e_2 is associated to an unexpected firing of transition t_2, enabled by a marking reached after the firing of a fault transition, while the occurrence of event e_3 is associated to a firing of transition t_4, enabled after the firing of transition t_2, having a firing duration shorter than its firing interval lower bound.

The final identified model is the one of Fig. 2(c) including the fault transition t_{f1} whose firing at time $\tau_4 = 3$ has enabled the one of transition t_2 at step 5 and the extension of the transition t_4 firing interval of the amount $\Delta l = 0.5$. The resulting interpretation of γ, updated with the fault transition firing, is $\mathfrak{S} = (\{t_1\}, 0)\ (\{t_1, t_3\}, 1)\ (\{t_2, t_3\}, 2)\ (\{t_4, t_{f1}\}, 3)\ (\{t_2\}, 3.4)\ (\{t_4\}, 3.9)$.

Instruction `identify FROM seq_demo_WI.m UNTIL 4` needs about 30 seconds to return the solution of the MILPP while the resolution of each one of two MILPPs (one for each observed discrepancy) built during instruction `repair_model UNTIL end` requires less than 1 second.

Currently, the step at which commuting between the two approaches is decided in an *a-priori* way from the user, further tool developments are in progress to provide a performance index able to establish the opportune step to execute the switch. The basic idea is the following: given an off-line observed sequence of length L, the identification of the model is stopped at a step q, then the ratio between the number of discrepancies detected in the following $L-q$ steps, assuming as nominal the identified model, and the number of these steps is evaluated. The commutation to the second identification approach occurs only when the value of such a ratio is less than a fixed threshold.

This example and other ones, as well as the whole IdentifyTPN tool, are available free of charge at the address `http://www.automatica.unisa.it/IdentifyTPN.php`.

5. CONCLUSION

IdentifyTPN is a tool for the identification of timed DESs, based on Time Petri net models and Mixed Integer Linear Programming Problem solvers. The tool has been developed by the Automatic Control Group of the University of Salerno and is available free of charge to interested readers. The main features of the tool have been shown to be:

(1) indentify a LTPN from a set of observations;
(2) repair a nominal LTPN making it able to generate a set of observations;
(3) identify a LTPN from a set of observations and a partial knowledge of nominal systems.

REFERENCES

Basile, F., Carbone, C., Chiacchio, P., Boel, R.K., and Avram, C.C. (2004). A hybrid model for urban traffic control. *2004 IEEE International Conference on Systems, Man and Cybernetics (SMC'04)*, 1795–1800.

Basile, F., Cordone, R., and Piroddi, L. (2015). A branch and bound approach for the design of decentralized supervisors in petri net models. *Automatica*, 52, 322–333.

Basile, F., Chiacchio, P., and Coppola, J. (2016a). A Novel Model Repair Approach of Timed Discrete-Event Systems With Anomalies. *IEEE Transactions on Automation Science and Engineering*, 13(4), 1541–1556.

Basile, F., Chiacchio, P., and Coppola, J. (2016b). Identification of labeled time Petri nets. *13th International Workshop on Discrete Event Systems (WODES 2016)*, 478–485.

Basile, F., Chiacchio, P., and Coppola, J. (2016c). Identification of Time Petri net models. *IEEE Transactions on Systems, Man and Cybernetics: Systems, doi=http://dx.doi.org/10.1109/TSMC.2016.2523929.*

Cabasino, M.P., Darondeau, P., Fanti, M.P., and Seatzu, C. (2015). Model identification and synthesis of discrete-event systems. In M. Zhou, H.X. Li, and M. Weijnen (eds.), *Contemporary Issues in Systems Science and Engineering*, IEEE/Wiley Press Book Series, 343–366. John Wiley & Sons, Inc.

Estrada-Vargas, A.P., Lopez-Mellado, E., and Lesage, J.J. (2010). A Comparative Analysis of Recent Identification Approaches for Discrete-Event Systems. *Mathematical Problems in Engineering.*

Fanti, M.P. and Seatzu, C. (2008). Fault diagnosis and identification of discrete event systems using Petri nets. *9th International Workshop on Discrete Event Systems (WODES 2008), Goteborg, Sweden*, 432–435.

Merlin, P.M. (1974). *A study of the recoverability of computing systems.* Ph.D. thesis, University of California, Irvine.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of IEEE*, 77(4), 541–580.

Roth, M., Lesage, J.J., and Litz, L. (2011). The concept of residuals for fault localization in discrete event systems. *Control Engineering Practice*, 19(9), 978–988.

Seatzu, C., Silva, M., and van Schuppen, J.H. (eds.) (2013). *Control of Discrete-Event Systems*, volume 433 of *Lecture Notes in Control and Information Sciences.* Springer.