Diagnosis of supervision patterns on bounded labeled Petri nets by Model Checking

Yannick Pencolé, Audine Subias

Diagnosis of supervision patterns on bounded labeled Petri nets by Model Checking

Yannick Pencolé and **Audine Subias**
LAAS-CNRS, Université de Toulouse, CNRS,INSA, Toulouse, France.
e-mail: (ypencole@laas.fr, subias@laas.fr)

Abstract

This paper investigates the problem of pattern diagnosis of systems modeled as bounded labeled Petri nets that extends the diagnosis problem on single fault events to more complex behaviours. An effective method to solve the diagnosis problem is proposed. It relies on a matching relation between the system and the pattern that turns the pattern diagnosis problem into a model-checking problem.

1 Introduction

Fault diagnosis of Discrete Event Systems (DES) is a generic problem that has been widely studied [1]. The original definition has been introduced in [2] in the automaton formalism. The fault diagnosis problem was based on a system model and on an explicit set of fault events. From this early approach several contributions were proposed exploring different modeling formalisms among them, communicating automata ([3], [4], [5]) or Petri nets ([6], [7], [8], [9], [10]) leading to various diagnosis methods. This original problem has also been extended to consider more complex faulty behaviours than simple fault events. In [11] the authors consider as faulty, behaviours that violate a given formal specification such as reaching a dead locking state, and propose to use linear-time temporal logic (LTL) to express these complex behaviours. In [12] *supervision patterns* are introduced by means of labelled transition systems to define complex behaviours one may be interested in to diagnose. Supervision patterns as occurrence of one fault, occurrence of multiple faults, ordered occurrence of events faults but also multiple occurrence of a fault are considered by the authors. In [13], such patterns are called *diagnosis conditions* and are formally represented with LTL properties with past operators. The advantage of such approaches is their generalizing power to tackle a large spectrum of diagnosis problems. Moreover, the use of patterns emphasizes on the separation between the behavioral model of the system and the objectives of the diagnosis tasks defined as patterns [14]. Another advantage is the possibility of pattern reusability to address others diagnosis problems on others systems.

In this article we revisit the diagnosis of patterns in the framework of Petri nets. We propose a concise way to model patterns that stresses pattern advantages. The diagnosis problem is posed as a *pattern matching* problem, so that the proposed approach does not derive a diagnoser-like machine like in [12] but a machine that captures the matching between the system and a pattern. The construction of this machine is based on a specific product of Petri nets called the *system-pattern product*. The diagnosis problem of a supervision pattern for a given sequence of observations is then solved by determining the intersection between the system's evolutions that produce the input sequence and those that match the pattern. What we want to emphasize in this paper is how this task can be totally and efficiently entrusted to a model checker fed by Petri nets. An interesting point is that the proposed approach can be fully coupled with a pattern diagnosability analysis of systems [15].

The Petri net formalism used in the approach and the system and patterns modeling are presented in Section 2. Section 3 states the diagnosis problem into a pattern matching problem. Section 4 focuses on the construction of the Petri machine that capture the matching. The implementation of the diagnosis problem as a model checking problem is given Section 5. We present experimental results in Section 6. Finally, conclusions and some perspectives are given Section 7.

2 Modeling framework

Patterns aim at modeling complex faulty behaviours or any normal behaviour of interest. This section presents the modeling framework of patterns and systems.

To have a complete characterization of the diagnosis problem, we use Labeled Petri nets with transition *priorities* ([16]). This formalism extends the classical Labeled Petri net. The use of priorities is a convenient way to manage transition conflicts in a model and then to enhance the expressiveness of the model. In particular, transition *priorities* will be used here to overcome the problem of modeling the interactions between the system and a pattern. This section first presents the formalism, called *L-type Labeled Prioritized Petri Nets*, that is used throughout this paper. Then the modeling of the system and the pattern are presented.

2.1 L-type Labeled Prioritized Petri Nets

Definition 1 (LLPPN). *An L-type Labeled Prioritized Petri Net (LLPPN for short) is a 7-uple $N = \langle P, T, A, \succ, \ell, \Sigma, Q\rangle$ where:*

- *P is a finite set of nodes called places;*
- *T is a finite set of nodes called transitions and $P \cap T = \varnothing$;*
- *$A \subseteq (P \times T) \cup (T \times P)$ is a binary relation transitive, not reflexive and not symetrical [17].*

- *Σ is a finite set of transition labels (events);*
- *$\ell : T \to \Sigma$ is the transition labeling function;*
- *Q is the set of final markings.*

The state of an LLPPN is defined as a *marking*. A marking M is a function $M : P \to \mathbb{N}$ which maps any place of the net to a number of tokens contained in this place.

A *marked LLPPN* is a couple $\langle N, M_0\rangle$ also denoted as the 8-tuple $\langle P, T, A, \succ, \ell, \Sigma, Q, M_0\rangle$. The *preset* $\mathrm{pre}(n)$ of a node n is the set of nodes $\mathrm{pre}(n) = \{n' \in P \cup T : (n', n) \in A\}$ and the *postset* $\mathrm{post}(n)$ of a node n is the set of nodes $\mathrm{post}(n) = \{n' \in P \cup T : (n, n') \in A\}$. By construction the preset and postset of a transition (respectively a place) are two sets of places (respectively two sets of transitions). An LLPPN is a classical Petri Net formalism with some additional constraints or pieces of information.

Static priorities between transitions are introduced (relation $\succ$) to manage transition conflicts. Priorities add expressiveness to the Petri net and modify the model's semantics. Indeed a transition cannot be fired if a transition with higher priority is firable at the same time: a net transition t is *firable* from a given marking M if and only if $(\forall p \in \mathrm{pre}(t), M(p) > 0) \wedge (\forall t' \in T, t' \succ t \implies \exists p' \in \mathrm{pre}(t'), M(p') = 0)$. This firing rule ensures that any firable transition t is firable in the usual sense (when $\succ = \varnothing$) but it avoids the choice of firing transitions with a lower priority that are also firable in the usual sense.

Firing a transition t consumes a token from any place in the preset and adds a token in any place of the postset. From a marking M, the fire of a transition t leads to the new marking M' such that $M' = M \setminus \mathrm{pre}(t) \cup \mathrm{post}(t)$, which is denoted $M \xrightarrow{t} M'$. A marking M is *reachable* in a marked LLPPN if it exists a sequence of firable transitions $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots \xrightarrow{t_k} M$, also denoted $M_0 \xrightarrow{r} M$ where r is the sequence $t_1 t_2 \dots t_k$, the sequence r is called a run of the net. For a given LLPPN N and a given marking M, the set of reachable markings from M is denoted: $R(N, M)$.

An LLPPN also contains a transition labeling function ℓ that maps any transition to a label of a set Σ, this is the way to associate physical events with transitions. Last but not least, an LLPPN has a set of final markings Q. The set Q is the way to define the language that is associated with an LLPPN (L-type language of [18]).

Let $r = t_1 \dots t_k \in T^*$ and let $\ell(r)$ denote the sequence $\ell(t_1)\dots\ell(t_k) \in \Sigma^*$, the language generated by a given LLPPN is defined as follows:

Definition 2 (Language of an LLPPN)**.** *The language generated by an LLPPN $N = \langle P, T, A, \succ, \ell, \Sigma, Q, M_0\rangle$ is:*

$$\mathcal{L}(N) = \{\ell(r) : r \in T^* \wedge M_0 \xrightarrow{r} M \wedge M \in Q\}.$$

2.2 System model

We propose in this paper to model a system over the set of events Σ as an LLPPN $\Theta = \langle P, T, A, \succ, \ell, \Sigma, Q, M_0\rangle$. As introduced in [2], systems generate prefix-closed regular languages: for any run r of the system, any of its prefix $r' : r = r'r''$ is indeed a run. It follows that such a system can be represented by a bounded LLPPN (regular language) and with a set of final markings corresponding to the set of reachable markings $Q = R(\Theta, M_0)$ (prefix-closed language). The set of events Σ generated by the system is partitioned into two subsets: Σ_o is the set of observable events and Σ_u the set of unobservable ones.

Definition 3 (System model)**.** *The model of a system is an LLPPN $\Theta = \langle P, T, A, \succ, \ell, \Sigma_o \cup \Sigma_u, Q, M_0\rangle$ such that:*

- *Q is the set of reachable markings $R(\Theta, M_0)$;*
- *the Petri net is bounded ($\exists n \in \mathbb{N}^+ : \forall M \in R(\Theta, M_0), \forall p \in P, M(p) \le n$).*

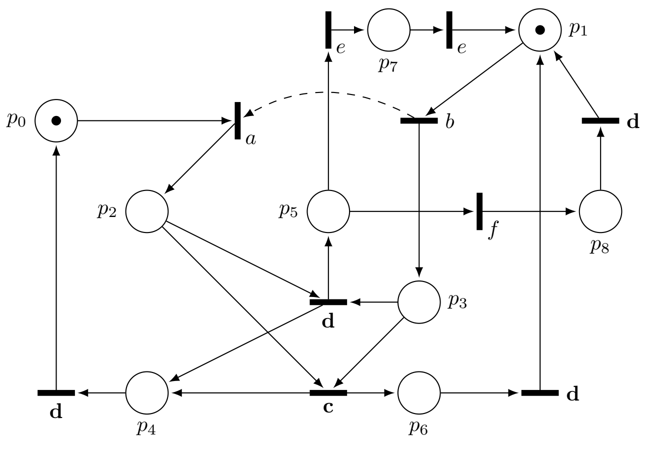

Figure 1: System example Θ_1 with $\Sigma_o = \{c, d\}$.

Figure 1 presents an example of such a model Θ_1. Bold events **c** and **d** are the observable events of the system whereas the events $\{a, b, e, f\} = \Sigma_u$ are the unobservable events. The dashed line represents the priority between the transitions of event b and a. It ensures that the transition labeled with b is always fired before the one labeled with a in a marking M such that $M(p_0) \ge 1 \wedge M(p_1) \ge 1$ (as the initial marking of the system).

2.3 Pattern model

The notion of pattern requires the definition of a general and flexible framework allowing to design a large spectrum of diagnosis problems including simple faulty event, multiple faults, fault repetitions....and more generally any behaviour of interest (faulty or not). The use of fault pattern helps to clearly separate the system and the diagnosis objectives (see [14]). The reusability is then reinforced to tackle new problems. For complex patterns this separation principle improves the applicability. Indeed, a modification of the system model in order to merge with the pattern is a tricky task that might even be impracticable in real cases. This would suppose to search in the system model for all the traces where the pattern has occurred and to modify them accordingly to the pattern model. Moreover, the system model would have to be modified each time a new pattern has to be analyzed.

Definition 4 (Pattern)**.** *A pattern Ω is an LLPPN*

$$\Omega = \langle P, T, A, \succ, \ell, \Sigma, Q, M_0\rangle$$

such that

1. *$M_0 \notin Q$ i.e. the language of the pattern does not contain ε;*
2. *from any reachable marking M of $R(\Omega, M_0)$ there exists a sequence of transitions $r \in T^*$ such that $M \xrightarrow{r} M'$ and $M' \in Q$;*
3. *from any reachable marking M of $R(\Omega, M_0)$ there is no event $e \in \Sigma$ that labels more than one firable transition;*

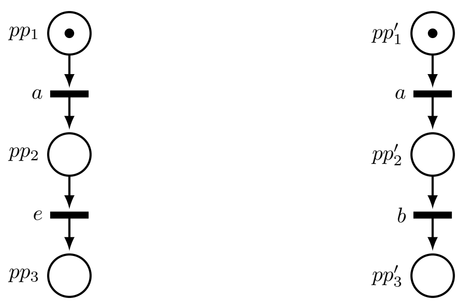

Figure 2: Two sequence patterns $\Omega_{a\to e}$ and $\Omega_{a\to b}$ with $Q_{\Omega_{a\to e}} = \{M : M(pp_3) = 1\}$ and $Q_{\Omega_{a\to b}} = \{M : M(pp'_3) = 1\}$.

4. $\forall M \in Q, \forall M' : (\exists t \in T, M \xrightarrow{t} M') \Rightarrow M' \in Q$;

5. $\succ = \emptyset$, *(no priority).*

Firstly, Figure 2 presents two similar patterns that will be used as running examples. The set of final markings of $\Omega_{a\to e}$ only contains one reachable marking, the one with $pp_3 = 1$. The language associated to $\Omega_{a\to e}$ is $\mathcal{L}(\Omega_{a\to e}) = \{ae\}$ that is the sequence of event a followed by e.

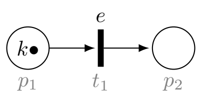

Figure 3: Pattern (Ω_1) about k occurrences of an event e with the final markings $Q_{\Omega_1} = \{M : M(p_2) = k\}$.

Figure 3 shows a pattern Ω_1 representing k occurrences of the event e ($k \in \mathbb{N}^+$). Figure 4 presents the LLPPN model of a pattern Ω_2 representing the occurrence of n events $\{e_1, e_2, \ldots, e_n\}$ in no fixed order. The language $\mathcal{L}(\Omega_1)$ generated by pattern Ω_1 is $\mathcal{L}(\Omega_1) = \{e^k\}$ that is the sequence of exactly k events e. The language $\mathcal{L}(\Omega_2)$ gathers any possible interleavings of the events $\{e_1, \ldots, e_n\}$, the cardinality of the language is $|\mathcal{L}(\Omega_2)| = n!$.

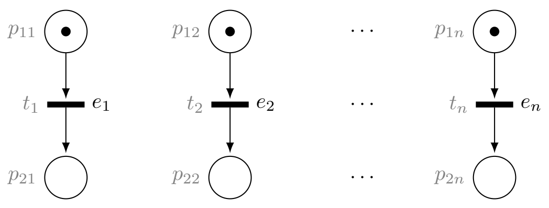

Figure 4: Pattern (Ω_2) about the occurrence of n events in no fixed order with the final markings $Q_{\Omega_2} = \{\{p_{21}, p_{22}, \ldots, p_{2n}\}\}$.

In this way, a pattern can easily be modeled directly disregarding any notion of system language contrarily to [12] or to ([19]). This concise way to model pattern reinforces the separation between the system and the diagnosis objectives. The way to handle the complex interactions between a system and a pattern is then not part of the modeling phase. It is managed by a specific product operator that is defined in Section 6.

3 Diagnosis problem

Intuitively speaking, pattern diagnosis is the problem of determining the set of patterns that have effectively occurred and that could explain a sequence of observations. The formal definition of the diagnosis problem then depends on the notion of pattern occurrence that is defined below.

A pattern Ω has occurred in a run r of the system if it is possible to extract from the event sequence $\rho = \ell(r) \in \mathcal{L}(\Theta)$ an ordered set of events that is a word of the language $\mathcal{L}(\Omega)$ generated by the pattern, in this case we say that the sequence ρ *matches* the pattern Ω and is denoted $\rho \Supset \Omega$ (we also say for convenience that the run r matches the pattern Ω if $\ell(r) \Supset \Omega$).

Definition 5 (Matching of a pattern). *A sequence $\rho \in \Sigma^*$ matches a pattern Ω ($\rho \Supset \Omega$) if there exists at least a sequence σ of $\mathcal{L}(\Omega)$ that is matched by ρ ($\rho \Supset \sigma$).*

Definition 5 relies on the definition of sequence matching.

Definition 6 (Sequence matching). *A sequence $\rho \in \Sigma^*$ matches another sequence $\sigma \in \Sigma^*$, denoted $\rho \Supset \sigma$, if:*

- *σ is empty ($\sigma = \varepsilon$); or*
- *$\sigma = s.\sigma_1, s \in \Sigma, \sigma_1 \in \Sigma^*$ and there exist two sequences $\rho_0, \rho_1 \in \Sigma^*$ such that:*
  1. *$\rho = \rho_0 s \rho_1$;*
  2. *$\rho_1 \Supset \sigma_1$.*

A sequence ρ matches a sequence σ if ρ contains any event of the sequence σ and these events occur in the same order as in σ. Obviously ρ can contain several collections of events that gather the events of σ in the same order; in this case σ occurs several times in ρ. The pattern ρ matches σ if σ occurs at least once. Figure 5 illustrates how the sequence $\rho = bcaabcac$ matches the sequence $\sigma = acc$. Definition 6 is recursively applied three times till $\sigma_3 = \varepsilon$. In this example, σ occurs twice in ρ. Back to the example of Figure 1, the system Θ_1 matches both the patterns $\Omega_{a\to e}$ and $\Omega_{a\to b}$ of Figure 2. The sequence $ba\mathbf{d}e$ is a possible run of Θ_1 and $ba\mathbf{d}e \Supset ae$. Θ_1 also matches $\Omega_{a\to b}$ as the sequence $ba\mathbf{d}eeb\mathbf{d}$ is a possible run of Θ_1 and $ba\mathbf{d}eeb\mathbf{d} \Supset ab$.

Based on the notion of pattern, the classical fault diagnosis problem on DESs can be generalized as proposed in [12]. Let $\mathscr{P}_{\Sigma_1\to\Sigma_2}$: $\Sigma_1^* \to \Sigma_2^*$ denote the projection function of any sequence of Σ_1^* to a sequence of Σ_2^* defined by:

$$\begin{cases} \mathscr{P}_{\Sigma_1\to\Sigma_2}(\varepsilon) = \varepsilon, & \\ \mathscr{P}_{\Sigma_1\to\Sigma_2}(\rho.s) = \mathscr{P}_{\Sigma_1\to\Sigma_2}(\rho) & \text{if } \rho \in \Sigma_1^*, s \notin \Sigma_2, \\ \mathscr{P}_{\Sigma_1\to\Sigma_2}(\rho.s) = \mathscr{P}_{\Sigma_1\to\Sigma_2}(\rho).s & \text{if } \rho \in \Sigma_1^*, s \in \Sigma_2. \end{cases}$$

For a given pattern Ω, the diagnosis problem consists in defining an Ω-diagnoser function that takes as input a sequence of observable events and that produces one of the three symbols $\{\Omega{-}\mathit{certain}, \Omega{-}\mathit{safe}, \Omega{-}\mathit{ambiguous}\}$ ([20]).

Definition 7 (Ω-diagnoser). *Let Θ be the model of a system based on the set of events $\Sigma = \Sigma_u \cup \Sigma_o$, an Ω-diagnoser is a function*

$$\Delta_\Omega : \Sigma_o^* \to \{\Omega{-}\mathit{certain}, \Omega{-}\mathit{safe}, \Omega{-}\mathit{ambiguous}\}$$

such that:

- *$\Delta_\Omega(\sigma) = \Omega{-}\mathit{certain}$ if for any run $\rho \in \mathcal{L}(\Theta)$ that is consistent with σ (i.e. $\mathscr{P}_{\Sigma\to\Sigma_o}(\rho) = \sigma$), $\rho \Supset \Omega$;*

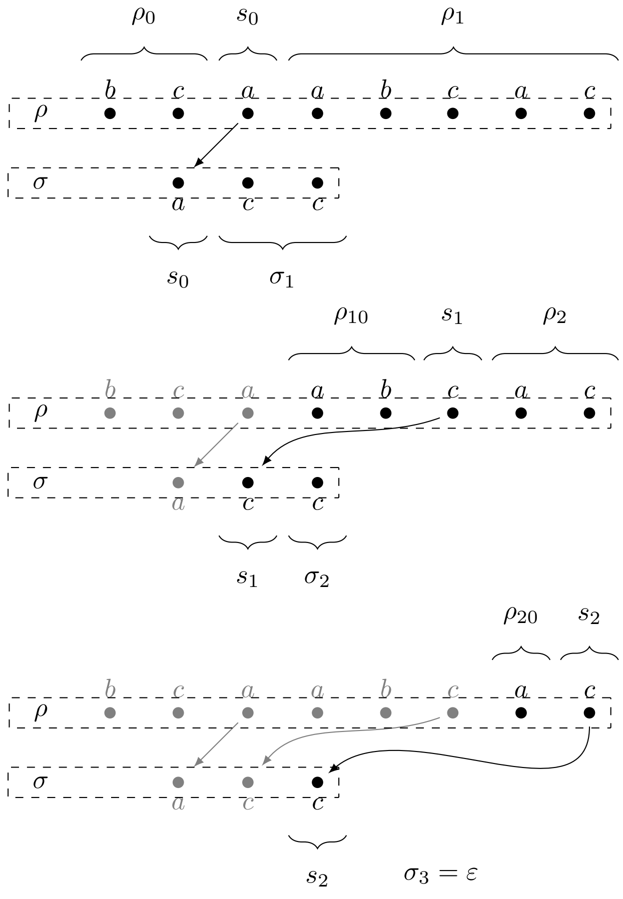

Figure 5: How $\rho = bcaabcac$ matches $\sigma = acc$: $\rho \Supset \sigma$.

- $\Delta_\Omega(\sigma) = \Omega-safe$ *if for any run* $\rho \in \mathcal{L}(\Theta)$ *that is consistent with* σ, $\rho \not\Supset \Omega$*;*
- $\Delta_\Omega(\sigma) = \Omega-ambiguous$ *otherwise.*

Considering now a set of patterns $\Omega_1, \ldots, \Omega_n$, the diagnoser function of a system can be defined as $\Delta : \Sigma_o^* \rightarrow \prod_{i=1}^{n} \{\Omega_i-certain, \Omega_i-safe, \Omega_i-ambiguous\}$ such that $\Delta(\sigma) = (\Delta_{\Omega_1}(\sigma), \ldots, \Delta_{\Omega_n}(\sigma))$.

As an example, consider first the following observable sequence $\sigma_1 = \mathbf{dd}$ from the system Θ_1 (Figure 1) then $\Delta_{\Omega_{a\rightarrow e}}(\sigma_1) = \Omega_{a\rightarrow e}-ambiguous$ and $\Delta_{\Omega_{a\rightarrow b}}(\sigma_1) = \Omega_{a\rightarrow b}-ambiguous$. In both cases, it is possible to find in Θ_1 two runs which are consistent with σ_1 (their observable projection on Σ_o is exactly σ_1), one run matches the pattern but not the other one. For instance consider the run $ba\mathbf{d}eeb\mathbf{d}$ of Θ_1 that matches both the pattern $\Omega_{a\rightarrow e}$ and the pattern $\Omega_{a\rightarrow b}$ and the run $ba\mathbf{d}f\mathbf{d}$ that does not match any of the two patterns. Consider now the observable sequence $\sigma_2 = \mathbf{cddc}$, then $\Delta_{\Omega_{a\rightarrow e}}(\sigma_2) = \Omega_{a\rightarrow e}-safe$ and $\Delta_{\Omega_{a\rightarrow b}}(\sigma_2) = \Omega_{a\rightarrow b}-certain$. There is no run in Θ_1 consistent with σ_2 that contains an event e so none of them can match $\Omega_{a\rightarrow e}$. On the other hand, any run consistent with σ_2 starts with ba and must contain one more b so any of them matches $\Omega_{a\rightarrow b}$.

4 Matching determination

We introduce a specific product operator to combine the system Θ and the pattern Ω in order to retain in the system any run that effectively matches the pattern: this product is called the system-pattern product and is denoted: $\Theta \ltimes \Omega$. Informally speaking, $\Theta \ltimes \Omega$ consists of the union of the places of Θ and Ω. The set of transitions is composed of the set of transitions of Θ and a set of synchronized and prioritized transitions replacing the transitions from Ω.

Definition 8 (system-pattern product)**.** *Let* $\Theta = \langle P_\Theta, T_\Theta, A_\Theta, \succ_\Theta, \ell_\Theta, \Sigma_\Theta, Q_\Theta, M_{\Theta 0}\rangle$*,* $\Omega = \langle P_\Omega, T_\Omega, A_\Omega, \varnothing, \ell_\Omega, \Sigma_\Omega, Q_\Omega, M_{\Omega 0}\rangle$ *be respectively a system and a pattern such that* $P_\Theta \cap P_\Omega = \varnothing$*,* $T_\Theta \cap T_\Omega = \varnothing$ *and* $\Sigma_\Omega \subseteq \Sigma_\Theta$*, the system-pattern product* $\Theta \ltimes \Omega$ *is the LLPPN* $\langle P, T, A, \succ, \ell, \Sigma, Q, M_0\rangle$ *defined as follows.*

- $P = P_\Theta \cup P_\Omega$.
- $T = T_\Theta \cup T_s$ *such that*
  - $T_s = \bigcup_{l \in \Sigma_\Omega} \{t_\Theta \| t_\Omega : \exists t_\Theta \in T_\Theta \wedge \exists t_\Omega \in T_\Omega \wedge \ell_\Theta(t_\Theta) = \ell_\Omega(t_\Omega) = l\}$ *is the set of synchronized transitions, the transition denoted* $t_\Theta \| t_\Omega$ *resulting from the synchronization of* $t_\Theta \in T_\Theta$ *and* $t_\Omega \in T_\Omega$*.*
- $A = A_\Theta \cup A_s$ *such that*

$$\begin{aligned} A_s = \{&(p,t) : p \in P_\Theta, t = t_\Theta \| t_\Omega, (p, t_\Theta) \in A_\Theta\} \\ \cup \{&(p,t) : p \in P_\Omega, t = t_\Theta \| t_\Omega, (p, t_\Omega) \in A_\Omega\} \\ \cup \{&(t,p) : p \in P_\Theta, t = t_\Theta \| t_\Omega, (t_\Theta, p) \in A_\Theta\} \\ \cup \{&(t,p) : p \in P_\Omega, t = t_\Theta \| t_\Omega, (t_\Omega, p) \in A_\Omega\}. \end{aligned}$$

- $\succ = \succ_\Theta \cup \succ_s$*, with:*

$$\begin{aligned} &\succ_s = \{(t_s, t_\Theta) : t_s = t_\Theta \| t_\Omega \in T_s\} \cup \\ &\bigcup_{t_s \in T_s} (\{(t_s, t) : (t_\Theta, t) \in \succ_\Theta\} \cup \{(t, t_s) : (t, t_\Theta) \in \succ_\Theta\}) \\ &\cup \{(t_\Theta^1 \parallel t_\Omega^1, t_\Theta^2 \parallel t_\Omega^2) : t_\Theta^1 \parallel t_\Omega^1 \in T_s, \\ &t_\Theta^2 \parallel t_\Omega^2 \in T_s, (t_\Theta^1, t_\Theta^2) \in \succ_\Theta\}. \end{aligned}$$

- $\forall t = t_\Theta \| t_\Omega \in T_s, \ell(t) = \ell_\Theta(t_\Theta), \forall t \in T \cap T_\Theta,$ $\ell(t) = \ell_\Theta(t)$.
- $\Sigma = \Sigma_\Theta$.
- $Q = \{M_\Theta \cup M_\Omega : M_\Theta \in Q_\Theta \wedge M_\Omega \in Q_\Omega\}$.
- $M_0 = M_{\Theta 0} \cup M_{\Omega 0}$.

Intuitively, the pattern Ω is applied as a filter on the system Θ by using the product $\Theta \ltimes \Omega$. Any transition of label e in the system is synchronized with a transition of the pattern Ω labeled with the same event e. The product is asymmetric in the sense that $\Theta \ltimes \Omega$ contains all the transitions of Θ whereas the transitions of Ω with such an event e have been replaced by a set of synchronized transitions with Θ. Given the current marking of $\Theta \ltimes \Omega$ and the occurrence of a new event e, two cases can happen. If the current marking of $\Theta \ltimes \Omega$ enables a synchronized transition labeled with e, as it is prioritized, it will be triggered (the event e is therefore part of the pattern), otherwise only a transition from the system Θ can be triggered in $\Theta \ltimes \Omega$ (the system keeps running by producing the event e but e is not part of the pattern).

With the help of this product, it is possible to identify in the system Θ the set of runs where the pattern Ω has occurred. The following results assert this intuition.

Let $\Theta_\Omega = \Theta \ltimes \Omega$ and $M_{\Theta_\Omega 0}$ be the initial marking of Θ_Ω. Let $M_{\Theta 0}$ (resp. $M_{\Omega 0}$) be the initial marking of Θ (resp. Ω).

Theorem 1. *Let* Θ *be the LLPPN of a system over the alphabet* Σ *and* Ω *be the LLPPN of a pattern:*

$$\mathcal{L}(\Theta \ltimes \Omega) = \{\rho \in \mathcal{L}(\Theta) : \rho \Supset \Omega\}.$$

Theorem 1 is the fundamental property of the operator $\ltimes$. The set of runs that are characterized by the language $\mathcal{L}(\Theta \ltimes \Omega)$ are exactly the set of runs that match the pattern Ω [15].

5 Implementation by Model-Checking

Through the Petri model of the system-pattern product ($\Theta \ltimes \Omega$), we have a model that determines for each possible run of the system if this evolution matches or not the pattern. This model can then be exploited to solve the diagnosis problem.

For an observable input sequence generated by the system (σ), the pattern diagnosis problem can be solved by determining the intersection between the system's evolutions that generate this input sequence and those that match the pattern. This intersection can be characterized through the synchronized product between the Petri net model of the system-pattern product and the Petri net model O of the observed input sequence (σ) produced by the system : $\Theta_\Omega || O$ [21].

Considering $\sigma = e_1, e_2, \dots e_k$, the Petri net model O is an $LLPPN$ defined by: $O = \langle P_O, T_O, A_O, \succ_O, \ell, \Sigma_O, Q_O, M_{O0}\rangle$ with:

- $P_O = \bigcup_{i=1}^{k+1} \{p_i\}$ and $T_O = \bigcup_{i=1}^{k} \{t_i\}$;
- $A_O = \bigcup_{i=1}^{k} \{(p_i, t_i), (t_i, p_{i+1})\}$
- $\succ_O = \varnothing$
- $\ell(t_i) = e_i, \forall t_i \in T$ and $i \in 1..k$;
- $\Sigma_O = \{e_i \dots e_k\}$;
- $Q_O = \{\{p_{k+1}\}\}$
- $M_{O0} = \{p_1\}$

Once the synchronized product $\Theta_\Omega || O$ built, it is then possible to use model checking techniques to determine the results produced by the Ω-diagnoser. Here we give a complete description of the implementation by Model-Checking.

5.1 Model checking problem

We propose here to implement the Ω-diagnoser function by using the TINA toolkit of [22] and [17] that provides all the necessary tools to directly generate the Kripke structure on which the model-checking problem is defined [23]. TINA actually generates *enriched labeled Kripke structures* from the analyzed Petri nets to check properties written in SE-LTL (State/Event Linear Temporal Logic) which extends LTL in the way presented here above.

A formula ψ is a SE-LTL formula if it is a universally quantified formula

$$\psi ::= \forall \varphi$$

such that

$$\begin{array}{lcl} \varphi & ::= & r \mid \neg\varphi \mid \varphi \vee \varphi \mid\mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \Box\varphi \mid \Diamond\varphi \mid \varphi \mathbf{U} \varphi \\ r & ::= & e \mid e \vartriangle e \\ e & ::= & p \mid a \mid c \mid e \triangledown e \end{array}$$

with p a place symbol, a a transition symbol, $c \in \mathbb{N}$, $\vartriangle \in \{=, <, >, \leq, \geq\}$ and $\triangledown \in \{+, -, *, /\}$. The operators $\bigcirc$ (next), $\Box$ (always), $\Diamond$ (eventually) and $\mathbf{U}$ (until) have their usual LTL semantics.

The Ω-diagnoser is based on two questions written in SE-LTL concerning the Petri net model $\Theta_\Omega || O$:

1. Question $\varphi_{CERTAIN}$: Are we sure that the pattern is recognized in each system's behaviour that produces the observations ? If the result is false the following question is asked to the model checker:
2. Question φ_{SAFE}: Are we sure that the pattern is never recognized in each system's behaviour that produces the observations ?

More formally, each question corresponds to a SE-LTL formula. Given M a marking of $O \parallel \Theta_\Omega$, $M_{|\Omega}$ denotes the restriction of M to Ω i.e. the marking of the pattern's places. In a similar way $M_{|O}$ denotes the restriction of M to O.

1. $\varphi_{CERTAIN} \equiv \Box((M_{|O} \in Q_O) \Rightarrow (M_{|\Omega} \in Q_\Omega))$, in others words, is it always true ($\Box$) that if the system generates ($M_{|O} \in Q_O$) then ($\Rightarrow$) it matches the pattern ($M_{|\Omega} \in Q_\Omega$).
2. $\varphi_{SAFE} \equiv \Box((M_{|O} \in Q_O) \Rightarrow (M_{|\Omega} \notin Q_\Omega))$, n others words, is it always true ($\Box$) that if the system generates ($M_{|O} \in Q_O$) then ($\Rightarrow$) it does not match the pattern ($M_{|\Omega} \notin Q_\Omega$).

The complete and automated implementation of the method to solve the pattern diagnosis problem on a system Θ is:

1. Compute $\Theta_\Omega = \Theta \ltimes \Omega$.
2. Compute O the LLPPN that represents the sequence of observations σ.
3. Compute the synchronisation of Θ_Ω and O: $\Theta_\Omega || O$
4. Compute $\varphi_{CERTAIN}$ from $\Theta_\Omega || O$ and check $\varphi_{CERTAIN}$ on the Kripke structure of $\Theta_\Omega || O$.
5. If the result is false, then compute φ_{SAFE} from $\Theta_\Omega || O$ and check φ_{SAFE} on the Kripke structure of $\Theta_\Omega || O$.
6. $\Delta_\Omega(\sigma)$ returns $\Omega-$*certain* if $\varphi_{CERTAIN}$ is true, else it returns $\Omega-$*safe* if φ_{SAFE}, otherwise it returns $\Omega-$*ambiguous*.

6 Case study

6.1 Description of the system

The case study that we propose in this paper is a product transportation system. It is a two-level system composed of two product sites (namely sites 1 and 2) at level 2 where a set of products is stored and two assembly stations at level 1 that request products from level 2. A lift is used between the two levels. Each site has a conveyor belt to move a product from the site to the lift and each station also has a conveyor belt to get the products from the lift. Figure 6 presents the system model. Once a product is detected on the site 1, a *Product1* signal (noted $Pr1$) is emitted. The product is then put in a box and becomes available. The box is then pushed and sent into the lift (action $Push_1$ that starts with event P_1 and ends with event EP_1). Site 2 behaves in a similar manner. A product from site 1 has a priority access to the lift. In the Petri net model of Figure 6, this priority relation is indicated by a dashed edge between the transitions labeled

Obs.	nS(Γ_1)	nT(Γ_1)	$\varphi^1_{CERTAIN}$?	φ^1_{SAFE}?
Pr1	4	3	F	T
Pr2	7	8	F	T
D	12	13	F	F
U	18	19	F	F
D	23	25	F	F
ELReq2	26	30	F	F
ERReq2	31	37	F	F
ELReq1	32	38	T	F
ERReq1	33	39	T	F
U	34	40	T	F

Table 1: Diagnosis results on the pattern Ω_1.

Obs.	$\varphi^2_{CERT.}$?	φ^2_{SAFE}?	$\varphi^3_{CERT.}$?	φ^3_{SAFE}?
Pr1	F	T	F	T
Pr2	F	T	F	T
D	F	F	F	T
U	T	F	F	F
D	T	F	F	F
ELReq2	T	F	F	F
ERReq2	T	F	F	F
ELReq1	T	F	T	F
ERReq1	T	F	T	F
U	T	F	T	F

Table 2: Diagnosis results on the pattern Ω_2 and Ω_3.

by P_1 and P_2. After a product has been pushed into the lift, the lift goes down (action *Down*) to reach level 1 (signal D detects the end of the *Down* action). At this level the box is directed to the station that makes the request (either request Req_1 or request Req_2). If it is request Req_1, then the box is pushed on the conveyor belt of station 1 (action $PushReq_1$ that starts with event Req_1 and ends with event $EPReq_1$). Then the conveyor belt performs a move-left action ($LeftReq_1$, $ELReq_1$) to deliver the box with the product and a move-right action to go back to initial position ($RightReq_1$, $ERReq_1$). The behaviour of station 2 is similar except that its conveyor belt moves right first ($RightReq_2$, $ERReq_2$) and then moves left ($LeftReq_2$, $ELReq_2$). Once the box has been pushed on a conveyor belt, the lift goes up (action *Up* ending with the emission of signal U) to level 2.

6.2 Monitoring of a single event class

We first present some experiments with a single event pattern Ω_1 as defined in Figure 3 with $k = 1$ and $e = Req_1$. The aim of this first example is to show how the proposed framework is able to diagnose the occurrence of single events as the classical diagnosis methods do. The pattern Ω_1 represents the occurrence of the event Req_1 in the system. Table 1 shows the obtained results for the following scenario:

$$\sigma = Pr1.Pr2.D.U.D.ELReq2.ERReq2.ELReq1$$
$$.ERReq1.U$$

In order to show the evolution of the diagnosis, any line of Table 1 represents one step of the scenario σ (i.e line 1 is *Pr1*, line 2 is *Pr1.Pr2*, etc). The second column shows the number of states in the corresponding Kripke structure and the third column shows the number of transitions. The first two lines show a Ω_1-safe diagnosis. Diagnoses between lines 3 and 7 are $\Omega_1-ambiguous$. Finally lines 8-10 show an $\Omega_1-certain$ diagnosis. These results can be intuitively explained as follows: *Req1* cannot occur before the lift goes down (lines 1-2) and as long as *ELReq1* is not observed there is an ambiguity about the occurrence of *Req1*.

In classical diagnosis methods, events are sometimes partitioned into a set of fault classes. Here, also it is possible to represent this type of event class. With a pattern Ω_2 as represented in Figure 4, it is possible to represent one occurrence of event $e_1 = Req_1$ or one occurrence of event $e_2 = Req_2$ (two transitions t_1 and t_2 only), i.e. an event of class $\{Req_1, Req_2\}$. In this example, the set of accepting markings is composed of the markings $\{M_1, M_2\}$ where $M_1(p_{21}) = 1$ and for any other place p $M_1(p) = 0$ and $M_2(p_{22}) = 1$ and for any other place p $M_2(p) = 0$. In this example, the $\varphi_{CERTAIN}$ question can have the following form: let p_{obs} denote the last place of the sequence of observations.

$$\varphi^2_{CERTAIN} = \Box((p_{obs} = 1) \Rightarrow (p_{21} = 1 \vee p_{22} = 1)).$$

Table 2 shows the diagnosis results (left columns) on the same scenario as above for Ω_2. When the first D is observed one of the Req may silently happen afterwards but not necessary hence the ambiguity. But as soon as we observe U, Ω_2 has certainly occurred.

6.3 Monitoring of the occurrence of multiple events

Another interesting pattern has the same structure as the one of Figure 4, it is the occurrence of both events *Req1* and *Req2* where, as explained in Figure 4, the only accepting marking is $M(p_{21}) = 1$ and $M(p_{22}) = 1$ and $M(p) = 0$ for the other two places p. In this case the $\varphi_{CERTAIN}$ question becomes:

$$\varphi^3_{CERTAIN} = \Box((p_{obs} = 1) \Rightarrow (p_{21} = 1 \wedge p_{22} = 1)).$$

This type of pattern is interesting if we want to supervise that a set of n specific types of event have occurred. Table 2 shows the diagnosis results (right columns) on the same scenario as above for this pattern. The diagnosis becomes certain (line 8) as soon as we both observe *ELReq2* and *ELReq1*.

6.4 Monitoring of a complex behaviour

The previous pattern examples show that the presented framework can model classical diagnosis problems where we look for the occurrence of a set of single events. However the purpose of this framework is to allow for the diagnosis of more complex behaviours. Back to our example, the global function of the system is to transfer items from conveyors at Level 2 to conveyors at Level 1. Figure 6 actually models the behaviour of the system but no single faulty event is present (any event that can happen is normal).

Figure 7 presents a complex behaviour that one might want to diagnose within the system of Figure 6. This pattern $\Omega_3(m, n)$ models the consecutive occurrence of $m = 3$ events of type Req_1 that are not interleaved with occurrence of Req_2 events (the left conveyor keeps having its requests fulfilled while the right conveyor get stuck) and this pattern should occur $n = 2$ times (the pattern has a counter). This pattern is made up of normal events but defines a behaviour

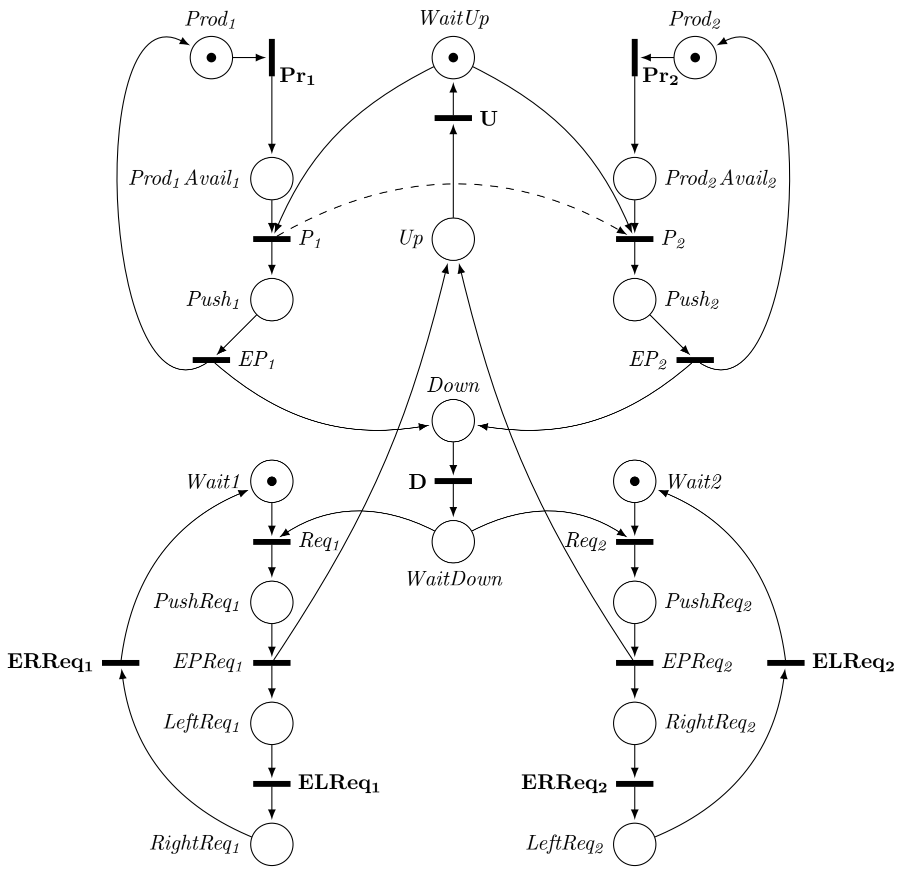

Figure 6: Case study: the product transportation system

Scenario	Size	nb(Req_1)	counter	$nS(\Gamma)$	$nT(\Gamma)$	Time (ms)	Result
S_1	20	5	1	92	107	11	*safe*
S_2	200	5	1	1028	1223	11	*safe*
S_3	2000	5	1	10388	12383	235	*safe*
S_4	5000	5	1	25988	30983	1105	*safe*
S_5	2000	2	10	107534	128379	1993	*ambiguous*
S_6	2000	5	10	10388	12383	228	*safe*
S_7	2000	10	10	10388	12383	239	*safe*
S_8	2000	2	10	7201	8400	156	*certain*
S_9	3500	10	10	17009	20266	577	*certain*
S_{10}	1000	5	2	907987	2155290	15745	*certain*
S_{11}	1000	5	10	1201381	2877901	21018	*certain*
S_{12}	2000	10	8	2737559	6482932	65066	*certain*
S_{13}	2000	20	8	5999626	14245740	244379	*certain*
S_{14}	1750	20	8	4812356	11444801	148629	*safe*

Table 3: Diagnosis results on the pattern Ω_3.

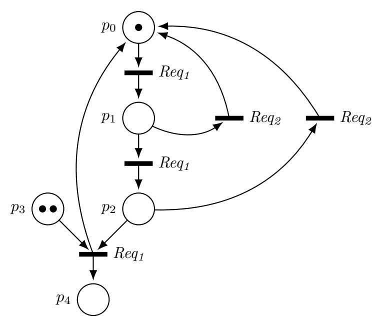

Figure 7: Pattern $\Omega_3(3,2)$: two occurrences of three consecutive requests Req_1: accepting marking M, $M(p_4) = 2$.

that could be unexpected (both conveyors at Level 1 should get items *regularly*).

Table 3 presents some experimental results for a set of scenarios $S_1 \dots S_{14}$ where the occurrence of the pattern $\Omega_3(m,n)$ is diagnosed. For each scenario, Table 3 shows the size of the observable sequence (column 2), the number m of Req_1 events in the pattern $\Omega_3(m,n)$ (column 3) behaviouras well as the number n of occurrences (counter). The size of the involved Kripke structure as a number of states ($nS(\Gamma)$) and transitions ($nT(\Gamma)$) for a given scenario is also given. The time here represents the computation time of all the diagnosis process (the computation of the Kripke structure and the verification of the questions). Scenarios $S_1 \dots S_4$ diagnose the occurrence of $\Omega_3(5,1)$ based on a certain amount of σ cycles (see Section 6.2). The sequence σ being composed of 10 events, S_1 diagnoses 2 cycles of σ, S_2 diagnoses 10 of them, etc. Based on σ, 5 consecutive occurrences of Req_1 can never happen that is why the result is always safe.

Scenarios $S_5 \dots S_7$ diagnoses the occurrence of $\Omega_3(m,10), m = 2,5,10$ so here we attempt to diagnose that m consecutive occurrences of Req_1 happen 10 times. For $m = 5,10$, the result is safe for the same reason as above. For $n = 2$, there is an ambiguity. Indeed, consider two consecutive cycles of σ, once $ELReq1$ is observed in the first cycle, it is possible that one Req_1 has occurred just before $ELReq1$. Within the second cycle, before observing $ELReq2$, one second occurrence of Req_1 may have happened (followed by one occurrence of Req_2), so it is possible (but not certain) that two consecutive occurrences of Req_1 have occurred and, as there are 200 cycles, this can happen 10 times, hence the ambiguous result. Scenario S_8 is similar as S_5 but the observable sequence is a set of cycles σ' where any $ELReq2$, $ERReq2$ in σ have been replaced by $ELReq1, ERReq1$, the diagnosis is therefore certain. Scenario S_9 presents a scenario composed of 100 cycles of σ followed by 75 cycles of σ' followed by 100 cycles of σ followed by 75 cycles of σ' (i.e. 350 cycles): 10 consecutive occurrences of Req_1 happen more than 10 times in S_9. The last five scenarios $S_{10} \dots S_{14}$ rely on the fact that the initial state of the system is unknown (unknown states of the conveyors and the lift). These scenarios illustrate the fact that the method can be applied on an observation window [24] that does not contain the first observations of the system.

7 Conclusion

This paper adresses the problem of patterns diagnosis by a generic framework for pattern modeling and model checking technics. The diagnosis method is designed in the framework of Petri nets (L-type Labeled Prioritized Petri Nets) and it can be viewed as a pattern matching based method. The construction of a Petri net machine capturing the matching of a pattern by the system is done by means of a specific Petri net product. Then a translation of the diagnosis problem into a model-checking problem allows an efficient resolution that is entrusted to the Petri net analyser TINA. The paper reports the results of experimenting this pattern diagnosis approach on an illustrative example. Future work could consolidate some aspects of the method. In particular, the expressivity of patterns that currently generate regular language may be extended to context-free-language. Another research direction is to extend the work to time systems. We currently investigate the use of Time Petri Nets to diagnose more interesting behaviours.

References

[1] Janan Zaytoon and Stéphane Lafortune. Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37:308–320, 2013.

[2] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *Transactions on Automatic Control*, 40(9):1555–1575, 9 1995.

[3] Laurence Rozé and Marie-Odile Cordier. Diagnosing discrete-event systems : extending the diagnoser approach to deal with telecommunication networks. *Journal on Discrete-Event Dynamic Systems : Theory and Applications (JDEDS)*, 12(1):43–81, 2002.

[4] Yannick Pencolé and Marie-Odile Cordier. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, 164(2):121–170, 5 2005.

[5] Gianfranco Lamperti and Marina Zanella. Flexible diagnosis of discrete-event systems by similarity-based reasoning techniques. *Artificial Intelligence*, 170(3):232–297, 2006.

[6] Albert Benveniste, Éric Fabre, Stefan Haar, and Claude Jard. Diagnosis of asynchronous discrete-event systems: a net unfolding approach. *Transactions on Automatic Control*, 48(5):714–727, 5 2003.

[7] Sahika Genc and Stéphane Lafortune. Distributed diagnosis of place-bordered Petri nets. *IEEE Transactions on Automation Science and Engineering*, 4(2):206–219, 2007.

[8] Dimitry Lefebvre and Catherine Delherm. Diagnosis of DES with Petri net models. *IEEE Transaction Automation Science and Engineering*, 4(1):114–118, 2007.

[9] Francesco Basile, Pasquale Chiacchio, and Gianmaria De Tommasi. On K-diagnosability of Petri nets via integer linear programming. *Automatica*, 48(9):2047–2058, 9 2012.

[10] Maria Paola Cabasino, Alessandro Giua, and Carla Seatzu. Diagnosability of discrete-event systems using

labeled Petri nets. *Automation Science and Engineering, IEEE Transactions on*, 11(1):144–153, 2014.

[11] Shengbing Jiang and Ratnesh Kumar. Failure diagnosis of discrete-event systems with linear-time temporal logic specifications. *Transactions on Automatic Control*, 49(6):934–945, 6 2004.

[12] Thierry Jéron, Hervé Marchand, Sophie Pinchinat, and Marie-Odile Cordier. Supervision patterns in discrete event systems diagnosis. In *8th International Workshop on Discrete Event Systems*, pages 262–268, Ann Arbor, MI, United States, 6 2006.

[13] Marco Bozzano, Alessandro Cimatti, Marco Gario, and Stefano Tonetta. Formal design of asynchronous fault detection and identification components using temporal epistemic logic. *Logical Methods in Computer Science*, Volume 11, Issue 4, November 2015.

[14] Marina Zanella and Gianfranco Lamperti. Diagnosis of discrete-event systems by separation of concerns, knowledge compilation, and reuse. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI04)*, pages 838–842, 2004.

[15] Houssam-Eddine Gougam, Yannick Pencolé, and Audine Subias. Diagnosability analysis of patterns on bounded labeled prioritized Petri nets. *Discrete Event Dynamic Systems*, 27(1):143–180, 2017.

[16] Bernard Berthomieu, Florent Peres, and François Vernadat. Bridging the gap between timed automata and bounded time Petri nets. In *4th International Conference Formal Modeling and Analysis of Timed Systems*, pages 82–97, Paris, France, 9 2006.

[17] Bernard Berthomieu, Florent Peres, and François Vernadat. Model-checking bounded prioriterized time Petri nets. In *Automated Technology for Verification and Analysis*, volume 4762 of *LNCS*, pages 523–532. Springer Verlag, 2007.

[18] James L Peterson. Petri nets. *ACM Computing Surveys*, 9(3):223–252, 1977.

[19] Houssam-Eddine Gougam, Audine Subias, and Yannick Pencolé. Supervision patterns: Formal diagnosability checking by Petri net unfolding. In *4th IFAC Workshop on Dependable Control of Discrete Systems*, pages 73–78, York, United Kingdom, 9 2013.

[20] Yannick Pencolé, Anika Schumann, and Dmitry Kamenetsky. Towards low-cost fault diagnosis in large component-based systems. In *6th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, pages 1473–1478, Beijing, China, 8 2006.

[21] Michel Hack. Petri net languages. Technical Report 124, M.I.T. Project MAC, Computation Structures Group, Massachusetts Institute of Technology, 1975.

[22] Bernard Berthomieu, P.-O Ribet, and François Vernadat. The tool tina – construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14):2741–2756, 2004.

[23] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT press, 1999.

[24] Xingyu Su and Alban Grastien. Diagnosis of des by independent windows. In *24th International Workshop on Principles of Diagnosis (DX 2013)*, pages 148–153, 2013.