A New Product Construction for the Diagnosability of Patterns in Time Petri Net

Éric Lubat[1] and Silvano Dal Zilio[1] and Didier Le Botlan[1] and Yannick Pencolé[1] and Audine Subias[1]

Abstract— We propose a method to decide the diagnosability of patterns in labeled Time Petri nets (TPN) that gracefully extends a classic approach for the diagnosability of single faults. Our approach is based on a new technique for computing the language intersection of TPN and on an associated extension of the State Class Graph construction. Our approach has been implemented and we report on some experimental results.

I. INTRODUCTION

Diagnosability is a basic property of Discrete Event Systems that relates to the "observability" of concealed events. Basically, it means that every failure (a distinct instance of unobservable event) can be eventually detected after a finite number of observations. In this work, we are interested in the diagnosability of systems modeled using labeled Time Petri nets (TPN); an extension of Petri nets in which we can associate timing constraints to transitions [1]. This means that we take into account the date at which events are observed and that we want to detect failures in a bounded time. We are also interested in detecting patterns [2] in the occurrence of unobservable events. Our motivation here is to handle a broad class of diagnosis problems in a unified way, such as dealing with permanent or intermittent faults; fault sequences; etc.

We tackle the diagnosability problem using model-checking techniques; which means that we reduce diagnosability to the problem of checking the validity of a temporal logic formula on a finite-state model. This approach relies on a new class of TPN, called Product Time Petri Net (PTPN), that extends a model introduced in [3]. The idea is to enrich Petri nets with a composition operator ($\times$), semantically equivalent to synchronous product, that can be used between transitions with incompatible timing constraints. Basically, given two TPN N and N', the product $N \times N'$ is a net where every "observable event" from N fires synchronously with a similar event from N'; meaning together and at the same date. We show in [3] that it is possible to extend the classical State Class Graph construction to this model. This gives an efficient method for building a finite-state representation of the traces in the intersection of (the timed language of) two or more TPN.

This paper makes several contributions. First, we describe how to use PTPN to decide the diagnosability of a system. Our approach extends the classical techniques based on the twin-plant construction and relies on an LTL model-checker. We also describe a direct, on-the-fly algorithm for

diagnosability. Next, we show that we can define a notion of *pattern diagnosability* as a straightforward extension of the single fault problem. Our approach has been implemented in a tool called Twina [3].

The rest of the paper is organized as follows. In Section II, we discuss some previous works on diagnosability, with a focus on methods that consider Petri nets and their extensions. Section III gives technical details on TPN, Product TPN and their semantics. We describe our method for checking diagnosability in Sect. IV and briefly report on some experimental results before concluding.

II. RELATED WORK

The problem of fault diagnosis was introduced by Sampath et al [4] and is well studied in the context of discrete event systems [5], [6]. The problem has also been studied on timed models, see for instance the work of Tripakis [7] with Time Automata. In these works, diagnosability is often reduced to properties on the *trace language* of systems and their composition. Several work address the problem of fault diagnosis for labeled Petri nets (PN), without time [8]. There are some reasons to justify using PN instead of finite state automata in this context. For example, PN are well-suited for modeling notions such as concurrency or causality in a very compact way; they provide several notions of composition that help compositional reasoning; there is a large choice of tools for their analysis and verification; etc.

A. Diagnosability of Petri Nets

Diagnosability is a decision problem related to fault diagnosis. A system is diagnosable if we can always detect that a fault has occurred using only the record of the observable events (and in a finite number of steps) [8]. There are several efficient methods for checking the diagnosability of single faults in PN [9]. We can roughly divide existing techniques in two groups: (1) those trying to find a *critical pair*; and (2) "diagnoser-based" methods, trying to find executions with *uncertain states*. We explain these terms below.

The idea with *critical pair* is to execute two copies of the system, in lock step on their observable labels. A critical pair is an infinite trace where one copy has a fault and not the other; and a system is diagnosable if it has no critical pairs. The *twin-plant* method of [10] is representative of this group. The drawback of this approach is that we may have more states in the twin-plant than in the system. An advantage is that this method is conceptually simple.

Methods that use *uncertain states* build a state graph (an automaton) that is deterministic and only contains active

states and observable events. Each state is tagged with a diagnosability information: Normal, Certain, or Uncertain; where Uncertain means that we can reach the state both through a fault or without. In this case, a system is diagnosable if every state eventually leads to a Normal or Certain one, see e.g. [11]. A drawback is that we build a deterministic automaton that may be much bigger than the original (non-deterministic) state graph. Something we observe in practice.

B. Diagnosability of Time Petri Nets

Unfortunately, many of the techniques proposed for PN are not suited when we consider time constraints. One problem is that the state space of a TPN is typically infinite when we use a dense time model; that is when time delays can be arbitrarily small. Therefore we need to work on an abstraction of the transition system. A solution to this problem was proposed in [12], where the authors define a state space abstraction based on State Class Graph (SCG). Basically, a state class captures a convex set of constraints on the time at which transitions can fire. This approach is used in several model-checking tools, such as Tina [13].

Another problem with TPN is that we cannot build the composition of two transitions that have non-trivial (meaning different from $[0, \infty)$) time constraints. Therefore, it is not possible to syntactically define a "twin-plant" from a TPN. To solve this problem, we propose to use Product TPN (PTPN), a new model where we can define "groups of transitions" that should fire synchronously.

Some works address the diagnosability of TPN [14], [15], [16], but all rely on some kind of post-processing phase. While they propose substantially different methods, they all rely on a variation of the SCG construction of [12]. The approach in [16] starts by building a Modified SCG that over-approximate the possible (timed) executions. The system is diagnosable if no critical pair is found at his point. Indeed, time can only limit executions, not add new behavior. If a candidate critical pair is found, it is necessary to solve a Linear Programming problems (LPP) to check whether this scenario is feasible. This approach has several limitations. In particular, it may require to solve a large number of LPP. The other approaches are diagnoser-based and suffers from the drawbacks mentioned earlier. In [14], the authors define a notion of Augmented State Class (ASC) graphs, which are SCG with diagnosability information, and use a method to split time intervals in order to only keep deterministic paths in the ASC graph. The interval splitting phase may create a large number of new active states that can lead to a state explosion problem. The approach in [15] relies on a combination of SCG and an enumeration of all the firing sequences between active states.

We propose a direct extension of the twin-plant construction to the case of TPN, without any post-processing of traces. We use this method to decide the single fault diagnosability problem on TPN and show that it can be simply extended to decide the diagnosability of patterns. An advantage of our approach is that we can adapt it easily to several extensions of TPN: priorities between transitions; inhibitor arcs; capacity arcs; etc.

III. TIME PETRI NETS, PRODUCT TPN AND OTHER TECHNICAL BACKGROUND

A *Time Petri Net* (TPN) is a net where each transition, t, is associated with a (static) time interval $\mathbf{I}_s(t)$ that constrains the time at which it can fire. A transition is enabled when there are enough tokens in its input places. Once enabled, transition t can fire if it stays enabled for a duration θ that is in the interval $\mathbf{I}_s(t)$. In this case, t is said *time enabled*.

A TPN is a tuple $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, \mathbf{I}_s \rangle$ in which: $\langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$ is a net (with P and T the set of places and transitions); $\mathbf{Pre}, \mathbf{Post} : T \to P \to \mathbb{N}$ are the precondition and postcondition functions; $m_0 : P \to \mathbb{N}$ is the initial marking; and $\mathbf{I}_s : T \to \mathbb{I}$ is the *static interval function*. We use $\mathbb{I}$ for the set of all possible time intervals. To simplify our presentation, we only consider the case of closed intervals of the form $[l, h]$ or $[l, \infty[$, but our results can be extended to the general case.

We consider that transitions can be tagged using a countable set of labels, $\Sigma = \{a, b, \dots\}$. We also distinguish the special constant ϵ (not in Σ) for internal, silent transitions. In the following, we use a global labeling function $\mathcal{L}$ that associates a unique label in $\Sigma \cup \{\epsilon\}$ to every transition. The alphabet of a net is the collection of labels (in Σ) associated to its transitions. To reason about diagnosability, we also need to partition Σ into two sets: one for observable, the other for unobservable events. We will often use L to refer to the set of observable events.

We propose an extension of TPN in which it is possible to fire several transitions "synchronously". A Product TPN is the composition (N, R) of a net N, with transitions T, and a *(product) relation*, R, that is a collection of *firing sets* $r_1, \dots, r_n$ included in T (hence $R \subseteq \mathcal{P}(T)$, the powerset of T). The idea is that all the transitions in an element r of R should be fired at the exact same time. As a consequence, two transitions in r should have the same labels (we should use $\mathcal{L}(r) = a$ to say they have a common label a) and not interfere with each other (they should not share a common input place).

Definition 1 (PTPN). *A Product TPN (N, R) is the pair of a net $N = \langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, \mathbf{I}_s \rangle$ and a product relation $R \subseteq \mathcal{P}(T)$ such that, for every firing set r in R, transitions in r are independent and compatible: if both t_1 and t_2 are in r then $\mathcal{L}(t_1) = \mathcal{L}(t_2)$ and for every place p in P, $\mathbf{Pre}(t_1)(p) > 0 \Rightarrow \mathbf{Pre}(t_2)(p) = 0$.*

Next, we define the behavior of nets. Apart from the effect of the firing sets, the following definitions are quite standard, see for instance [17], [18].

A. A Semantics Based on Firing Domains

A *marking* m of a net $\langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$ is a mapping $m : P \to \mathbb{N}$ from places to natural numbers. A transition t in T is *enabled* at m if and only if $m \dot{\geqslant} \mathbf{Pre}(t)$, where $\dot{\geqslant}$ is the pointwise comparison between functions.

A *state* of a PTPN is a pair $s = (m, \varphi)$ in which m is a marking, and $\varphi : T \to \mathbb{I}$ is a mapping from transitions to time intervals, also called *firing domains*. Intuitively, if t is enabled at m, then $\varphi(t)$ contains the dates at which t can possibly fire in the future. For instance, when t is newly enabled, it is associated to its static time interval $\varphi(t) = \mathbf{I}_s(t)$. Likewise, a transition t can fire immediately only when 0 is in $\varphi(t)$ and it cannot remain enabled for more than its timespan, *i.e.* the maximal value in $\varphi(t)$.

For a given delay θ in $\mathbb{Q}_{\geq 0}$ and $\iota = [l, h]$ in $\mathbb{I}$, we denote $\iota - \theta$ the time interval ι shifted by θ. This operation is defined only when $\theta \leq h$, in which case $\iota - \theta = [\max(0, l - \theta), h - \theta]$. By extension, we use $\varphi \mathrel{\dot{-}} \theta$ for the partial function that associates the transition t to the value $\varphi(t) - \theta$. This operation is useful to model the effect of elapsed time on a state.

The semantics of a PTPN is a (labeled) Kripke structure, or Time Transition System (TTS), $\langle S, S_0, \to \rangle$, with two possible kinds of actions: either $s \xrightarrow{a} s'$, meaning that a set of transitions t with label a is fired from s; or $s \xrightarrow{\theta} s'$, with $\theta \in \mathbb{Q}_{\geq 0}$, meaning that we let a duration θ elapse from s. A transition t can fire from state (m, φ) if t is enabled at m and firable instantly. When we fire a set of transitions $r = \{t_1, \ldots, t_n\}$ from state (m, φ), a transition k (with $k \neq t$) is said to be *persistent* if k is also enabled in the marking $m - \sum_{t \in r} \mathbf{Pre}(t)$, that is if $m - \sum_{t \in r} \mathbf{Pre}(t) \mathrel{\dot{\geq}} \mathbf{Pre}(k)$. The other transitions enabled after firing r are called *newly enabled*.

Definition 2 (Semantics of PTPN). *The semantics of a PTPN (N, R), with $N = \langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, \mathbf{I}_s \rangle$, is the TTS $\langle S, s_0, \to \rangle$, also denoted $[\![(N, R)]\!]$, where S is the smallest set containing s_0 and closed by $\to$ such that:*
— the initial state is $s_0 = (m_0, \varphi_0)$ where φ_0 is the firing domain such that $\varphi_0(t) = \mathbf{I}_s(t)$ for every t enabled at m_0;
— the state relation $\to \subseteq S \times (\Sigma \cup \{\epsilon\} \cup \mathbb{Q}_{\geq 0}) \times S$ is such that for all state (m, φ) in S

(i) *if $r \in R$ with labels a and t is enabled at m and $0 \in \varphi(t)$ for all $t \in r$, then $(m, \varphi) \xrightarrow{a} (m', \varphi')$ where $m' = m - \sum_{t \in r} \mathbf{Pre}(t) + \sum_{t \in r} \mathbf{Post}(t)$ and φ' is a firing function such that $\varphi'(k) = \varphi(k)$ for any persistent transition and $\varphi'(k) = \mathbf{I}_s(k)$ elsewhere.*
(ii) *if $\varphi(t) - \theta$ is defined for all t enabled at m then $(m, \varphi) \xrightarrow{\theta} (m, \varphi \mathrel{\dot{-}} \theta)$.*

Transitions in the case (i) above are called *discrete*; those labelled with delays (case (ii)) are the *continuous*, or time elapsing, transitions.

A Product TPN (N, R) allows to fire multiple transitions simultaneously, constrained by the relation R. Therefore TPN form a natural subset of PTPN, the one where every firing set has only one transition. More precisely, we can always interpret a TPN N with transitions $\{t_1, \ldots, t_n\}$ as the PTPN (N, R_N), where R_N is the collection of singleton $\{\{t_1\}, \ldots, \{t_n\}\}$. In the following, we often omit the product relation in a PTPN when it is not needed, or obvious from the context. We should also simply use the term net, or the symbol N, to refer to a Product TPN.

B. Executions, Observations and Traces

An *execution* of a net N is a sequence of actions in its semantics, $[\![N]\!]$, that starts from the initial state. It is a time-event word $\alpha_1 \ldots \alpha_n$ over the alphabet containing both labels ($a, b, \cdots \in \Sigma$) and delays ($\theta \in \mathbb{Q}_{\geq 0}$), where we do not record silent transitions. In the following, we simplify executions in order to avoid the occurrence of two successive delays; this can be achieved by defining a structural congruence relation between sequences, $\equiv$, such that $\theta \theta' \equiv (\theta + \theta')$. This means that we can always consider executions of the form $\theta_0 a_0 \theta_1 a_1 \ldots$, that alternate between delays and labels. In this case, we say that the *date* of the event a_k in σ is $\theta_0 + \cdots + \theta_k$.

Given a set of observable labels $L \subseteq \Sigma$, the *L-observation* for an execution $\sigma = \alpha_1 \ldots \alpha_k$ is the sequence $\mathrm{obs}_L(\alpha_1) \ldots \mathrm{obs}_L(\alpha_k)$ such that $\mathrm{obs}_L(\alpha) = \alpha$ when $\alpha \in \mathbb{Q}_{\geq 0} \cup L$ and $\mathrm{obs}_L(\alpha) = 0$ otherwise. Hence $\mathrm{obs}_L(\sigma)$ is an execution that contains only the observable events in σ, in the same order and at the same date than in σ.

By contrast, a *trace* is the untimed word obtained from an execution when we keep only the discrete actions. Then the language of a TPN is the set of all its (finite) traces. By definition, the language of a TPN is prefix-closed; and it is regular when the net is bounded. The State Class Graph construction of [12] provides an effective method for computing a finite representation of the traces in a bounded TPN. We can do the same with Product TPN using the SCG construction defined in a previous work [3].

We can illustrate our definitions with a simple example (see Fig. 1). Executions for the net N_1 (left) are sequences of the form $\theta_0 a \theta_1 b$ (and their prefix), provided that $\theta_1 \geq 1$; the $\{b\}$-observations are of the form θb, with $\theta \geq 1$; and the only maximal trace in N_1 is $a\, b$.

We can define the set of observations that are common to two nets, N_1 and N_2, using a classic product operation between transition systems. Assume $K_1 = \langle S_1, s_1^0, \to_1 \rangle$ and $K_2 = \langle S_2, s_2^0, \to_2 \rangle$ are two TTS and L is a set of (observable) labels. The product of two TTS over L, denoted $K_1 \|_L K_2$, is the TTS $\langle (S_1 \times S_2), (s_1^0, s_2^0), \to \rangle$ such that $\to$ is the smallest relation obeying the following rules:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \quad \alpha \in (\Sigma \setminus L) \cup \{\epsilon\}}{(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2)} \qquad \frac{s_2 \xrightarrow{\alpha}_2 s_2' \quad \alpha \in (\Sigma \setminus L) \cup \{\epsilon\}}{(s_1, s_2) \xrightarrow{\alpha} (s_1, s_2')}$$

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \quad s_2 \xrightarrow{\alpha}_2 s_2' \quad \alpha \in \mathbb{Q}_{\geq 0} \cup L}{(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2')}$$

It is the case that the "common observations" in N_1 and N_2 (relative to L) are exactly the observations in the TTS product $[\![N_1]\!] \|_L [\![N_2]\!]$. This has a direct application when we try to find a critical pair in a TPN N, since it amounts to finding an observation in $[\![N]\!] \|_L [\![N]\!]$ where the first component had an occurrence of a fault and not the second.

Theorem 1. *There is an execution σ in $K_1 \|_L K_2$ if and only if there are two executions, σ_1 in K_1 and σ_2 in K_2, with the same observations: $\mathrm{obs}_L(\sigma) \equiv \mathrm{obs}_L(\sigma_1) \equiv \mathrm{obs}_L(\sigma_2)$.*

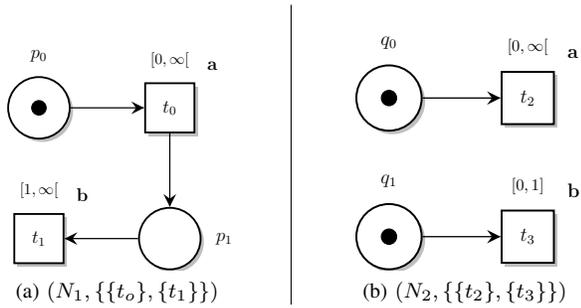(a) $(N_1, \{\{t_o\}, \{t_1\}\})$ (b) $(N_2, \{\{t_2\}, \{t_3\}\})$

Fig. 1: two examples of TPN

Proof (sketch). Given an execution σ from $K_1 \|_L K_2$ we can define its projection over each component; which gives σ_1 and σ_2. Reciprocally, we can always synchronize (over L) pairs of executions that have the same L-observations. $\square$

C. Synchronous Product of PTPN

Given two nets (N_1, R_1) and (N_2, R_2) with disjoint sets of places P_1, P_2 and transitions T_1, T_2, their product $(N_1, R_1) \times_L (N_2, R_2)$ is the PTPN (N, R) where N is the concurrent composition (juxtaposition) of N_1 with N_2, the net $\langle P_1 \cup P_2, T_1 \cup T_2, \mathbf{Pre}, \mathbf{Post}, m_0^1 \uplus m_0^2, \mathbf{I}_s \rangle$ with $\mathbf{Pre}(t)(p) = \mathbf{Pre}_i(t)(p)$ if and only if $t \in T_i$ and $p \in P_i$ with $i \in 1..2$, and 0 otherwise (same with $\mathbf{Post}$); and the product relation R is such that:

$$R = \bigcup_{a \in L} \{r_1 \cup r_2 \mid r_i \in R_i, \mathcal{L}(r_i) = a, i \in 1..2\}$$
$$\cup \bigcup_{a \in \Sigma \setminus L \cup \{\epsilon\}} \{r \mid r \in R_1 \cup R_2, \mathcal{L}(r) = a\}$$

Unlike the conventional *synchronous composition* operator between Petri nets, we do not merge transitions with the same labels but, instead, compose relations. But like with synchronization, our goal is to define an operation that is a *congruence*, meaning that $[\![N_1 \times_L N_2]\!]$ is equivalent to $[\![N_1]\!] \|_L [\![N_2]\!]$.

Theorem 2. *State graph* $[\![(N_1, R_1) \times_L (N_2, R_2)]\!]$ *is isomorphic to* $[\![(N_1, R_1)]\!]\|_L[\![(N_2, R_2)]\!]$.

Proof (sketch). By induction on the shortest path from the initial state, s_0, to a reachable state s in $[\![(N_1, R_1) \times_L (N_2, R_2)]\!]$, then a case analysis on the possible transitions from s. $\square$

We can illustrate how the product can constrain the behavior of nets using the examples of Fig. 1. Executions for N_2 (right) include sequences of the form $\theta_0 \, a \, \theta_1 \, b$ and $\theta_1 \, b \, \theta_0 \, a$, provided that $\theta_1 \leq 1$. The product $N_1 \times_{a,b} N_2$ is a PTPN with two firing sets, $\{t_0, t_2\}$ and $\{t_1, t_3\}$. The resulting net contains only the executions common to both N_1 and N_2: we have $\theta_0 \, a \, \theta_1 \, b$ only if $\theta_0 = 0$ (this is the only possibility to fire $\{t_1, t_3\}$ simultaneously); or we have $\theta_0 \, a \, \theta_1$ with $\theta_0 + \theta_1 \leq 1$. In the latter case, we reach a *time deadlock* at date 1 (a situation where time cannot progress and no transitions can fire). Indeed, at this date t_3 must urgently

fire (its firing domain is $[0, 0]$) but t_1 is not firable yet (its firing domain is $[1 - \theta, \infty[$).

Time deadlocks are important in the context of our work. They model the case of two executions that start with the same observation but that cannot be reconciled after some point; meaning that observable events are enough to eventually discriminate them.

Theorem 2 provides an effective method for checking the diagnosability of a TPN when it is bounded. We describe this method in the next section.

IV. SINGLE FAULT AND PATTERN DIAGNOSABILITY

We can easily define the *twin-plant* construction of a net N as the composition of two copies of N, say $N.1 \times_L N.2$. In the following, we consider that failures are transitions on a common unobservable label, say f. We say that the single fault f is diagnosable when we cannot find a (critical) pair of executions such that: (1) they have the same L-observations; and (2) only one of them eventually exhibits a failure (contains a transition labeled with f).

We use the general assumption that systems are *ultimately observable*; meaning that they do not block and that, on every execution, we always eventually find an observable event after a bounded number of transitions and within a bounded delay (which entails the absence of Zeno traces, like in [7]). Hence the fault f is diagnosable when executions with the same observations both fail, or if they block, meaning we cannot extend them with a compatible event.

By Th. 1, a critical pair in N corresponds to an infinite execution path in $[\![N.1]\!]\|_L[\![N.2]\!]$ where a fault occurs in one copy of N but not the other. We can characterize these executions using Linear temporal Logic (LTL) [19], which will provide an effective method to check for diagnosability. An LTL formula ϕ expresses constraints on the occurrences of events along an execution path. It is built from propositions, logical connectives, and a modality, $\Diamond \phi$, meaning that "property ϕ will eventually hold". We use propositions $f.i$ to denote an occurrence of f in component i, and $dead$ for deadlocks. Hence checking the absence of critical pairs amounts to checking that, on every path, formula $\Diamond(f.1 \vee dead)$ is valid only if $\Diamond(f.2 \vee dead)$ is valid. We can simplify this statement by taking into account the inherent symmetry of the problem.

Theorem 3. *a TPN N is diagnosable if and only if all the maximal executions of the product $N.1 \times_L N.2$ satisfy the LTL formula* $\varphi_D \overset{\text{def}}{=} (\Diamond f.1) \Rightarrow \Diamond(f.2 \vee dead)$.

Proof. Net N is diagnosable in the sense of [7] iff property φ_D is valid on all maximal executions in $[\![N.1]\!]\|_L[\![N.2]\!]$ (there is no infinite execution faulty in $N.1$ and not in $N.2$). The result follows from Th. 2. $\square$

Therefore, to check if N is diagnosable, we can simply generate the SCG for $N.1 \times_L N.2$ then use an LTL model-checker (like the tool Selt provided with Tina) to check property φ_D from Th. 3.
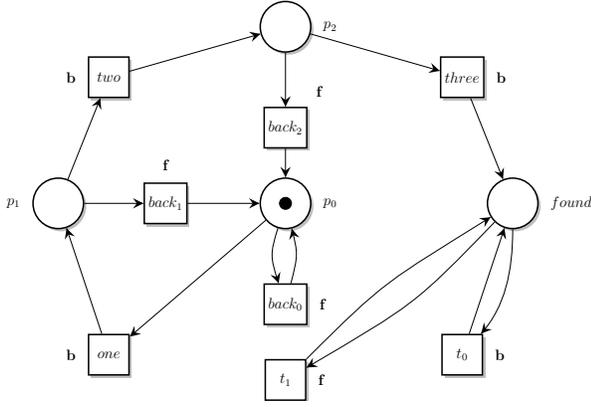
Fig. 2: pattern for "three consecutive b without f"

Fig. 3: marking graph for the pattern in Fig. 2

A. On-the-Fly Algorithm for Diagnosability

We have improved on the indirect method of Th 3 by defining a dedicated decision algorithm for diagnosability that is *on-the-fly* (it avoids computing the whole state space of the system when not necessary) and that has better memory usage. When the system is not diagnosable, we return a counter-example that is a trace in $N.1 \times_L N.2$ corresponding to a critical pair.

We remark that it is not necessary to fire transition $f.2$ to check property φ_D, since every execution that contains $f.2$ satisfies $\Diamond(f.2 \lor dead)$. Then, assuming we never fire $f.2$, a counter example to property $\Diamond(f.2 \lor dead)$ corresponds to a non-trivial Strongly Connected Component (SCC). Based on this idea, we propose an adaptation of Tarjan's SCC Algorithm [20] to "mark" states leading to a cycle (avoiding $f.2$) in the state class graph of $N.1 \times_L N.2$.

Our algorithm generates new nodes (state classes) following a depth-first search order; computes SCC; and mark nodes incrementally. We mark all the nodes in a cycle as soon as we find one. We also mark nodes that can lead to a marked one (either when we fire a transition or when we pop the stack of visited nodes). We report a counter-example for diagnosability as soon as we either: (1) reach a marked node by firing $f.1$; or (2) when we find a SCC containing $f.1$. The net is diagnosable if we finish exploring all the classes. Like with Tarjan's algorithm, the time complexity is linear in the size of the SCG for $N.1 \times_L N.2$.

B. Pattern Diagnosability

Our method can be naturally extended to check for the diagnosability of "patterns of unobservable events" [2]. In our case, a *pattern* M is a special instance of TPN. We denote F the set of labels occurring in M and we distinguish a place in M, say $found$, that is a witness for detection.

We say that pattern M *detects* the execution σ if $\mathrm{obs}_F(\sigma)$, the F-observation of σ, is an execution of M that "marks" place $found$: we can run the execution $\mathrm{obs}_F(\sigma)$ in $[\![M]\!]$ and it satisfies the linear property $\Diamond found$.

More generally, we say that pattern M is *detected* in N when we reach a state where place $found$ is marked in $N \times_F$
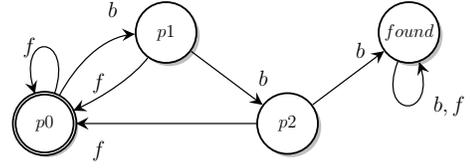
M. For instance, the pattern in Fig. 2 detects executions that have three consecutive occurrence of b without any f in-between. This can be inferred from the marking graph of the pattern (see Fig. 3).

Instead of defining a pattern as a regular language [2], or as a set of timed sequences [21], [22], we use a (prefix-closed) set of executions in $[\![M]\!]$. Hence we restrict ourselves to *time regular* languages, meaning sets of executions that can be "realized" with a TPN. This is enough to model every regular set of (untimed) traces.

We also want to make sure that a pattern does not interfere with the system it interacts with. For example, it should not prevent some executions of the system. To this end, we impose three well-formedness conditions on patterns:

1) patterns are *total*; they should always allow transitions on the labels in F, at any time (they never block or delay a transition):
2) patterns are *deterministic* (the same observations should lead to the same states)
3) labels in F are unobservable ($F \cap L = \emptyset$)

Constraints (1) and (2) can be expressed as a property over all states in $[\![M]\!]$, namely $\forall s \in [\![M]\!], a \in F, \theta \in \mathbb{Q}_{\geq 0}.\exists!t \in T, s', s'' \in [\![M]\!].(\mathcal{L}(t) = a \land s \xrightarrow{\theta} s' \xrightarrow{t} s'')$

By analogy with our previous definition of diagnosability, we say that pattern M is diagnosable if it is not possible to find a (critical) pair of executions such that M is found in one but never in the other. We can again reduce this question to a model-checking problem on a twin-plant; this time on the product of the system with the pattern.

Theorem 4. *Given a well-formed pattern M, with labels F, the net N is diagnosable for pattern M if and only if all the maximal executions of the product $(N.1 \times_F M.1) \times_L (N.2 \times_F M.2)$ satisfy $(\Diamond found.1) \Rightarrow \Diamond (found.2 \lor dead)$.*

Proof. By Th. 1, since M is total, the pattern is detected for the execution σ of N iff there is an (equivalent) execution σ_M in $[\![N]\!]\|_F[\![M]\!]$ and property $\Diamond found$ is valid for σ_M. Since M is deterministic, σ_M is unique, so it is not possible to find another execution, compatible with σ, where $found$ is not marked. Hence we are left with checking the diagnosability of the event $found$ in the net $N \times_F M$. The result follows from Th. 3 and the fact that $F \cap L = \emptyset$. $\qquad \square$

We can relax some of the well-formedness constraints in the proof of Theorem 4. For instance, we can replace "deterministic" with the weaker property that "detection is unambiguous": it is not possible to find an execution in M that leads to two markings, one where $found$ is marked, the

other not. Nonetheless, this presentation has some merits. For instance each condition can be checked automatically on the marking graph of M when the net is bounded and has no timing constraints.

V. Implementation and Experimental Results

We have extended our tool Twina with support for the declaration of product relations and for the construction of "twin machines". See https://projects.laas.fr/twina/ for installing the tool and for information necessary to reproduce our experiments.

Twina provides an option, -twin, to build the state class graph for the "twin-product" of a net given a fault label. The result can be used as input of the Selt model-checker, which is part of the Tina toolbox. We also provide an option, -diag, for testing diagnosability of single faults using our on-the-fly algorithm of Sect. IV-A. When the system is not diagnosable, it is possible to print a counter-example using the verbose output mode, option -v.

We provide several use cases on the tool website. We only report some of our results here due to space limitations.

One of our biggest model is a version of the WODES diagnosis benchmark of Giua [23], also used in [21], with the addition of timing constraints. This instance is not diagnosable. Diagnosability with the (indirect) approach from Th 3 takes about one hour of computation and generates almost 40 million classes. We return a result in about 5 s with option -diag (using the same computer) after computing less than 200 000 classes. The counter-example found is of length 63. We are also able to check the diagnosability of the examples taken from [15], [21] and obtain the same verdicts.

To check the diagnosability of a net, N, with respect to a F-pattern M, it is enough to compute the SCG for a system build from the "twin-plant" product of $N \times_F M$ and then use a model-checker to check property ϕ_D of Th. 4. Among other benchmarks, we have experimented this approach with an instance of the pattern in Fig. 2, on a timed version of the product transportation system found in [21]. The untimed system is not diagnosable (and it state graph has 14 270 markings). By varying the timing constraints, we can switch from a diagnosable behavior to an un-diagnosable one. The size of the resulting SCG vary from about 2 000 classes to almost 130 000 when the system is not diagnosable. In all these examples, computation time is less than a few seconds.

VI. Conclusion

We propose a constructive and unified approach for deciding the diagnosability of single failures and patterns in TPN. Our presentation emphasizes the connections between these two properties. For future work, we hope to benefit from this relationship in order to study more elaborate observability properties; for example a notion of opacity for patterns.

As such, our approach can be naturally extended to the diagnosability of timed pattern. Unfortunately, it was not possible to delve into this question due to space limitations. In this context, proving that the pattern is total and deterministic may be complex; and may require to consider priorities between transitions. Our approach could also be adapted to check for k-diagnosability (by limiting the number of observable events after a fault) or for Δ-diagnosability, by composing our twin-plant model with an "observer" that triggers an alarm if it is possible to let a delay Δ elapse after the first occurrence of a fault. We plan to study these questions in a future work.

References

[1] P. M. Merlin, "A study of the recoverability of computing systems," Ph.D. dissertation, University of California, Irvine, 1974.

[2] T. Jéron, H. Marchand, S. Pinchinat, and M.-O. Cordier, "Supervision patterns in discrete event systems diagnosis," in *International Workshop on Discrete Event Systems*, 2006.

[3] É. Lubat, S. Dal Zilio, D. L. Botlan, Y. Pencolé, and A. Subias, "A state class construction for computing the intersection of time Petri nets languages," in *Formal Modeling and Analysis of Timed Systems (FORMATS)*, ser. LNCS, vol. 11750. Springer, 2019.

[4] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems," *IEEE Transactions on automatic control*, vol. 40, no. 9, 1995.

[5] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Springer-Verlag, 2009.

[6] J. Zaytoon and S. Lafortune, "Overview of fault diagnosis methods for discrete event systems," *Annual Reviews in Control*, vol. 37, 2013.

[7] S. Tripakis, "Fault diagnosis for Timed Automata," in *7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, 2002.

[8] F. Basile, "Overview of fault diagnosis methods based on Petri net models," in *2014 European Control Conference (ECC)*, June 2014.

[9] M. P. Cabasino, A. Giua, S. Lafortune, and C. Seatzu, "A new approach for diagnosability analysis of Petri nets using verifier nets," *IEEE Trans. Automat. Contr.*, vol. 57, no. 12, 2012.

[10] S. Jiang, Z. Huang, V. Chandra, and R. Kumar, "A polynomial algorithm for testing diagnosability of discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 46, no. 8, 2001.

[11] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems," *Transactions on Automatic Control*, vol. 40, no. 9, 9 1995.

[12] B. Berthomieu and M. Menasche, "An enumerative approach for analyzing time Petri nets," in *Proceedings IFIP*, 1983.

[13] B. Berthomieu, P.-O. Ribet, and F. Vernadat, "The tool TINA–construction of abstract state spaces for Petri nets and time Petri nets," *International Journal of Production Research*, vol. 42, no. 14, 2004.

[14] B. Liu, M. Ghazel, and A. Toguyeni, "Diagnosis of labeled time Petri nets using time interval splitting," *IFAC Proceedings Volumes*, vol. 47, no. 3, 2014.

[15] X. Wang, C. Mahulea, and M. Silva, "Diagnosis of time Petri nets using fault diagnosis graph," *IEEE Transactions on Automatic Control*, vol. 60, no. 9, 2015.

[16] F. Basile, M. P. Cabasino, and C. Seatzu, "Diagnosability analysis of labeled time Petri net systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 3, 2017.

[17] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux, "Comparison of the expressiveness of timed automata and time Petri nets," in *Formal Modeling and Analysis of Timed Systems (FORMATS)*, ser. LNCS, vol. 3829. Springer, 2005.

[18] B. Berthomieu, F. Peres, and F. Vernadat, "Bridging the gap between timed automata and bounded time Petri nets," in *Formal Modeling and Analysis of Timed Systems (FORMATS)*, ser. LNCS, vol. 4202. Springer, 2006.

[19] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (SFCS)*. IEEE, 1977.

[20] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, 1972.

[21] H.-E. Gougam, Y. Pencolé, and A. Subias, "Diagnosability analysis of patterns on bounded labeled prioritized Petri nets," *Discrete Event Dynamic Systems*, vol. 27, no. 1, 2017.

[22] Y. Pencolé and A. Subias, "Timed pattern diagnosis in timed workflows: a model checking approach," *IFAC-PapersOnLine*, vol. 51, no. 7, 2018.

[23] A. Giua, "A benchmark for diagnosis," in *Benchmark Session of WODES'08 Int. Workshop on Discrete Event Systems*, 2007.