






# Aligning Event Logs to Resource-Constrained $\nu$ -Petri Nets

Dominique Sommers<sup>(✉)</sup>, Natalia Sidorova, and Boudewijn van Dongen

Department of Mathematics and Computer Science, Eindhoven University  
of Technology, Eindhoven, The Netherlands  
{d.sommers,n.sidorova,b.f.v.dongen}@tue.nl

**Abstract.** Systems with shared resources can be modeled and analyzed using high-level Petri nets in a natural way. Choosing a model type suitable for the use in conformance checking introduces challenges related to constraints the model should put on resource types and resource instances. In this paper, we propose a model for systems with shared resources based on resource-constrained Petri nets and  $\nu$ -Petri nets that can be used in the context of conformance checking. Our model allows for case and resource isolation, allowing for proper simulation of multiple cases involving shared resources.

With this minimal extension, we show that we can use existing state-of-the-art conformance checking techniques to compute alignments on complete event logs rather than on individual case instances. We show that previously undetected deviations caused by inter-case dependencies can now be exposed, providing valuable information regarding the exhaustive workflow in the process.

**Keywords:** Petri nets · Shared resources · Conformance checking · Inter-case dependencies

## 1 Introduction

Process models often include descriptions of resources executing activities within the process, since the availability of resources puts constraints on the process execution. Event logs, recording process executions, also often include indications which resource executed which activity and when, usually mentioning the exact person(s) or machine(s) that were involved in the activity. Such event logs can be used for conformance checking, i.e. checking whether and where the actual process behavior recorded in an event log deviates from the behavior prescribed by a process model.

Various types of conformance checking techniques that exist so far are primarily focused on the control flow of a process, without taking into account

---

This work is done within the project “Certification of production process quality through Artificial Intelligence (CERTIF-AI)”, funded by NWO (project number: 17998).

the information about resources, and do so by looking at individual cases going through the process separately. In the context of resources, it is crucial to consider all the cases going through the system at the same time, since these cases share resources available in the system.

Another challenge in the conformance checking of processes with shared resources is the necessity to consider not only resource types (e.g. whether it is a doctor or a nurse who has to perform a particular activity), but also the resource identity (e.g. which doctor performed a surgery). This information is critical for checking resource related constraints that should be imposed by the model, e.g. that the patient had a follow-up appointment with the doctor who conducted the surgery, or that the second-opinion appointment is not planned with the same doctor whom the patient already met.

In this paper, we first address the question how to model resource-constrained processes in order to enable conformance checking and then adapt a conformance checking method to dealing with resource-constrained processes. We build our model on basis of resource-constrained Petri nets [31] and  $\nu$ -Petri nets [22]. The model allows to specify resource types by using resource places, and case and resource identities by using case ids and resource ids as token colors. We use the alignment mechanism introduced in [1] as basis for our alignment method to do conformance checking on resource-constrained  $\nu$ -Petri nets.

**Related Work.** In [3] and [31], Petri nets are extended with resources to model availability of durable resources, as well as their claims and releases by cases running through the system.  $\nu$ -Petri nets [22] allow for case isolation as a minimal extension to classical Petri nets via name creation and name management. An advantage as opposed to more advanced Petri net extensions is that coverability and termination are decidable for  $\nu$ -Petri nets.

Other extensions such as Catalog Petri nets [10], synchronizing proclat models [9], resource and instance-aware workflow nets (RIAW-nets) [17], and DB-nets [18] inherit the functionality of  $\nu$ -Petri nets. Additionally, these extensions implement concepts from databases, shared resources, and proclat channels. We show that we do not require such additional functionality and aim for a minimal extension on Petri nets.

Many conformance checking techniques use alignments to directly connect the behavior of a system recorded in a log with the behavior allowed by a process model. Alignments can expose exactly where the recorded behavior and the model agree, which activities prescribed by the model are missing in the log and which log activities should not be performed according to the model [6, 28]. Rule checking techniques [14, 25] are conformance checking techniques that check if specific business rules are respected, and they can be useful in case the process model does not describe the whole process behavior. Case-replay techniques [4, 24, 28, 29] aim to identify specific deviations between modeled and observed behavior. We choose alignments as basis for our conformance checking method since they are designed for fully-specified processes (potentially with

invisible transitions) and target at discovering a broad range of deviations in the process behavior.

Conformance checking usually targets isolated cases from the workflow perspective. More advanced techniques consider resources and data on top of the control flow; in [7], the control flow is considered first, after which other perspectives are checked. This method can provide misleading results in case of shared resources, since resources put additional constraints on the control flow. In [15], this is partially mitigated by balancing the different perspectives in a customizable manner. More recently, a technique was proposed to consider all perspectives at once [16], but cases are still considered individually, also when they are run in parallel, and tokens are uncolored, making it impossible to capture resource ids in the model.

Alignments, as well as the other techniques, are computed primarily focusing on the detection of workflow deviations for individual cases. Work has been done to take into account multiple perspectives like data attributes and resources to check, besides the workflow, whether the correct data attributes and resources were involved [2, 7, 15, 16]. However, they still operate on a case-by-case basis. With resource-constrained Petri nets, violations regarding inter-case dependencies remain undetected.

**Outline.** This paper is organized as follows. Section 2 presents basic definitions related to Petri nets and event logs. In Sect. 3, we focus on processes with shared resources, introduce the notion of resource-constrained  $\nu$ -Petri nets and some modeling patterns. In Sect. 4, we steer towards the problem of conformance checking and investigate the missing link with inter-case dependencies caused by shared resources. In Sect. 5, we propose a solution exploiting  $\nu$ -Petri nets to compute alignments which allows for exposing violations. We conclude in Sect. 6 by discussing our contributions and directions for future work.

## 2 Preliminaries

In this section we present the notations that we will use throughout the paper.

### 2.1 Petri Nets

Petri nets can be used as a tool for the representation, validation and verification of workflow processes to provide insights in how the process behaves [21].

**Definition 1 (Multiset).** A multiset  $m$  over a set  $X$  is  $m : X \rightarrow \mathbb{N}$ , denoted as  $X^\oplus$ . The support  $\text{supp}(m)$  of a multiset  $m$  is the set  $\{x \in X \mid m(x) > 0\}$ .

For  $m_1, m_2 \in \mathbb{N}$ , we write  $m_1 \leq m_2$  if  $\forall x \in X : m_1(x) \leq m_2(x)$ , and  $m_1 < m_2$  if  $m_1 \leq m_2 \wedge m_1 \neq m_2$ . We define  $m_1 + m_2$  as  $(m_1 + m_2)(x) = m_1(x) + m_2(x)$  for all  $x \in X$ . For  $m_1 \geq m_2$ , we define  $m_1 - m_2$  as  $(m_1 - m_2)(x) = m_1(x) - m_2(x)$  for all  $x \in X$ .

In some cases we consider multisets over a set  $X$  as vectors of length  $|X|$ , where we assume arbitrary but fixed orderings of elements of  $X$ .

**Definition 2** (*Petri net*). A Petri Net [19] is a 3-tuple  $N = \langle P, T, \mathcal{F} \rangle$ , where  $P$  is the set of places,  $T$  is the set of transitions,  $P \cap T = \emptyset$ ,  $\mathcal{F} : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$  is the flow of the net. The incidence matrix  $F$  of a Petri net  $N$  is a matrix with a row for each place  $p \in P$  and a column for each transition  $t \in T$  and it is defined by  $F(p, t) = \mathcal{F}(t, p) - \mathcal{F}(p, t)$ .

We write  $P(N)$ ,  $T(N)$  and  $\mathcal{F}(N)$  to indicate that we refer to the set of places, the set of transitions and the flow relation of a net  $N$ .

$N_1 \cap N_2$ ,  $N_1 \cup N_2$ , and  $N_1 \subseteq N_2$  denote intersection, union, and subsets of nets, respectively, defined on the sets of nodes and arcs of  $N_1$  and  $N_2$ .

A labeled Petri net  $N = \langle P, T, \mathcal{F}, \ell \rangle$  additionally defines a labeling  $\ell : T \rightarrow \Sigma^\perp = \Sigma \cup \{\tau\}$  assigning each transition  $t$  a label  $\ell(t)$  from alphabet  $\Sigma$  or  $\ell(t) = \tau$  for silent transitions. We assume that the intersection, union and subsets are only defined for two labeled Petri nets  $N_1, N_2$  where  $\ell_1(t) = \ell_2(t)$  for any transition  $t \in T_1 \cap T_2$ .

**Definition 3** (*Post-set, Pre-set*). Given a transition  $t \in T$ , its pre-set  $\bullet t$  and post-set  $t^\bullet$  are multisets defined as follows:  $\bullet t(p) = \mathcal{F}(p, t)$  and  $t^\bullet(p) = \mathcal{F}(t, p)$  for  $p \in P$ . Correspondingly, for a place  $p \in P$  we have  $\bullet p(t) = \mathcal{F}(t, p)$  and  $p^\bullet(t) = \mathcal{F}(p, t)$  for  $t \in T$ .

**Definition 4** (*Marking*). A marking  $m : P \rightarrow \mathbb{N}$  of Petri net  $N = \langle P, T, \mathcal{F} \rangle$  assigns how many tokens each place contains. A marking defines the state of  $N$ .

**Definition 5** (*Enabling and firing of transitions, Reachable markings*). A transition  $t \in T$  is enabled for firing if and only if  $m \geq \bullet t$ . We denote the firing of  $t$  by  $m \xrightarrow{t} m'$ , where  $m'$  is the resulting marking after firing  $t$  and is defined by  $m' = m - \bullet t + t^\bullet$ . For a transition sequence  $\sigma = \langle t_1, \dots, t_m \rangle$  we write  $m \xrightarrow{\sigma} m'$  to denote the consecutive firing of transitions  $t_1$  to  $t_m$ . We also write  $m \xrightarrow{*} m'$  if there is some  $\sigma \in T^*$  such that  $m \xrightarrow{\sigma} m'$ .

The set of reachable markings  $\mathcal{R}(N, m)$  from marking  $m$  in a Petri net  $N$  is the set  $\{m' \mid m \xrightarrow{*} m'\}$ .

**Definition 6** (*Place invariant*). A place invariant [12] is a row vector  $I : P \rightarrow \mathbb{Q}$  such that  $I \cdot F = 0$ . We denote the set of all place invariants as  $\mathcal{I}_N$ , which is a linear subspace of  $\mathbb{Q}^P$ .

The main property of place invariants is that for any two markings  $m_1, m_2$  such that  $m_1 \xrightarrow{*} m_2$  and any place invariant  $I$  holds:  $I \cdot m_1 = I \cdot m_2$ .

**Definition 7** (*Distributed run*). A distributed run describes a partial order of transition occurrences represented as an acyclic occurrence net  $\pi$  [20]. An occurrence net  $\pi = \langle B, E, G \rangle$  is a Petri net where each place  $b \in B$  is called a condition, each  $e \in E$  is called an event, the transitive closure  $G^+$  is acyclic. Each  $b \in B$  has at most one pre-event and at most one post-event, i.e.  $|\bullet b| \leq 1$  and

$|b^\bullet| \leq 1$ . A labeled occurrence net  $\pi = \langle B, E, G, \ell \rangle$  is an unfolding of a Petri net  $N$  where each condition (event) is labeled with a set of labels of the form  $(x, id)$  where  $x$  refers to a place (transition) of  $N$ , and  $id$  is an instance identifier.

**Definition 8** (*Net system, Language, Execution sequence*). A Net system is a tuple  $SN = (N, m_i, m_f)$ , where  $N$  is a Petri Net and  $m_i$  and  $m_f$  are respectively the initial and final marking. The language of  $SN$  is the set  $\mathcal{L}(SN) = \{\sigma \in T^* \mid m_i \xrightarrow{\sigma} m_f\}$  of all full firing sequences of  $SN$ .

An execution sequence in a net system  $SN = (N, m_i, m_f)$  is a distributed run of steps, starting at the initial marking  $m_i$  and ending at the final marking  $m_f$ .

## 2.2 Event Logs

An event log records action executions as events where each event records at least the action that occurred, the time of occurrence and the case identifier of the case in which the action occurred. Often resources are also recorded as event attributes, e.g. the actors executing the action. Typically, there are several types of resources, and it is generally known beforehand which resources of which types are involved in which actions.

**Definition 9** (*Cases, Resources*).  $Id_c$  denotes the set of case identifiers. An identifier of case  $c$  is denoted as  $id_c$ .

$R = \{r_1, \dots, r_m\}$  is the set of resource types. Each resource instance with an identifier  $id_r$  belongs to some resource type  $r \in R$ .  $Id_r$  denotes the set of resource instances of type  $r \in R$ . We assume that  $Id_r \cap Id_{r'} = \emptyset$  for any  $r \neq r'$ .  $Id_R = \bigcup_{r \in R} Id_r$  denotes the set of the resource instances of all types.

Note that if one would want to capture resource instances with multiple types  $R' \subseteq R$ , a new type should be constructed containing all types from  $R'$ .

With the notation on cases, resources and resource types, we can define events, an event log and its traces in an abstract manner:

**Definition 10** (*Event, Event log, Trace*). An event  $e$  is a tuple  $\langle a, ts, id_c, Id_\rho \rangle$ , with an activity name  $a \in \Sigma$ , a timestamp  $ts$ , a case identifier  $id_c$  and a set of resource instance identifiers  $Id_\rho \subseteq Id_R$ . Such an event represents that activity  $a$  occurred at timestamp  $ts$  for case  $id_c$  and is executed by resource instances from  $Id_\rho$  belonging to possibly different resource types.

An event log  $L$  is a (partially) ordered set of events. These events can be split into traces, defined as projections e.g. on the case identifiers or on the resource identifiers.

For a process modeled by a Petri net, an activity name corresponds to a transition name or a transition label of the corresponding transition of a (labeled) Petri net. With the projection on case identifiers, we get the events from individual cases, as is mainly used in classical process mining and with projection on resource identifiers, we can get the events from individual resource instances, which are entities in their own right, providing the perspective of a single or multiple resource types.

### 3 Modeling Resource-Constrained Case Handling Systems

A classical Petri net models a process execution using transition firings and the corresponding changes of markings without making distinction between different cases on which the modeled system works simultaneously. To create a case view, Workflow nets [27] model processes from the perspective of a single case. Systems in which cases share resources need to be modeled in a different way, providing information both about cases and resources. In this paper, we extend the notion of RCWF-nets [31], resource-constrained workflow nets with resource places and id-tokens identifying cases, by loosening some structural restrictions on Petri nets and including information about resource instances working on cases. To achieve that, we make use of  $\nu$ -Petri nets [22].

#### 3.1 Resource-Constrained Petri Nets

Let  $R$  be the set of all resource types. Following the definition of [31], we model each resource type  $r \in R$  by a place  $p_r$ , where the resources (tokens) are located when they are available. We extend the RCWF-nets definition by adding a place  $\bar{p}_r$  for each resource type  $r$ . Tokens on  $\bar{p}_r$  represent resources working on cases. The structural condition  $\mathcal{F}(p_r, t) + \mathcal{F}(\bar{p}_r, t) = 0$  is imposed on the net, which implies that a token can e.g. be moved from  $p_r$  to  $\bar{p}_r$  to show that the resource gets occupied, moved from  $\bar{p}_r$  to  $p_r$  to show that the resource becomes available, or there could be tests whether there are free/occupied resources.

We consider *durable* resources only, meaning that resources can neither be created nor destroyed, so in the corresponding net system with initial and final marking  $m_i$  and  $m_f$ ,  $m_i(p_r) = m_f(p_r)$  and  $m_i(\bar{p}_r) = m_f(\bar{p}_r)$ , for any resource type  $r \in R$ . The net obtained from a resource-constrained net  $N$  by removing all resource places  $P_r$  together with their incoming and outgoing arcs is called the *production net* of  $N$ .

$T_r \subseteq T$  denotes the set of activities in which resource instances of type  $r$  are involved. We define  $T_r^{in} = p_r^\bullet$  and  $T_r^{out} = \bullet p_r$ , where resource instances of type  $r$  are claimed and released, respectively. Note that both  $T_r \setminus (T_r^{in} \cup T_r^{out})$  and  $T_r^{in} \cap T_r^{out}$  may be nonempty, since a resource can be claimed by an activity and then released only after executing several other activities, or it can be claimed and immediately released by an activity.

We add modeling restrictions on  $p_r$  and  $\bar{p}_r$  to exploit structural characteristics of the Petri net later in Sect. 3.3.

**Definition 11** (*Resource-constrained net system*). *Let  $R$  be a set of resource types. We define the set of availability resource places  $P_r = \{p_r \mid r \in R\}$  and the set of occupancy resource places  $\bar{P}_r = \{\bar{p}_r \mid r \in R\}$ . A resource-constrained net system  $SN = (N, m_i, m_f)$  is a regular net system with resource-constrained*

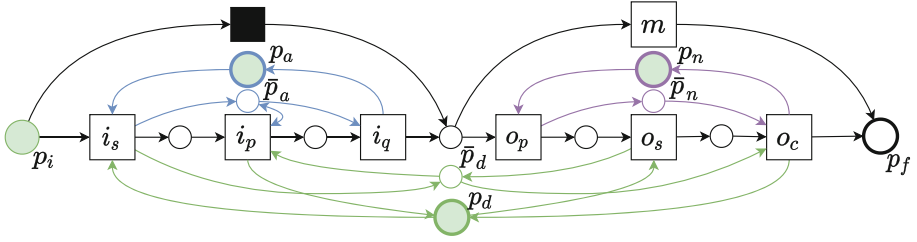


Fig. 1. Running example Petri net.

Petri net  $N = \langle P, T, \mathcal{F} \rangle$  where  $P = P_p \uplus P_r \uplus \bar{P}_r$ , with  $P_p$  the production places. We have the following modeling restrictions on  $p_r$  and  $\bar{p}_r$ :

1.  $\forall t \in T [\bullet t(p_r) + \bullet t(\bar{p}_r) = t^\bullet(p_r) + t^\bullet(\bar{p}_r)]$ , i.e.  $\mathcal{F}(p_r, t) + \mathcal{F}(\bar{p}_r, t) = 0$ .
2.  $m_i(p_r) = m_f(p_r)$  and  $m_i(\bar{p}_r) = m_f(\bar{p}_r) = 0$ ;

Restriction 1 from Definition 11 enforces the place invariant (1, 1) for each pair of the availability and occupancy resource places  $p_r$  and  $\bar{p}_r$ , which trivially follows from the definition of place invariants. This implies that  $m(p_r) + m(\bar{p}_r) = m_i(p_r)$  for any marking  $m$  reachable from the initial marking  $m_i$ . Restriction 2 requires that all resource tokens are returned to the availability resource place when the net reaches its final marking.

Typically, a variant of the soundness property is imposed on the net system to guarantee that the final marking is reachable from any marking reachable from the initial marking.

### 3.2 Running Example

As a running example, we use the Petri net representation of a simple process, see Fig. 1. This Petri net models a hospital process in which three types of resources are involved: doctors (modeled with resource places  $p_d$  and  $\bar{p}_d$ ), doctor assistants (places  $p_a$  and  $\bar{p}_a$ ) and nurses (places  $p_n$  and  $\bar{p}_n$ ). Patients undergo two phases of a treatment. The first phase is the intake where a doctor together with an assistant first discuss patient symptoms (transition  $i_s$ ), after which the doctor provides the plan of approach (transition  $i_p$ ), and finally the patient asks questions to the assistant (transition  $i_q$ ). In case of emergency, the whole intake phase can be skipped, which is modeled by the black (silent) transition. The second phase is either medication collection (transition  $m$ ) or operation. The latter is subdivided in preparation (transition  $o_p$ ) done by a nurse after which the surgery (transition  $o_s$ ) and close up (transition  $o_c$ ) are performed by the nurse and a doctor, ending the process.

An assistant is actively involved in the whole intake phase of the process, which is emphasized by the test arc between the place  $p_a$  and transition  $i_p$ . Note that the nurse is not involved in the surgery, although it is not released during the entire operation phase.

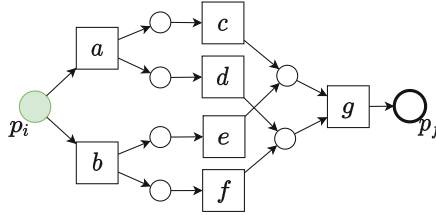


Fig. 2. Example Petri net in need of case isolation.

### 3.3 $\nu$ -Petri Nets

Resource-constrained Petri nets are especially useful when multiple cases are present simultaneously. An example in Fig. 2 shows that distinguishing tokens belonging to different cases is essential for capturing process behavior of simultaneously running cases in a correct way. The shown net does not have the separability property [30]. Trace  $\langle a, c, \underline{b}, f, g, d, e, g \rangle$  can be replayed on the Petri net without differentiating between the case ids. However, this trace cannot be formed as an interleaving shuffle of the traces of two separate cases, since each firing of transition  $g$  uses tokens belonging to two different cases. Thus we need a mechanism preventing firings of transitions that mix tokens belonging to different resources. Moreover, we also need a mechanism allowing us to keep track of resource instances. In our running example, we need e.g. a possibility to extend the model with a constraint that the doctor who performed the intake is also the doctor who performs the surgery later in the process.

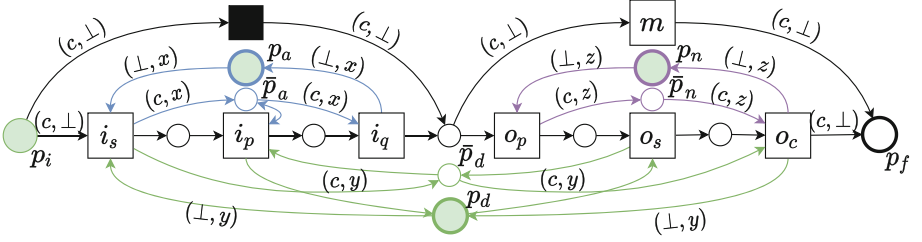
We use  $\nu$ -Petri nets to provide case and resource isolation.  $\nu$ -Petri nets, also referred to as Petri nets with names, extend regular Petri nets with the capability of name management. The expressive power of a  $\nu$ -Petri net strictly surpasses that of Petri nets and they essentially correspond to the minimal object-oriented Petri nets of [11]. In a  $\nu$ -Petri net, names can be created, communicated and matched which can be used to deal with authentication issues [23], correlation or instance isolation [8]. Name management is formalized by replacing ordinary tokens by distinguishable ones, thus adding color the Petri net.

We first give the definition of regular  $\nu$ -Petri nets from [22] (see Definition 12), after which we show how we extend the definition to work with resource-constrained Petri nets. Colors are handled by matching variables labeling the arcs of the Petri nets, taken from a fixed set  $Var$  and a set of special variables  $\mathcal{Y} \subset Var$  as defined in Definition 12.

**Definition 12** ( $\nu$ -Petri net [22]). A  $\nu$ -Petri net is a tuple  $\nu\text{-}N = \langle P, T, \mathcal{F} \rangle$ , with a set of places  $P$  and a set of transitions  $T$  with  $P \cap T = \emptyset$ , and a flow function  $\mathcal{F} : (P \times T) \cup (T \times P) \rightarrow Var^\oplus$  such that for every  $t \in T$ ,  $\mathcal{Y} \cap pre(t) = \emptyset$  and  $post(t) \setminus \mathcal{Y} \subseteq pre(t)$ , where  $pre(t) = \bigcup_{p \in P} supp(\mathcal{F}(p, t))$  and  $post(t) =$

$\bigcup_{p \in P} supp(\mathcal{F}(t, p))$ .  $\mathcal{Y} \subset Var$  denotes a set of special variables ranged by  $\nu, \nu_1, \dots$  to instantiate fresh names.





**Fig. 3.** Process model  $M$ :  $\nu$ -Petri net representation of the running example (note that some arc labels are omitted for clarity).

A marking of  $\nu$ - $N$  is a function  $m : P \rightarrow Id^\oplus$ .  $Id(m)$  denotes the set of names in  $m$ , i.e.  $Id(m) = \bigcup_{p \in P} supp(m(p))$ .

A mode  $\mu$  of a transition  $t$  is an injection  $\mu : Var(t) \rightarrow Id$ , that instantiates each variable to an identifier.

For a firing of transition  $t$  with mode  $\mu$ , we write  $m \xrightarrow{t, \mu} m'$ .  $t$  is enabled with mode  $\mu$  if  $\mu(\mathcal{F}(p, t)) \subseteq m(P)$  for all  $p \in P$  and  $\mu(\nu) \notin Id(m)$  for all  $\nu \in \Upsilon \cap Var(t)$ . The reached state after the firing of  $t$  with mode  $\mu$  is the marking  $m'$ , given by:

$$m'(p) = m(p) - \mu(\mathcal{F}(p, t)) + \mu(\mathcal{F}(t, p)) \text{ for all } p \in P \quad (1)$$

$\nu$ -Petri nets support instance isolation: we use case ids and resource ids as token colors and require tokens involved in a transition firing to have matching colors. This allows for separating multiple instances simultaneously running in the Petri net. We build on Definition 12 to define resource constrained Petri nets with matching on case instances and resource instances. Instance isolation is achieved by extending the colored tokens to *multi-colored*, for which we have two sets of variables,  $Var_c$  and  $Var_r$ , for case and resource isolation respectively, instead of the single set  $Var$  in  $\nu$ -Petri nets. This requires modifications in the standard definition of arcs  $\mathcal{F}$ , marking  $m$  and mode  $\mu$  of the  $\nu$ -Petri net. The definition of transition firings remains the same.

**Definition 13** (Resource-constrained  $\nu$ -Petri net). Let  $C^\perp$  be the set of case ids  $C$  extended with ordinary tokens, i.e.  $\bullet \in C$ , and  $R^\perp$  be the set of resource ids extended with ordinary tokens.

A resource-constrained  $\nu$ -Petri net  $N = \langle P, T, \mathcal{F} \rangle$  is a Petri net system with  $\mathcal{F} : (P \times T) \cup (T \times P) \rightarrow (Var_c^\perp \times Var_r^\perp)^\oplus$ , where  $Var_c$  denote case variables and  $Var_r$  denote resource variable.

A marking of  $N$  is a function  $m : P \rightarrow (C^\perp \times R^\perp)^\oplus$  with case ids  $C$  and resources  $R$ , which is a mapping from places to multisets of colored tokens.

A mode of a transition  $t$  is an injection  $\mu : (Var_c^\perp \times Var_r^\perp)(t) \rightarrow (C^\perp \times R^\perp)$ , that instantiates each variable to an identifier.

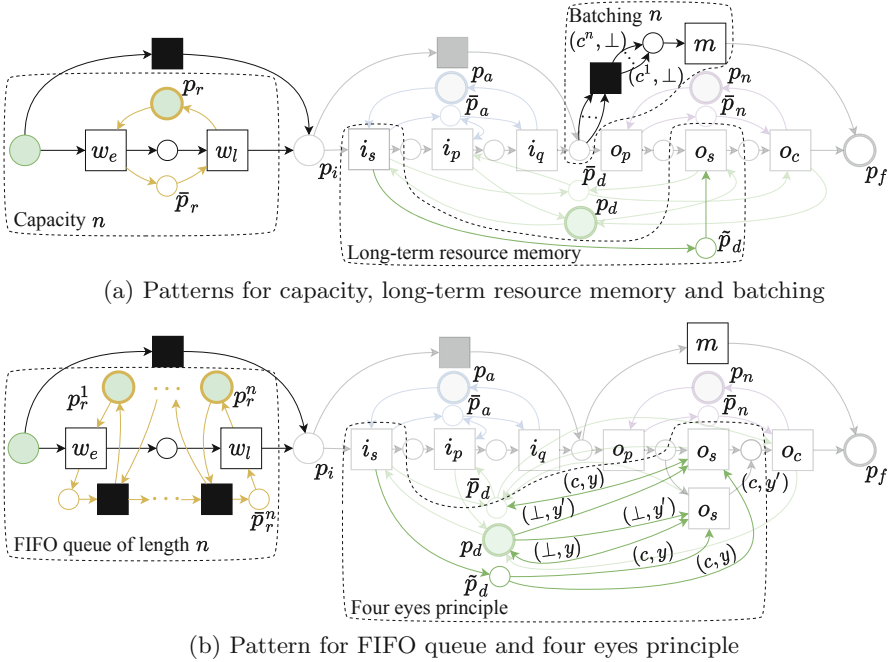


Fig. 4. Modeling patterns extending the running example showing possible use cases.

The mode determines the case and resource ids of the tokens to consume and produce in a transition firing. Note the role of the occupancy resource places in this definition: they allow to keep track of resources working on individual cases. The place invariant naturally holds for resource-constrained  $\nu$ -Petri nets as well.

Figure 3 shows the  $\nu$ -Petri net for our running example.

Note that this example does not include the functionality of new name creation although this could be useful to exploit in some processes. E.g., consider a production process where components are assembled into products. Each component has a unique identifier, of which a subset is merged into a product for which a fresh identifier should be produced, requiring new name creation in the process model.

### 3.4 Modeling Patterns

Resource-constrained  $\nu$ -Petri nets open up a number of possibilities in terms of simulating resource-constrained processes, where case and resource isolation are critical for correct simulation.

We illustrate a number of modeling patterns in Fig. 4, building on our running example:

- **Capacity, FIFO queues** A subprocess in Fig. 4a on the left, prior to the intake, models a waiting room with a capacity limited by the initial number

of tokens on the resource place  $p_r$ . Figure 4b shows a FIFO (first in first out) version of the waiting room. For the FIFO queue with capacity  $n$ ,  $n$  availability and  $n$  occupancy resource places are needed for this pattern to lead patient tokens (with their case ids) through the queue.

- **Long term resource memory:** Place  $\tilde{p}_d$  with  $\mathcal{F}(i_s, \tilde{p}_d) = \mathcal{F}(\tilde{p}_d, o_s) = (c, y)$  models a long term resource memory with respect to the doctor, ensuring that the resource instance of type doctor that was involved in the intake of a patient is also the same instance that performs the operation for that patient. This construct still allows the resource to be available for other cases (patients) in between the intake and the surgery.
- **Four eyes principle:** Alternatively, as shown in Fig. 4b on the right,  $\tilde{p}_d$  can also be used to add the opposite restriction with respect to the doctor resource: the doctor performing the operation ( $d_2$ ) should be different from the doctor involved in the intake ( $d_1$ ). This pattern is known as the four eyes principle, meaning that two resources involved in a process should not be equal. Note that the starting transition of the second subprocess should be duplicated, since the intake-doctor  $d_1$ , who is not involved in the surgery, could be residing in either  $p_d$  or  $\tilde{p}_d$ .
- **Batching:** Batch processing can be modeled with multiple arrows, like the arrows connected to the silent transition in the net firing before transition  $m$ . In this case, a pharmacy may only replenish their inventory for e.g. three orders of medication at once. This pattern is similar to the one in classical Petri nets.

## 4 Alignments on Resource-Constrained Petri Nets

Several state-of-the art techniques in conformance checking use alignments to relate the recorded executions of a process with a model of this process [1]. A traditional alignment shows how a trace can be replayed on the process model by a sequence of moves representing either a synchronous move, a log move or a model move, denoted as  $\binom{a}{a}$ ,  $\binom{a}{\gg}$  and  $\binom{\gg}{a}$  respectively. A synchronous move indicates that observed and modeled behavior agree, i.e. the execution of an activity observed in the log can be mimicked by performing this activity in the process model. A log move means that an activity from the log cannot be mimicked in the model, and a model move represents the fact that the model requires an execution of some activity, which is not observed in the log. Log moves and model moves can expose deviations of the real behavior from the model.

In this section we recapitulate the alignment mechanism in its classical form, with computations performed on a case-by-case basis, after which we show its shortcomings when dealing with resource-constrained Petri nets.

### 4.1 Traditional Case-by-Case Alignments

The foundational work for constructing alignments is presented in [1] and relies on two fundamental concepts: (1) a synchronous product of Petri nets and (2)

the marking equation. The synchronous product definition is tuned towards the setting of alignments and it is built for the Petri net model of the process and the trace Petri net (a Petri net representation of the (partially) ordered trace in the event log). The trace Petri net traditionally represents only individual cases from the event log, and consequently does not capture the interdependencies between multiple cases. The case-by-case alignment is then found by a depth-first search on the synchronous product Petri net using the  $A^*$  algorithm [1].

A *trace net system*  $SN_\sigma$  of a trace  $\sigma = \langle e_1, \dots, e_n \rangle$  is a net system with the set of transitions  $T^l = \{t_i^l \mid e_i \in \sigma\}$ , a connection place for every pair of transitions  $t_i^l$  and  $t_{i+1}^l$ , place  $p_i$  being the input place of  $t_1$  and place  $p_f$  being the output place of  $t_n$ ,  $m_i = \{p_i\}$  and  $m_f = \{p_f\}$ .

Given a net system  $SN$  modeling the considered process and a trace net system  $SN_\sigma$  modeling a trace from a log, a *synchronous product IISN* contains the places and the transitions of  $SN$  and  $SN_\sigma$  and additional transitions called synchronous moves: For each pair of transitions  $t^m \in T(SN), t^l \in T(SN_\sigma)$  with matching labels  $\ell(t^l) = \ell(t^m)$ , transition  $t^s$  is created with  $\bullet t^s = \bullet t^l \cup \bullet t^m$  and  $t^s \bullet = t^l \bullet \cup t^m \bullet$ . Thus  $IISN$  contains transitions  $T = T^s \cup T^l \cup T^m$ , where each  $t^s \in T^s$  can be traced back to a pair of a transition  $t^l \in T^l$  and a transition  $t^m \in T^m$ ,  $T^l$  is the set of transitions of the trace net system and  $T^m$  is the set of transitions of the process model  $SN$ . While  $T^s$  transitions represent synchronous moves in both the trace net and the process model,  $T^l$  transitions represent log moves and  $T^m$  transitions represent model moves.

The core alignment question is now formalized as follows: given a synchronous product Petri net with a cost function assigning a non-negative cost to each transition firing, find a distributed run from the initial marking to the final marking with the lowest total costs. Synchronous moves have zero costs, since they represent a match between the trace and the model behavior.

Let  $IISN = \langle P, T, \mathcal{F} \rangle$  be a synchronous product Petri net with  $T = T^s \uplus T^l \uplus T^m$  partitioned into sets of transitions corresponding to synchronous moves, log moves and model moves respectively and let  $(IISN, m_i, m_f)$  a corresponding net system. Furthermore let  $c : T \rightarrow \mathbb{R}^+$  a cost function.

An *alignment* is a distributed run  $\gamma \in \{\gamma \in T^* \mid (IISN, m_i) \xrightarrow{\gamma} (IISN, m_f)\}$ .

An *optimal alignment* is an alignment  $\gamma$  such that  $c(\gamma) \leq c(\gamma')$  holds for any alignment  $\gamma'$ .

Optimal alignments can be computed for individual cases in an event log using an  $A^*$  based search strategy [1, 6, 29] where ILP is utilized as a heuristic function, or logic programming [5] is used. Other methods focus on approximations of alignments [26] or provide divide-and-conquer strategies [13]. Although we will not go into the details on the exact workings of these methods, we point out that they all have one fundamental property in common: they all reason over the synchronous product Petri net.

## 4.2 Unexposed Deviations; the Need for Multi-case and -resource Alignments

When talking about case-by-case alignments with resource-constrained Petri nets, some deviations remain unexposed. Recall that for simulating resource-constrained Petri nets, we need a  $\nu$ -Petri net representation to correctly isolate the cases and resources. Similar issues emerge when computing alignments case by case using indistinguishable resources. Referring back to the modeling patterns presented in Sect. 3.4, we show some event logs for the extended running example process models from Fig. 4 for which case-by-case alignments fail to expose deviations in resource-constrained Petri nets:

- **Multitasking:** consider the partial event log  $L_1$  given by

$$L_1 = \langle \dots, \langle i_s, \{d_1, a_1\} \rangle, \langle \underline{i_s}, \{d_1, a_2\} \rangle, \langle i_p, \{d_1, a_1\} \rangle, \langle \underline{i_p}, \{d_1, a_2\} \rangle, \dots \rangle$$

where the timestamp is abstracted away and the case identifier is denoted by the activity color (and additionally by the bar position). The recorded behavior in  $L_1$  shows that doctor  $d_1$  is multitasking on the intake subprocesses of two patients. The resource-constrained  $\nu$ -Petri net does not accept this behavior since  $i_s$  claims the doctor and the doctor is released again only after  $i_p$  is executed. Case-by-case alignments consider every case in isolation and therefore they do not expose any deviations in  $L_1$ .

- **Resource switching:** consider the partial event log  $L_2$  given by

$$L_2 = \langle \dots, \langle i_s, \{d_1, a_1\} \rangle, \langle \underline{i_s}, \{d_2, a_2\} \rangle, \langle i_p, \{d_2, a_1\} \rangle, \langle \underline{i_p}, \{d_1, a_2\} \rangle, \dots \rangle$$

The behavior recorded in  $L_2$  shows that doctors  $d_1$  and  $d_2$  swapped patients during the intake subprocess, which is not allowed according to the process model where the resources have names (colors).

Furthermore, consider the partial event log  $L_3$  given by

$$L_3 = \langle \dots, \langle i_s, \{d_1, a_1\} \rangle, \langle i_p, \{d_1, a_1\} \rangle, \dots, \langle o_s, \{d_2, a_1\} \rangle, \dots \rangle$$

$L_3$  shows that doctor  $d_2$  performed an surgery on a patient whose intake was done by doctor  $d_1$ , although the long-term resource memory place  $\tilde{p}_d$  in the process model implies that the doctor performing the surgery is the same as the one who did the intake.

These deviations remain undetected by the traditional alignments, computed on classical Petri nets with black tokens.

- **Capacity violations:** consider the partial event log  $L_4$  given by

$$L_4 = \langle \langle w_e, \{r_1\} \rangle, \langle \underline{w_e}, \{r_1\} \rangle, \langle \overline{w_e}, \{r_1\} \rangle, \langle w_l, \{r_1\} \rangle, \langle \underline{w_l}, \{r_1\} \rangle, \langle \overline{w_l}, \{r_1\} \rangle, \dots \rangle$$

$L_4$  shows the behavior from the waiting room subprocess, where a maximum capacity of two patients is in place. Similar to the first example, case-by-case leaves this deviation undetected.

- **Overtaking in FIFO queues:** consider the partial event log  $L_5$  given by

$$L_5 = \langle \langle w_e, \{r_1\} \rangle, \langle \underline{w_e}, \{r_1\} \rangle, \langle \underline{w_l}, \{r_1\} \rangle, \langle w_l, \{r_1\} \rangle, \dots \rangle$$

The recorded behavior in  $L_5$  does not violate the waiting room’s capacity, but the patients leave in a different order than how they arrived, while the process model imposes the FIFO pattern. Such deviations can only be exposed when aligning multiple cases simultaneously.

- **Batching violations:** consider the partial event log  $L_6$  given b

$$L_6 = \langle \dots, \langle m, \emptyset \rangle, \langle \underline{m}, \emptyset \rangle, \dots \rangle$$

Recall that  $m$  is only enabled after e.g. three (with  $n = 3$ ) tokens are in  $\bullet o_p$  in order to fire the connected silent transition in the process model.  $L_6$  shows that  $m$  occurred for two patients only, deviating from the model. With case-by-case alignments, it is impossible to align  $m$ . It also would be impossible in case there were three patients according to the log, since it requires multiple cases being processed simultaneously.

This clearly shows that case-by-case conformance checking is not sufficient and resource identities are essential for detecting deviations from standard resource-related constraints. Therefore, we introduce multi-case and multi-resource alignments using  $\nu$ -Petri nets that align the *complete* event log to the process model, allowing to expose the deviations listed above.

## 5 Computing Multi-case and -resource Alignments

Our approach to computing multi-case alignments is based on the traditional alignment-based approach using the synchronous product Petri net. Instead of representing individual cases in the trace Petri net, we capture the complete event log there in order to consider inter-case relations. To retain the case isolation when aligning, we transform the process model given by a resource-constrained Petri net into the resource-constrained  $\nu$ -Petri net (Definition 13).

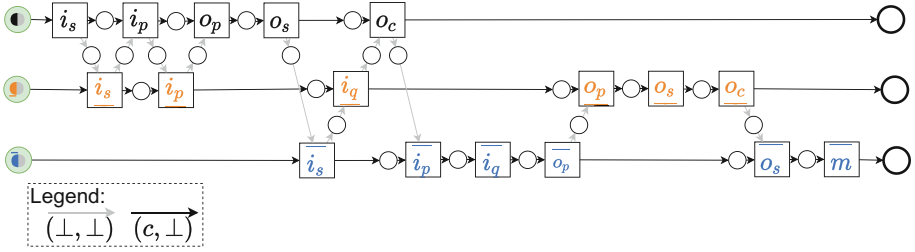
As a running example we take the Petri net from Fig. 1 and the event log:

$$L = \langle \dot{i}_s, \dot{i}_p, \dot{i}_p, o_p, o_s, \overline{\dot{i}_s}, \overline{\dot{i}_q}, o_c, \overline{\dot{i}_p}, \overline{\dot{i}_q}, \overline{o_p}, \overline{o_p}, \overline{o_s}, \overline{o_c}, \overline{o_s}, \overline{m} \rangle$$

where the colors (and bar positions) represent the case identifiers. The timestamps and involved resource instances are omitted and made implicit by the ordering and the transitions from the process model respectively. For this example, there is a single resource instance for each resource type.

### 5.1 Approach

Recall that alignments are computed by taking the synchronous product Petri net constructed from the process model Petri net and a Petri net representation of the trace, called the trace Petri net. With a resource constrained  $\nu$ -Petri net



**Fig. 5.** Running example trace  $\nu$ -Petri net

as the process model, we have to modify the definition of the trace Petri net that takes into account the case identifiers of the events in the event log. We achieve this by constructing multiple trace Petri nets for each trace in the event log projected on the case identifiers. The cases are then differentiated by turning this into a  $\nu$ -Petri net with the label  $(c, \perp)$  on the arcs. Additionally, we add places between the transitions to enforce the correct ordering as they occurred in the event log, with label  $(\perp, \perp)$  on its incoming and outgoing arcs. Formally, this trace  $\nu$ -Petri net is defined as follows:

**Definition 14** (*Trace  $\nu$ -Petri net*).  $\nu$ -SN =  $\langle P, T, \mathcal{F}, \ell \rangle$ , a labeled trace  $\nu$ -Petri net, is constructed from an event log  $L = \langle e_i^{id_c, a} \rangle_{1 \leq i \leq n}$ , with  $id_c$  and  $a$  denoting respectively the case identifier and activity of event  $e_i$ . For each event  $e_i^{id_c, a}$  with  $1 \leq i \leq n$ , we have a transition  $t_i^{id_c}$  with  $\ell(t_i) = a$ . Places are added between these transitions as follows<sup>1</sup>: With  $i$  from 1 to  $n$ , we add a place  $p_i^{id_c}$  between  $t_i^{id_c}$  and  $\min_{j > i} t_j^{id_{c'}}$  such that  $id_{c'} = id_c$  with  $\mathcal{F}(t_i^{id_c}, p_i^{id_c}) = \mathcal{F}(p_i^{id_c}, t_j^{id_{c'}}) = (c, \perp)$ . Furthermore, we add a place  $p_i^\tau$ , enforcing the original ordering, between  $t_i^{id_c}$  and  $t_{i+1}^{id_{c'}}$  if  $id_{c'} \neq id_c$  with  $\mathcal{F}(t_i^{id_c}, p_i^\tau) = \mathcal{F}(p_i^\tau, t_{i+1}^{id_{c'}}) = (\perp, \perp)$ .

Lastly, initial and final places are added for each case  $id_c \in Id_c$ :  $p_{in}^{id_c}$  to  $\min_i t_i^{id_c}$  and  $p_{out}^{id_c}$  to  $\max_j t_j^{id_c}$  with  $\mathcal{F}(p_{in}^{id_c}, t_i^{id_c}) = \mathcal{F}(t_j^{id_c}, p_{out}^{id_c}) = (c, \perp)$ . The initial and final marking is then defined by  $m_i(p_{in}^{id_c}) = m_f(p_{out}^{id_c}) = \{(id_c, \perp)\}$ .

The trace  $\nu$ -Petri net for the running example is shown in Fig. 5. With the redefined trace  $\nu$ -Petri net we can construct the synchronous product Petri net  $\nu$ -HISN consisting of the resource constrained  $\nu$ -Petri net and the trace  $\nu$ -Petri net.

Computing the multi-case alignments is now a matter of finding the distributed run in the  $\nu$ -HISN for which we can use *any* of the existing methods as described in Sect. 4.1. The optimal alignment is again the one with lowest cost.

Note that while  $\nu$ -Petri nets are inherently unbounded by generating fresh tokens, we can decide on the tokens to be generated beforehand by preprocessing the event log, and can therefore retain boundedness.

<sup>1</sup> Adding a place  $p$  between two transitions  $t_a$  and  $t_b$  denotes a single arc from  $t_a$  to  $p$  and a single arc from  $p$  to  $t_b$ .

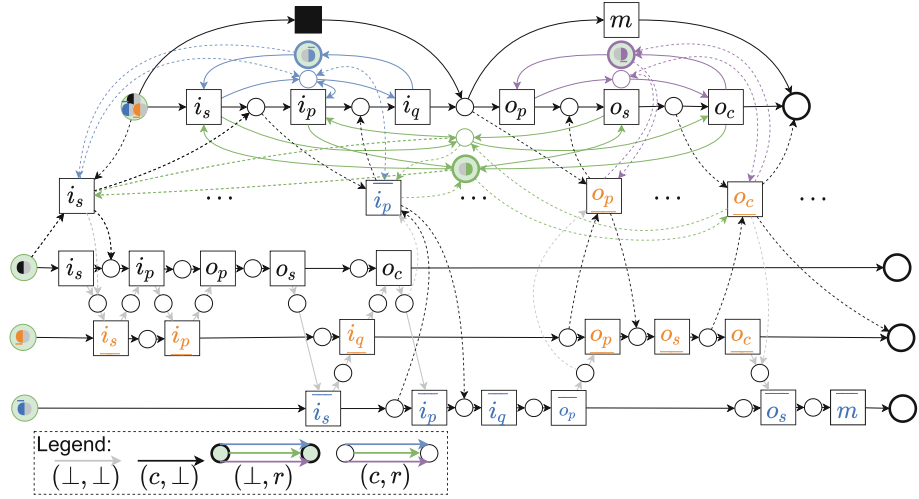


Fig. 6. Running example synchronous product  $\nu$ -Petri net

For our running example we get the synchronous product  $\nu$ -Petri net as shown in Fig. 6 and the corresponding optimal alignment is the shortest path through  $HSN$  and is shown in Table 1. Note that not all synchronous transitions are visualized in the figure for clarity reasons.

Table 1. Alignment from the naive method

$\gamma^{(L)}$		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17		
$L$		$i_s$	$\gg$	$\underline{i_s}$	$i_p$	$\underline{i_p}$	$o_p$	$o_s$	$\underline{i_s}$	$\underline{i_q}$	$o_c$	$\gg$	$\underline{i_p}$	$i_q$	$o_p$	$\underline{o_p}$	$\underline{o_s}$	$\underline{o_c}$	$o_s$	$\underline{m}$
$M$		$\gg$	$\tau$	$\underline{i_s}$	$\gg$	$\underline{i_p}$	$o_p$	$o_s$	$\gg$	$\underline{i_q}$	$o_c$	$\underline{i_s}$	$\underline{i_p}$	$i_q$	$\gg$	$\underline{o_p}$	$\underline{o_s}$	$\underline{o_c}$	$\gg$	$\underline{m}$

### 5.2 Multi-case and -resource Alignments in Action

With the examples listed in Sect. 4.2 of undetected deviations with traditional alignments, we show here how multi-case and -resource alignments expose them. Note that resource attributes for some event logs are abstracted away when this is implicit from the data (only a single resource was involved). For others, the resource attribute is denoted in the superscript of the event.

- $\gamma^{(L_1)}$  shows an optimal alignment for  $L_1$  computed from the method described above. With multi-case and -resource alignments, it is not possible anymore for all moves to be aligned synchronously because of the doctor's availability during the intake process.



$$\gamma^{(L_1)} = \left| \frac{L_1 \mid i_s \gg \underline{i_s} \ i_p \ \underline{i_p}}{M_e \mid i_s \ i_p \ \underline{i_s} \ \gg \ \underline{i_p}} \right|$$

- $\gamma^{(L_2)}$  shows an optimal alignment for  $L_2$ , where we see that  $i_p$  and  $\underline{i_p}$  should have occurred with doctors  $d_1$  and  $d_2$  respectively according to the model exposing that they have switched positions in the recorded behavior.

$$\gamma^{(L_2)} = \left| \frac{L_2 \mid i_s^{d_1} \ \underline{i_s^{d_2}} \ \gg \ i_p^{d_2} \ \gg \ \underline{i_p^{d_1}}}{M_e \mid i_s^{d_1} \ \underline{i_s^{d_2}} \ i_p^{d_1} \ \gg \ \underline{i_p^{d_2}} \ \gg} \right| \quad \gamma^{(L_3)} = \left| \frac{L_3 \mid i_s^{d_1} \ i_s^{d_2} \ \dots \ \gg \ o_s^{d_2}}{M_e \mid i_s^{d_1} \ i_s^{d_2} \ \dots \ o_s^{d_1} \ \gg} \right|$$

- $\gamma^{(L_3)}$  shows an optimal alignment for  $L_3$ , where we see that the long term resource memory is violated and  $o_s$  should have been executed by doctor  $d_1$  instead of  $d_2$ .
- $\gamma^{(L_4)}$  shows an optimal alignment for  $L_4$ , revealing that  $w_l$  should have occurred before  $\underline{w_e}$  according to the capacity restriction in the model, i.e. the first patient should have left the waiting room before the third patient entered.

$$\gamma^{(L_4)} = \left| \frac{L_4 \mid w_e \ \underline{w_e} \ \gg \ \overline{w_e} \ w_l \ \underline{w_l} \ \overline{w_l}}{M_e \mid w_e \ \underline{w_e} \ w_l \ \overline{w_e} \ \gg \ \underline{w_l} \ \overline{w_l}} \right| \quad \gamma^{(L_5)} = \left| \frac{L_5 \mid w_e \ \underline{w_e} \ \gg \ \underline{w_l} \ w_l}{M_e \mid w_e \ \underline{w_e} \ w_l \ \underline{w_l} \ \gg} \right|$$

- $\gamma^{(L_5)}$  shows an optimal alignment for  $L_5$ . The model move on  $w_l$  shows that the first patient should leave before the second one does, exposing the FIFO violation in the waiting room process.
- $\gamma^{(L_6)}$  shows an optimal alignment for  $L_6$ , where the batching restriction is violated. The model moves show that an added third patient should have been included in order to execute  $m$  for the three patients.

$$\gamma^{(L_6)} = \left| \frac{L_6 \mid \overline{\overline{\tau}} \ \overline{\overline{\tau}} \ \overline{\overline{\tau}} \ m \ \underline{m}}{M_e \mid \overline{\tau} \ \overline{\tau} \ \underline{m} \ m \ \underline{m}} \right|$$

### 5.3 Relaxing the Synchronous Product Petri Net to Detect Resource-Related Deviations

With the alignments generated as described above it could be difficult to interpret the exposed deviations, especially in terms of the added model moves: was the activity executed but not logged, was it executed by a “wrong” resource and therefore not executable in the model, or was it definitely not executed because no appropriate resource was available?

In this section we show how we can use simple model transformations on the synchronous product Petri net to allow for additional behavior (at some costs), so we can interpret resource-related deviations in more detail. To show these transformations, we use an example subprocess with two activities  $a$  and  $b$  in which a resource of type  $r$  is involved. The corresponding Petri net is shown in Fig. 7, where the transitions  $\{\bar{a}, \bar{b}, \tau_1, \tau_2, \tau_3\}$  are added to the model.

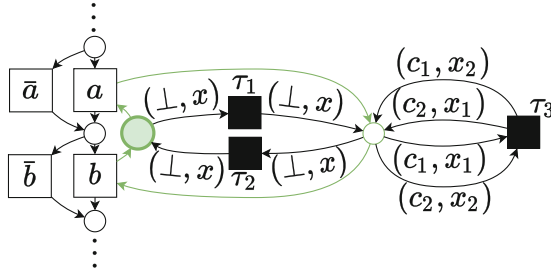


Fig. 7. Resource relaxations for improved alignment interpretability.

**Multitasking.** From alignment  $\gamma^{(L_1)}$  from Sect. 5.2 it is not immediately clear that the resource instance was multitasking. The model transformation in Fig. 7 using silent transitions  $\tau_1$  and  $\tau_2$  between  $p_r$  and  $\bar{p}_r$  allows for turning a resource instance from available to occupied and vice versa at any point in time. These silent transitions give additional interpretation to the alignment, showing where the resource might have been released or claimed to work on another case, while it was not allowed by the model. With  $c(\tau_1) > 0$  and  $c(\tau_2) > 0$ , model moves with  $\tau_1$  or  $\tau_2$ , which we denote as *resource moves*, are only selected when necessary.

**Model Moves Not Claiming Resources.** Transitions  $\bar{a}$  and  $\bar{b}$  allow for model moves not claiming the resource(s), which we call *control flow moves*. It is important to note that these transitions have no corresponding “resource-free” synchronous move in the synchronous product Petri net  $HSN$ , since they represent a relaxed version of the model move. Therefore, the cost of a control flow move should be higher than the cost of the corresponding model move (that do claim the resources), i.e.  $c(\bar{a}) > c(a)$  and  $c(\bar{b}) > c(b)$ .

This transformation allows to compute more sensible alignments in cases when a work item is skipped and the resources necessary for it were occupied, making a model move impossible. In such cases, the alignment without control flow moves would fit the model moves in time ranges where the needed resources were available, potentially causing conflicts in earlier or later stages of the process. Therefore, with the additional model moves, the resource claim is bypassed.

$\tau_1$  and  $\tau_2$  as introduced above are necessary to avoid deadlocks that can arise in case a control flow move mimics a transition that claims or releases resources.

**Resource Switching.** From alignments  $\gamma^{(L_2)}$  and  $\gamma^{(L_3)}$  from Sect. 5.2 it is not immediately clear that the resource instances have switched or the incorrect resource instance is involved. The model transformation in Fig. 7 using  $\tau_3$  connected to  $\bar{p}_r$  and possibly  $\tilde{p}_r$  allows the resources to take over each other’s work. For the resource switch from  $\gamma^{(L_3)}$ , additionally  $\tau_1$  and  $\tau_2$  are necessary to get  $d_1$  into  $\tilde{p}_d$ . With  $c(\tau_3) > 0$ , a model move with  $\tau_3$  would only reside in the optimal alignment would it be necessary and it is interpretable showing the

resource instance that took over work for a case from a specific resource instance. A model move with  $\tau_3$  is also denoted as a resource move.

**Resource Type Relaxation.** Recall that the resource instances as defined above are strictly typed. In case we want to allow a resource to execute tasks belonging to other resource types (at some cost), e.g.  $n$  and  $a$ , the following model transformation is sufficient: add a place  $p_{n,a}$  with  $\bullet p_{n,a} = \bullet p_n \cup \bullet p_a$  and  $p_{n,a}^\bullet = p_n^\bullet \cup p_a^\bullet$ . A resource instance that could be involved in activities from  $n$  and  $a$  resides initially in  $p_{n,a}$  from where it is able to do both.

Furthermore, when we would want to allow some types, e.g. for type  $r$ , to be unnamed, a model transformation making all tokens on place  $p_r$  the same color suffices. Note that this was already done for the waiting room resource instances for alignments  $\gamma(L_4)$  and  $\gamma(L_5)$  from Sect. 5.2.

These, and possible other model transformations could be used to enrich the alignment providing more interpretability.

## 6 Conclusion

In this paper we proposed a model for processes with shared resources using some features of resource-constrained workflow nets and  $\nu$ -Petri nets. Our model allows to distinguish both cases and resources. This opens up possibilities in terms of modeling intricate inter-case dependencies and shared resources, including long-term resource memory, while still offering an option to exploit structural properties like well-structuredness of the control flow for e.g. conformance checking.

We showed that traditional alignments for conformance checking fail to detect some deviations that can arise in processes with shared resources. With our extended  $\nu$ -Petri net representation of the process model, and a newly defined trace  $\nu$ -Petri net containing the complete event log, we showed that the techniques for computing alignments can be utilized to expose violations on inter-case dependencies and usage of shared resources.

Our proposed extension to  $\nu$ -Petri nets is a minimal extension that is sufficient for computing alignments on event logs without redundant functionality that other, possibly more sophisticated, extensions may offer as discussed in Sect. 1.

**Future Work.** Computing alignments on a case-by-case basis is already a complex problem in terms of computational power [6]. In principle, the complexity increases when multiple traces together with resource information are considered. At the same time, resource information available in the log can narrow the actual search space. We plan to look into preprocessing techniques and into structural reductions and decompositions for the Petri net to reduce the search space when computing the alignments.

## References

1. Adriansyah, A.: Aligning observed and modeled behavior. Ph.D. thesis, Mathematics and Computer Science (2014)
2. Alizadeh, M., Lu, X., Fahland, D., Zannone, N., van der Aalst, W.M.P.: Linking data and process perspectives for conformance analysis. *Comput. Secur.* **73**, 172–193 (2018)
3. Barkaoui, K., Petrucci, L.: Structural analysis of workflow nets with shared resources (1998)
4. Berti, A., van der Aalst, W.M.P.: A novel token-based replay technique to speed up conformance checking and process enhancement. *Trans. Petri Nets Other Model. Concurr.* **15**, 1–26 (2021)
5. Boltenhagen, M., Chatain, T., Carmona, J.: Optimized sat encoding of conformance checking artefacts. *Computing* **103**(1), 29–50 (2021)
6. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: *Conformance Checking*. Springer, Heidelberg (2018)
7. de Leoni, M., van der Aalst, W.M.P.: Aligning event logs and process models for multi-perspective conformance checking: an approach based on integer linear programming. In: Daniel, F., Wang, J., Weber, B. (eds.) *BPM 2013. LNCS*, vol. 8094, pp. 113–129. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40176-3\\_10](https://doi.org/10.1007/978-3-642-40176-3_10)
8. Decker, G., Weske, M.: Instance isolation analysis for service-oriented architectures. In: 2008 IEEE International Conference on Services Computing, vol. 1, pp. 249–256. IEEE (2008)
9. Fahland, D.: Describing behavior of processes with many-to-many interactions. In: Donatelli, S., Haar, S. (eds.) *PETRI NETS 2019. LNCS*, vol. 11522, pp. 3–24. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-21571-2\\_1](https://doi.org/10.1007/978-3-030-21571-2_1)
10. Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: Petri nets with parameterised data: modelling and verification (extended version). arXiv preprint [arXiv:2006.06630](https://arxiv.org/abs/2006.06630) (2020)
11. Kummer, O.: Undecidability in object-oriented Petri nets. In: *Petri Net Newsletter*. Citeseer (2000)
12. Lautenbach, K.: Liveness in Petri Nets. Bonn Interner Bericht ISF. Selbstverl, GMD (1975)
13. Lee, W.L.J., Verbeek, H.M.W., Munoz-Gama, J., van der Aalst, W.M.P., Sepúlveda, M.: Replay using recomposition: alignment-based conformance checking in the large. In: *BPM (Demos)* (2017)
14. Letia, I.A., Goron, A.: Model checking as support for inspecting compliance to rules in flexible processes. *J. Vis. Lang. Comput.* **28**, 100–121 (2015)
15. Mannhardt, F., De Leoni, M., Reijers, H.A., Van Der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. *Computing* **98**(4), 407–437 (2016)
16. Mozafari Mehr, A.S., de Carvalho, R.M., van Dongen, B.: Detecting privacy, data and control-flow deviations in business processes. In: Nurcan, S., Korthaus, A. (eds.) *CAiSE 2021. LNBIP*, vol. 424, pp. 82–91. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-79108-7\\_10](https://doi.org/10.1007/978-3-030-79108-7_10)
17. Montali, M., Rivkin, A.: Model checking Petri nets with names using data-centric dynamic systems. *Formal Aspects Comput.* **28**(4), 615–641 (2016)

18. Montali, M., Rivkin, A.: DB-Nets: on the marriage of colored Petri nets and relational databases. In: Koutny, M., Kleijn, J., Penczek, W. (eds.) *Transactions on Petri Nets and Other Models of Concurrency XII*. LNCS, vol. 10470, pp. 91–118. Springer, Heidelberg (2017). [https://doi.org/10.1007/978-3-662-55862-1\\_5](https://doi.org/10.1007/978-3-662-55862-1_5)
19. Murata, T.: Petri nets: properties, analysis and applications. *Proc. IEEE* **77**(4), 541–580 (1989)
20. Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains. In: Kahn, G. (ed.) *Semantics of Concurrent Computation*. LNCS, vol. 70, pp. 266–284. Springer, Heidelberg (1979). <https://doi.org/10.1007/BFb0022474>
21. Peterson, J.L.: *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR (1981)
22. Rosa-Velardo, F., de Frutos-Escrig, D.: Decision problems for Petri nets with names. arXiv preprint [arXiv:1011.3964](https://arxiv.org/abs/1011.3964) (2010)
23. Rosa-Velardo, F., de Frutos-Escrig, D., Marroquín-Alonso, O.: On the expressiveness of mobile synchronizing Petri nets. *Electron. Notes Theor. Comput. Sci.* **180**(1), 77–94 (2007)
24. Rozinat, A., Van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. *Inf. Syst.* **33**(1), 64–95 (2008)
25. Taghiabadi, E.R., Gromov, V., Fahland, D., van der Aalst, W.M.P.: Compliance checking of data-aware and resource-aware compliance requirements. In: Meersman, R., et al. (eds.) *OTM 2014*. LNCS, vol. 8841, pp. 237–257. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-662-45563-0\\_14](https://doi.org/10.1007/978-3-662-45563-0_14)
26. Taymouri, F., Carmona, J.: An evolutionary technique to approximate multiple optimal alignments. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) *BPM 2018*. LNCS, vol. 11080, pp. 215–232. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-98648-7\\_13](https://doi.org/10.1007/978-3-319-98648-7_13)
27. Van Der Aalst, W.: Data science in action. In: Van Der Aalst, W. (ed.) *Process Mining*, pp. 3–23. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49851-4\\_1](https://doi.org/10.1007/978-3-662-49851-4_1)
28. Van der Aalst, W., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisc. Rev. Data Min. Knowl. Discov* **2**(2), 182–192 (2012)
29. Dongen, B.F.: Efficiently computing alignments. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) *BPM 2018*. LNCS, vol. 11080, pp. 197–214. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-98648-7\\_12](https://doi.org/10.1007/978-3-319-98648-7_12)
30. van Hee, K., Sidorova, N., Voorhoeve, M.: Soundness and separability of workflow nets in the stepwise refinement approach. In: van der Aalst, W.M.P., Best, E. (eds.) *ICATPN 2003*. LNCS, vol. 2679, pp. 337–356. Springer, Heidelberg (2003). [https://doi.org/10.1007/3-540-44919-1\\_22](https://doi.org/10.1007/3-540-44919-1_22)
31. Van Hee, K., Sidorova, N., Voorhoeve, M.: Resource-constrained workflow nets. *Fundamenta Informaticae* **71**(2, 3), 243–257 (2006)