Checking Conformance between Colored Petri Nets and Event Logs*

Julio C. Carrasquel[1], Khalil Mecheraoui[1,2], and Irina A. Lomazova[1]

[1] National Research University Higher School of Economics,
Myasnitskaya ul. 20, 101000 Moscow, Russia
jcarrasquel@hse.ru, k_mecheraoui@esi.dz, ilomazova@hse.ru
[2] University of Constantine 2 — Abdelhamid Mehri,
Nouvelle ville Ali Mendjeli BP : 67A, 25000 Constantine, Algeria

Abstract. Event logs of information systems consist of recorded traces, describing executed activities and involved resources (e.g., users, data objects). Conformance checking is a family of process mining techniques that leverage such logs to detect whether observed traces deviate w.r.t some specification model (e.g., a Petri net). In this paper, we present a conformance checking method using colored Petri nets (CPNs) and event logs. CPN models allow not only to specify a causal ordering between system activities, but also they allow to describe how resources must be processed upon activity executions. By replaying each trace of an event log on top of a CPN, we present how this method detects: (1) control-flow deviations due to unavailable resources, (2) rule violations, and (3) differences between modeled and real produced resources. We illustrate in detail our method using the study case of trading systems, where orders from traders must be correctly processed by a platform. We describe experimental evaluations of our method to showcase its practical value.

Keywords: Process mining, conformance checking, Petri nets, colored Petri nets, trading systems, order books.

1 Introduction

Conformance checking is a family of process mining techniques to diagnose whether or not a system process is being executed as described by its specification model [1, 3]. Two main inputs are considered in such methods: *event logs* and *process models*. On the one hand, an event log describes *real behavior* of a process. It consists of recorded traces, each of them consisting of executed activities and *resources* involved in such executions. Resources may be users or data objects processed by a system. On the other hand, a process model allows to describe *expected behavior* of a system process, based on its specification. Regarding the model notation, conformance checking methods consider Petri nets — a formalism for modeling and analysis of concurrent distributed systems [15].

* This work is supported by the Basic Research Program at the National Research University Higher School of Economics.

In particular, Petri nets allow to specify the *control-flow* of a system, that is, a causal ordering between system activities (e.g., activity `a` must be followed by `b`). Thus, conformance checking methods use Petri nets and event logs to determine, for instance, to which degree the modeled control-flow is being complied by the real system, as observed in the recorded traces. For example, "a loan approval was executed, but it was not inspected before, and this must not happen according to the model". This is why conformance checking has become a research subject of interest in several application domains, i.e., for auditing business processes [20].

Nonetheless, most of the conformance checking methods merely focus on the control-flow aspect (i.e., only considering event activities), thereby neglecting other valuable information recorded in event logs, for example, processed resources. This imposes severe limitations in study cases where the system's correct execution can be only determined by checking involved resources in events (i.e., "a trade can be executed if a buy order and a sell order are available").

To address such limitation, certain conformance methods propose the use of enriched models, such as in [13], where Petri nets with data (DPN) are employed. In DPNs, data variables are attached to transitions (representing activities), and thus this model allows to specify data constraints on activity executions (for instance, "a loan is rejected if the requested amount is higher than a threshold"). In DPNs, however, the system's control-flow is still defined separately and data objects play a minor role, being statically attached to transitions. In consequence, a conformance checking method with DPNs does not allow to clearly validate whether dynamic resources are evolving as expected, while they are processed by the system, nor how new resource states may affect the overall system execution.

In this paper, we present a conformance checking method between event logs and colored Petri nets (CPN) [11] — a Petri net extension resembling the object-oriented paradigm. In CPNs, tokens carry values, representing object instances of some classes (called *colors*). Besides, arc expressions adjacent to transitions allow to specify how objects are transformed upon activity executions. Our conformance method is based on replaying each trace of an event log on top of a CPN model. When replaying each trace, the distinct observed resources (object instances) are injected as tokens in the model. Then, for each event of a trace, we try to fire a transition associated to the activity executed in the event, and selecting as input tokens the ones which represent the real resources observed in the event. Following such scheme, we explain in this work how our method can detect three kinds of deviations: (1) control-flow deviations caused by the absence of resources (i.e., it is not possible to fire a transition with the resources indicated in the event); (2) rule violations (e.g., according to priority rules on transitions, some resources must be served first); and (3) differences between modeled and real resources regarding their evolution along a trace (for instance, after a transition firing, the resulting values of produced tokens must be equal to their corresponding resources observed in an event).

For this method, we consider a specific class of CPNs with certain restrictions. For instance, all tokens in a model must be unique (e.g., using identifiers).

Also, each token involved in a transition firing must be of a different class. These restrictions come to be natural in various information systems where, for example, objects can be distinguished. In the next sections, we explain in detail these restrictions, describing how they guarantee a correct and efficient replay.
We illustrate our method throughout the paper with the study case of trading systems [9]. These systems receive buy/sell orders from agents to trade securities (company shares), placing them in lists called *order books*. Then, orders in a same book are matched to produce trades. In a system, there can be as many order books as securities are traded (e.g., an order to buy `3` stocks of the company `yandex` is placed in the order book "`yandex`"). Event logs of these systems consist of traces, each of them related to a trading session in an order book (see Fig. 1).

trace	timestamp	activity	buy order				sell order			
			id	tsub	price	qty	id	tsub	price	qty
001	09:13:07.536	submit buy order	Wpl	09:13:07.536	22.00	3				
001	09:13:07.537	new buy order	Wpl	09:13:07.536	22.00	3				
001	09:13:07.544	submit sell order					Wpm	09:13:07.544	19.00	1
001	09:13:07.545	new sell order					Wpm	09:13:07.544	19.00	1
001	09:13:07.565	submit sell order					Wpn	09:13:07.565	21.00	3
001	09:13:07.566	new sell order					Wpn	09:13:07.565	21.00	3
001	09:13:07.581	trade 2	Wpl	09:13:07.536	22.00	2	Wpm	09:13:07.544	19.00	0
001	09:13:07.582	trade 3	Wpl	09:13:07.536	22.00	0	Wpn	09:13:07.565	21.00	1
001	09:13:11.236	discard sell order					Wpn	09:13:07.565	21.00	0

Fig. 1: Log trace of a trading system. Each row shows an activity fired and orders involved, with attributes `id`, arrival time (`tsub`), `price` and quantity (`qty`).

Given a CPN modeling a trading session, and an event log of real sessions (i.e., see Fig. 2), our method detects the following deviations: (1) control-flow errors due to absent resources, e.g., trades occurred with unavailable orders, (2) violation of priority rules when serving orders, and (3) differences between modeled and real produced resources (e.g., an order attribute was incorrectly modified).

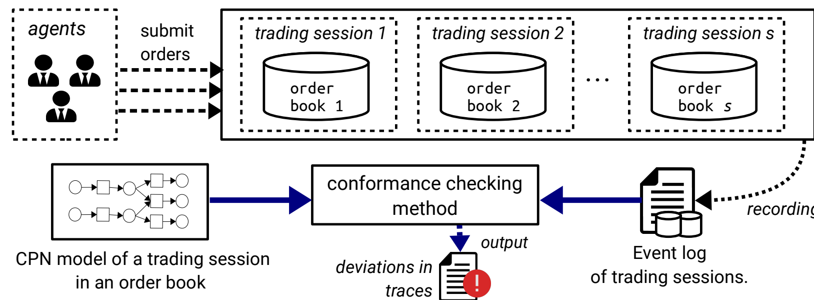


Fig. 2: Validating trading sessions via conformance checking with CPNs.

The remainder of this paper is structured as follows. In Section 2, we introduce CPNs, its formal definition and execution semantics. In Section 3, we describe event logs. In Sections 4 and 5, we describe our conformance checking method, its implementation and experimental validation. In Section 6, we conclude our paper with a discussion on the novelty of our contribution. Also, we briefly mention how our method compares to other conformance proposals, as well as methods within the sphere of data science.

2 Colored Petri Nets

In this section, we present colored Petri nets (CPN), using as an example the model of a trading session in an order book. Then, we introduce the formal definition of CPNs and their execution semantics, as well as we consider some model restrictions. In general, Petri nets consist of two kinds of nodes: *transitions* modeling activities, and *places* storing *tokens*, which model buffers with resources. Pictorially, transitions and places are drawn as boxes and circles respectively. Directed arcs connect input places to transitions, and transitions to output places. Activity executions and resource processing are modeled by *transition firings*, consuming and producing tokens in input and output places respectively.

Colors and tokens. CPNs are an extension of Petri nets, where tokens carry values of some types. Formally, a token is a tuple $(d_1, ..., d_n) \in D_1 \times ... \times D_n$, s.t. $\{D_1, ..., D_k\} \subseteq \mathfrak{D}$ are *data types* from a *data type domain* of interest $\mathfrak{D}$. A cartesian product $D_1 \times ... \times D_n$ between any combination of data types from $\mathfrak{D}$ is a *color*. We denote by Σ the set of all colors that can be obtained from $\mathfrak{D}$. Resembling the object-oriented paradigm, colors denote object classes, whereas tokens are object instances. For example, we define buy and sell order classes with colors $\mathtt{OB} = O_\mathtt{B} \times \mathbb{N} \times \mathbb{R}^+ \times \mathbb{N}$ and $\mathtt{OS} = O_\mathtt{S} \times \mathbb{N} \times \mathbb{R}^+ \times \mathbb{N}$, where $O_\mathtt{B}$ and $O_\mathtt{S}$ are sets of order identifiers, $\mathbb{N}$ is the set of natural numbers, and $\mathbb{R}^+$ is the set of positive real numbers. In Fig. 3, tokens stored in p_1 and p_2 represent buy and sell orders, e.g., the token $(\mathtt{b1}, \mathtt{1}, \mathtt{22.0}, \mathtt{3})$ in place p_1 models a buy order with identifier `b1`, submitted in time `1`, to buy `3` stocks at a price per unit of `22.0`.

Places. Places store tokens of a specific color. We define a function `color`, mapping each place to a color in Σ. In Fig. 3, places p_1 and p_2 are the initial places for incoming buy and sell orders, so $\mathtt{color}(p_1) = \mathtt{OB}$ and $\mathtt{color}(p_2) = \mathtt{OS}$. Places p_3 and p_4 denote buffers of buy/sell orders, received by the platform, whereas places p_5 and p_6 model the buy and sell side of an order book. Places p_7 and p_8 store filled orders that traded successfully, and finally places p_9 and p_{10} store canceled orders.

Arc Expressions. Arcs are labeled with expressions to formally indicate how tokens are processed upon transition firings. We consider a language of expressions $\mathcal{L}$. Each expression is of the form $(e_1, ..., e_n)$ s.t., for each $i \in \{1, ..., n\}$, e_i is either a constant, a variable, or a function. We define a function $\mathcal{E}$ that maps each arc to an expression from $\mathcal{L}$. Let us consider some examples in Fig. 3. The expressions $\mathcal{E}(p_1, t_1) = \mathcal{E}(t_1, p_3) = (\mathtt{o}, \mathtt{ts}, \mathtt{pr}, \mathtt{q})$ in arcs (p_1, t_1) and (t_1, p_3) specify that, when transition t_1 fires, one token in p_1 shall be consumed from place p_1 and transferred (without modifications) to place p_3. This is how we model processing of resources. In such firing, variables in the expression are binded to token values, e.g., $(\mathtt{o}, \mathtt{ts}, \mathtt{pr}, \mathtt{q}) = (\mathtt{b1}, \mathtt{1}, \mathtt{22.0}, \mathtt{3})$. As another example, let us consider transition t_6. It specifies a trade where a buy order is partially filled (`q2` out of `q` stocks were bought), so the order should return with its remainder to the buy side (place p_5). The expression $\mathcal{E}(t_6, p_5) = (\mathtt{o}, \mathtt{ts}, \mathtt{pr}, \mathtt{q} - \mathtt{q2})$ in arc (t_6, p_5) makes such modification, decrementing the buy order's stock quantity by `q2`, s.t. `q2` is the stock quantity of the sell order binded from place p_6.

Transitions and Activity labels. We consider a function Λ, mapping each transition to a label from a finite set $\mathcal{A}$ of activity labels. Thus, as shown in Fig. 3, each transition represents an activity in a trading session. Transitions t_1, t_2 model submission of incoming orders from participants. Transitions t_3, t_4 model insertion of submitted orders in an order book side. Then, a trade may occur between a buy order and a sell order. In particular, transition t_5 (activity `trade1`) models a trade where both orders were filled (all their stocks were bought/sold). Transitions t_6, t_7 (activities `trade2` and `trade3`) model the situation where only one of the orders is filled, whereas the second one is partially filled (returning to the order book). Finally, transitions t_8 and t_9 represent activities to discard orders from the order book.

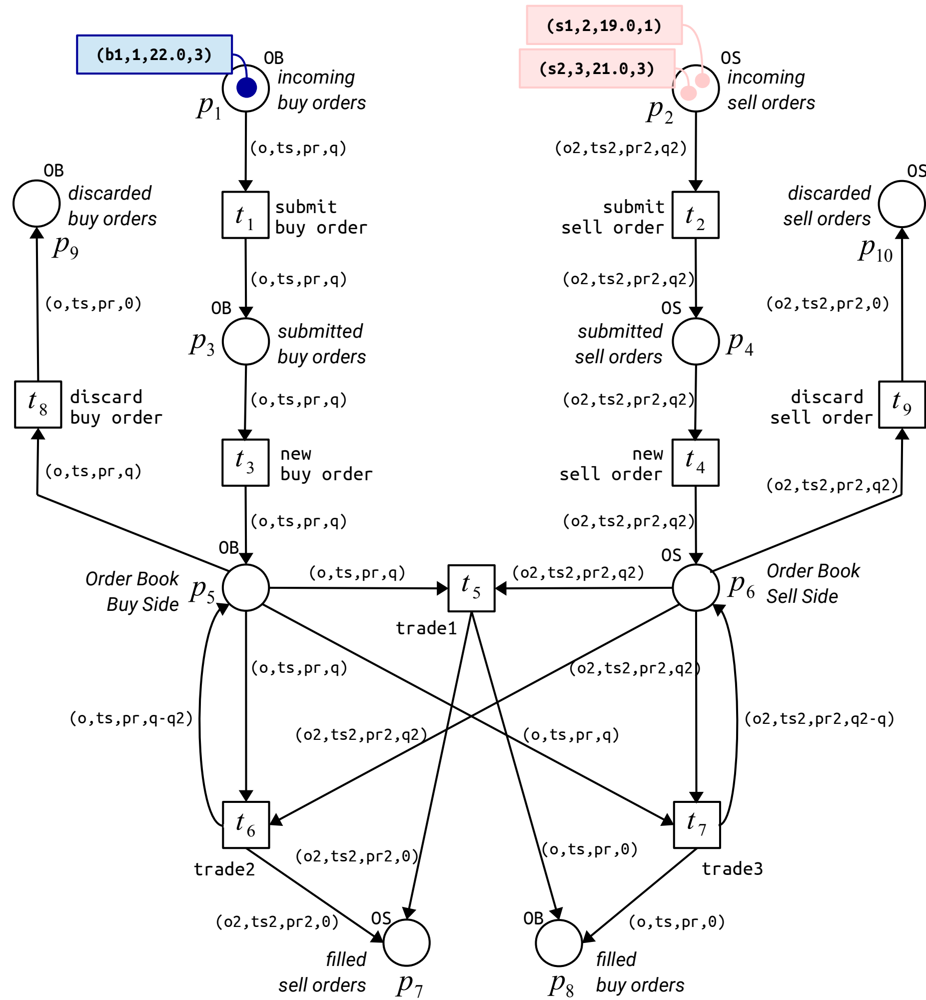


Fig. 3: CPN modeling a trading session in an order book.

Definition 1 (Colored Petri net). *A colored Petri net is a 6-tuple* $CP = (P, T, F, \texttt{color}, \mathcal{E}, \Lambda)$*, where:*

- P *is a finite set of places;*
- T *is a finite set of transitions, s.t.* $P \cap T = \emptyset$*;*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a finite set of directed arcs (called the flow relation);*
- $\texttt{color} : P \rightarrow \Sigma$ *is a place-coloring function, mapping each place to a color in* Σ*, such that* Σ *is a finite set of colors;*
- $\mathcal{E} : F \rightarrow \mathcal{L}$ *is an arc-labeling function, mapping each arc* r *to an expression of a language* $\mathcal{L}$*, s.t.* $\texttt{color}(\mathcal{E}(r)) = \texttt{color}(p)$ *where* p *is a place adjacent to an arc* r*;*
- $\Lambda : T \rightarrow \mathcal{A}$ *is an activity-labeling function, mapping each transition to an element in* $\mathcal{A}$*, s.t.* $\mathcal{A}$ *is a finite set of activity labels,* $\forall t, t' \in T : \Lambda(t) \neq \Lambda(t')$*.*

We consider CPNs with the following restrictions. On the one hand, all tokens are unique, so each place stores a set of tokens (not a multiset). Notice also that only one token can be consumed at once from each input place (e.g., see Fig. 3). On the other hand, for each transition t, each input place of t is of a different color. These restrictions come to be natural in many information systems. As exemplified with the model in Fig. 3, all tokens are unique having distinct identifiers. Also, orders can be modified (e.g., to update their stock size), but cannot disappear when processing them (e.g., all orders in initial places p_1, p_2 must arrive to places p_7, p_8 if they trade all stocks, or to places p_9, p_{10} if they are canceled). In Section 4, we explain how these restrictions guarantee that the conformance checking method performs a correct and efficient replay.

We now define execution semantics of our model. Let $CP = (P, T, F, \texttt{color}, \mathcal{E}, \Lambda)$ be a colored Petri net. A *marking* M is a function, mapping each place $p \in P$ to a set of tokens $M(p)$, according to its color. We denote by M_0 an initial marking. Markings model system states, e.g., in Fig. 3, the initial marking of the net models the start of a simple trading session, with orders yet not submitted and with an empty order book. A *binding* b of a transition $t \in T$ is a function, assigning a value $b(\texttt{v})$ to each variable $\texttt{v}$ occurring in arc expressions adjacent to t. Let be ${}^{\bullet}t$ be a set of input places of a transition $t \in T$. Transition t is *enabled* in marking M w.r.t. a binding b iff $\forall p \in {}^{\bullet}t : b(\mathcal{E}(p,t)) \in M(p)$, that is, each input place of t has at least one token to be consumed. The *firing* of an enabled transition t in a marking M w.r.t. to a binding b yields a new marking M' such that $\forall p \in P : M' = M(p) - \{b(\mathcal{E}(p,t))\} \cup \{b(\mathcal{E}(t,p))\}$.

3 Event Logs

In this section, we now introduce *event logs*, describing how they are structured.

Definition 2 (Event Log). *An event log of is a finite set of traces* $L = \{\sigma_1, ..., \sigma_s\}$ *where, for each* $i \in \{1, ..., s\}$*, a trace* $\sigma_i = \langle e_1, ..., e_m \rangle$ *is a finite sequence of events, s.t.* $m = |\sigma_i|$ *is the trace length.*

Each event e *in a trace is a tuple of the form* $(a, \{r_1, ..., r_k\})$ *where* $a \in \mathcal{A}$ *is an activity label, s.t.* $\mathcal{A}$ *is a finite set of activity labels, and for each* $j \in \{1, ..., k\}$*, we say that* r_j *is a resource involved in the execution of activity* a*.*

Table 1: Example of a trace σ in an event log L of a simple trading session.

event (e)	**activity** (a)	**resources** ($R(e)$)
e_1	`submit buy order`	(`b1, 1, 22.0, 3`)
e_2	`new buy order`	(`b1, 1, 22.0, 3`)
e_3	`submit sell order`	(`s1, 2, 19.0, 1`)
e_4	`new sell order`	(`s1, 2, 19.0, 1`)
e_5	`submit sell order`	(`s2, 3, 21.0, 3`)
e_6	`new sell order`	(`s2, 3, 21.0, 3`)
e_7	`trade 2`	(`b1, 1, 22.0, 2`), (`s1, 2, 19.0, 0`)
e_8	`trade 3`	(`b1, 1, 22.0, 0`), (`s2, 2, 21.0, 1`)
e_9	`discard sell order`	(`s2, 2, 21.0, 0`)

As an example, Table 1 shows a trace σ of an event log L. Each event e in σ indicates which activity was executed and a set of involved resources, e.g., in event e_2, activity `new buy order` was executed, placing order (`b1, 1, 22.0, 3`) in the order book. We introduce function $R(e)$ to return the set of resources involved in an event e. As introduced in Section 1, since our conformance method aims to associate observed resources in an event with tokens in a CPN model, we assume that each resource $r \in R(e)$ is a tuple belonging to some color in Σ, s.t. Σ is the set of all possible colors in a CPN model. With slight abuse of notation, we use $\texttt{color}(r)$ to denote the color of a resource r. For example, $(\texttt{b1}, 1, 22.0, 3)$ in event e_1 is a buy order, so $\texttt{color}((\texttt{b1}, 1, 22.0, 3)) = \texttt{OB}$, where `OB` defines the structure of buy orders, as we exemplified in Section 2.

For each resource $r = (r^{(1)}, ..., r^{(n)})$ in an event $e = (a, R(e))$, its tuple components $r^{(1)}, ..., r^{(n)}$ represent the state of the resource after the execution of activity a. In other words, after executing a, some attributes of r could have been modified. However, we shall assume that the first component of r, i.e., $r^{(1)}$, is the *resource identifier*, which cannot be modified by any activity. For compactness, we denote by $\texttt{id}(r) = r^{(1)}$ the identifier of $r = (r^{(1)}, ..., r^{(n)})$, e.g., $\texttt{id}(r_1) = \texttt{b1}$ for $r_1 = (\texttt{b1}, 1, 22.0, 3)$.

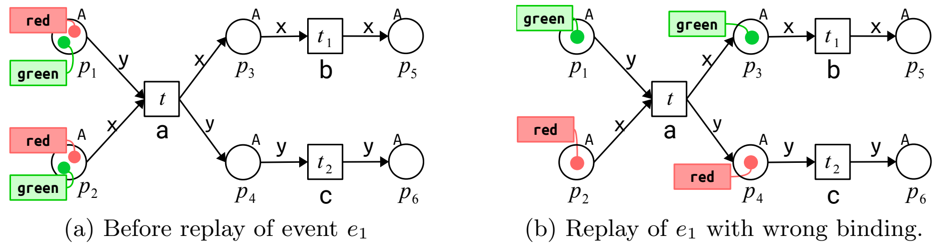
By using identifiers, we consider that resources can be distinguished (as we assumed with tokens in a model). This allows us to identify the distinct objects involved in a trace, e.g., in Table 1 we identify three distinct resources: one buy order `b1`, and two sell orders `s1` and `s2`. Also, this allows us to track how a resource is modified. For example, let us consider the order `s2`, which initially had `3` stocks in event e_5. In event e_8 its stock size was reduced to `1` after executing a trade, and then in event e_9 its stock size went to `0` after the order was discarded.

Let $r = (r^{(1)}, ..., r^{(n)})$ be a resource. For $j \in \{1, ..., n\}$, we have that each resource attribute $r^{(j)}$ can be accessed using an attribute name. Also, all resources of the same color share the same set of attribute names. For instance, for the color of buy orders `OB`, we consider the attribute names $\{\texttt{id}, \texttt{tsub}, \texttt{price}, \texttt{qty}\}$. We define a *member access function* `#`, such that given a resource $r = (r^{(1)}, ..., r^{(n)})$ and the name of the jth-component, it returns the value of $r^{(j)}$, i.e., $\#(r, \texttt{name}_j) = r^{(j)}$. For simplicity, we use notation $\texttt{name}_j(r)$ instead of $\#(r, \texttt{name}_j)$. For example, for $r = (\texttt{b1}, 1, 22.0, 3)$, we have that $\texttt{tsub}(r) = 1$, $\texttt{price}(r) = 22.0$, and $\texttt{qty}(r) = 3$.

4 Conformance Checking using Colored Petri Nets and Event Logs

In this section, we present a conformance checking method for CPNs (cf. Def. 1) and event logs (cf. Def. 2). Before describing our method, we first explain the restrictions that CPNs must satisfy to guarantee a correct and efficient replay.

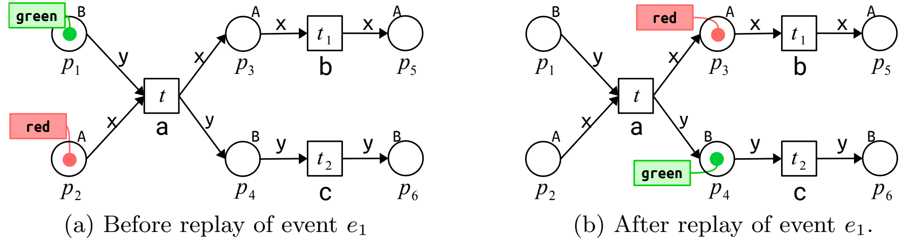
Model restrictions. As described in Section 2, tokens (in all model markings) must be unique (to have distinct identifiers), and for each transition t, all input places of t are of different colors. We illustrate the need of such restrictions with the next example. Consider the replay of trace $\sigma = \langle e_1, e_2 \rangle = \langle (\mathtt{a}, \{\mathtt{green}, \mathtt{red}\}), (\mathtt{b}, \{\mathtt{red}\}) \rangle$ on the CPN of Fig. 4(a). All places and variables `x`, `y` in arcs are of a same color `A`. The CPN breaks the restrictions: there are clones with identifiers `green` and `red`, and t has input places of the same color.



(a) Before replay of event e_1 (b) Replay of e_1 with wrong binding.

Fig. 4: Replay of trace σ on a CPN model without restrictions (a). After replaying event e_1 with wrong binding b_1, event $e_2 = (\mathtt{b}, \{\mathtt{red}\})$ cannot be replayed (b).

To replay event $e_1 = (\mathtt{a}, \{\mathtt{green}, \mathtt{red}\})$, we may fire t ($\Lambda(t) = \mathtt{a}$) with binding $b_1 = \langle \mathtt{x} = \mathtt{green}, \mathtt{y} = \mathtt{red} \rangle$ or $b_2 = \langle \mathtt{x} = \mathtt{red}, \mathtt{y} = \mathtt{green} \rangle$. Consider to fire t with binding b_1, yielding the marking of Fig. 4(b). Now, event $e_2 = (\mathtt{b}, \{\mathtt{red}\})$ cannot be replayed. The event does not showcase a real deviation, but a wrong binding selection: if t fires with binding b_2, then e_2 can be replayed. Backtracking (return to previous events and to try other bindings) is needed in such situations to assert if a deviation has been found, or instead previous firings with wrong bindings blocked the replay. Backtracking may be computationally expensive. Instead, let us consider now the model of Fig. 5(a) where restrictions are complied.



(a) Before replay of event e_1 (b) After replay of event e_1.

Fig. 5: CPN model with the restrictions: the `green` value now is of a color `B`; after replaying e_1 (with the only allowed binding), $e_2 = (\mathtt{b}, \{\mathtt{red}\})$ can be replayed.

Now, to replay $e_1 = (\mathtt{a}, \{\mathtt{green}, \mathtt{red}\})$, there is only one binding associated to the observed resources in event e_1, that is, $b = \langle \mathtt{x} = \mathtt{red}, \mathtt{y} = \mathtt{green} \rangle$. No other binding may be selected as `green` and `red` values are now forced to come from separate sources. After firing t, $e_2 = (\mathtt{b}, \{\mathtt{red}\})$ is guaranteed to be replayed, so backtracking is avoided. In this way, our method associates each observed resource to a token from a specific input place (such token cannot appear in other place), and thus only one binding can satisfy the event replay. Thus, with these restrictions, the method guarantees a correct and less expensive replay.

Conformance Checking Method. We proceed now to explain our conformance checking method, based on individual replay of each trace on top of a CPN. As introduced before, the method seeks to fire transitions, labeled with activities indicated in the events, and selecting input tokens according to resources observed in the events. If the latter is not possible or, as we will present, other deviations are found, the trace replay is stopped. As output, this method returns a list of non-fitting traces (not completely replayed), and a fitness metric. This metric indicates a degree of conformance between a CPN and an event log.

Algorithm 1: Conformance Checking using CPNs

Input: $CP = (P, T, F, \mathtt{color}, \mathcal{E}, \Lambda)$, a CPN with an empty initial marking;
$P_0 \subseteq P$, a non-empty set of initial places;
L, an event log (finite set of traces);

Output: $L_{\mathtt{error}}$ - set of non-fitting traces;
`fitness` - degree of conformance;

1 $L_{\mathtt{error}} \leftarrow \emptyset$; $\mathtt{fitness} \leftarrow 0$;
2 **foreach** $\sigma \in L$ **do**
3 $\quad M \leftarrow \emptyset$; $M \leftarrow \mathtt{populateInitialPlaces}(P_0, R(\sigma))$;
4 $\quad$ **foreach** $e = (a, R(e))$ in σ **do**
5 $\quad\quad t \leftarrow \mathtt{selectTransition}(a)$;
6 $\quad\quad$ **if** $\mathtt{controlFlowDeviation}({}^\bullet t, M, R(e))$ **then:** $\mathtt{add}(\sigma, L_{\mathtt{error}})$; **break**;
7 $\quad\quad b \leftarrow \mathtt{selectBinding}(t, M, R(e))$;
8 $\quad\quad$ **if** $\mathtt{ruleViolation}(t, M, b)$ **then:** $\mathtt{add}(\sigma, L_{\mathtt{error}})$; **break**;
9 $\quad\quad M \leftarrow \mathtt{fire}(t, M, b)$;
10 $\quad\quad$ **if** $\mathtt{corruptedResources}(t^\bullet, M, R(e))$ **then:** $\mathtt{add}(\sigma, L_{\mathtt{error}})$; **break**;
11 $\quad$ **endfor**
12 **endfor**
13 $\mathtt{fitness} \leftarrow 1 - (|L_{\mathtt{error}}| \ / \ |L|)$;
14 **return** $(L_{\mathtt{error}}, \mathtt{fitness})$;

Initial setting. Algorithm 1 presents our method whose input is a CPN with an empty initial marking, an event log L, and a set of initial places $P_0 \subseteq P$. At the start of each trace replay, each place in P_0 is populated with the distinct resources in σ, according to its color (function `populateInitialPlaces`).

Let us consider the replay of σ in Table 1 on the CPN of Fig. 3. The CPN shows a marking after the distinct resources in σ were placed in the initial places: buy order `b1` in p_1, and sell orders `s1` and `s2` in p_2. For each resource to insert as a token in an initial place, we set its token values according to its first occurrence in a trace, e.g., `b1` is placed with values $(\texttt{b1}, 1, 22.0, 3)$ as shown in event e_1.

Control-flow deviation due to the absence of input resources. In a trace σ, after setting the model marking according to the distinct resources in σ, we start to replay σ on the CPN. For each event $e = (a, R(e))$ in σ, we try to fire a transition t, s.t. $\Lambda(t) = a$. To fire, we check if, in a current marking M, each resource involved in e is contained in an input place of t. Let $^\bullet t$ be the set of input places of t. To this aim, we check the truth value of the next formula:

$$\forall p \in {}^\bullet t : \exists! r \in R(e)\ \exists (d_1, ..., d_n) \in M(p) : \texttt{id}(r) = d_1 \wedge \texttt{color}(r) = \texttt{color}(p)$$

The function `controlFlowDeviation` checks if the previous formula evaluates to false. If so, the replay is stopped, e.g., some resource in $R(e)$ is not available in an input place. Otherwise, a binding is selected (function `selectBinding`), s.t. a token $(d_1, ..., d_n)$ will be consumed from each input place p, i.e., $b(\mathcal{E}(p,t)) = (d_1, ..., d_n)$, and each token corresponds to a resource r in $R(e)$, i.e., $\texttt{id}(r) = d_1$. For example, let us consider the replay of a trace sigma σ'_1 on the CPN of Fig. 3. Let us assume that σ'_1 consists of the first six events of Table 1 (submission of buy order `b1` and sell orders `s1`,`s2`) plus the two events shown below.

e_7	`discard sell order`	(s1, 2, 19.0, 0)
e_8	`trade 2`	(b1, 1, 22.0, 2), (s1, 2, 19.0, 0)

Event e_8 (`trade 2` between `b1` and `s1`) will not be replayed as `s1` was discarded in e_7 (moved to place p_{10}). Hence, `s1` is not anymore available in the sell side (place p_6). Clearly, a trade cannot be executed with a canceled order. In this way, we can detect deviating events with unavailable resources.

Rule violations. Let b be the selected binding to fire t according to the resources in $R(e)$. For each input place p of t, we want to check if the token to consume is the one that should be selected (among all possible ones in p) according to some rule. For example, in trading systems, it is mandatory to know if a priority rule is being complied, e.g., a buy order with highest price must trade before other buy orders. Thus, we define a *marking dependent rule* $\Phi(t)$ as follows:

$$\Phi(t) \equiv \bigwedge_{\forall p \in {}^\bullet t} \phi_p(M(p), b(\mathcal{E}(p,t)))$$

where each $\phi_p(M(p), b(\mathcal{E}(p,t)))$ is a local rule in place p. In Algorithm 1, we set $\texttt{ruleViolation}(t, M, b) \equiv \neg\Phi(t)$. Before firing t, if a rule $\phi_p(M(p), \mathcal{E}(p,t))$ is violated, the trace replay is stopped. Otherwise, if all rules are complied or no rule was defined for transition t, then t fires with selected binding b. For the CPN of Fig. 3, let us assign the rule below to transitions t_5, t_6, t_7 (trade activities).

$$\Phi(t) \equiv \phi_{\texttt{BUY}}(M(p_5), r_1)\ \wedge\ \phi_{\texttt{SELL}}(M(p_6), r_2).$$

$$\phi_{\texttt{BUY}}(M(p_5), r_1) \equiv \forall_{(\texttt{o,ts,pr,q}) \in M(p_5)\ \texttt{id}(r_1) \neq \texttt{o}} : (\texttt{price}(r_1) > \texttt{pr}) \\ \vee (\texttt{price}(r_1) = \texttt{pr} \wedge \texttt{tsub}(r_1) < \texttt{ts})$$

$$\phi_{\texttt{SELL}}(M(p_6), r_2) \equiv \forall_{(\texttt{o,ts,pr,q}) \in M(p_6)\ \texttt{id}(r_2) \neq \texttt{o}} : (\texttt{price}(r_2) < \texttt{pr}) \\ \vee (\texttt{price}(r_2) = \texttt{pr} \wedge \texttt{tsub}(r_2) < \texttt{ts})$$

where r_1 and r_2 are the buy and sell orders to consume. The local rule $\phi_{\texttt{BUY}}$ for place p_5 states that r_1 must be the order with highest price (or with earlier submitted time than other order with same highest price). The local rule $\phi_{\texttt{SELL}}$ for place p_6 is defined similarly, but stating that r_2 must be the order with lowest price. Notably, $\phi_{\texttt{BUY}}(M(p_5), r_1) \wedge \phi_{\texttt{SELL}}(M(p_6), r_2)$ is a price-time priority rule, that trading sessions must comply. For instance, let us consider the replay of the trace in Table 1 on CPN of Fig. 3. It can be observed that this rule is being complied when executing trades, e.g., sell order `s1` is served before order `s2`. For other trace, if the rule is not complied, the replay of that trace is stopped.

Checking differences between modeled and real produced resources. We aim to exploit the fact that each event $e = (a, R(e))$ has information about the new state of each resource after executing a. Recall that each resource in $R(e)$ could have been modified by a. Let us consider the firing of a transition t w.r.t a selected binding b, and yielding a new marking M. Then, we proceed to check whether each resource in real life, after executing a, was modified as performed in the model, after firing t. Let $t^\bullet$ be the set of output places of t. Then, we verify if the following formula is satisfied:

$$\forall p \in t^\bullet : \exists! r \in R(e)\ \exists (d_1, ..., d_n) \in M(p) : r = (d_1, ..., d_n) = b(\mathcal{E}(t, p))$$

In Algorithm 1, the function `corruptedResources` checks if the previous formula evaluates to false. If so, the replay is stopped, i.e., some resource in $R(e)$ was not modified as indicated by the output arc expressions of t. For example, let us consider the replay of a trace σ_2' on the CPN of Fig. 3. Let us assume that σ_2' consists of the first six events of Table 1 plus the event shown below.

e_7	trade 2	(b1, 1, 22.0, 1), (s1, 2, 19.0, 0)

Before the execution of event e_7, we recall that orders `b1` and `s1` have the following states: $(\texttt{b1}, 1, 22.0, 3)$ and $(\texttt{s1}, 2, 19.0, 1)$. After replaying event e_7 in the CPN, the token `b1` is transformed to $(\texttt{b1}, 1, 22.0, 2)$. As specified by the arc expression $\mathcal{E}(t_6, p_5)$, the stock size of `b1`, which is `3`, was decremented by the stock size `s1`, which is `1`. Thus, the resulting stock size of `b1` is `2`. However, event e_7 states that after trading the stock size of order `b1` is `1`. Evidently, the stock size of `b1` was corrupted when trading. For such a case, the trace replay is stopped.

The output of Algorithm 1 is a set of non-fitting traces $L_{\texttt{error}}$, e.g., traces with any kind of the deviations explained before. Also, the algorithm computes a *fitness metric* (a ratio of completely replayed traces) as follows: $\texttt{fitness} = 1 - (|L_{\texttt{error}}| \ / \ |L|)$ where $L_{\texttt{error}} \subseteq L$. Since $|L| \geq |L_{\texttt{error}}| \rightarrow \texttt{fitness} \in [0, 1]$.

We close this section with a brief analysis on the time complexity of our method. Whilst we do not carry out a deeper study, for example, using asymptotic notation, we do identify the crucial parameters that mainly influence the time performance of our algorithm. Let $CP = (P, T, F, \texttt{color}, \mathcal{E}, \Lambda)$ be a colored Petri net and $L = \{\sigma_1, ..., \sigma_s\}$ be an event log, as described in Definitions 1 and 2. Now, let us examine the required operations to replay a single event $e = (a, R(e))$ of a trace σ (i.e., see lines 4-10 of Algorithm 1). On the one hand, these operations seek to fire a transition t s.t. $\Lambda(t) = a$, which it may require up to $|T|$ transitions to visit. On the other hand, a common term of these functions is to compare each resource in $R(e)$ against its corresponding token in an input place of t and, after firing, in an output place of t (e.g., when checking availability of resources). Both, the sets of input and output places of t are bounded by $|P|$. Thus, it can be inferred that the number of steps required to replay a single event is in the order of magnitude $|T| + (|R(e)| \cdot |P|)$. With this term, it is clear to identify that the computing time required for replaying an event is mainly affected by the number of places $|P|$ in a CPN and the number of observed resources $|R(e)|$ in an event. Afterwards, it is easy to see that this event replay routine will be repeated at most $|\sigma| \cdot |L|$ times, s.t. $\forall \sigma' \in L : |\sigma| \geq |\sigma'|$, that is, $|\sigma|$ is the maximum number of events per trace, whereas $|L|$ is the number of traces in the event log.

5 Implementation and Experimental Validation

In this section, we describe the implementation and evaluation of our conformance method using colored Petri nets. We developed this implementation in Python programming language. We have carried out experimental works with both real and artificially generated event logs, which allow us to show the practical value of our method for detecting system deviations. The implementation and all material of our experiments are available in our project repository [7].

Our solution is supported by a Python library called SNAKES [19]. This library facilitates the prototyping of high-level classes of Petri nets, including CPNs. This allows us to instantiate CPN models as Python objects, which can be used as input to our method. Fig. 6 illustrates the organization of our prototypical implementation. Users of our solution simply need to invoke a program called the "conformance checker". This program receives three parameters: an option indicating the conformance method to use (e.g., replay with CPNs), an event log stored in a log repository, and a Petri net model stored in a model repository. This generic organization allows us to seamlessly extend our solution, incorporating other conformance methods and models of our research.

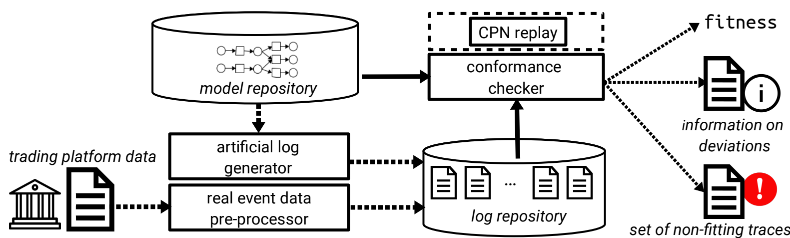


Fig. 6: Organization of our prototypical implementation.

```
=============== CONFORMANCE RESULTS ================
Total number of traces: 4
Non-fitting traces: 3

Fitness : 0.2500
Control-flow deviations detected: 1
Rule violations deviations detected: 1

Resource corruptions detected: 1

Deviations written in file: deviations_14081.csv
Non-fitting traces cloned to file: nf_traces_14081.csv
====================================================
```

Fig. 7: Output summary resulting after the execution of our method.

Upon execution of our method, the program provides command-line messages, indicating resulting metrics, i.e., fitness. Besides, it generates two files: a file consisting of a set of non-fitting traces (i.e., traces that were not completely replayed) and a file with specific information of trace deviations: in which event a trace replay was stopped, which kind of deviation occurred, and comments that may help engineers to localize failures. For example, Fig. 7 shows a resulting message fragment after replaying an event log, where three traces suffered from the kinds of deviations explained in the previous section (e.g., control-flow deviation, rule violation, or resource corruption). In addition, Fig. 8 shows a fragment of an output file specifying the deviating events in each non-fitting trace.

TRACE	EVENT OCCURRED AT	ACTIVITY	DEVIATION	ADDITIONAL INFORMATION
1111037	18-02-2019T09:13:07.581	trade2	CONTROL-FLOW	resource with id: s1 is not available.
1111038	18-02-2019T09:13:07.581	trade2	RULE-VIOLATION	resource with id: s2 does not have priority over other resources in the same place
1111039	18-02-2019T09:13:07.581	trade2	RESOURCE-CORRUPTED	resource with id: b1 has observed state: ('b1',1,22.0,3) but expected state was: ('b1',1,22.0,2)

Fig. 8: Fragment of specific deviation diagnostics generated by our method.

Experiment with a real event log. We conducted an experimental work using an event log from a trading platform. This log was obtained by pre-processing data from a real trading system. The data is a recorded set of Financial Information Exchange (FIX) protocol messages [8]. These messages were exchanged by participants and a platform during trading sessions, so they encapsulate activities executed by both agents and the platform. We developed a pre-processor in Java which extracts such event log from this set of messages (also available via [7]). The event log consists of individual traces, each of them related to a trading session in an order book. In particular, the expected behavior in each of these trading sessions is specified by the CPN model shown in Fig. 3.

Table 2: Event log characteristics and obtained conformance results.

Event log characteristics		*Conformance results*	
Number of pre-processed FIX Messages	552935	Number of non-fitting traces	8
Number of traces (trading sessions)	73	Fitness	0.890
Total number of events in the log	2259	Rule violations detected	1
Average number of events per trace	30	Resource corruptions detected	7

Table 2 shows characteristics of the event log. Also, it presents conformance results, resulting from the execution of our method using the mentioned log and the CPN of Fig 3. The table shows the number and kinds of deviations detected. The fact that the majority of traces were completely replayed evidences that most of the trading sessions comply with the model. Regarding the non-fitting traces, the obtained file with information about deviations indicate that the rule violations and resource corruptions originated in trade activities. The obtained information may support experts to confirm whether a failure is occurring in those activities, or instead the CPN model should be slightly refined.

Experiments with artificial event logs. In addition to the previous experiment, we considered to test our method with slightly more stressed scenarios, where system runs may be hampered by sporadic (yet critical) deviations. More precisely, we aimed to evaluate the impact of each of the kinds of deviations previously introduced when a system is managing a certain number of resources (e.g., buy or sell orders). Thus, we considered an experiment where we slightly modified the correct specification model (the CPN of Fig. 3), obtaining three "incorrect" variants (see Table 3). These variants can be seen as instances of a correct trading platform, but sporadically suffering from one kind of deviation.

Table 3: Description of "incorrect" variants to generate artificial event logs.

Model variant	*Description of the deviation that may occur*
System `A`	*Control-flow deviation*: with 5% of probability, activities `discard buy order` or `discard sell order` do not cancel orders, so orders keep in the order book (places p_5 and p_6 in Fig. 3), and may continue to trade.
System `B`	*Rule violation*: with 2% of probability, and upon execution of activities `trade1`, `trade2`, or `trade3`, this variant does not respect the priority rule, i.e., buy/sell orders with highest/lowest prices are not served first.
System `C`	*Resource corruption*: With 5% of probability, stock quantities of buy/sell orders change to 0 when placing them in the order book (misbehavior of activities `new buy order` or `new sell order` of Fig. 3).

We built models of these variants (using SNAKES) to generate artificial event logs, thereby representing observed behavior of the "incorrect" instances. Recall that the fitness metric considered in this work relates to the number of completely replayed traces. We used this metric to assess the probability that a system run can be jeopardized by the occurrence of a deviation. Table 4 presents results of this experiment. Each cell indicates the average fitness value after executing our method, between the correct model (Fig. 3) and ten artificial event logs from an 'incorrect" variant, and with a certain number of resources per class (e.g., number of buy orders and sell orders). All event logs consist of 500 traces.

Table 4: Fitness between the CPN (Fig. 3) and logs of each incorrect variant.

Model variant	*5 resources per class*	*25 resources per class*
System `A`	0.6436	0.05854
System `B`	0.9816	0.8569
System `C`	0.6196	0.05852

Let us consider variants `A` and `C`. Albeit the probability of a deviation is expected to be low (i.e., 5%), the average ratio of correct traces is only above 60%. Also, when the number of resources processed by a system inscreases, the fitness value notably decreases. This is an expected pattern since when considering more resources during a system run, the length of traces are enlarged (i.e., more activities processing resources), and the probability of one deviation is increased.

6 Discussion and Conclusion

Conformance checking methods detect deviations in system processes using event logs and process models. These methods use replay [18], as seen in this paper, or alignments [2], which relate traces with model executions. A limitation of these methods is that they only focus on control-flow, i.e., whether system's activities comply a causal ordering. Whilst certain proposals tackle this issue with slightly enriched models (e.g., data Petri nets) [12, 13], they use notations whose backbone does not allow to describe transformation of objects. Tackling such problem, we presented in this paper a conformance method using colored Petri nets — an extension where tokens carry data of some classes (colors). Arc expressions specify how tokens are transformed upon transition firings. The method replays events on a CPN, firing transitions labeled with an event's activity, and choosing as input tokens the ones related to observed resources in the event. We showed deviations that can be detected: errors due to absent resources, rule violations or resource corruptions. We provided a prototype and experiments [7].

To make feasible the use of CPNs, we considered restrictions, e.g., tokens must be unique and cannot be destroyed. Also, all input places of a transition must be of different colors. We described how the latter allows us to avoid backtracking when replaying a trace. Interestingly, by looking the CPN in Fig. 3, this restricted model resembles a union of workflow nets [1], a Petri net class used in process mining. Each "lane" processing a resource class can be seen as a workflow net, and some transitions allow interaction between these workflows.

Through the paper, we illustrated our method using trading systems [9]. In this regard, a direction of our research focuses on the definition of formal models to analyze different aspects of these systems [4–6]. However, it is easy to see that the method provided in this work can be easily applied in other domains.

There are similar approaches for checking system's compliance in the broader field of data science, e.g., passive analysis [10] or action rules [17]. Also, event logs with resources are similar to (multi-dimensional) sequence databases in data mining [16]. However, such methods do not use formal models such as Petri nets. Instead, we showed how rules to comply can be exhaustively described into a single model, so that traces can be systematically compared against such model.

For future work, we want to enhance our method, replaying traces after one deviation is found. When a transition cannot fire due to an absent resource, we cannot consider injection of "missing" tokens (as proposed in [1]) since we would violate resource uniqueness in a model. Instead, we plan to study an approach based on "moving" tokens. Also, we plan to research how this method may relate to another work, where we check conformance of system-agent interactions [14].

References

1. van der Aalst, W.: Process Mining: Data Science in Action. Springer (2016)
2. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B., van der Alst, W.: Measuring Precision of Modeled Behavior. Information Systems and e-Business Management **13**(1), 37–67 (2015)
3. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking: Relating Processes and Models. Springer (2018)
4. Carrasquel, J.C., Lomazova, I.A.: Modelling and Validation of Trading and Multi-Agent Systems: An Approach Based on Process Mining and Petri Nets. In: van Dongen, B., Claes, J. (eds.) Proc. of the ICPM Doctoral Consortium. CEUR, vol. 2432 (2019)
5. Carrasquel, J.C., Lomazova, I.A., Itkin, I.L.: Towards a Formal Modelling of Order-driven Trading Systems using Petri Nets: A Multi-Agent Approach. In: Lomazova, I.A., Kalenkova, A., Yavorsky, R. (eds.) Modeling and Analysis of Complex Systems and Processes (MACSPro). CEUR, vol. 2478 (2019)
6. Carrasquel, J.C., Lomazova, I.A., Rivkin, A.: Modeling Trading Systems using Petri Net Extensions. In: Köhler-Bussmeier, M., Kindler, E., Rölke, H. (eds.) Int. Workshop on Petri Nets and Software Engineering (PNSE). CEUR, vol. 2651 (2020)
7. Conformance Checking with Colored Petri Nets: Project Repository (Github): `https://github.com/jcarrasquel/hse-uamc-conformance-checking`
8. FIX Community - Standards: `https://www.fixtrading.org/standards/`
9. Harris, L.: Trading and Exchanges: Market Microstructure for Practitioners. Oxford University Press (2003)
10. Itkin, I., Yavorskiy, R.: Overview of Applications of Passive Testing Techniques. In: Lomazova, I., Kalenkova, A., Yavorsky, R. (eds.) Modeling and Analysis of Complex Systems and Processes (MACSPro). CEUR, vol. 2478 (2019)
11. Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer (2009)
12. de Leoni, M., Munoz-Gama, J., Carmona, J., van der Aalst, W.: Decomposing Alignment-Based Conformance Checking of Data-Aware Process Models. In: Meersman, R., Panetto, H., Dillon, T., Missikoff, M., Liu, L., Pastor, O., Cuzzocrea, A., Sellis, T. (eds.) On the Move to Meaningful Internet Systems (OTM). pp. 3–20. LNCS, Springer (2014)
13. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.: Balanced multi-perspective checking of process conformance. Computing **98**(4), 407–437 (2016)
14. Mecheraoui, K., Carrasquel, J.C., Lomazova, I.A.: Compositional Conformance Checking of Nested Petri Nets and Event Logs of Multi-Agent Systems. CoRR **abs/2003.07291** (2020)
15. Murata, T.: Petri nets: Properties, analysis and applications. Proc. of the IEEE **77**(4), 541–580 (1989)
16. Pinto, H., Han, J., Pei, J., Wang, K., Chen, Q., Dayal, U.: Multi-Dimensional Sequential Pattern Mining. In: Int. Conference on Information and Knowledge Management. pp. 81–88. ACM (2001)
17. Ras, Z.W., Wyrzykowska, E., Tsay, L.: Action rules mining. In: Encyclopedia of Data Warehousing and Mining, pp. 1–5. IGI Global (2009)
18. Rozinat, A., van der Alst, W.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems **33**(1), 64–95 (2008)
19. SNAKES - Petri net library: `https://snakes.ibisc.univ-evry.fr/`
20. van der Aalst, W., van Hee, K., van der Werf, J., Verdonk, M.: Auditing 2.0: Using Process Mining to Support Tomorrow's Auditor. Computer **43**(3), 90–93 (2010)