Searching for Deviations in Trading Systems: Combining Control-Flow and Data Perspectives*

Julio C. Carrasquel and Irina A. Lomazova

HSE University
Myasnitskaya ul. 20, 101000 Moscow, Russia
`jcarrasquel@hse.ru`, `ilomazova@hse.ru`

Abstract. Trading systems are software platforms that support exchange of securities (e.g., company shares) between participants. In this paper, we present a method to search for deviations in trading systems by checking conformance between colored Petri nets and event logs. Colored Petri nets (CPNs) are an extension of Petri nets, a formalism for modeling of distributed systems, which allow to describe an expected causal ordering between system activities and how data attributes of domain-related objects (e.g., orders to trade) must be transformed. Event logs consist of traces corresponding to runs of a real system. By comparing CPNs and event logs, different types of deviations can be detected. Using this method, we report the validation of a real-life trading system.

Keywords: process mining; conformance checking; Petri nets; colored Petri nets; trading systems

1 Introduction

Trading systems are software platforms that support exchange of securities (e.g., company shares) between participants [9]. In these systems, orders are submitted by users to indicate what securities they aim to buy or sell, how many stocks and their price. Investors buy securities with promising returns, whereas companies sell their shares to gain capital. These are some of the reasons why trading systems are a vital element in global finances, requiring software processes in these systems to guarantee their correctness. Among these processes, a crucial one is the management of orders in order books. Order books are two-sided priority lists where buy orders and sell orders that aim to trade the same security are matched for trading. A trading system must handle and match orders in these books according to its specification. Nonetheless, trading systems may be prone to deviate from their specification due to software errors or malicious users. This is why the validation of processes in trading systems, such as the management of orders, is a task of utmost importance. In this light, domain experts constantly seek for novel ways to detect system *deviations*, that is, to localize precise differences between a real system and its specification [10].

---

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

To detect deviations in trading systems, we have proposed the use of *conformance checking* [2]. Conformance checking is a family of process mining techniques to search for differences between formal models describing *expected behavior* of processes and event logs that record *real behavior* of such processes [1]. Event logs consist of traces related to runs of processes; a trace is an ordering sequence of events, where each event indicates an activity executed. To model expected behavior, we consider *Petri nets* — a well-known formalism for modeling and analysis of distributed systems [13]. Petri nets allow to describe the *control-flow* perspective of processes, that is, activities and their causal ordering (e.g., "a trade between two orders is preceded by the submission of both orders").

For trading systems, models should describe not only control-flow, but also how *data attributes* of objects such as orders change upon the execution of activites (e.g., "stocks of a sell order decrease by the number of stocks sold in a trade"). We resort to colored Petri nets (CPNs) to combine both control-flow and data perspectives [11]. CPNs are an extension of Petri nets where tokens progressing through the net carry data from some domains (referred to as "colors"). CPNs allow us to describe how trading systems handle objects (represented by tokens) and how their data attributes are transformed. This is an advantage over data-aware Petri net models used in other conformance methods, which do not directly relate data to objects [12]. In [3–5] we presented how CPNs, as well as other Petri net extensions, allow to model different processes in trading systems.

We then developed conformance methods to replay traces of trading system processes on CPNs modeling specifications. Replay consists in the execution of a model according to the information in events of a trace [14]. Deviations are found when a model cannot be executed as an event indicates. In [6] we focused on deviations related to control-flow, and proposed a strategy to force the execution of CPNs if deviations are found. Conversely, in [7] we use replay to analyze object data attributes. Particularly, we check if data attributes of objects are transformed by a real system in the same way that its model does.

In this paper, we present a conformance method which combines both approaches presented in [6, 7]. The following kinds of deviations in a system can be detected when replaying a system's trace on a CPN that models the system specification: (*i*) *control-flow deviation:* the real system invoked an activity involving certain objects, by skipping some activities that should have processed before such objects; (*ii*) *priority rule violation:* an object was served before other objects with higher priority; (*iii*) *resource corruption:* object attributes were not transformed as the model specifies; (*iv*) *non-proper termination:* an object was not fully processed by the real system. The method returns a file with precise information about all deviations detected. We developed a prototypical implementation of the method, which we use to validate the management of orders in a real-life trading system. An experiment with artificial data is also reported.

The rest of this paper is structured as follows. Sections 2 and 3 introduce the CPN models and event logs used in our method. Section 4 presents the conformance method. Section 5 reports the prototype and experiments conducted. Finally, Section 6 presents the conclusions.

2 Colored Petri Nets

Petri nets [13] are bipartite graphs consisting of two kinds of nodes: places and transitions. Places (drawn as circles) represent resource buffers, conditions, or virtual/physical locations. Transitions (drawn as boxes) account for system activities. Places may store tokens, which represent control threads, resources, etc. Transitions consume tokens from input places and produce them in output places. In particular, we consider colored Petri nets (CPNs) where tokens carry data belonging to some data domains ("colors") [11]. As an example, Fig. 1 depicts a CPN modeling the specification of a trading system managing buy orders and sell orders in one order book. Places p_1 and p_2 are sources for incoming buy and sell orders; p_3 and p_4 are buffers for submitted orders; p_5 and p_6 model the buy/sell side of the order book, whereas p_7 and p_8 are sinks for orders that traded or were canceled. Transitions t_1 and t_2 model submission of orders by users; t_3 and t_4 model insertion of orders in order book sides. Then, trades may occur between orders. Transition t_5 (activity `trade1`) models a trade where both orders are filled (all stocks were bought/sold); t_6, t_7 (activities `trade2` and `trade3`) model the cases where only one of the orders is filled, whereas the second one is partially filled (returning to the order book). Transitions t_8 and t_9 represent cancellation of orders.

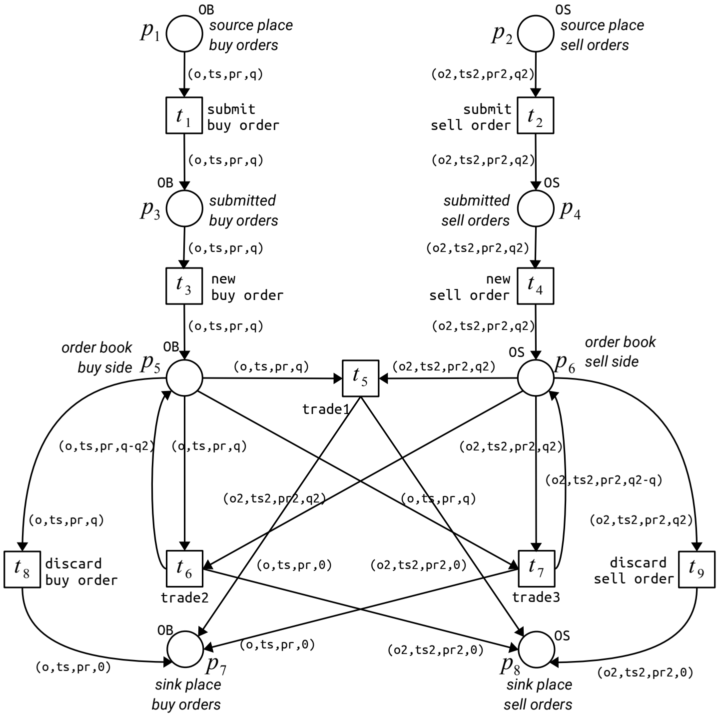


Fig. 1: CPN model of a trading system operating one order book.

Let $\mathfrak{D}$ be a finite set of *data domains*. A Cartesian product $D_1 \times ... \times D_n$, $n \geq 1$, between a combination of data domains $D_1, ..., D_n$ from $\mathfrak{D}$ is called a *color*. Σ denotes the set of all possible colors defined over $\mathfrak{D}$. A token is a tuple $(d_1, ..., d_n) \in \mathtt{C}$ such that $\mathtt{C}$ is a color in Σ. In the CPN of Fig. 1, we consider two 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 identifiers for buy orders and sell orders, $\mathbb{N}$ is the set of natural numbers (including zero) and $\mathbb{R}^+$ is the set of positive real numbers. These colors are used to represent buy/sell orders equipped with identifiers, arrival time, price and stock quantity. For example, a token $(\mathtt{b1}, \mathtt{1}, \mathtt{22.0}, \mathtt{5})$ represents an order with identifier `b1`, submitted in time `1`, to buy `5` stocks at price `22.0` per stock. Thus, colors represent classes of objects, whereas tokens are object instances. Arcs are labeled with expressions to specify how tokens are processed. We fix a language of expressions $\mathcal{L}$, where each expression is of the form $(e_1, ..., e_n)$ such that, for each $i \in \{1, ..., n\}$, e_i is either a constant from a domain in $\mathfrak{D}$, a variable typed over an element in $\mathfrak{D}$, or a function whose domain and range are elements in $\mathfrak{D}$.

Definition 1 (Colored Petri net). *Let $\mathfrak{D}$ be a finite set of data domains, let Σ be a set of colors defined over $\mathfrak{D}$, let $\mathcal{L}$ be a language of expressions, and let $\mathcal{A}$ be a set of activity labels. A colored Petri net is a 6-tuple $CP = (P, T, F, \mathtt{color}, \mathcal{E}, \Lambda)$, where:*

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

We define restrictions that must hold for CPNs in order to model processes in trading systems that manage different kinds of objects such as buy/sell orders [6, 7]. We call *conservative-workflow* CPNs the models that comply such restrictions. In the following, for a transition $t \in T$ in a CPN, ${}^\bullet t = \{p \in P \mid (p, t) \in F\}$ denotes the set of *input places* of t, and $t^\bullet = \{p \in P \mid (t, p) \in F\}$ denotes the set of *output places* of t.

Definition 2 (Conservative-Workflow Colored Petri Net). *Let $CP = (P, T, F, \mathtt{color}, \mathcal{E}, \Lambda)$ be a CPN defined over a set of colors Σ. We say that CP is a conservative-workflow CPN iff:*

1. *CP is a conservative colored Petri net where tokens do not disappear or duplicate. For every transition $t \in T$:*
   - $\forall\, p \in {}^\bullet t \;\; \exists!\, p' \in t^\bullet : \mathcal{E}(p, t) = (\mathtt{v}_1, ..., \mathtt{v}_n) \wedge \mathcal{E}(t, p') = (\mathtt{w}_1, ..., \mathtt{w}_n) \wedge\; \mathtt{v}_1 = \mathtt{w}_1$.
   - $\forall\, p \in t^\bullet \;\; \exists!\, p' \in {}^\bullet t : \mathcal{E}(p', t) = (\mathtt{v}_1, ..., \mathtt{v}_n) \wedge \mathcal{E}(t, p) = (\mathtt{w}_1, ..., \mathtt{w}_n) \wedge\; \mathtt{v}_1 = \mathtt{w}_1$.
2. *For every $j \in \{1, ..., k\}$, there exists one distinguished pair of places in P, a source i_j and a sink o_j, where $\mathtt{color}(i_j) = \mathtt{color}(o_j) = \mathtt{C}_j$ with $\mathtt{C}_j \in \Sigma$, and there exists a path from i_j to o_j s.t. for each place p in the path $\mathtt{color}(p) = C_j$. We respectively denote the sets of sources and sinks in CP by P_0 and P_F.*

3. $\forall t \in T : \forall p, p' \in {}^{\bullet}t \; p \neq p' \iff \texttt{color}(p) \neq \texttt{color}(p') \;\wedge\; \forall p, p' \in t^{\bullet} \; p \neq p' \iff \texttt{color}(p) \neq \texttt{color}(p')$*, i.e., for every transition t, places located within the set of input places of t have distinct colors. The same rule holds for places located in the set of output places of t.*

We close this section by defining execution semantics of our model. Let $CP = (P, T, F, \texttt{color}, \mathcal{E}, \Lambda)$ be a CPN. A *marking* M is a function, mapping every place $p \in P$ to a (possibly empty) set of tokens $M(p)$, s.t. $M(p) \subseteq \texttt{color}(p)$. A *binding* b of a transition $t \in T$ is a function, that assigns a value $b(\texttt{v})$ to each variable $\texttt{v}$ occurring in arc expressions adjacent to t, where $b(\texttt{v}) \in \texttt{type}(\texttt{v})$. 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 $\forall p \in P : M'(p) = M(p) \setminus \{b(\mathcal{E}(p, t))\} \cup \{b(\mathcal{E}(t, p))\}$.

3 Event Logs

Definition 3 (Event, Trace, Event Log). *Let $\mathfrak{D}$ be a finite set of data domains, let Σ be a set of colors defined over $\mathfrak{D}$, and let $\mathcal{A}$ be a finite set of activities. An event is a pair $(a, R(e))$ such that $a \in A$ and $\forall r \in R(e)$, r is a tuple of color $\texttt{C} \in \Sigma$, representing an object involved in the execution of activity a. A trace $\sigma = \langle e_1, ..., e_m \rangle$ is a finite sequence of events, s.t. $m = |\sigma|$ is the trace length. An event log L is a multiset of traces.*

Table 1: A trace σ of an event log, corresponding to a run in a trading system.

event (e)	**activity** (a)	**objects** ($R(e)$)
e_1	`submit buy order`	(`b1, 1, 22.0, 5`)
e_2	`new buy order`	(`b1, 1, 22.0, 5`)
e_3	`submit sell order`	(`s1, 2, 21.0, 2`)
e_4	`new sell order`	(`s1, 2, 21.0, 2`)
e_5	`new sell order`	(`s2, 3, 19.0, 1`)
e_6	`trade2`	(`b1, 1, 22.0, 4`), (`s1, 2, 21.0, 0`)

We denote as $\texttt{color}(r)$ the color of object $r \in R(e)$ in event e. For each object $r = (r^{(1)}, ..., r^{(n)})$ in an event $e = (a, R(e))$, its components $r^{(1)}, ..., r^{(n)}$ represent the state of r after the execution of a. We assume that the first component of r, $r^{(1)}$, is the *object identifier* which cannot be modified; $\texttt{id}(r) = r^{(1)}$ denotes the identifier of r. We consider that objects in a trace can be distinguished. $R(\sigma)$ denotes the set of distinct object identifiers in a trace σ, e.g., for Table 1, $R(\sigma) = \{\texttt{b1}, \texttt{s1}, \texttt{s2}\}$. Let $r = (r^{(1)}, ..., r^{(n)})$ be an object. For $j \in \{1, ..., n\}$, we consider that each attribute $r^{(j)}$ can be accessed using a name. Objects of the same color share the same set of attribute names, e.g., for color `OB` described in Section 2, we consider names $\{\texttt{id}, \texttt{tsub}, \texttt{price}, \texttt{qty}\}$; we fix a *member access function* `#`, that given an object $r = (r^{(1)}, ..., r^{(n)})$ and the name of the jth-attribute, it returns $r^{(j)}$, i.e., $\#(r, \texttt{name}_j) = r^{(j)}$.

For simplicity, we use $\texttt{name}_j(r)$ instead of $\texttt{\#}(r, \texttt{name}_j)$, e.g., for $r = (\texttt{b1}, \texttt{1}, \texttt{22.0}, \texttt{5})$, $\texttt{tsub}(r) = 1$, $\texttt{price}(r) = 22.0$, and $\texttt{qty}(r) = \texttt{5}$.

Finally, a criterion of *syntactical correctness* must hold for CPNs and event logs that serve as input to the method we propose. Let L be an event log, and let $CP = (P, T, F, \texttt{color}, \mathcal{E}, \Lambda)$ be a conservative-workflow CPN. We say that L is *syntactically correct* w.r.t. to CP iff, for every trace $\sigma \in L$, each event e in σ is syntactically correct. An event $e = (a, R(e))$ is syntactically correct w.r.t. to CP iff $\exists t \in T : \Lambda(t) = a \ \wedge \ \forall p \in {}^{\bullet}t \ \ \exists! r \in R(e) : \texttt{color}(r) = \texttt{color}(p) \ \wedge \ \forall r \in R(e) \ \ \exists! p \in {}^{\bullet}t : \texttt{color}(r) = \texttt{color}(p)$; that is, for every event $(a, R(e))$, there exists a transition t with activity label a, and each input place of t is mapped to exactly one event's object, and similarly each event's object is mapped to exactly one input place of t.

4 Conformance Method

We present a replay-based method to check conformance between a CPN and a trace of an event log. For each event in a trace, the method seeks to execute a model transition labeled with the event's activity, and consuming tokens that correspond to objects involved in the event. As mentioned in Section 1, four kinds of deviations can be detected in events: control-flow deviations, priority rule violations, resource corruptions, and non-proper termination of objects.

Algorithm 1 describes the replay method between a trace σ and a (conservative-workflow) CPN whose initial marking is empty. In addition to deviations, the method returns two counters: the number of *token jumps* `j`, i.e., the number of tokens that are moved to input places of transitions to force their firing, and the number of consumed/produced tokens `k`. At the start, each source place of the CPN is populated with the trace's distinct objects $R(\sigma)$ according to their color. For each object to insert as a token in a source place, we set its values according to its first occurrence in σ. As an example, let us consider the replay of trace σ in Table 1 on the CPN of Fig. 1: place p_1 is populated with buy orders $(\texttt{b1}, 1, \texttt{22.0}, \texttt{5})$, and p_2 with sell orders $(\texttt{s1}, 2, \texttt{21.0}, 2)$ and $(\texttt{s2}, 3, \texttt{19.0}, 1)$. Then, for each event $e = (a, R(e))$ in σ, a transition is selected to fire s.t. $\Lambda(t) = a$. To fire t, we check for every object $r \in R(e)$ whether its corresponding token in the model $(d_1, ..., d_n)$, $\texttt{id}(r) = d_1$, is located in input place p of t s.t. $\texttt{color}(p) = \texttt{color}(r)$. If the latter is not true for an object r, we look for its corresponding token in other places, which is moved to the input place p of t for tokens of $\texttt{color}(r)$. In such case, a *control-flow deviation* is registered and the number of token jumps increases (e.g., Lines 5-10).

Let us consider again the replay of σ in Table 1 on the CPN of Fig. 1. Let us assume that events e_1,...,e_4 were processed with no deviations detected. Now, consider $e_5 = (\texttt{new sell order}, \{(\texttt{s2}, 3, \texttt{19.0}, \texttt{1})\})$ which implies to fire transition t_4 consuming token with id. `s2`. In the current model marking, however, `s2` is not in place p_4, but in p_2. To execute the model according to e_5, token s_2 jumps to place p_4 as depicted in Fig. 2. This deviation relates to a sell order that was placed in the order book, but that illegally skipped activity `submit sell order`.

Algorithm 1: Object-Centric Replay with CPNs

```
Input: $CP = (P, T, F, \texttt{color}, \mathcal{E}, \Lambda)$ — conservative-workflow CPN;
       $P_0, P_F \subseteq P$ — non-empty sets of source and sink places;
       $\sigma$ — an event log trace;
Output: counter of token jumps ($\texttt{j}$) and consumed/produced tokens ($\texttt{k}$);
$\texttt{j} \leftarrow 0$; $\texttt{k} \leftarrow 0$;
populateSourcePlaces($P_0, R(\sigma)$);
foreach $e = (a, R(e))$ in $\sigma$ do
    $t \leftarrow$ selectTransition($a$);                    // $\exists! t \in T \; \Lambda(t) = a$
    foreach $r$ in $R(e)$ do
        if $\neg\exists(d_1, ..., d_n) \in M(p) : p \in {}^\bullet t \wedge \texttt{color}(p) = \texttt{color}(r) \wedge \texttt{id}(r) = d_1$ then
            registerDeviation(CONTROL_FLOW);
            jump(id($r$), $p$);
            $\texttt{j} \leftarrow \texttt{j} + 1$;
        endif
        if priorityRuleViolation($(d_1, ..., d_n), M(p)$) then
            registerDeviation(RULE_VIOLATION);
        endif
    endfor
    fire($t, R(e)$);
    $\texttt{k} \leftarrow \texttt{k} + |R(e)|$;
    foreach $r$ in $R(e)$ do
        let $d = (d_1, ..., d_n) : d_1 = \texttt{id}(r) \wedge d \in M(p) \wedge \texttt{color}(p) = \texttt{color}(r) \wedge p \in t^\bullet$
        if $d \neq r$ then
            registerDeviation(RESOURCE_CORRUPTED);
            $d \leftarrow r$;
        endif
    endfor
endfor
foreach $r$ in $R(\sigma)$ do
    if $\neg\exists(d_1, ..., d_n) \in M(p) : p \in P_F \wedge \texttt{color}(p) = \texttt{color}(r) \wedge \texttt{id}(r) = d_1$ then
        registerDeviation(NONPROPER_TERMINATION);
        jump(id($r$), $p$);
        $\texttt{j} \leftarrow \texttt{j} + 1$;
    endif
endfor
consumeAllObjectsFromSinkPlaces($P_F, R(\sigma)$);
$\texttt{k} \leftarrow \texttt{k} + |R(\sigma)|$;
return $(\texttt{j}, \texttt{k})$;
```

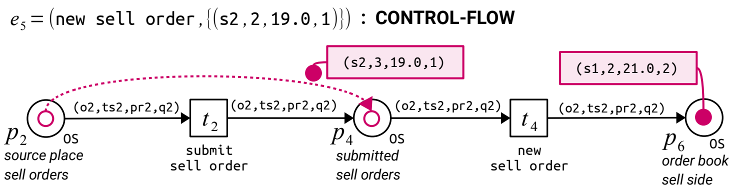


Fig. 2: Control-flow deviation: `s2` is not in p_4, so a jump is done to force replay.

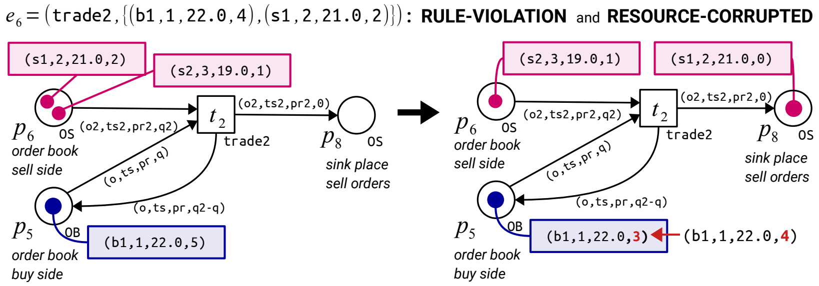


Fig. 3: Example of a priority rule violation and a resource corruption.

Prior to each transition firing, the method checks if each token to consume is the one that must be selected according to a *priority rule*. To this end, we shall assume that input CPNs may have priority rules on some transitions. Let b a selected binding to fire a transition t. We define a priority rule on t as $\Phi(t) = \bigwedge_{\forall p\in {}^\bullet t} \phi_p(M(p), b(\mathcal{E}(p,t)))$, s.t. $b(\mathcal{E}(p,t))$ is the token to consume from input place p, and $\phi_p(M(p), b(\mathcal{E}(p,t)))$ is a priority local rule on p; $\phi_p(M(p), b(\mathcal{E}(p,t)))$ holds if $b(\mathcal{E}(p,t))$ must be consumed before other tokens in $M(p)$. Algorithm 1 checks the truth value of $\Phi(t)$ by checking if the local rule of each input place p of t is violated, i.e., in line 11, function `priorityRuleViolation`$((d_1,...,d_n), M(p))$ evaluates to true iff $\Phi(t)$ is defined and $\phi_p(M(p),(d_1,...,d_n))$ does not hold. If the function returns true, then a *priority rule violation* is registered as token $(d_1,...,d_n)$ should not have been consumed before other tokens in p. For example, let us assign $\Phi(t) = \phi_{\texttt{BUY}}(M(p_5), r_1) \land \phi_{\texttt{SELL}}(M(p_6), r_2)$ to transitions t_5, t_6, and t_7 (trade activities) in the CPN of Fig. 1, such that:

$$\begin{aligned}\phi_{\texttt{BUY}}(M(p_5), r_1) = \forall_{(\texttt{o},\texttt{ts},\texttt{pr},\texttt{q})\in M(p_5)\ \texttt{id}(r_1)\neq\texttt{o}} &: (\texttt{price}(r_1) > \texttt{pr})\\ &\lor (\texttt{price}(r_1) = \texttt{pr}\ \land\ \texttt{tsub}(r_1) < \texttt{ts})\\ \phi_{\texttt{SELL}}(M(p_6), r_2) = \forall_{(\texttt{o},\texttt{ts},\texttt{pr},\texttt{q})\in M(p_6)\ \texttt{id}(r_2)\neq\texttt{o}} &: (\texttt{price}(r_2) < \texttt{pr})\\ &\lor (\texttt{price}(r_2) = \texttt{pr}\ \land\ \texttt{tsub}(r_2) < \texttt{ts})\end{aligned}$$

where r_1 and r_2 are buy and sell orders to consume; the local rule $\phi_{\texttt{BUY}}$ on place p_5 states that r_1 must be the order with highest price (or with earliest submitted time if other orders have the same price). The local rule $\phi_{\texttt{SELL}}$ on p_6 is defined similarly, but r_2 must be the order with lowest price. Let us consider event e_6 in Fig. 3: the rule on p_6, to prioritize sell orders with lowest price, is violated as order `s1` with price `21.0` is consumed before `s2` with price `19.0`.

After firing a transition according to an event, we search for *resource corruptions*. Specifically, we check if values of every transferred token are equal to the values of corresponding objects in the event; this detects if a system transformed object attributes as expected, e.g., in Fig. 3, after the trade of `1` stock between `b1` and `s1`, the stock quantity of `b1` decreased from `5` to `3`; however, event e_6 shows that the `b1`'s stocks changed to `4`, indicating that `b1` was corrupted; in case of these deviations, values of the corrupted token are updated according to the values of its corresponding object in the event, e.g., in Fig. 3, `b1`'s stocks change to `4`.

After replaying a trace, we check *non-proper termination*, that is, whether the system did not fully process all objects. We check if all objects reside in their corresponding sinks. After the replay of the trace in Table 1 on the CPN of Fig. 1, orders `b1` and `s2` did not arrive to their sinks. These are orders that were not fully handled by the trading system. For these deviations, the method moves these tokens to their sinks, increasing the counter of token jumps `j`. When all tokens are in the sinks, they are consumed by the environment, and the counter of transfers k increases by the number of tokens consumed. Finally, the ratio $1 - \mathtt{j}/\mathtt{k}$ can be used as a fitness metric to measure the extent to which a system (as observed in the trace) complies with the CPN, e.g., if the result of such ratio is 1, then all behavior observed in the trace complied with the model.

5 Prototype and Experiments

We implemented a software prototype[1] of the method proposed using SNAKES [15], a Python library for simulating CPNs. We aimed at detecting deviations in the management of a subset of order books in a real-life trading system. We validated order books with only *day limit orders*, orders that buy/sell stocks at a fixed price, and that must trade or cancel by the end of a day. The orders considered are not amended once they are submitted. The system expected behavior is described by the CPN of Fig. 1. The experimental setting is illustrated in Fig. 4. The method takes as input the CPN of Fig. 1 and an event log where each trace corresponds to the management of an order book during a day. The log was extracted from a set of Financial Information Exchange (FIX) protocol messages [8]. The messages were exchanged by users and the system during a day, informing activities executed and status of orders. The recorded set consists of 552935 FIX messages, whereas the log obtained from such set consists of 73 traces (order books) and 2259 events, with a mean of 30.94 events per trace.

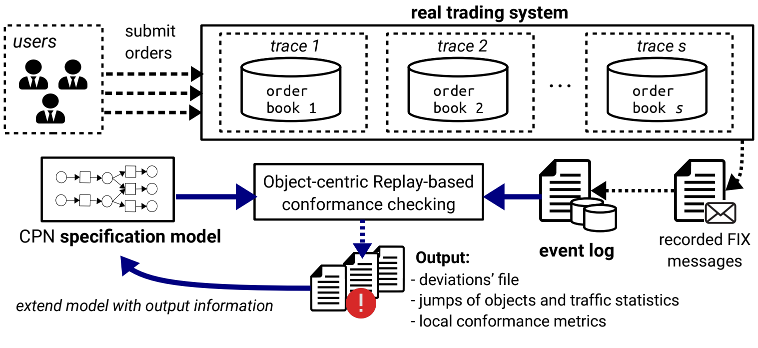
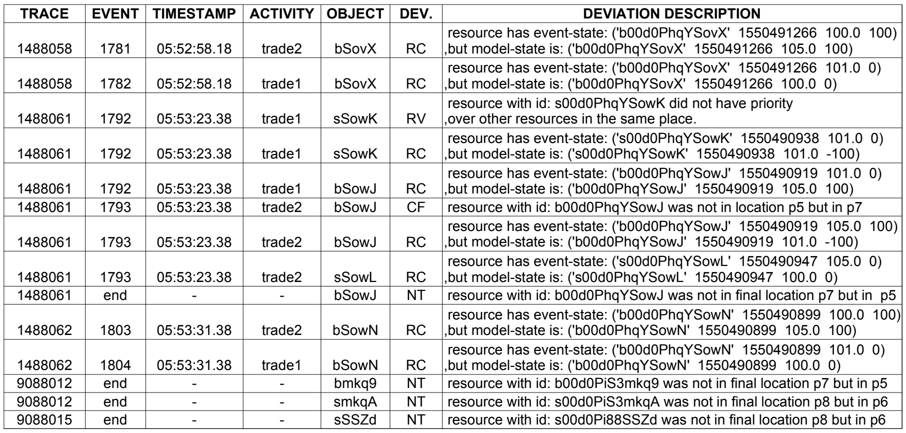


Fig. 4: Input/output of the conformance method for validating trading systems.

---

[1] https://github.com/jcarrasquel/hse-uamc-conformance-checking

A fragment of the deviations file computed by the method is shown in Fig. 5. The file lists deviations detected in events of different traces of the input log. Each line describes precise information of a deviation in the real system: the trace (order book), event number, timestamp, and activity where the error occurred, the object affected, and the kind of deviation detected. Also, a detailed description of the deviation is automatically generated. In this experiment, most of the deviations relate to corruption of orders when executing trades: the prices of some orders changed upon the execution of trades, e.g., in event 1781 the price of order with id. `bSovX` changed its price from 105 to 100 after trading, and such transformation is not described in the CPN. Thus, this information about deviations can be used by experts to confirm if this is a failure in the system, or instead the model should be slightly refined.



TRACE	EVENT	TIMESTAMP	ACTIVITY	OBJECT	DEV.	DEVIATION DESCRIPTION
1488058	1781	05:52:58.18	trade2	bSovX	RC	resource has event-state: ('b00d0PhqYSovX' 1550491266 100.0 100) ,but model-state is: ('b00d0PhqYSovX' 1550491266 105.0 100)
1488058	1782	05:52:58.18	trade1	bSovX	RC	resource has event-state: ('b00d0PhqYSovX' 1550491266 101.0 0) ,but model-state is: ('b00d0PhqYSovX' 1550491266 100.0 0)
1488061	1792	05:53:23.38	trade1	sSowK	RV	resource with id: s00d0PhqYSowK did not have priority ,over other resources in the same place.
1488061	1792	05:53:23.38	trade1	sSowK	RC	resource has event-state: ('s00d0PhqYSowK' 1550490938 101.0 0) ,but model-state is: ('s00d0PhqYSowK' 1550490938 101.0 -100)
1488061	1792	05:53:23.38	trade1	bSowJ	RC	resource has event-state: ('b00d0PhqYSowJ' 1550490919 101.0 0) ,but model-state is: ('b00d0PhqYSowJ' 1550490919 105.0 100)
1488061	1793	05:53:23.38	trade2	bSowJ	CF	resource with id: b00d0PhqYSowJ was not in location p5 but in p7
1488061	1793	05:53:23.38	trade2	bSowJ	RC	resource has event-state: ('b00d0PhqYSowJ' 1550490919 105.0 100) ,but model-state is: ('b00d0PhqYSowJ' 1550490919 101.0 -100)
1488061	1793	05:53:23.38	trade2	sSowL	RC	resource has event-state: ('s00d0PhqYSowL' 1550490947 105.0 0) ,but model-state is: ('s00d0PhqYSowL' 1550490947 100.0 0)
1488061	end	-	-	bSowJ	NT	resource with id: b00d0PhqYSowJ was not in final location p7 but in p5
1488062	1803	05:53:31.38	trade2	bSowN	RC	resource has event-state: ('b00d0PhqYSowN' 1550490899 100.0 100) ,but model-state is: ('b00d0PhqYSowN' 1550490899 105.0 100)
1488062	1804	05:53:31.38	trade1	bSowN	RC	resource has event-state: ('b00d0PhqYSowN' 1550490899 101.0 0) ,but model-state is: ('b00d0PhqYSowN' 1550490899 100.0 0)
9088012	end	-	-	bmkq9	NT	resource with id: b00d0PiS3mkq9 was not in final location p7 but in p5
9088012	end	-	-	smkqA	NT	resource with id: s00d0PiS3mkqA was not in final location p8 but in p6
9088015	end	-	-	sSSZd	NT	resource with id: s00d0Pi88SSZd was not in final location p8 but in p6

Fig. 5: Fragment of deviations detected (`DEV`): resource corruptions (`RC`), priority rule violations (`RV`), control-flow deviations (`CF`), non-proper termination (`NT`)

.

In a second experiment, we show how information obtained during replay, about token jumps and transfers, can be used to enhance an input CPN for visualizing deviations. Using SNAKES, we built a model representing a trading system, similar to the CPN of Fig. 1, but with some undesired behavior that shall be uncovered as control-flow deviations: orders may skip activities `submit buy order` and `submit sell order`, e.g., this may represent malicious users submitting unverified orders via back-doors. Also, activity `new sell order` may lead some orders to a deadlock. As input for our method, we consider the model of Fig. 1 and an artificial event log, that records the system's behavior. The log was generated by our solution, running the CPN that represents the faulty system.

The log consists of 100 traces and 4497 events, with an average of 44.97 events per trace. In each trace, there is an average of 10 buy orders and 10 sell orders.

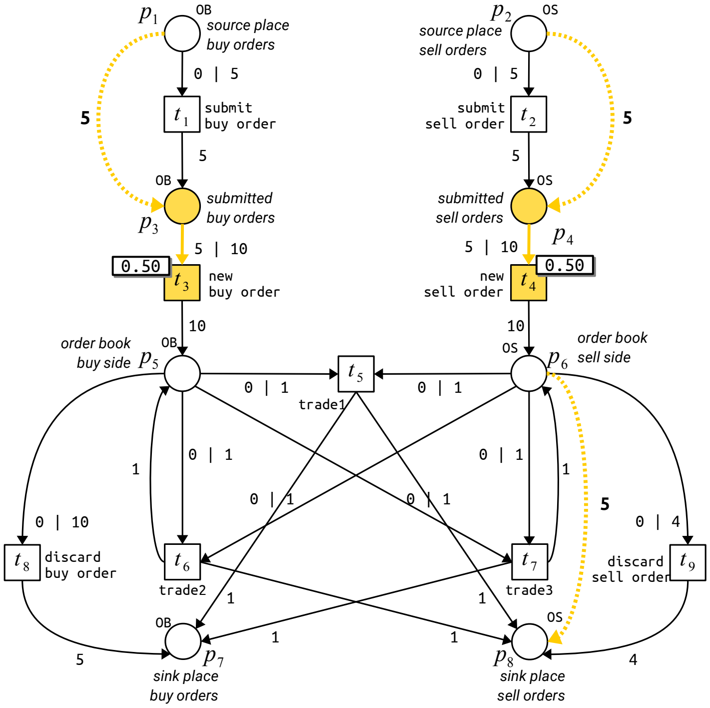


Fig. 6: Specification model extended with diagnostics computed by our method.

Upon the execution of the method, control-flow deviations are detected and reveal the undesired behavior previously described. When detecting such deviations, tokens jump between places via unforeseen model paths in order to continue the replay. Information about token jumps in each place of the CPN, as well as token transfers are registered by our solution. Fig. 6 illustrates how such information is used to enhance the input CPN model. Dotted lines represent token jumps related to the deviations mentioned: jumps from p_1 to p_3, and from p_2 to p_4 are from orders that illegally skipped activities `submit buy order` and `submit sell order`. Also, jumps from p_6 to p_8 relate to orders that got locked after executing `new sell order`. The method detects such locked orders when checking non-proper termination. Input arcs and dotted lines indicate the (rounded) average number of transferred/jumped tokens, considering all log traces. The software prototype tracks the proportion of token transfers/jumps flowing through model components. *Local conformance metrics* are computed using such proportions to measure how deviations affect precise system parts. For example, `new buy order` has a measure of 0.5, meaning that 5 out of 10 objects processed by the activity complied with the model path. We refer to [6] for formal definitions and a further discussion about these local measures.

6 Conclusions

In this paper, we presented a conformance method to search for deviations in trading systems. Different deviations are detected by replaying a system's trace on a CPN. We validated the management of orders in a real system and revealed precise deviations. Another experiment showed how conformance diagnostics can be added to a CPN to display control-flow deviations. A direction for further research may study how to visualize more complex deviation patterns.

References

1. van der Aalst, W.: Process Mining: Data Science in Action. Springer, 2nd edn. (2016)
2. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking: Relating Processes and Models. Springer, 1st edn. (2018)
3. 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)
4. 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)
5. 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)
6. Carrasquel, J.C., Mecheraoui, K.: Object-centric replay-based conformance checking: Unveiling desire lines and local deviations. Modeling and Analysis of Information Systems **28**(2), 146–168 (2021)
7. Carrasquel, J.C., Mecheraoui, K., Lomazova, I.A.: Checking conformance between colored petri nets and event logs. In: Analysis of Images, Social Networks and Texts. LNCS, vol. 12602, pp. 435–452. Springer (2021)
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., Gromova, A., Sitnikov, A., Legchikov, D., Tsymbalov, E., Yavorskiy, R., Novikov, A., Rudakov, K.: User-Assisted Log Analysis for Quality Control of Distributed Fintech Applications. In: IEEE International Conference On Artificial Intelligence Testing (AITest). pp. 45–51. IEEE (2019)
11. Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer, 1st edn. (2009)
12. Mannhardt, F., Leoni, de, M., Reijers, H., van der Aalst, W.: Balanced multi-perspective checking of process conformance, Computer, vol. 98. Springer (2015)
13. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (1989)
14. Rozinat, A., van der Aalst, W.: Conformance checking of processes based on monitoring real behavior. Information Systems **33**(1), 64–95 (2008)
15. SNAKES: A flexible high-level Petri net library: `https://snakes.ibisc.univ-evry.fr/`