MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 2, 2021
journal homepage: www.mais-journal.ru

COMPUTER SYSTEM ORGANIZATION

Object-Centric Replay-Based Conformance Checking: Unveiling Desire Lines and Local Deviations

J. C. Carrasquel[1], K. Mecheraoui[2]

DOI: 10.18255/1818-1015-2021-2-146-168

[1]HSE University, 20 Myasnitskaya str., Moscow 101000, Russia.
[2]University of Constantine 2 – Abdelhamid Mehri, Nouvelle ville Ali Mendjeli BP : 67A, 25000 Constantine, Algeria.

MSC2020: 68N30
Research article
Full text in English

Received May 3, 2021
After revision May 25, 2021
Accepted June 2, 2021

Conformance checking methods diagnose to which extent a real system, whose behavior is recorded in an event log, complies with its specification model, e.g., a Petri net. Nonetheless, the majority of these methods focus on checking isolated process instances, neglecting interaction between instances in a system. Addressing this limitation, a series of object-centric approaches have been proposed in the field of process mining. These approaches are based on the holistic analysis of the multiple process instances interacting in a system, where each instance is centered on the handling of an object. Inspired by the object-centric paradigm, this paper presents a replay-based conformance checking method which uses a class of colored Petri nets (CPNs) – a Petri net extension where tokens in the model carry values of some types (colors). Particularly, we consider conservative workflow CPNs which allow to describe the expected behavior of a system whose components are centered on the end-to-end processing of distinguishable objects. For describing a system's real behavior, we consider event logs whose events have sets of objects involved in the execution of activities. For replay, we consider a jump strategy where tokens absent from input places of a transition to fire move from their current place of the model to the requested places. Token jumps allow to identify desire lines, i.e., object paths unforeseen in the specification. Also, we introduce local diagnostics based on the proportion of jumps in specific model components. The metrics allow to inform the severity of deviations in precise system parts. Finally, we report experiments supported by a prototype of our method. To show the practical value of our method, we employ a case study on trading systems, where orders from users are matched to trade.

Keywords: Process Mining; Conformance Checking; Petri Nets; Colored Petri Nets

INFORMATION ABOUT THE AUTHORS

Julio C Carrasquel correspondence author	orcid.org/0000-0003-3557-797X. E-mail: jcarrasquel@hse.ru Postgraduate Student.
Khalil Mecheraoui	orcid.org/0000-0001-9906-6074. E-mail: k_mecheraoui@esi.dz Assistant Professor.

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

For citation: J. C. Carrasquel and K. Mecheraoui, "Object-Centric Replay-Based Conformance Checking: Unveiling Desire Lines and Local Deviations", *Modeling and analysis of information systems*, vol. 28, no. 2, pp. 146-168, 2021.

Объектно-ориентированная проверка соответствия модели на основе воспроизведения журнала событий: выявление желаемого поведения и локальных отклонений

Х. С. Карраскель[1], Х. Мешерауи[2]

DOI: 10.18255/1818-1015-2021-2-146-168

[1]Национальный исследовательский университет “Высшая школа экономики”, ул. Мясницкая, д. 20, г. Москва, 101000 Россия.
[2]Университет Константина 2 - Абделхамид Мехри, Нувель виль Али Менджели BP : 67A, 25000 Константин, Алжир.

УДК 519.686.2
Научная статья
Полный текст на английском языке

Получена 3 мая 2021 г.
После доработки 25 мая 2021 г.
Принята к публикации 2 июня 2021 г.

Методы проверки соответствия позволяют установить, в какой степени реальная система, поведение которой регистрируется в журнале событий, соответствует ее модели, например, в виде сети Петри. Большинство таких методов направлены на проверку изолированных экземпляров процесса и игнорируют взаимодействие между экземплярами в системе. Для преодоления этого ограничения в области интеллектуального анализа данных был предложен ряд объектно-ориентированных подходов. Эти подходы основаны на целостном анализе нескольких экземпляров процесса, взаимодействующих в системе, где каждый экземпляр соответствует некоторому объекту. В этой статье объектно-ориентированный подход применяется к методу проверки соответствия между журналами событий и цветными сетями Петри (CPN) – расширением сетей Петри, в котором фишки в модели представляют собой значения некоторых типов (цветов). В частности, мы рассматриваем консервативные CPN потоков работ, которые позволяют описывать ожидаемое поведение системы, в которой компоненты соответствуют обработке различных объектов. Реальное поведение системы описано в журнале событий, в котором события атрибутированы участвующими в них объектами. Для воспроизведения журнала событий мы используем стратегию прыжков, когда фишки, необходимые для срабатывания перехода, перемещаются из своих текущих позиций во входные позиции этого перехода. Прыжки фишек позволяют идентифицировать линии желаний, то есть поведения объектов, не предусмотренные в спецификации. Также мы представляем локальную диагностику, основанную на доле прыжков в поведении конкретных компонент модели. Эти метрики позволяют судить о серьезности отклонений в тех или иных частях системы. Наконец, мы приводим эксперименты, выполненные с помощью программного прототипа. Практическая ценность нашего метода показана на примере моделирования торговых систем, при котором устанавливаются соответствия между заявками пользователей и сделками.

Ключевые слова: Process Mining; проверка соответствия; сети Петри; Раскрашенные сети Петри

ИНФОРМАЦИЯ ОБ АВТОРАХ

Хулио С Карраскель
автор для корреспонденции | orcid.org/0000-0003-3557-797X. E-mail: jcarrasquel@hse.ru
аспирант.

Халил Мешерауи | orcid.org/0000-0001-9906-6074. E-mail: k_mecheraoui@esi.dz
преподаватель.

Финансирование: Исследование выполнено при поддержке Программы фундаментальных исследований НИУ ВШЭ.

Для цитирования: J. C. Carrasquel and K. Mecheraoui, “Object-Centric Replay-Based Conformance Checking: Unveiling Desire Lines and Local Deviations”, *Modeling and analysis of information systems*, vol. 28, no. 2, pp. 146-168, 2021.

1. Introduction

Process mining is a discipline that focuses on the analysis of system processes on the basis of event logs and formal models [1]. Event logs consist of recorded traces, which describe the *real behavior* of systems. As formal models, most process mining methods use Petri nets — a formalism for analysis of concurrent distributed systems [2]. Thus, process mining methods allow, for instance, to discover models describing the real processes from event logs, or to check compliance of real processes by comparing their logs with models describing *expected behavior*. The former kind of method is called *process discovery*, whereas the latter is referred to as *conformance checking* [3]. These methods have increasingly gained attention, resulting in a plethora of case studies from business organizations [4] and innovative research applications, e.g., see [5–7].

Nevertheless, the majority of process mining methods have hitherto consisted on the individual analysis of *isolated* process instances, thereby neglecting their interaction with other instances in the system. This assumption falls short, and may throw out a wrong analysis, especially when there is a strong dependency between the life-cycles of instances (for example, "a buy order is filled only if a sell order is in the system, and both orders are in certain locations").

To overcome such limitations, *object-centric* approaches have emerged as a popular paradigm in the field of process mining [8, 9]. The common denominator of these approaches lies in the holistic analysis of the multiple processes interacting in a system, where each process is centered on the management of a class of objects. Thus, within this research direction, novel multi-instance modeling notations [10, 11], process discovery methods [12], and formal verification techniques [13] have been proposed.

As for conformance checking, certain methods have been proposed to validate multiple perspectives of a process beyond the control-flow, i.e., the correct ordering of activities [14, 15]. Particularly, these methods make use of Petri nets with data (DPNs) to detect deviations caused by data corruptions (e.g., "a loan approval was wrongly executed due to a requested amount higher than expected"). However, the backbone of DPNs is an ordinary Petri net used to describe the individual execution of a single process instance, and data elements are only statically attached to model transitions. Thus, this model does not allow to describe and validate the dynamic and concurrent interaction of multiple instances within a system.

In this paper, inspired by the object-centric paradigm, we present a conformance checking method to diagnose whether systems that manage different kinds of objects comply with their specification. The method uses a class of colored Petri nets (CPNs) — a Petri net extension where tokens carry values representing objects of different types (called *colors*) [16]. Particularly, we consider *conservative workflow* CPNs with multiple source and sink places. In this model, tokens cannot disappear or duplicate, and they move through paths whose endpoints are specific pairs of source and sink places. In this way, the model allows to describe the expected behavior of systems with components centered on the end-to-end processing of distinct objects of a certain type. For describing the system's real behavior, we consider traces of event logs, where events consist of executed activities and sets of object identifiers. The latter indicates which objects were involved in an event's activity. As we will present, the characteristics proposed for both CPNs and event logs allow us not only to keep track of individual objects, but also to provide an algorithm with linear time complexity.

Our method is based on replaying individually each trace of an event log on top of a CPN model. When replaying each trace, the distinct objects are injected as tokens in source places of the model. Then, for each event of the trace, we seek to fire a transition labeled with the event's activity, and selecting tokens from the transition's input places that correspond to the event's object identifiers. If a token related to an event's object is not in a requested input place, we consider a *jump strategy*, where the missing token is moved from its current location in the model to the requested input place. This allows to force a transition firing, and to keep replaying a trace to find more deviations. The method reports all token jumps between places and their frequency. As we will present, this information can be added to the input CPN model to unveil so-called *desire lines* [17], i.e., actual paths of objects which are unforeseen in the specification model.

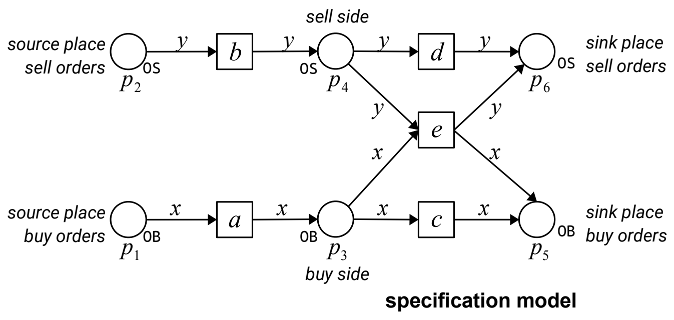
In addition, we present *local conformance metrics* based on the proportions of token transfers and jumps through specific components of a CPN model. By leveraging the correspondence of parts of a real system with components of a CPN model (e.g., activities with transitions and system locations with places), these metrics then allow to diagnose local deviations and their severity in concrete parts of a system. Finally, we report an experimental evaluation supported by a prototypical implementation of our method. To showcase the practical value of our approach, we shall make use of a case study on trading systems, where orders from users are matched to trade.

The remainder of this paper is structured as follows. Section 2 introduces a motivating example illustrating the use of our method in trading systems. Sections 3 and 4 present the class of CPNs and event logs, which we use in our method. Section 5 presents the conformance checking method, as well as the conformance metrics. Section 6 reports experimental evaluations supported by our prototype. Section 7 discusses the related work, and finally Section 8 presents the conclusions.

2. Motivating Example: Checking Conformance in Trading Systems

To give the reader an idea on how our method can be applied on a specific domain, let us consider the validation of *trading systems* in stock exchanges [18]. In trading systems, buy orders and sell orders from users are crossed to trade securities (e.g., stocks of a company). Orders that aim to trade the same kind of securities are placed in two-sided lists called *order books* (e.g., orders that buy or sell securities of "`rosneft`" are placed in the order book "`rosneft`"). In a trading system, there are as many order books as kinds of securities that can be traded.

Fig. 1 presents a CPN modeling the specification of a trading system operating one order book. It describes how the system is expected to manage both object classes, buy orders and sell orders. CPNs consist of two kinds of nodes: places and transitions. Places (drawn as circles) represent locations. For example, places p_1 and p_2 are source places for orders, p_3 and p_4 model buy and sell sides of the order book, and finally p_5 and p_6 are sink places for canceled or filled orders. Conversely, transitions (drawn as boxes) model system activities. Transitions consume tokens from input places, producing them back in output places. Thus, transitions a and b model submission of orders, transitions c and d represent cancellation of orders, whereas transition e models a trade execution between two orders. Albeit this example abstracts from other activities in a trading system, it will allow us to clearly illustrate our method.

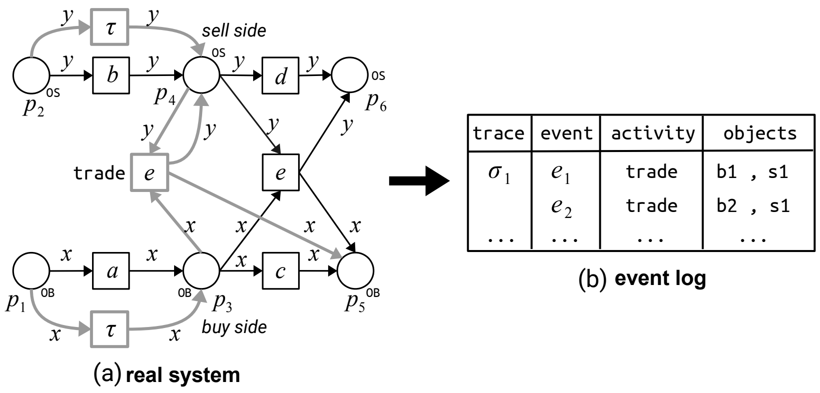


a	`new buy order`
b	`new sell order`
c	`cancel buy order`
d	`cancel sell order`
e	`trade`

Fig. 1: CPN model specifying the *expected behavior* of a trading system operating an order book

Like other distributed technologies, trading systems are prone to failures, e.g., due to errors during the system's development or due to malicious users hacking the system. Thus, trading systems are sensitive to deviate from their specification. For instance, let us assume a trading system initially built upon the specification of Fig. 1, but whose real behavior is actually described in Fig. 2(a). In this real system, buy and sell orders are "silently" allowed to trade without being accepted in the order book, i.e., skipping activities a and b. Also, an undesired variant of activity e allows sell orders to trade an unrestricted number of times.

For system engineers, it can be hard to determine these deviations and to which extent they have violated the system. Fortunately, event logs of this system record this misbehavior (e.g., Fig. 2(b)). For instance, the two first events in the trace σ_1 of Fig. 2(b) exemplify the mentioned problems of this system: a buy order `b1` and a sell order `s1` have traded, skipping activities *a* and *b*. Also, the sell order `s1` illegally trades in the next event. Powered by this kind of logs, we introduce how our conformance method unveils these deviations.



trace	event	activity	objects
σ_1	e_1	trade	b1 , s1
	e_2	trade	b2 , s1
...	...	...	...

Fig. 2: CPN describing the *real behavior* of a trading system with undesired behavior highlighted in gray (a). Our method takes an *event log* (b) from this real system.

Fig. 3 illustrates the setting of our method for detecting the aforementioned deviations. As input, the method takes the CPN specification model of Fig. 1, and an event log of the real system as exemplified in Fig. 2. In this example, each trace of an event log corresponds to all events executed in a specific order book. As output, the method produces a report listing all token jumps detected (representing objects skipping activities), global and local conformance metrics, as well as other traffic statistics. These reports can be used, for example, to extend the specification model in order to clearly identify deviations and their magnitude.

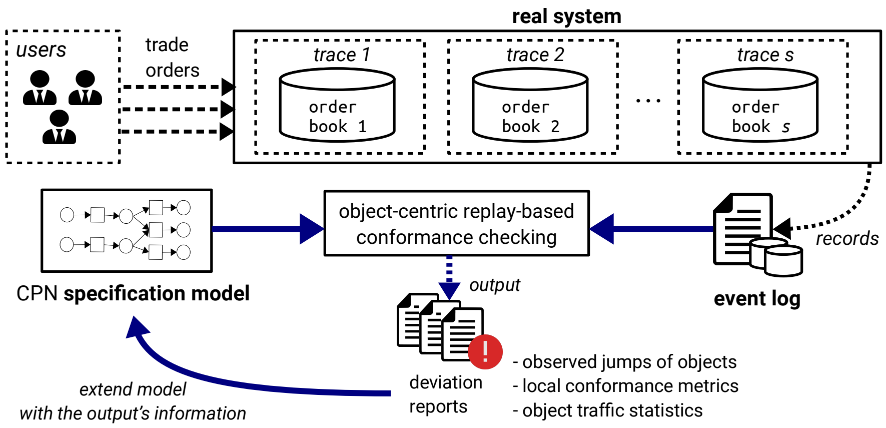

real system
users
trade orders
trace 1
order book 1
trace 2
order book 2
trace s
order book s
...
object-centric replay-based conformance checking
CPN specification model
records
event log
output
deviation reports
- observed jumps of objects
- local conformance metrics
- object traffic statistics
extend model with the output's information

Fig. 3: Validating real behavior of a trading system via object-centric replay-based conformance.

Fig. 4 shows an example of the specification model extended with the mentioned outputs of our method. Token jumps and their average frequencies on traces appear as dotted lines between places. Note how these jumps directly correlate with the components in Fig. 2(a) that describe undesired behavior of the real system, i.e., orders skipping activities *a* and *b* jump from places p_1, p_2 to places p_3 and p_4. Also, components of the extended CPN are labeled with traffic statistics. For instance, places indicate how many tokens were transferred from them (`k`), and how many of them actually arrived to the place by a jump (`j`).

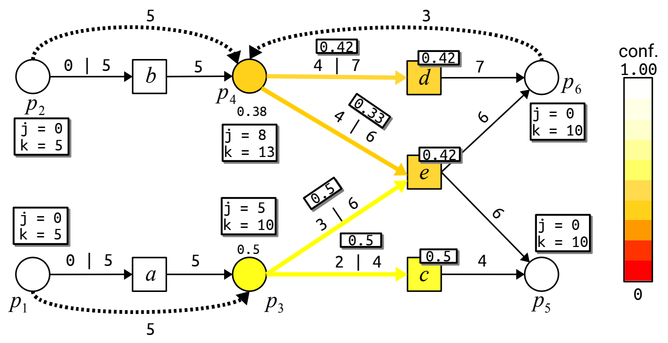


Fig. 4: Specification model of Fig. 1 enriched with the output information of our conformance method.

By leveraging the association between model's transitions and places with system's activities and locations, local conformance metrics allow to diagnose the severity of deviations on precise system components. For instance, one of these metrics checks the proportion of token jumps to input places of a concrete transition to force its firing. Then, this allows to measure the proportion of objects executing a precise activity without following the specification path. For example, activity *d* consumed 7 orders, but 4 of them were not ready at the location related to place p_4. Thus, this activity is associated with a measure of 0.42, i.e., 42% of the objects were at the required location when executing *d*, whereas the rest were improperly involved in this activity. As we will present in the next sections, similar local metrics can be associated to places (locations) and arcs from places to transitions. Finally, as depicted in Fig. 4, the metrics can be combined with the notion of a *heat map* to clearly visualize which components of a system have been violated more, e.g., if the local measure of a component is close to 0, then such a component is painted in red.

3. Colored Petri Nets

In this section, we present formal definitions and execution semantics for a class of colored Petri nets (CPNs). As introduced in Section 1, CPNs are an extension of Petri nets where tokens carry values of some types (also referred to as *colors*). For example, types may account for object classes, whereas tokens carry object identifiers. Let $\mathfrak{D} = \{D_1, ..., D_k\}$ be a finite set of types. Each token in a CPN carries a value $\mathtt{d}$ of some type $D \in \mathfrak{D}$. For instance, in the model of Fig. 1 two types are defined: `OB` for buy orders and `OS` for sell orders. Places are mapped to types in $\mathfrak{D}$ to indicate the kind of tokens they contain, e.g., $\mathtt{type}(p_1) = \mathtt{OB}$.

Arcs are labeled with expressions to describe how tokens are processed upon transition firings. We consider that each expression consists of a typed variable. Let $\mathcal{V}$ be a finite set of typed variables. We denote by $\mathtt{type(v)}$ the type of a variable $\mathtt{v} \in \mathcal{V}$ s.t. $\mathtt{type(v)} \in \mathfrak{D}$. We define a function W that maps each arc to a variable in $\mathcal{V}$. For example, in Fig. 1, expressions $W(p_1, t_1) = W(t_1, p_3) = \mathtt{x}$ specify that one buy order is transferred from place p_1 to p_3 upon the firing of t_1. Let $\mathcal{A}$ be a finite set of *activities*. To relate transitions with real system activities, we fix an *activity-labeling function* Λ that maps transitions to elements in $\mathcal{A}$.

Definition 1 (**Colored Petri Net**). *Let $\mathfrak{D}$ be a finite set of types, let $\mathcal{V}$ be a finite set of variables typed over $\mathfrak{D}$, and let $\mathcal{A}$ be a finite set of activities. A colored Petri net is a 6-tuple $CP = (P, T, F, \mathtt{type}, W, \Lambda)$ 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);*
- *$\mathtt{type} : P \longrightarrow \mathfrak{D}$ is a place-typing function, mapping each place to a type in $\mathfrak{D}$;*
- *$W : F \longrightarrow \mathcal{V}$ is an arc-labeling function, mapping each arc r to a variable in $\mathcal{V}$. $\forall r \in F$, if r is adjacent to a place $p \in P$, then $\mathtt{type}(W(r)) = \mathtt{type}(p)$;*
- *$\Lambda : T \longrightarrow \mathcal{A}$ is an activity-labeling function, s.t. $\forall t, t' \in T : t \neq t' \iff \Lambda(t) \neq \Lambda(t')$, i.e., transitions are mapped to distinct activities.*

We now define execution semantics for the CPNs defined above. Let $CP = (P, T, F, \texttt{type}, W, \Lambda)$ be a CPN. A *marking* M in a CPN is a function, mapping every place $p \in P$ to a (possibly empty) set of tokens $M(p)$, such that $M(p) \subseteq \texttt{type}(p)$. We denote by M_0 an initial marking. 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, such that $b(\texttt{v}) \in \texttt{type}(\texttt{v})$. Let ${}^\bullet t$ and $t^\bullet$ be respectively the sets of input places and output places of a transition $t \in T$. Transition t is *enabled* in marking M w.r.t. a binding b iff $\forall p \in {}^\bullet t \; : \; b(W(p,t)) \in M(p)$, 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 $\forall p \in P \; : \; M'(p) = M(p) \setminus \{b(W(p,t))\} \cup \{b(W(t,p))\}$.

As introduced in Section 1, we make use of CPNs to model systems consisting of components centered on the *end-to-end processing* of different types of *distinguishable objects*. To faithfully describe these systems, CPNs shall be characterized by the following properties: on the one hand, CPNs must be *conservative*, that is, tokens cannot disappear or duplicate upon transition firings; on the other hand, models must be *workflow-oriented* with a pair of source and sink places for every type defined in the model, such that tokens move in a path between a source and a sink corresponding to their type. We thus define *conservative workflow* CPNs.

Definition 2 (Conservative-Workflow Colored Petri Net). *Let $\mathfrak{D} = \{D_1, ..., D_k\}$ be a finite set of types such that $k \geq 1$, and let $CP = (P, T, F, \texttt{type}, W, \Lambda)$ be a CPN defined over $\mathfrak{D}$. We say that CP is a conservative-workflow CPN if and only if:*

1. *CP is a conservative colored Petri net where:*
   - $\forall \, t \in T \;\; \forall \, p \in {}^\bullet t \;\; \exists! \; p' \in t^\bullet \; : \; W(p,t) = W(t,p')$.
   - $\forall \, t \in T \;\; \forall \, p \in t^\bullet \;\; \exists! \; p' \in {}^\bullet t \; : \; W(p',t) = W(t,p)$.
2. *For every $j \in \{1, ..., k\}$, there exists one distinguished pair of places in P, a source place i_j and a sink place o_j in P, where $\texttt{type}(i_j) = \texttt{type}(o_j) = D_j$ with $D_j \in \mathfrak{D}$, and there exists a path in CP from i_j to o_j such that for every place p in the path $\texttt{type}(p) = D_j$. We denote by P_0 and P_F the sets of source places and sink places in CP.*
3. $\forall t \in T : \;\; \forall p, p' \in {}^\bullet t \;\, p \neq p' \iff \texttt{type}(p) \neq \texttt{type}(p') \;\wedge\; \forall p, p' \in t^\bullet \;\, p \neq p' \iff \texttt{type}(p) \neq \texttt{type}(p')$, *i.e., for every transition t, places located within the set of input places of t have distinct types. The same rule holds for places located in the set of output places of t.*

As input models for our method, we shall consider conservative workflow CPNs. We briefly explain Definition 2. Firstly, a CPN is conservative iff for every variable $\texttt{v}$ occurring in an input arc of a transition t, $\texttt{v}$ occurs exactly once in an output arc of t. Similarly, each variable occurring in an output arc of t shall occur exactly once in an input arc of t. This implies that when token values in input places are binded to variables upon transition firings, then such values are transferred to output places, without disappearing or being duplicated. In this way, our conformance method will be able to unambiguously associate every distinct object in a trace with a token in the model.

Also, according to Definition 2, CPNs shall be workflow-oriented. More precisely, a workflow CPN has distinct k source places and k sink places where k is the number of types defined in the model. Each pair of source and sink places of the same type are connected by a path whose intermediate places are also of the same type. In our conformance method, distinct objects in a trace are injected as tokens in source places. Then, tokens move in paths according to the information of objects recorded in events. Upon termination, the method will check whether these tokens arrived to their corresponding sink places. It can be inferred that all places of the same type form a subnet within a workflow CPN, where each sub-net represents a system component handling end-to-end processing of a concrete object class.

Finally, Definition 2 states that the model does not have transitions with input places of the same type. The latter allows to relate every object type with exactly one input place in each transition. In Section 5, we discuss how this syntactic restriction contributes to providing an algorithm with linear time complexity.

4. Event Logs

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

Definition 3 (Event Log, Trace, Event). *An event log 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.*

Let $\mathcal{A}$ be a finite set of activities, and let $\mathfrak{D}$ be a finite set of types. For every trace σ in L, each event e in σ is a tuple of the form $e = (a, R(e))$, where $a \in \mathcal{A}$ is an activity label, and $R(e) = \{r_1, ..., r_k\}$ is a finite set of objects. For each $j \in \{1, ..., k\}$, we say that $r_j \in R(e)$ is an object of type D involved in the execution of activity a, such that $D \in \mathfrak{D}$.

Table 1: An event log L with two examples of traces, which correspond to runs in a trading system.

trace	**event** (e)	**activity** (a)	**objects** ($R(e)$)
σ_1	e_1	`new buy order`	`b1`
	e_2	`new sell order`	`s1`
	e_3	`new sell order`	`s2`
	e_4	`trade`	`b1` , `s1`
	e_5	`cancel sell order`	`s2`
σ_2	e_1	`new buy order`	`b1`
	e_2	`trade`	`b1` , `s1`
	e_3	`trade`	`b2` , `s1`
	e_4	`new sell order`	`s2`

Table 1 presents an event log with two traces related to end-to-end runs in a trading system. Events indicate activities executed and objects involved, e.g., event e_4 in trace σ_1 indicates an execution of activity `trade` with two objects involved: buy order `b1` and sell order `s1`. We consider that all objects in a trace are distinguishable by having distinct identifiers. We denote by $R(\sigma)$ the set of distinct objects in a trace, e.g., $R(\sigma_1) = \{\texttt{b1}, \texttt{s1}, \texttt{s2}\}$. With slight abuse of notation, we denote the type of an object r by $\texttt{type}(r)$.

To guarantee the proper execution of our method, event logs must comply with a criterion of *syntactical correctness* with respect to the CPN used in the method. Let L be an event log, and let $CP = (P, T, F, \texttt{type}, W, \Lambda)$ 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 $\exists t \in T : \Lambda(t) = a \wedge \forall p \in {}^{\bullet}t \;\; \exists! r \in R(e) : \texttt{type}(r) = \texttt{type}(p) \wedge \forall r \in R(e) \;\; \exists! p \in {}^{\bullet}t : \texttt{type}(r) = \texttt{type}(p)$. That is, for every event $e = (a, R(e))$, there exists a transition t labeled with activity a, and each input place of t is associated with exactly one event's object, and similarly each event's object is associated with exactly one input place of t.

5. Object-Centric Replay-Based Conformance Checking

5.1. The Algorithm

In this section, we present our conformance checking method. The method is based on the replay strategy described in [1] with some adaptations for the class of CPNs and event logs described in Sections 3 and 4. Particularly, we shall assume that input models are conservative workflow CPNs (i.e., see Definition 2), whereas event logs are syntactically correct to these models.

The method replays individually each trace of an event log on a CPN. We consider that the input CPN has an empty initial marking. When replaying a trace σ, distinct objects in σ are firstly inserted as tokens in source places of the CPN, according to their type. Then, for each event $e = (a, R(e))$ in σ, the method seeks to fire a transition t labeled with activity a, and consuming the tokens that correspond to the event's objects, i.e., elements in $R(e)$. If a token corresponding to an event's object is not in an input place of t, then we consider a *jump strategy* where the missing token is moved from their current location in the model to the requested input place. This allows to force transition firings, and to keep replaying a trace to find more deviations.

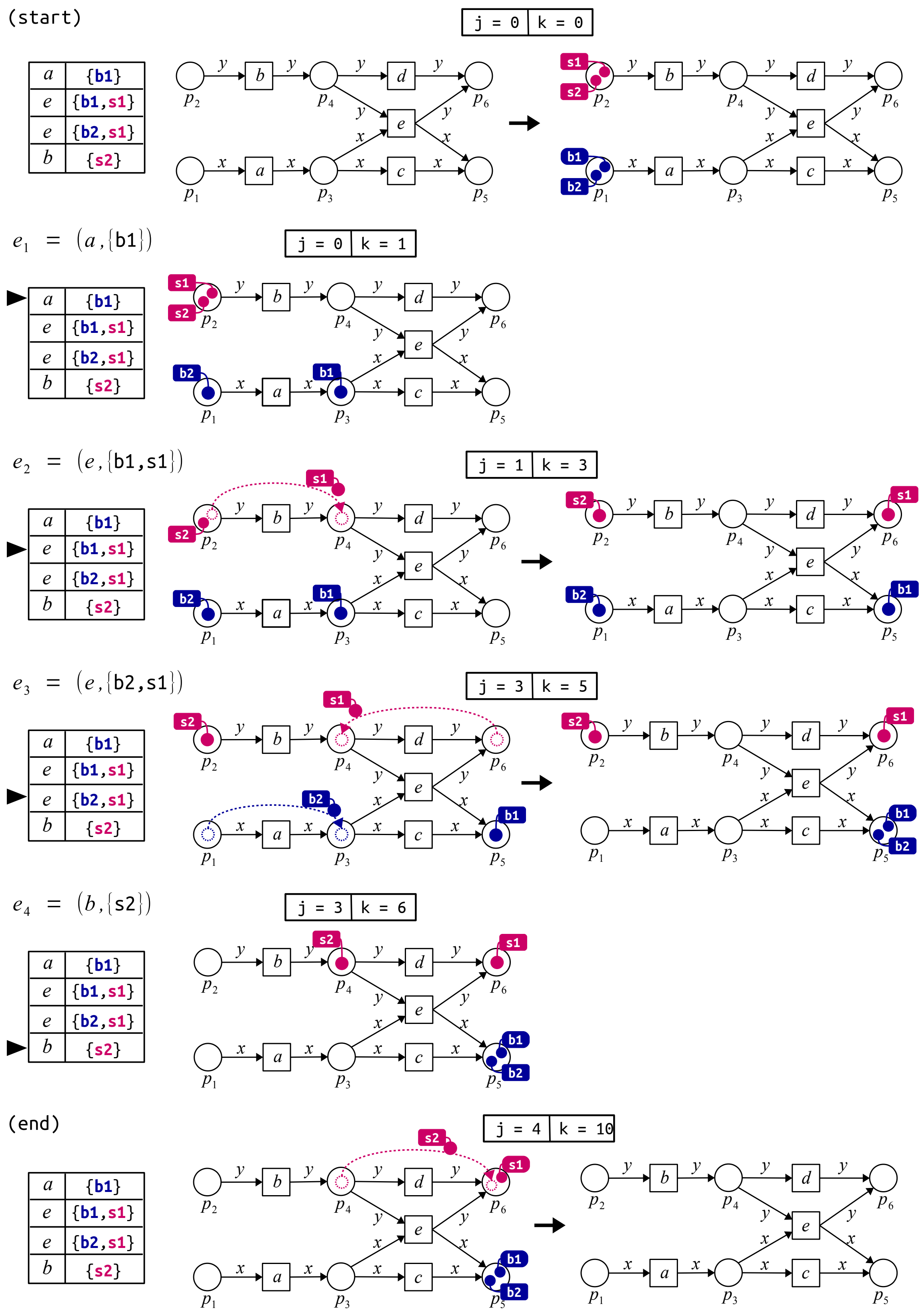
Algorithm 1 describes the method. As output, the method returns two integer counters `j` and `k`. The value `j` is the total number of token jumps, whereas the value `k` is the total number of tokens transferred from input places to output places upon transition firings. A ratio between these values `j` and `k` allows to measure the discrepancy between a trace and a CPN. In the following, we illustrate the use of the algorithm, whereas at the end of this part we introduce conformance measures based on these counters.

Algorithm 1: Replay on CPNs with a token jump strategy

```
Input: $CP = (P, T, F, \mathtt{type}, W, \lambda)$ — conservative-workflow CPN with marking $M$ initially empty;
       $P_0, P_F \subseteq P$ — non-empty sets of source and sink places;
       $\sigma$ — an event log trace;
Output: j — number of token jumps;
        k — number of tokens consumed/produced;
1  j ⟵ 0; k ⟵ 0;
2  populateSourcePlaces($P_0, R(\sigma)$);
3  foreach $e = (a, R(e))$ in $\sigma$ do
4      $t$ ⟵ selectTransition($a$);
5      foreach $r$ in $R(e)$ do
6          if $r \notin M(p)$ such that $p \in {}^{\bullet}t \wedge \mathtt{type}(p) = \mathtt{type}(r)$ then      // location[$r$] $\neq p$
7              jump($r, p$);
8              j ⟵ j + 1;
9          endif
10     endfor
11     fire($t, R(e)$);
12     k ⟵ k + $|R(e)|$;
13 endfor
14 foreach $r$ in $R(\sigma)$ do
15     if $r \notin M(p)$ such that $p \in P_F \wedge \mathtt{type}(p) = \mathtt{type}(r)$ then
16         jump($r, p$);
17         j ⟵ j + 1;
18     endif
19 endfor
20 consumeAllObjectsFromSinkPlaces($P_F, R(\sigma)$);
21 k ⟵ k + $|R(\sigma)|$;      // count final transfers: consumption of all distinct objects from the sinks
22 return (j, k);
```

To illustrate how the algorithm works, we will consider the example depicted in Fig. 5, which describes step-by-step the replay of trace σ_2 in Table 1 on the CPN of Fig. 1. For compactness, transition names are used instead of activity labels. Firstly, our method extracts all distinct objects of the trace, inserting them in source places according to their type (function `populateSourcePlaces`). For example, four objects are extracted from σ_2 — buy orders `b1`, `b2` and sell orders `s1`, `s2`. Thus, the source place p_1 for buy orders ($\mathtt{type}(p_1) = \mathtt{OB}$) is populated with tokens `b1` and `b2`, and the source place p_2 for sell orders ($\mathtt{type}(p_2) = \mathtt{OS}$) is populated with tokens `s1` and `s2`.

After populating source places with the distinct objects, we start to replay the trace on the CPN. As described in lines 3-13 of Algorithm 1, for each event $e = (a, R(e))$, the following steps are performed. We select a transition t in the CPN labeled with activity a. Then, for every object $r \in R(e)$, we check if the input place p of t contains object r. If the latter is not true, then r is moved from its current location in the model to place p. Each token jump is counted by incrementing the value of counter `j`. Afterwards, when all observed objects in $R(e)$ are located in the input places of transition t, then t fires. The transition consumes such objects from input places, transferring them to its output places. The counter `k` is incremented by the number of tokens transferred.



a	{b1}
e	{b1,s1}
e	{b2,s1}
b	{s2}

(start)
j = 0 | k = 0
$e_1 = (a,\{\text{b1}\})$
j = 0 | k = 1
$e_2 = (e,\{\text{b1,s1}\})$
j = 1 | k = 3
$e_3 = (e,\{\text{b2,s1}\})$
j = 3 | k = 5
$e_4 = (b,\{\text{s2}\})$
j = 3 | k = 6
(end)
j = 4 | k = 10
p_1 p_2 p_3 p_4 p_5 p_6
a b c d e
x y
s1 s2 b1 b2

Fig. 5: Replay of trace σ_2 of Table 1 on top of the CPN of Fig. 1 using Algorithm 1.

As an example, let us focus on the replay of $e_2 = (e, \{\mathtt{b1}, \mathtt{s1}\})$ depicted in Fig. 5. The sell order s1 is not in place p_4. This event corresponds to the situation of a trade between b1 and s1, but s1 was still not allowed to trade. However, to continue to replay, the object s1 jumps from its current location (place p_2) to place p_4. As observed later in Fig. 5, the same situation occurs on $e_3 = (e, \{\mathtt{b2}, \mathtt{s1}\})$, where both tokens are absent from the input places of the transition to fire. Thus, after forcing the replay of e_2 note how we can detect another deviating events.

After replaying all events in a trace, we check if all distinct objects arrived to their corresponding sink places. This final step allows to validate, for example, if the real system completely processed all objects. Lines 14-21 of Algorithm 1 describe this final step. For instance, in Fig. 5, object s2 was left in an intermediate place. This can be interpreted as a corrupt order that should have traded or been canceled at the end of a day. Hence, we force this token to jump to its sink place, which is p_6 since $\mathtt{s2} \in \mathtt{OS}$ and $\mathtt{type}(p_6) = \mathtt{OS}$. When all tokens are in the sink places of the CPN, they are consumed by the "environment". Note that the counter of transfers k is incremented by the number of all distinct objects consumed in this final step.

Time Complexity. We briefly analyze the time complexity of our method. Let $n = (\sum_{\forall e \in \sigma} |R(e)|)$ be the number of objects recorded in all events of a trace σ. Let $T(n)$ be the time taken to execute Algorithm 1. We sketch out that $T(n)$ is $O(n)$, where $O(n)$ is the standard asymptotic notation, referring that the execution time of the method *linearly growths* according to the number of objects n in all events of a trace. This bound can be guaranteed under the assumption that access to elements of a CPN model only requires constant time, i.e., the time taken to visit a transition or a place given its name is negligible.

Let us define $T(n) = T_{\mathtt{init}}(n) + T_{\mathtt{rep}}(n) + T_{\mathtt{end}}(n)$, where $T_{\mathtt{init}}(n)$ is the execution time of the function populateSourcePlaces (line 2 in Algorithm 1), $T_{\mathtt{rep}}(n)$ is the time taken to replay all events in trace σ (lines 3-13), and finally $T_{\mathtt{end}}(n)$ is the time taken to consume all tokens from sink places (lines 14-20). First, the function populateSourcePlaces visits all objects in all events of a trace σ, looking for the set of all distinct objects $R(\sigma)$, so this operation takes up to n steps. Then, each distinct object $r \in R(\sigma)$ is inserted in the source place of type $\mathtt{type}(r)$, which can take up to $|R(\sigma)| \leq n$ steps. Thus, $T_{\mathtt{init}}(n)$ is $O(n)$.

Now, we sketch that the time taken $T_{\mathtt{rep}}(n)$ to replay all events is $O(n)$. This part of the algorithm performs n iterations as it visits each object $r \in R(e)$ of every event e in σ. Below, we rewrite Lines 3-13 of Algorithm 1 illustrating that the operations performed for every object can be performed in constant time.

```
1  foreach $e = (a, R(e))$ in $\sigma$ do
2    foreach $r$ in $R(e)$ do
3      $p \leftarrow$ inputPlace[$a$, type($r$)];
4      if location[$r$] $\neq p$ then                  // $r \notin M(p)$ such that $p \in {}^{\bullet}t \;\wedge\;$ type($p$) = type($r$)
5        marking[location[$r$]].remove($r$);
6        marking[p].insert($r$);
7        location[$r$] $\leftarrow p$;                                              // jump($r, p$)
8        j $\leftarrow$ j + 1;
9      endif
10     marking[$p$].remove($r$);
11     marking[outputPlace[$a$, type($r$)]].insert($r$);  // part of firing: $r$ transferred to an output place
12     location[$r$] $\leftarrow$ outputPlace[$a$, type($r$)];
13     k $\leftarrow$ k + 1;
14   endfor
15 endfor
```

In the code above, we consider that the CPN is stored in the following associative arrays: location tracks the position of each object in the CPN; marking stores the tokens contained by every place, and inputPlace and outputPlace indicates input/output places of a transition given a type. Notably, as shown above in Line 3, each object is directly related by its type to exactly one input place as we consider CPNs whose transitions have input places of distinct types (i.e., Definition 2) and events are syntactically correct w.r.t. to the CPN.

Then, if the associative arrays representing the CPN guarantee constant time to access, remove and insert elements, then it follows that the operations for every object in each event are performed in constant time. Thus, the execution time of the trace replay only depends on the number of objects n in the trace, following that $T_{\texttt{rep}}(n)$ is $O(n)$. The time taken $T_{\texttt{end}}(n)$ in Lines 14-20 of Algorithm 1 is also $O(n)$ as $R(\sigma) \leq n$ distinct objects are consumed from sink places of the model. Finally, since $T_{\texttt{init}}(n)$, $T_{\texttt{rep}}(n)$, and $T_{\texttt{end}}(n)$ are $O(n)$, then the execution time $T(n) = T_{\texttt{init}}(n) + T_{\texttt{rep}}(n) + T_{\texttt{end}}(n)$ of Algorithm 1 is also $O(n)$.

Fitness Metric. We introduce a global metric, namely fitness, to measure the overall degree of conformance between a trace of an event log and a CPN. It allows to quantify to which extent the behavior seen in the trace complies with the CPN model. The metric is based on a proportion of the total number of token jumps `j` and tokens transferred `k`.

Definition 4 (Fitness). *Let σ be a trace, and let CP be a colored Petri net. Let* `j` *be the total number of token jumps and, let* `k` *be the total number of tokens transferred, computed in Algorithm 1 with σ and CP as input. Then, the (global) fitness metric* $\texttt{fit}(\sigma, CP)$ *is defined as:*

$$\texttt{fit}(\sigma, CP) = 1 - \frac{\texttt{j}}{\texttt{k}}$$

We shall demonstrate that $0 \leq \texttt{fit}(\sigma, CP) \leq 1$. Let us focus on counter `k`. In each event e, we transfer $|R(e)|$ tokens, as we force to replay all event's objects. Also, all distinct objects are consumed at the end of the method. Thus, we have that $k = (\sum_{\forall e \in \sigma} |R(e)|) + |R(\sigma)|$ with $|R(\sigma)| > 0$. Regarding the counter `j`. Let $\texttt{j}_e$ be the number of token jumps made in an event e of a trace σ. In every event e, we know that at most $|R(e)|$ jumps can be made, so $0 \leq \texttt{j}_e \leq |R(e)|$. Let $\texttt{j}_F$ be the number of token jumps of the distinct objects to sink places as they remained in intermediate places after the replay (e.g., see Lines 14-21 in Algorithm 1). We know that $0 \leq \texttt{j}_F \leq |R(\sigma)|$. Now, let us formulate the total number of jumps as $\texttt{j} = (\sum_{\forall e \in \sigma} \texttt{j}_e) + \texttt{j}_F$. Then, it follows that $\texttt{j} \leq \texttt{k}$. Therefore, $0 \leq \texttt{fit}(\sigma, CP) \leq 1$.

We now extend the definition of fitness for an event log as the average of the fitness values, which are computed upon the replay of each trace in the event log on top of a colored Petri net.

Definition 5 (Fitness of an event log). *Let L be an event log, and let CP be a colored Petri net. We denote by* $\texttt{fit}(L, CP)$ *the average fitness value obtained upon replaying individually every trace σ on top of CP using Algorithm 1, where:*

$$\texttt{fit}(L, CP) = \frac{1}{|L|} \sum_{\forall \sigma \in L} \texttt{fit}(\sigma, CP)$$

For example, let us consider the replay of traces σ_1 and σ_2 of Table 1 on top of the CPN of Fig. 1. After the execution of Algorithm 1 with σ_1, the obtained fitness value is $\texttt{fit}(\sigma_1, CP) = 1 - \frac{0}{9} = 1$, whereas with σ_2, we have that $\texttt{fit}(\sigma_2, CP) = 1 - \frac{4}{10} = 0.6$. These global measures may be interpreted as follows: the overall system's behavior observed in σ_1 complies completely with the specification model. Conversely, $\texttt{fit}(\sigma_2, CP) = 0.6$ indicates that only 60% of the overall behavior observed in σ_2 complied with specification model. Then, by considering both traces of the log using Definition 5, $\texttt{fit}(L, CP) = 0.8$ gives an estimation on how in average the system, as observed in the logs, complies the specification model.

5.2. Local Conformance Diagnostics

In the previous part of this section, we introduced a global conformance measure, namely fitness (Definitions 4 and 5). This measure allows to quantify to which extent the real system, as observed in logs, comply with the specification. Whilst such a metric provides an overall compliance estimation for the whole system, in many applications is required to provide *local diagnostics*, i.e., in which precise system components deviations are occurring, and in which magnitude.

In this part, we present *local conformance metrics* that are related to precise components of a system, and which can be computed upon the execution of our conformance checking method. Our approach is based on the direct association between real components of a system and components of a CPN input model. Recall that activities correspond to transitions, real locations to places, and the relation between a location and an activity is represented by an arc. Thus, by keeping track of the proportion of token transfers and jumps flowing through a model component, we can precisely indicate the number and magnitude of deviations occurred in the part of the real system that such model component represents. In what follows, we introduce these metrics, and we illustrate their usage in an example.

Definition 6 (Place-conformance). *Let σ be a trace, and let $p \in P$ be a place in a CPN. Let $\mathtt{k}_\sigma(p)$ be the number of tokens consumed from p upon the replay of σ, and let $\mathtt{j}_\sigma(p)$ denote the number of token jumps to place p upon the replay of σ. The place-conformance $\mathtt{fit}_\sigma(p)$ is defined as:*

$$\mathtt{fit}_\sigma(p) = \begin{cases} 1 - \dfrac{\mathtt{j}_\sigma(p)}{\mathtt{k}_\sigma(p)} & : \quad \mathtt{k}_\sigma(p) > 0 \\ \perp & : \quad \textit{otherwise} \end{cases}$$

The *place-conformance* $\mathtt{fit}_\sigma(p)$ compares the number of tokens consumed from place p, when replaying trace σ, with how many of them actually jumped to p to force transition firings. Hence, this metric can be seen as the proportion of objects that comply with be at the location represented by place p upon the execution of any activity that requires objects from such a location. If $\mathtt{fit}_\sigma(p)$ is close to 1, then almost no tokens jumped to p, e.g., objects are respecting the path established in the specification. Conversely, if $\mathtt{fit}_\sigma(p)$ is close to 0, then most of the tokens are jumping to place p, e.g., the majority of the objects skip previous activities that precede the location represented by p. Note that if $\mathtt{k}_\sigma(p) = 0$, then $\mathtt{fit}_\sigma(p)$ is not defined, i.e., assessments cannot be computed since no traffic flowed through place p during the replay of σ.

Definition 7 (Flow-conformance). *Let σ be a trace, and let $(p, t) \in F$ be an input arc in a CPN, s.t. $p \in P \land t \in T$. Let $\mathtt{k}_\sigma(p, t)$ be the number of tokens consumed from p to fire t when replaying trace σ, and let $\mathtt{j}_\sigma(p, t)$ denote the number of tokens that jumped to place p to force the firing of t. The flow-conformance $\mathtt{fit}_\sigma(p, t)$ is defined as:*

$$\mathtt{fit}_\sigma(p,t) = \begin{cases} 1 - \dfrac{\mathtt{j}_\sigma(p,t)}{\mathtt{k}_\sigma(p,t)} & : \quad \mathtt{k}_\sigma(p,t) > 0 \\ \perp & : \quad \textit{otherwise} \end{cases}$$

Definition 8 (Transition-conformance). *Let σ be a trace, and let $t \in T$ be a transition in a CPN. Let us define the active pre-set of transition t in trace σ as ${}^\bullet t_\sigma = \{p \mid p \in {}^\bullet t \land \mathtt{k}_\sigma(p, t) > 0\}$. The transition-conformance $\mathtt{fit}_\sigma(t)$ is defined as:*

$$\mathtt{fit}_\sigma(t) = \begin{cases} \dfrac{1}{|{}^\bullet t_\sigma|} \sum\limits_{\forall p \in {}^\bullet t_\sigma} \mathtt{fit}_\sigma(p,t) & : \quad |{}^\bullet t_\sigma| > 0 \\ \perp & : \quad \textit{otherwise} \end{cases}$$

Definitions 7 and 8 follow the same principle of place-conformance. Given the replay of a trace σ, the *flow-conformance* $\mathtt{fit}_\sigma(p,t)$ compares the number of tokens transferred, through the arc from place p to transition t, with how of many them jumped to p to force specifically the firing of t. This can be interpreted as the proportion of objects that comply to be at the location related to place p when executing activity $\Lambda(t)$.

The *transition-conformance* $\mathtt{fit}_\sigma(t)$ is the mean value of the flow-conformance between t and all the input places from which t consumes tokens. Thus, $\mathtt{fit}_\sigma(t)$ diagnoses how many of the objects consumed by activity $\Lambda(t)$, from all its required locations, correspond to outliers.

We now exemplify the usage of the local conformance metrics. Let us recall our motivating example of trading systems shown in Section 2. Fig. 6(a) shows a model representing a real trading system S_0 whose behavior is slightly deviated from the specification model of Fig. 1. In particular, some sell orders in S_0 can be submitted to the sell side of an order book (place p_4) by skipping activity b. Now, let us consider a trace σ from S_0. Let us assume that σ corresponds to the observed interaction of 20 distinct objects (i.e., 10 buy orders and 10 sell orders). We then ran our method to check conformance between σ and the model of Fig. 1. Let us suppose $\mathtt{j} = 5$ and $\mathtt{k} = 55$ as the total number of jumps and token transfers computed by Algorithm 1, thereby obtaining a fitness value $\mathtt{fit}(\sigma, CP) = 1 - \frac{5}{55} = 0.909$ (i.e., Definition 4). Nevertheless, it becomes more insightful to diagnose conformance in precise system components. To this aim, we calculate the local conformance metrics previously introduced and we extend the specification of the specification model of Fig. 1 with such metrics. The model is also extended with relevant traffic statistics of token jumps and transfers in the model components (i.e., see Fig. 6(b)).

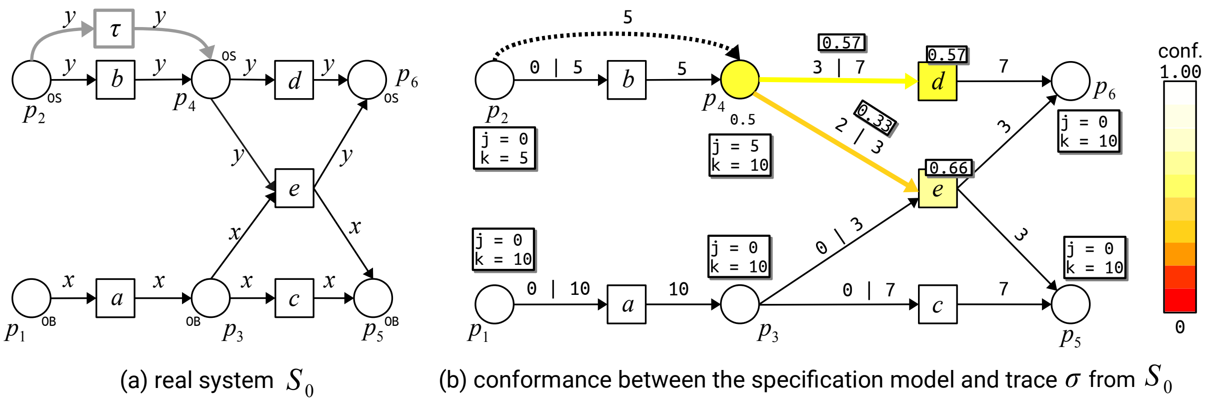


Fig. 6: Conformance results between a trace σ of system S_0 (a) and the specification (b).

We briefly discuss the specification model of Fig. 6(b) which has been enriched with conformance diagnostics. First, the number of transferred tokens $\mathtt{k}_\sigma(p)$ and token jumps $\mathtt{j}_\sigma(p)$ are shown besides each place p, e.g., $\mathtt{k}_\sigma(p_2) = 10$ and $\mathtt{j}_\sigma(p_2) = 5$. Jumps between places are displayed as dotted lines labeled with their frequency. In this example, jumps corresponded to 5 sell orders that in the system S_0 moved to the sell side of the order book (place p_4) using the silent action τ. This action is not recorded in trace σ nor allowed by the specification model shown in Fig. 1. Then, upon the replay of activities d and e, these sell orders were not in place p_4, but in place p_2. Hence, to force replay, these 5 tokens jumped from place p_2 to place p_4.

Input arcs are labeled using notation $\mathtt{j}_\sigma(p,t) \mid \mathtt{k}_\sigma(p,t)$ where $\mathtt{k}_\sigma(p,t)$ is the number of tokens consumed by transition t from place t, and $\mathtt{j}_\sigma(p,t)$ indicates the number of jumps to place p to force the firing of t. As an example, the label of input arc (p_4, d) indicates that 7 sell orders were consumed by activity d, but 3 of them were not ready in place p_4 before the firing. Output arcs are simply labeled with the number of resources transferred from a transition to a place.

Local conformance diagnostics are displayed on components of the specification model. For example, for place p_4, the place-conformance $\mathtt{fit}_\sigma(p_4) = 0.5$. This can be interpreted that only half of the time a sell order was ready in the sell side of the order book upon the execution of activity d or e. The flow-conformance $\mathtt{k}_\sigma(p_4, e) = 0.33$, i.e., activity e consumed a sell order 3 times, but two of these orders were not ready in place p_4. The transition-conformance $\mathtt{fit}_\sigma(e) = 0.66$ is the mean value between $\mathtt{fit}_\sigma(p_3, e) = 1$ and $\mathtt{fit}_\sigma(p_4, e) = 0.33$. This means that buy orders were always available in place p_3, whereas only a third of the time sell orders were available in place p_4. As a result, 66% of the time no deviations were observed upon the execution of activity e.

Notably, a practical benefit of the use of local conformance metrics is their combination with the notion of a *heat map*. For example, in Fig. (see Fig. 6(b)) a *heat bar* is displayed in the right side of the model denoting that the lower the conformance measure of a component, then the more red that such a component is painted. This allows us to quickly identify which components experienced more deviations. For instance, in Fig 6(b), it can be easily seen that deviations are localized in the component of the system related to sell orders.

Finally, note that the introduced local measures are computed based on the information provided by a single trace σ. We close this section by extending definitions of these measures to event logs. Considering now all traces in the log, they shall allow to diagnose the average magnitude of deviations in precise components of the system.

Definition 9 (Place-conformance of an event log). *Let L be an event log, and let $p \in P$ be a place in a CPN. Let us define $L_p = \{\sigma \mid \sigma \in L \ \land \ \mathtt{k}_\sigma(p) > 0\}$. The place-conformance $\mathtt{fit}_L(p)$ is defined as:*

$$\mathtt{fit}_L(p) = \begin{cases} \dfrac{1}{|L_p|} \sum\limits_{\forall \sigma \in L_p} \mathtt{fit}_\sigma(p) & : \quad |L_p| > 0 \\ \bot & : \quad \textit{otherwise} \end{cases}$$

Definition 10 (Flow-conformance of an event log). *Let L be an event log, and let $(p, t) \in F$ be an input arc in a CPN, s.t. $p \in P \land t \in T$. Let us define $L_{(p,t)} = \{\sigma \mid \sigma \in L \ \land \ \mathtt{k}_\sigma(p, t) > 0\}$. The flow-conformance $\mathtt{fit}_L(p, t)$ is defined as:*

$$\mathtt{fit}_L(p, t) = \begin{cases} \dfrac{1}{|L_{(p,t)}|} \sum\limits_{\forall \sigma \in L_{(p,t)}} \mathtt{fit}_\sigma(p, t) & : \quad |L_{(p,t)}| > 0 \\ \bot & : \quad \textit{otherwise} \end{cases}$$

Definition 11 (Transition-conformance of an event log). *Let L be an event log, and let $t \in T$ be a transition in a CPN, s.t. $p \in P \land t \in T$. Let us define $L_t = \{\sigma \mid \sigma \in L \ \land \ \exists p : \mathtt{k}_\sigma(p, t) > 0\}$. The transition-conformance $\mathtt{fit}_L(t)$ is defined as:*

$$\mathtt{fit}_L(t) = \begin{cases} \dfrac{1}{|L_t|} \sum\limits_{\forall \sigma \in L_t} \mathtt{fit}_\sigma(t) & : \quad |L_t| > 0 \\ \bot & : \quad \textit{otherwise} \end{cases}$$

6. Implementation and Experimental Evaluation

We have developed a prototypical implementation of our method in the Python programming language. Our solution is supported by SNAKES [19] — a library which facilitates the prototyping of high-level classes of Petri nets, including CPNs. In the following, we describe the functioning of our prototype, as well as we report an experimental evaluation of our method. The prototype and all the material of our experiment is freely available in our project repository [20].

Fig. 7 illustrates the organization of our prototypical implementation. Users of our solution simply need to invoke a program called "conformance checker". The program receives three input arguments: an option indicating the conformance method to use (e.g., replay with CPN using jumps), an event log formatted as a comma-separated value (CSV) file, and a CPN model. CPN models are built as Python scripts. This generic organization allows us to seamlessly extend our solution, incorporating other methods of our research. In addition, as depicted in Fig. 7, our prototype has an independent routine to generate artificial event logs (as defined in Definition 5) by running CPN models. When running a CPN to generate a trace, if two or more transitions are enabled in a given marking, then the routine randomly selects one of such transitions to fire.

As an example, Fig. 8 depicts the execution of our prototype in a command-line interface with the aforementioned input arguments (option 1 stands for the conformance method presented in this paper). Upon successful execution, the program generates an output folder with CSV files corresponding to: jumps detected, frequency per trace, and their average (file `jumps.csv`), traffic statistics and the local conformance metrics presented in Section 5 per each component type (files `arcs.csv`, `transitions.csv` and `places.csv`), and finally the file `traceFitness.csv` reports the total number of token jumps, token transfers and the resulting fitness per trace and their average (Definitions 4 and 5). As we motivated in Section 2, the information provided in these reports can be used to extend the specification model, so that to quickly identify observed deviations and their magnitude in concrete components of the system.

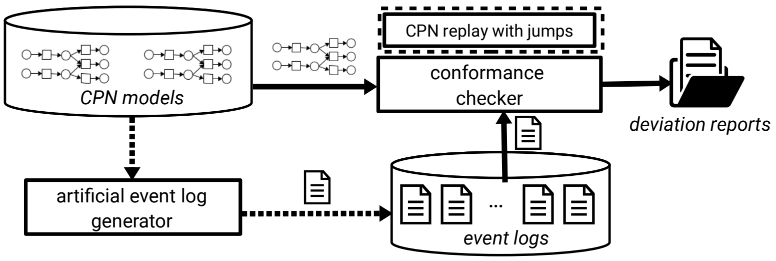
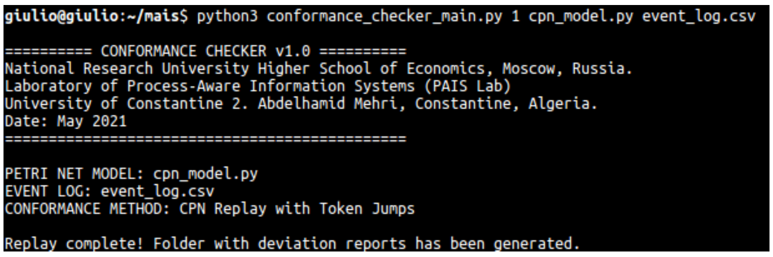


Fig. 7: Organization of the prototypical implementation of our conformance method.



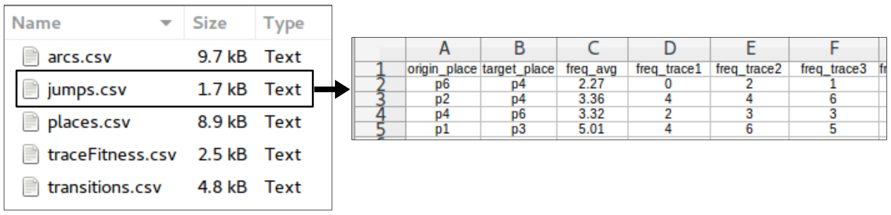
```
giulio@giulio:~/mais$ python3 conformance_checker_main.py 1 cpn_model.py event_log.csv

========== CONFORMANCE CHECKER v1.0 ==========
National Research University Higher School of Economics, Moscow, Russia.
Laboratory of Process-Aware Information Systems (PAIS Lab)
University of Constantine 2. Abdelhamid Mehri, Constantine, Algeria.
Date: May 2021
==============================================

PETRI NET MODEL: cpn_model.py
EVENT LOG: event_log.csv
CONFORMANCE METHOD: CPN Replay with Token Jumps

Replay complete! Folder with deviation reports has been generated.
```

(a) run of the prototype in the command-line interface



Name	Size	Type
arcs.csv	9.7 kB	Text
jumps.csv	1.7 kB	Text
places.csv	8.9 kB	Text
traceFitness.csv	2.5 kB	Text
transitions.csv	4.8 kB	Text

	A	B	C	D	E	F
1	origin_place	target_place	freq_avg	freq_trace1	freq_trace2	freq_trace3
2	p6	p4	2.27	0	2	1
3	p2	p4	3.36	4	4	6
4	p4	p6	3.32	2	3	3
5	p1	p3	5.01	4	6	5

(b) folder with deviation reports and fragment of the file `jumps.csv`

Fig. 8: Execution of the prototype in the interface (a) and example of generated files (b).

Experimental Evaluation. Using our prototype, we conducted an experiment with three event logs, artificially generated from different trading system models S_1, S_2, and S_3. Each event log was replayed on top of the specification model of Fig. 1. These models represent replicas of a trading system, according to the specification, but each of them with undesired behavior is increasingly added. For instance, system S_2 is a variation of S_1, but with a subtle modification to slightly increase its difference with the specification model. Table 2 describes each replica and its event log generated. Each event log consists of 100 traces and 20 resources (i.e., 10 buys orders and 10 sell orders). The aim of this experiment is two-fold: to showcase the use of local conformance measures presented in Section 5, computed with all traces of an event log, and to study the stability of the proposed measures, e.g., how much the metrics are affected by subtle increases of undesired behavior.

Table 2: Experimental settings

event log	*system source*	*number of events (total, avg. per trace)*	*system description (undesired behavior)*
L_1	S_1 (Fig. 9(a))	(2610, 26)	sell orders and buy orders skip activity b and a.
L_2	S_2 (Fig. 9(b))	(2726, 27)	S_1 + activity e may return sell orders to place p_4.
L_3	S_3 (Fig. 9(c))	(2575, 26)	S_2 + activity b may take sell orders to a deadlock place p_7.

Before discussing the conformance results, let us review the specification model in Fig. 1. According to the specification, in a system with no deviations each object must be transferred exactly 3 times: an order is submitted (activity a or b), then it trades or is canceled (activities c, d or e), and finally the order is consumed from a sink place. This implies that the replay of traces on the specification model, with 20 objects each, must count 60 token transfers (that is, 6000 transfers for an event log with 100 objects). However, as shown in Table 2, each system variant presents certain undesired behavior, so objects can move between certain locations disobeying the specification model. Hence, when replaying event logs of these system variants on top of the specification, observed deviations will incite token jumps between places, and less expected token transfers through the model structure. The latter is evidenced in columns *resources transferred* and *jumps detected* of Table 3. For ease of representation, the averages of transfers and jumps have been rounded.

Table 3 summarizes the results upon the replay of each event log on top of the specification model. For each variant, it can be observed how the introduction of a single subtle deviation induces more token jumps during replay, e.g., more objects flowing through paths unforeseen in the specification. It can be observed that the latter causes a monotonic and tenuous decrease in the average trace fitness (e.g., Definition 5). However, more intriguing becomes to identify where deviations have occurred and their magnitude. To this aim, items (d), (e), and (f) in Fig. 9 presents the specification model of Fig. 1 extended with local conformance diagnostics, computed when checking conformance in each variant, and which are associated with model components (Definitions 9—11). Input arcs and dotted lines representing jumps indicate the rounded average number of transferred/jumped tokens that flowed through them, considering all traces of a log variant. The introduction of certain undesired behavior in each variant is unveiled by our method as a jump, showing how such undesired behavior induces more deviations in a precise component. The latter impacts on the conformance-related measures, used to paint the model to clearly identify where deviations occurred more.

Table 3: Conformance results.

event log	*system source*	*fitness*	*resources transferred (total, avg. per trace)*	*jumps detected (total, avg. per trace)*	*kinds of jumps (origin, target, avg. frequency per trace)*
L_1	S_1	0.7974	(4999, 50)	(1001, 10)	$(p_2, p_4, 5)$, $(p_1, p_3, 5)$
L_2	S_2	0.7607	(5309, 53)	(1263, 13)	$(p_2, p_4, 5)$, $(p_1, p_3, 5)$, $(p_6, p_4, 3)$
L_3	S_3	0.7425	(5058, 51)	(1306, 13)	$(p_2, p_4, 3)$, $(p_1, p_3, 5)$, $(p_6, p_4, 2)$, $(p_4, p_6, 3)$

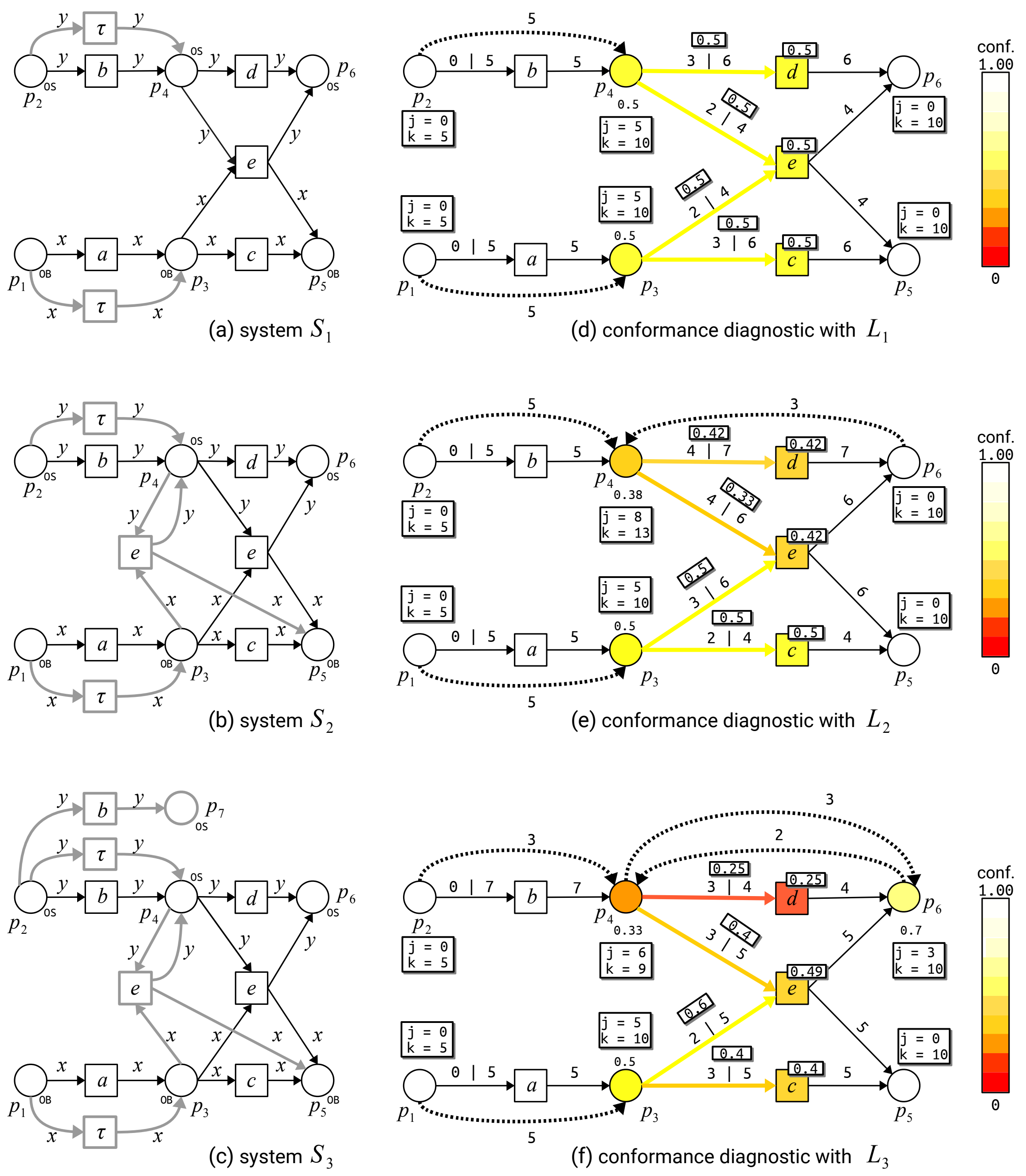

(a) system S_1
(d) conformance diagnostic with L_1
(b) system S_2
(e) conformance diagnostic with L_2
(c) system S_3
(f) conformance diagnostic with L_3
conf.
1.00
0

Fig. 9: Conformance checking between each event log of a system variant and the specification of Fig. 1.

.

7. Related Work

Conformance checking relies to a great extent on the expressive power of the models used to describe expected behavior of systems under evaluation. In this regard, state-of-the-art conformance methods use workflow nets (WF-nets) which is an ordinary Petri net employed to describe the control-flow of a process executed in isolation, that is, one single process instance after another. Thus, methods using WF-nets do not lend themselves for validating multiple instances concurrently interacting in a system. In contrast, the use of more expressive models overcoming the mentioned limitation becomes a demand in real-world applications.

For example, papers [21, 22] present case studies where multi-instance modeling notations are needed to diagnose the behavior of objects interacting within modern computational environments. In what follows, we review how different works address this challenge by using various kinds of Petri net extensions, and focusing on their application for conformance checking and similar techniques.

Proclets are one of the earliest proposals in process mining to study and validate the interaction of process instances in a system [23]. Each instance runs in an independent workflow, and semantics are provided to describe communication between workflows. Among its different applications, the concept of Proclets evolved into *artifact-centric processes* for conformance checking [24]. Also, the decomposition of the conformance checking problem has been studied for artifact-centric processes [25] in order to perform replay between each artifact and its corresponding sub-log. Another variant of this model is reported in [26], where the authors check conformance of artifacts which are modeled using UML state and activity diagrams.

Object-centric Petri nets is a notation recently proposed to describe in a single model the interaction of multiple cases (process instances) [8, 27]. Object-centric Petri nets can be seen as another subclass of CPNs with certain characteristics. For instance, arcs that transfer an arbitrary number of objects are introduced to describe one-to-many or many-to-many interactions. In [27], the authors thus present a method to discover an object-centric Petri net from event logs. In particular, these event logs are built in *extensible object-centric* (XOC) format [28]. The usage of object-centric Petri nets for conformance checking presents open challenges. For instance, the model does not provide a direct assignment of objects to concrete variables, so multiple bindings may be chosen. In this light, the fact that one event can be associated with multiple valid bindings implies the need for recursive strategies of non-linear time complexity, e.g., backtracking.

Another notable research direction in conformance checking is the validation of additional behavioral dimensions (perspectives) of a single process [14, 15] such as time or data constraints. In [14] the authors present an alignment-based conformance method using Petri nets with data (DPNs). On the one hand, alignments find differences between a model and a trace by computing the shortest path in the state space of a synchronous product net, i.e., a Petri net composed by the input model and the trace [29]. On the other hand, a DPN is a WF-net whose transitions are equipped with data variables which can be read/written upon transition firings. In this way, additional process perspectives to analyze are encoded within these variables. In contrast, DPNs are not appropriate for analysis of multiple process instances, e.g., tokens are black dots and data values do not flow through the model. This is why, for instance, we have opted for a model based on CPNs whose tokens carry object identifiers.

CPNs have already been considered in the process mining field. For instance, Rozinat et al. considered the discovery of CPN-based models for simulation [30, 31]. Notably, in paper [32], we proposed a conformance checking method with a class of CPNs whose tokens are tuples carrying object identifiers and attributes, thereby allowing to detect various kinds of deviations. For instance, using logs whose events record the state of object attributes after the event's activity execution, the method detects if such attributes were transformed as specified by the CPN. Also, the method was applied to check compliance of a real-world trading system w.r.t. its specification. In this regard, we refer the reader to papers [33–36], which present our studies on the extraction of event logs and Petri net-based modeling of real-world trading systems. However, in the method presented in [32], the replay of a trace is stopped upon the occurrence of the first deviating event, and also local diagnostics on system components are not provided. Thus, the approach of token jumps and local conformance diagnostics presented in this paper can be used to extend such methods.

Nested Petri nets is an extension where tokens can be Petri nets themselves, which allow to describe the inner behavior of objects [37]. The model becomes useful when it is crucial not only to analyze the flow of objects within a system, but also to validate the inner behavior of these objects. For instance, in paper [38], we studied the conformance problem between a nested Petri net and an event log of a multi-agent system. We proposed a compositional approach where the behavior of each agent can be checked separately.

Nested Petri Nets have also been used in the related fields of adaptive process modeling and verification [39, 40]. Finally, another recent research direction on modeling and validation of object-centric systems focuses on the use of models that combine Petri nets with data persistence models such as relational databases. The Information Systems Modeling Language (ISML) [41] and catalog-nets [42] are examples of this research direction. For instance, in these works, methods are proposed for verifying the integrity of objects in the Petri net and their representation in databases.

8. Conclusion

In this paper, we have presented a replay-based conformance checking method to validate whether a system, whose components manage interacting objects of different classes, complies with its specification. As the modeling language for the specification, we considered a subclass of colored Petri nets, whereas for describing real behavior we considered event logs where events are equipped with sets of involved objects. These objects refer to the individual resources involved in the execution of an activity. Regarding the subclass of CPNs, we particularly considered conservative workflow CPNs which faithfully characterizes systems handling the end-to-end processing of distinguishable objects. Among the syntactic constraints, we considered that transitions do not have input places of the same type. The latter assures in our setting only one binding for an event to replay. Consequently, we can provide an algorithm that, unlike alignments, has linear time complexity. Noteworthy to mention that the constraint of distinct types for input places can be dropped at the price of losing linear time complexity. For such a case, multiple bindings then can be associated to an event, and thus the algorithm shall look for the correct one using, e.g., recursive-based strategies such as backtracking.

The replay strategy of our method has been based on populating tokens in the model that correspond to distinct objects observed in the trace. For an event to replay, if objects are not located as tokens in specified input places of the transition to fire, we proposed a jump strategy to move tokens from their current location in the model to the required places. Interestingly, this approach not only allows us to force transition firings to find more deviations in a trace, but also recorded jumps between places unveil the so-called *desire lines*, i.e., paths of objects which are unforeseen in the specification [17].

Leveraging the fact that real locations and activities directly correspond to concrete components of a CPN, we proposed local conformance diagnostics that can be used to clearly identify the severity of deviations in precise parts of a system. Besides, we presented a prototypical implementation of our method, and we illustrated its usage with a case study on trading systems. The prototype is freely available for its usage and extension. In this regard, the prototype may be upgraded to provide automatic enhancement of an input CPN model with conformance diagnostics.

The work presented in this paper may give ground for different research directions. For instance, previous works on conformance checking based on Petri net models whose tokens carry object attributes or object inner behavior (e.g., [32, 38]) can be extended with the strategies presented in this paper, e.g., use of jumps and local conformance diagnostics. Also, it may be of interest to study other variants for this method, e.g., where tokens not only represent distinct objects, but also relationships between each other. This would imply, for instance, that the state of the objects do not correspond to a single location, but their state is in some sense distributed among places, similar to the approaches of ISML and catalog nets. The latter would make to the approach presented in this paper more intriguing and challenging.

References

[1] W. van der Aalst, *Process Mining: Data Science in Action*, 2nd. Springer, 2016.

[2] T. Murata, "Petri nets: Properties, analysis and applications", *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[3] J. Carmona, B. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking: Relating Processes and Models*, 1st. Springer, 2018.

[4] L. Reinkemeyer, *Process Mining: Principles, Uses Cases, and Outlook*. Springer, 2020.

[5] V. Rubin, A. Mitsyuk, I. Lomazova, and W. van der Aalst, "Process Mining Can Be Applied to Software Too!", in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ACM, 2014.

[6] F. Leotta, M. Mecella, and J. Mendling, "Applying Process Mining to Smart Spaces: Perspectives and Research Challenges", in *Advanced Information Systems Engineering Workshops*, A. Persson and J. Stirna, Eds., ser. LNBIP, vol. 215, Springer, 2015, pp. 298–304.

[7] F. Mannhardt, P. Arnesen, and A. D. Landmark, "Estimating the Impact of Incidents on Process Delay", in *2019 International Conference on Process Mining (ICPM)*, IEEE, 2019, pp. 49–56.

[8] W. van der Aalst, "Object-Centric Process Mining: Dealing with Divergence and Convergence in Event Data", in *Software Engineering and Formal Methods*, P. C. Ölveczky and G. Salaün, Eds., ser. LNCS, vol. 11724, Springer, 2019, pp. 3–25.

[9] D. Fahland, "Artifact-Centric Process Mining", in *Encyclopedia of Big Data Technologies*, S. Sakr and A. Y. Zomaya, Eds. Springer, 2019, pp. 108–117. DOI: 10.1007/978-3-319-77525-8_93.

[10] J. M. E. M. van der Werf and A. Polyvyanyy, "The Information Systems Modeling Suite", in *Application and Theory of Petri Nets and Concurrency*, R. Janicki, N. Sidorova, and T. Chatain, Eds., ser. LNCS, vol. 12152, Springer, 2020, pp. 414–425.

[11] D. Fahland, "Describing Behavior of Processes with Many-to-Many Interactions", in *Application and Theory of Petri Nets and Concurrency*, S. Donatelli and S. Haar, Eds., ser. LNCS, vol. 11522, Springer, 2019, pp. 3–24.

[12] W. M. P. van der Aalst and A. Berti, "Discovering Object-centric Petri Nets", *Fundamenta informaticae*, vol. 175, no. 1/4, pp. 1–40, 2020.

[13] S. Ghilardi, A. Gianola, M. Montali, and A. Rivkin, "Petri Nets with Parameterised Data", in *Business Process Management*, D. Fahland, C. Ghidini, J. Becker, and M. Dumas, Eds., ser. LNCS, vol. 12168, Springer, 2020, pp. 55–74.

[14] F. Mannhardt, M. Leoni, de, H. Reijers, and W. van der Aalst, *Balanced multi-perspective checking of process conformance*, ser. Computer. Springer, 2015, vol. 98, pp. 407–437.

[15] M. de Leoni and W. van der Aalst, "Data-Aware Process Mining: Discovering Decisions in Processes Using Alignments", ser. Symposium on Applied Computing (SAC 2013), ACM, 2013, pp. 1454–1461.

[16] K. Jensen and L. M. Kristensen, *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*, 1st. Springer, 2009.

[17] W. van der Aalst, "Desire Lines in Big Data", in *Encyclopedia of Social Network Analysis and Mining*, R. Alhajj and J. Rokne, Eds. 2014, pp. 351–364. DOI: 10.1007/978-1-4614-6170-8_396.

[18] L. Harris, *Trading and Exchanges: Market Microstructure for Practitioners*. Oxford University Press, 2003.

[19] F. Pommereau, "SNAKES: A Flexible High-Level Petri Nets Library", in *Application and Theory of Petri Nets and Concurrency*, R. Devillers and A. Valmari, Eds., ser. LNCS, vol. 9115, Springer, 2015, pp. 254–265.

[20] Github, *Object-centric Replay-based Conformance Checking Project Repository*, https://github.com/jcarrasquel/hse-uamc-conformance-checking.

[21] G. Meroni, L. Baresi, M. Montali, and P. Plebani, "Multi-party business process compliance monitoring through IoT-enabled artifacts", *Information Systems*, vol. 73, pp. 61–78, 2018.

[22] R. Seiger, F. Zerbato, A. Burattin, L. García-Bañuelos, and B. Weber, "Towards IoT-driven Process Event Log Generation for Conformance Checking in Smart Factories", in *2020 IEEE 24th International Enterprise Distributed Object Computing Workshop (EDOCW)*, IEEE, 2020, pp. 20–26.

[23] W. van der Aalst, P. Barthelmess, C. Ellis, and J. Wainer, "Proclets: A Framework for Lightweight Interacting Workflow Processes", *International Journal of Cooperative Information Systems*, vol. 10, no. 04, pp. 443–481, 2001.

[24] D. Fahland, M. de Leoni, B. van Dongen, and W. van der Aalst, "Behavioral Conformance of Artifact-Centric Process Models", in *LNBIP*, W. Abramowicz, Ed., vol. 87, Berlin, Heidelberg: Springer, 2011, pp. 37–49.

[25] D. Fahland, M. de Leoni, B. van Dongen, and W. van der Aalst, "Conformance Checking of Interacting Processes with Overlapping Instances", in *Business Process Management*, S. Rinderle-Ma, F. Toumani, and K. Wolf, Eds., ser. LNCS, vol. 6896, Springer, 2011, pp. 345–361.

[26] M. Estañol, J. Munoz-Gama, J. Carmona, and E. Teniente, "Conformance Checking in UML Artifact-Centric Business Process Models", *Software and Systems Modeling*, vol. 18, no. 4, pp. 2531–2555, 2019.

[27] W. van der Aalst and A. Berti, "Discovering Object-Centric Petri Nets", *Fundamenta Informaticae*, vol. 175, 2020.

[28] G. Li, E. G. L. de Murillas, R. M. de Carvalho, and W. van der Aalst, "Extracting Object-Centric Event Logs to Support Process Mining on Databases", in *Information Systems in the Big Data Era*, J. Mendling and H. Mouratidis, Eds., Springer, 2018, pp. 182–199.

[29] A. Adriansyah, "Aligning observed and modeled behavior", PhD thesis, Eindhoven University of Technology (TU/e), 2014.

[30] A. Rozinat, R. Mans, M. Song, and W. van der Aalst, "Discovering colored Petri nets from event logs", *International Journal on Software Tools for Technology Transfer*, vol. 10, no. 1, pp. 57–74, 2008.

[31] A. Rozinat, R. Mans, M. Song, and W. van der Aalst, "Discovering simulation models", *Information Systems*, vol. 34, no. 3, pp. 305–327, 2009.

[32] J. C. Carrasquel, K. Mecheraoui, and I. A. Lomazova, "Checking Conformance Between Colored Petri Nets and Event Logs", in *Analysis of Images, Social Networks and Texts*, W. van der Aalst, V. Batagelj, D. I. Ignatov, M. Khachay, O. Koltsova, A. Kutuzov, S. O. Kuznetsov, I. A. Lomazova, N. Loukachevitch, A. Napoli, A. Panchenko, P. M. Pardalos, M. Pelillo, A. V. Savchenko, and E. Tutubalina, Eds., ser. LNCS, vol. 12602, Springer, 2021, pp. 435–452.

[33] J. C. Carrasquel, S. Chuburov, and I. A. Lomazova, "Pre-processing Network Messages of Trading Systems into Event Logs for Process Mining", in *Tools and Methods of Program Analysis*, ser. CCIS, vol. 1288, Springer, 2021, pp. 88–100.

[34] J. C. Carrasquel, I. A. Lomazova, and I. L. Itkin, "Towards a Formal Modelling of Order-driven Trading Systems using Petri Nets: A Multi-Agent Approach", in *Modeling and Analysis of Complex Systems and Processes (MACSPro)*, I. A. Lomazova, A. Kalenkova, and R. Yavorsky, Eds., ser. CEUR, vol. 2478, 2019.

[35] J. C. Carrasquel and I. A. Lomazova, "Modelling and Validation of Trading and Multi-Agent Systems: An Approach Based on Process Mining and Petri Nets", in *Proc. of the ICPM Doctoral Consortium*, B. van Dongen and J. Claes, Eds., ser. CEUR, vol. 2432, 2019.

[36] J. C. Carrasquel, I. A. Lomazova, and A. Rivkin, “Modeling Trading Systems using Petri Net Extensions”, in *Int. Workshop on Petri Nets and Software Engineering (PNSE)*, M. Köhler-Bussmeier, E. Kindler, and H. Rölke, Eds., ser. CEUR, vol. 2651, 2020.

[37] I. A. Lomazova, “Nested Petri Nets – a Formalism for Specification and Verification of Multi-Agent Distributed Systems”, *Fundamenta Informaticae*, vol. 43, pp. 195–214, 2000.

[38] K. Mecheraoui, J. C. Carrasquel, and I. A. Lomazova, “Compositional Conformance Checking of Nested Petri Nets and Event Logs of Multi-Agent Systems”, in *Modeling and Analysis of Complex Systems and Processes (MACSPro)*, A. Shapoval, V. Popov, and I. Makarov, Eds., ser. CEUR, vol. 2795, 2020.

[39] I. A. Lomazova, “Nested Petri Nets for Adaptive Process Modeling”, in *Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, A. Avron, N. Dershowitz, and A. Rabinovich, Eds. Springer, 2008, vol. 4800, pp. 460–474.

[40] K. V. Hee, O. Oanea, A. Serebrenik, N. Sidorova, M. Voorhoeve, and I. Lomazova, “Checking Properties of Adaptive Workflow Nets”, *Fundam. Informaticae*, vol. 79, pp. 347–362, 2007.

[41] J. M. E. M. van der Werf and A. Polyvyanyy, “The Information Systems Modeling Suite”, in *Application and Theory of Petri Nets and Concurrency*, R. Janicki, N. Sidorova, and T. Chatain, Eds., ser. LNCS, vol. 12152, Springer, 2020, pp. 414–425.

[42] S. Ghilardi, A. Gianola, M. Montali, and A. Rivkin, “Petri Nets with Parameterised Data”, in *Business Process Management*, D. Fahland, C. Ghidini, J. Becker, and M. Dumas, Eds., ser. LNCS, vol. 12168, Springer, 2020, pp. 55–74.