Decomposing Alignment-based Conformance Checking of Data-aware Process Models

Massimiliano de Leoni[1⋆], Jorge Munoz-Gama[2⋆⋆], Josep Carmona[2], and Wil M.P. van der Aalst[1]

[1] Eindhoven University of Technology, Eindhoven (The Netherlands)
[2] Universitat Politècnica de Catalunya, Barcelona (Spain)

m.d.leoni@tue.nl, jmunoz@cs.upc.edu, jcarmona@cs.upc.edu, w.m.p.v.d.aalst@tue.nl

Abstract. Process mining techniques relate observed behavior to modeled behavior, e.g., the automatic discovery of a Petri net based on an event log. Process mining is not limited to process discovery and also includes conformance checking. Conformance checking techniques are used for evaluating the quality of discovered process models and to diagnose deviations from some normative model (e.g., to check compliance). Existing conformance checking approaches typically focus on the control-flow, thus being unable to diagnose deviations concerning data. This paper proposes a technique to check the conformance of data-aware process models. We use so-called *Petri nets with Data* to model data variables, guards, and read/write actions. Data-aware conformance checking problem may be very time consuming and sometimes even intractable when there are many transitions and data variables. Therefore, we propose a technique to decompose large data-aware conformance checking problems into smaller problems that can be solved more efficiently. We provide a general correctness result showing that decomposition does not influence the outcome of conformance checking. The approach is supported through ProM plug-ins and experimental results show significant performance improvements. Experiments have also been conducted with a real-life case study, thus showing that the approach is also relevant in real business settings.

Keywords: Process Mining, Conformance Checking, Divide-and-Conquer Techniques, Multi-Perspective Process Modelling

1 Introduction

Nowadays, most organizations document and analyze their processes in some form, and with it, the practical relevance of process mining is increasing as more and more event data becomes available. Process mining techniques aim to discover, monitor and improve real processes by extracting knowledge from event logs. The two most prominent process mining tasks are: (i) *process discovery:* learning a process model from example behavior recorded in an event log, and (ii) *conformance checking:* diagnosing

⋆ When conducting most of this research work, Dr. de Leoni was also affiliated with University of Padua and financially supported by the Eurostars - Eureka project PROMPT (E!6696).
⋆⋆ Supported by FPU Grant (AP2009-4959) and project FORMALISM (TIN-2007-66523)

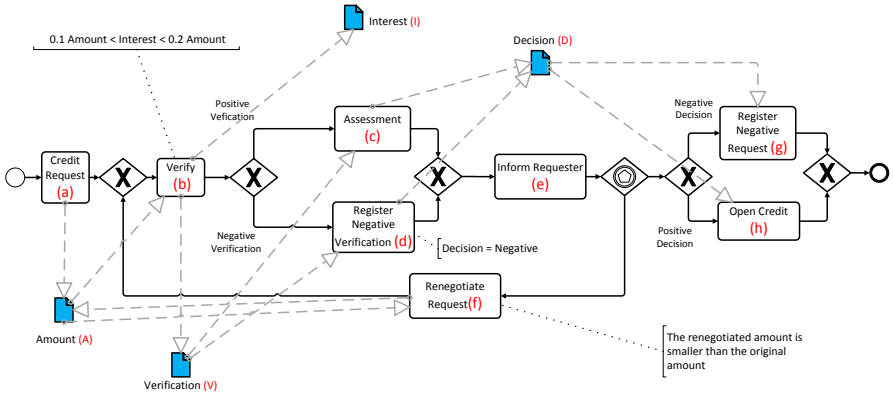

Fig. 1: Example of a (simplified) process to request loans. The dotted arcs going from a transition to a variable denote write operations; the arcs towards a transition denote read operations, i.e. the transition requires accessing the current variables' value. In the paper, each transition is abbreviated into a lower-case letter (e.g. a) and each variable is represented as a upper-case letter (e.g. A). The abbreviations are shown in brackets after the name of the transitions or variable names.

and quantifying discrepancies between observed behavior and modeled behavior [1]. Models that faithfully conform the reality are necessary to obtain trustful analysis and simulation results, for certification and regulation purposes, or simply to gain insight into the process.

Most of the work done in conformance checking in the literature focuses on the control-flow of the underlying process, i.e. the ordering of activities. There are various approaches to compute the fraction of events or traces in the log that can be replayed by the model [2,3].

In a data-aware process model, each case, i.e. a process instance, is characterized by its case variables. Paths taken during the execution may be governed by guards and conditions defined over such variables. A process model specifies the set of variables and their possible values, guards, and write/read actions. Since existing conformance checking techniques typically completely abstract from data, resources, and time, many deviations remain undetected. Therefore, the event log may record executions of process instances that appear fully conforming, even when it is not the case. Rigorous analysis of the data perspective is needed to reveal such deviations.

Let us consider the process that is modeled as BPMN diagram in Figure 1. It models the handling of loans requests from customers. It is deliberately oversimplified to be able to explain the concepts more easily. The process starts with a credit request where the requestor provides some documents to demonstrate the capability of paying the loan back. These documents are verified and the interest amount is also computed. If the verification step is negative, a negative decision is made, the requestor is informed and, finally, the negative outcome of the request is stored in the system. If verification

is positive, an assessment is made to take a final decision. Independently of the assessment's decision, the requestor is informed. Moreover, even if the verification is negative, the requestor can renegotiate the loan (e.g. to have lower interests) by providing further documents or by asking for a smaller amount. In this case, the verification-assessment part is repeated. If both the decision and verification are positive and the requestor is not willing to renegotiate, the credit is opened. Let us consider the following trace:[3]

$$\sigma_{ex} = \langle(\mathbf{a}, \emptyset, \{(A, 4000)\}), (\mathbf{b}, \{(A, 4000)\}, \{(I, 450), (V, \mathsf{false})\}), (\mathbf{c}, \{(V, \mathsf{false})\}, \{(D, \mathsf{true})\}), (\mathbf{e}, \emptyset, \emptyset), (\mathbf{f}, \{(A, 4000)\}, \{(A, 5000)\}), (\mathbf{b}, \{(A, 5000)\}, \{(I, 450), (V, \mathsf{false})\}), (\mathbf{d}, \{(V, \mathsf{false})\}, \{(D, \mathsf{false})\}), (\mathbf{e}, \emptyset, \emptyset), (\mathbf{h}, \{(D, \mathsf{true})\}, \emptyset)\rangle$$

Seen from a control-flow perspective only (i.e. only considering the activities' ordering), the trace seems to be fully conforming. Nonetheless, a number of deviations can be noticed if the data perspective is considered. First of all, if activity c is executed, previously activity b could not have resulted in a negative verification, i.e. V is set to false. Second, activity f cannot write value 5000 to variable A, as this new value is larger than the previous value, i.e. 4000. Furthermore, if the decision and verification are both negative, i.e. both V are D are set to false, then h cannot be executed at the end.

The identification of non-conforming traces clearly has value in itself. Nonetheless, organizations are often interested in explanations that can steer measures to improve the quality of the process. *Alignments* aim to support more refined conformance checking. An alignment aligns a case in the event log with an execution path of the process model as good as possible. If the case deviates from the model, then it is not possible to perfectly align with the model and a best matching scenario is selected. Note that for the same deviation, multiple explanations can be given. For instance, the problem that h was executed when it was not supposed to happen can be explained in two ways: (1) h should not have occurred because V and D are both set to false ("control-flow is wrong") and (2) V and D should both have been set to true because h occurs ("data-flow is wrong"). In order to decide for the most reasonable explanation, costs are assigned to deviations and we aim to find the explanation with the lowest cost. For instance, if assigning a wrong value to V and D is less severe than executing h wrongly, the second explanation is preferred. The seminal work in [3] only considers alignments in the control-flow part, thus ignoring the data-perspective aspect of conformance.

As we detail in Section 2.4, finding an alignment of an event log and a data-aware process model is undecidable in the general case. However, to make the problem decidable, works [4,5] put forward the limitation that guards need to be linear (in)equations. Readers are also referred to them for a state-of-the-art analysis of data-aware conformance checking. These works also show that, even with that limitation, the problem of finding an alignment of an event log can become intractable since the problem's complexity is exponential on the size of the model, i.e. the number of activities and data variables. In this paper, while keeping the limitations mentioned above, we aim to speed

[3] Notation $(\mathbf{act}, r, w)$ is used to denote the occurrence of activity act that writes and reads variables according to functions w and r, e.g., $(\mathbf{b}, \{(A, 4000)\}, \{(I, 450), (V, \mathsf{false})\})$ is an event corresponding to the occurrence of activity **b** while reading value 4000 for variable A and writing values 450 and false to variables I and V respectively. $(\mathbf{e}, \emptyset, \emptyset)$ corresponds to the occurrence of activity **e** without reading/writing any variables.

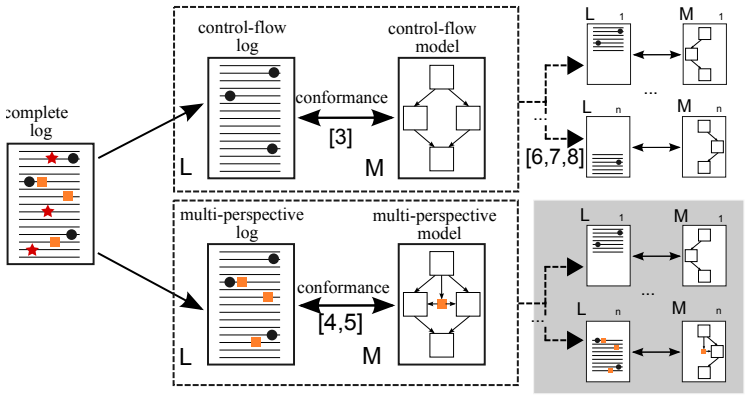

Fig. 2: Positioning the contribution of this paper with respect to the state of the art: the gray area identifies the novelty of the proposed technique.

up the computation of alignments by using a divide-and-conquer approach. The data-aware process model is split into smaller partly overlapping model fragments. For each model fragment a sublog is created by projecting the initial event log onto the activities used in the fragment. Given the exponential nature of conformance checking, this may significantly reduce the computation time. If the decomposition is done properly, then any trace that fits into the overall model also fits all of the smaller model fragments and vice versa. Figure 2 positions the contribution of this paper with respect to the state-of-the-art alignment-based techniques. The top part identifies the alignment-based conformance checking that considers the control flow, only. The bottom part refers to the alignment-based conformance checking techniques that account for data, as well. Regarding the control-flow only, several approaches have been proposed to decompose process mining problems, both for discovery and conformance checking. As described in [6], it is possible to decompose process mining problems in a variety of ways. Special cases of this more general theory are passages [7] and SESE-based decomposition [8]. However, these approaches are limited to control-flow. Indeed, some techniques exist that also consider the data aspects (i.e. [4,5]) but without exploiting the possibility of decomposing the data-aware model. In this paper, we extend the control-flow approaches mentioned above to also take data into account, which coincides with the gray area in Figure 2. Finally, the work in [9] (and similar) for data-aware conformance on declarative models is orthogonal to the contributions listed in Figure 2, that are focused on procedural models.

The decomposed data-aware conformance checking approach presented in this paper has been implemented as plug-ins for the ProM framework. We conducted experiments related to a real-life case study as well as with several synthetic event logs. Experimental results show that data-aware decomposition may indeed be used to significantly reduce the time needed for conformance checking and that the problem is practically relevant since models of real processes can actually be decomposed.

Preliminaries are presented in Section 2. Section 3 introduces our approach for data-aware decomposition. Section 4 describes different algorithms for instantiating the gen-

eral results presented in Section 3. Section 5 reports on experimental results. Section 6 concludes the paper.

2 Preliminaries

2.1 System Nets

Petri nets and their semantics are defined as usual: a Petri net is a tuple (P, T, F) with P the set of places, T the set of transitions, $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ the flow relation. A place p is an input place of a transition t iff $(p, t) \in F$; similarly, p is an output place of t iff $(t, p) \in F$. The marking of a Petri net is a multiset of tokens, i.e., $M \in \mathbb{B}(P)$. For some multiset $M \in \mathbb{B}(P)$, $M(p)$ denotes the number of times element p appears in M. The standard set operators can be extended to multisets, $M_1 \uplus M_2$ is the union of two multisets.

Firing a transition t in a marking M consumes one token from each of its input places and produces one token in each of its output places. Furthermore, transition t is enabled and may fire in M if there are enough tokens in its input places for the consumptions to be possible, i.e. iff for each input place s of t, $M(s) \geq 1$. Some of the transitions corresponds to piece of work in the process; each of those transitions are associated with a label that indicates the activity that it represents.

Definition 1 (Labeled Petri net). *A labeled Petri net $PN = (P, T, F, l)$ is a Petri net (P, T, F) with labeling function $l \in T \not\rightarrow \mathcal{U}_A$ where $\mathcal{U}_A$ is some universe of activity labels.*[4]

Transitions without a label are invisible transitions, also known as τ-transitions. They are introduced for routing purposes but they do not represent actual pieces of work. As such, their execution is not recorded in the event logs.

Definition 2 (System Net). *A system net $SN = (PN, M_{init}, M_{final})$ is a triplet where $PN = (P, T, F, l)$ is a labeled Petri net, $M_{init} \in \mathbb{B}(P)$ is the initial marking, and $M_{final} \in \mathbb{B}(P)$ is the final marking. $\mathcal{U}_{SN}$ is the* universe of system nets*.*

Definition 3 (System Net Notations). *Let $SN = (PN, M_{init}, M_{final}) \in \mathcal{U}_{SN}$ be a system net with $PN = (P, T, F, l)$.*

- $T_v(SN) = dom(l)$ *is the set of* visible transitions *in SN,*
- $A_v(SN) = rng(l)$ *is the set of corresponding* observable activities *in SN,*
- $T_v^u(SN) = \{t \in T_v(SN) \mid \forall_{t' \in T_v(SN)}\ l(t) = l(t') \Rightarrow t = t'\}$ *is the set of* unique *visible transitions in SN (i.e., there are no other transitions having the same visible label), and*
- $A_v^u(SN) = \{l(t) \mid t \in T_v^u(SN)\}$ *is the set of corresponding* unique *observable activities in SN.*

In the remainder, for the formal definitions and proved theorems in Section 3, we need to introduce the concept of union of two nets. For this, we need to merge labeling functions. For any two partial functions $f_1 \in X_1 \not\rightarrow Y_1$ and $f_2 \in X_2 \not\rightarrow Y_2$: $f_3 =$

[4] Symbol $\not\rightarrow$ is used to denote partial functions.

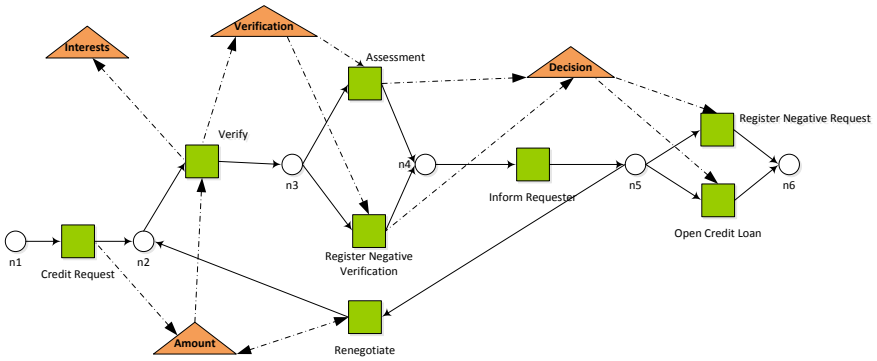

Fig. 3: Pictorial representation of a Petri net with Data that models the process earlier described in terms of BPMN diagram (cf. Figure 1). Places, transitions and variables are represented as circles, rectangles and triangles, respectively. The dotted arcs going from a transition to a variable denote the writing operations; the reverse arcs denote the read operations, i.e. the transition requires accessing the current variables' value.

$f_1 \oplus f_2$ is the union of the two functions. $f_3 \in (X_1 \cup X_2) \not\to (Y_1 \cup Y_2)$, $dom(f_3) = dom(f_1) \cup dom(f_2)$, $f_3(x) = f_2(x)$ if $x \in dom(f_2)$, and $f_3(x) = f_1(x)$ if $x \in dom(f_1) \setminus dom(f_2)$.

Definition 4 (Union of Nets). *Let $SN^1 = (N^1, M^1_{init}, M^1_{final}) \in \mathcal{U}_{SN}$ with $N^1 = (P^1, T^1, F^1, l^1)$ and $SN^2 = (N^2, M^2_{init}, M^2_{final}) \in \mathcal{U}_{SN}$ with $N^2 = (P^2, T^2, F^2, l^2)$ be two system nets.*

- *$l^3 = l^1 \oplus l^2$ is the union of l_1 and l_2,*
- *$N^1 \cup N^2 = (P^1 \cup P^2, T^1 \cup T^2, F^1 \cup F^2, l^3)$ is the union of N^1 and N^2, and*
- *$SN^1 \cup SN^2 = (N^1 \cup N^2, M^1_{init} \uplus M^2_{init}, M^1_{final} \uplus M^2_{final})$ is the union of system nets SN^1 and SN^2.*

2.2 Petri nets with Data

A Petri net with Data is a Petri net with any number of variables (see Definitions 5 and 6 below). Petri nets with data can be seen as an abstracted version of high-level/colored Petri nets [10]. Colored Petri nets are extremely rich in expressiveness; however, many aspects are unimportant in our setting. Petri nets with data provide precisely the information needed for conformance checking of data-aware models and logs.

Definition 5 (Variables and Values). *$\mathcal{U}_{VN}$ is the universe of variable names. $\mathcal{U}_{VV}$ is the universe of values. $\mathcal{U}_{VM} = \mathcal{U}_{VN} \not\to \mathcal{U}_{VV}$ is the universe of variable mappings.*

In this type of nets, transitions may read from and/or write to variables. Moreover, transitions are associated with guards over these variables, which define when these they can fire. A guard can be any formula over the process variables using relational operators ($<, >, =$) as well as logical operators such as conjunction ($\wedge$), disjunction ($\vee$),

and negation ($\neg$). A variable v appear as v_r or v_w, denoting the values read and written by the transition for v. We denote with $Formulas(V)$ the universe of such formulas defined over a set V of variables. In the remainder, given a set $V \subset \mathcal{U}_{VN}$ of variable names, we denote $V_R = \{v_r : v \in V\}$ and $V_W = \{v_w : v \in V\}$.

Formally, a *Petri net with Data* (DPN) is defined as follows:

Definition 6 (Petri net with Data). *A Petri net with Data* $DPN = (SN, V, val, init, read, write, guard)$ *consists of*

- *a system net* $SN = (PN, M_{init}, M_{final})$ *with* $PN = (P, T, F, l)$,
- *a set* $V \subseteq \mathcal{U}_{VN}$ *of data variables,*
- *a function* $val \in V \rightarrow \mathcal{P}(\mathcal{U}_{VV})$ *that defines the values admissible for each variable, i.e.,* $val(v)$ *is the set of values that variable* v *can have,*[5]
- *a function* $init \in V \rightarrow \mathcal{U}_{VV}$ *that defines the initial value for each variable* v *such that* $init(v) \in val(v)$ *(initial values are admissible),*
- *a read function* $read \in T \rightarrow \mathcal{P}(V)$ *that labels each transition with the set of variables that it reads,*
- *a write function* $write \in T \rightarrow \mathcal{P}(V)$ *that labels each transition with the set of variables that it writes,*
- *a guard function* $guard \in T \rightarrow Formulas(V_W \cup V_R)$ *that associates a guard with each transition such that, for any* $t \in T$ *and for any* $v \in V$*, if* v_r *appears in* $guard(t)$ *then* $v \in read(t)$ *and if* v_w *appears in* $guard(t)$ *then* $v \in write(t)$.

$\mathcal{U}_{DPN}$ *is the* universe of Petri nets with data.

The notion of bindings is essential for the remainder. A *binding* is a triplet (t, r, w) describing the execution of transition t while reading values r and writing values w. A binding is valid if:

1. $r \in read(t) \rightarrow \mathcal{U}_{VV}$ and $w \in write(t) \rightarrow \mathcal{U}_{VV}$
2. for any $v \in read(t)$: $r(v) \in val(v)$, i.e., all values read should be admissible,
3. for any $v \in write(t)$: $w(v) \in val(v)$, i.e., all values written should be admissible.
4. Guard $guard(t)$ evaluate true.

More specifically, let us introduce variable assignment $\chi_b : (V_R \cup V_W) \not\rightarrow \mathcal{U}_{VV})$ which is defined as follows: for any $v \in read(t)$, $\chi(v_r) = r(v)$ and, for any $v \in write(t)$, $\chi(v_w) = w(v)$. A binding (t, r, w) makes $guard(t)$ evaluate true if the evaluation of $guard(t)$ wrt. χ_b returns true.

A marking (M, s) of a Petri net with Data DPN has two components: $M \in \mathbb{B}(P)$ is the *control-flow marking* and $s \in \mathcal{U}_{VM}$ with $dom(s) = V$ and $s(v) \in val(v)$ for all $v \in V$ is the *data marking*. The initial marking of a Petri net with Data DPN is $(M_{init}, init)$. Recall that $init$ is a function that defines the initial value for each variable.

$(DPN, (M, s))[b\rangle$ denotes that a binding b is enabled in marking (M, s), which indicates that each of its input places $\bullet t$ contains at least one token (control-flow enabled), b is valid and and $s{\restriction_{read(t)}} = r$ (the actual values read match the binding).[6]

[5] $\mathcal{P}(X)$ is the powerset of X, i.e., $Y \in \mathcal{P}(X)$ is and only if $Y \subseteq X$.

[6] $f{\restriction_Q}$ is the function projected on Q: $dom(f{\restriction_Q}) = dom(f) \cap Q$ and $f{\restriction_Q}(x) = f(x)$ for $x \in dom(f{\restriction_Q})$. Projection can also be used for bags and sequences, e.g., $[x^3, y, z^2]{\restriction_{\{x,y\}}} = [x^3, y]$ and $\langle y, z, y\rangle{\restriction_{\{x,y\}}} = \langle y, y\rangle$.

Table 1: Definitions of the guards of the transitions in Fig. 3. Variables and transition names are abbreviated as described in Figure 1. Subscripts r and w refer to, respectively, the values read and written for that given variable.

Transition	Guard
Credit Request	true
Verify	$0.1 \cdot A_r < I_w < 0.2 \cdot A_r$
Assessment	$V_R = \mathsf{true}$
Register Negative Verification	$V_r = \mathsf{false} \wedge D_w = \mathsf{false}$
Inform Requester	true
Renegotiate Request	$V_r = \mathsf{false} \wedge A_w < A_r$
Register Negative Request	$D_r = \mathsf{false}$
Open Credit	$D_r = \mathsf{true}$

An enabled binding $b = (t, r, w)$ may *occur*, i.e., one token is removed from each of the input places $\bullet t$ and one token is produced for each of the output places $t\bullet$. Moreover, the variables are updated as specified by w. Formally: $M' = (M \setminus \bullet t) \uplus t\bullet$ is the control-flow marking resulting from firing enabled transition t in marking M (abstracting from data) and $s' = s \oplus w$ is the data marking where $s'(v) = w(v)$ for all $v \in write(t)$ and $s'(v) = s(v)$ for all $v \in V \setminus write(t)$. $(DPN, (M, s))[b\rangle(DPN, (M', s'))$ denotes that b is enabled in (M, s) and the occurrence of b results in marking (M', s').

Figure 3 shows a Petri net with Data DPN_{ex} that models the same process as represented in Figure 1 as BPMN diagram, and Table 1 illustrates the conditions of the guards of the transitions of DPN_{ex}. The labeling function l is such that the domain of l is the set of transitions of DPN_{ex} and, for each transition t of DPN_{ex}, $l(t) = t$. In other words, the set of activity labels coincides with the set of transitions.

Let $\sigma_b = \langle b_1, b_2, \ldots, b_n\rangle$ be a sequence of bindings. $(DPN, (M, s))[\sigma_b\rangle(DPN, (M', s'))$ denotes that there is a set of markings $(M_0, s_0), (M_1, s_1), \ldots, (M_n, s_n)$ such that $(M_0, s_0) = (M, s)$, $(M_n, s_n) = (M', s')$, and $(DPN, (M_i, s_i))[b_{i+1}\rangle(DPN, (M_{i+1}, s_{i+1}))$ for $0 \leq i < n$. A marking (M', s') is *reachable* from (M, s) if there exists a σ_b such that $(DPN, (M, s))[\sigma_b\rangle(DPN, (M', s'))$.

$\phi_f(DPN) = \{\sigma_b \mid \exists_s\ (DPN, (M_{init}, init))[\sigma_b\rangle(DPN, (M_{final}, s))\}$ is the *set of complete binding sequences*, thus describing the behavior of DPN.

Definition 7 (Union of Petri nets with Data). *Let* $DPN^1 = (SN^1, V^1, val^1, init^1, read^1, write^1, guard^1)$ *and* $DPN^2 = (SN^2, V^2, val^2, init^2, read^2, write^2, guard^2)$ *with* $V^1 \cap V^2 = \emptyset$. $DPN^1 \cup DPN^2 = (SN^1 \cup SN^2, V^1 \cup V^2, val^1 \oplus val^2, init^1 \oplus init^2, read^3, write^3, guard^3)$ *is the union such that*

- $read^3(t) = read^1(t)$, $write^3(t) = write^1(t)$, *and* $guard^3(t) = guard^1(t)$ *if* $t \in T^1 \setminus T^2$,
- $read^3(t) = read^2(t)$, $write^3(t) = write^2(t)$, *and* $guard^3(t) = guard^2(t)$ *if* $t \in T^2 \setminus T^1$, *and*
- $read^3(t) = read^1(t) \cup read^2(t)$, $write^3(t) = write^1(t) \cup write^2(t)$, *and* $guard^3(t) = guard^1(t) \cdot guard^2(t)$ *if* $t \in T^1 \cap T^2$.

2.3 Event Logs and Relating Models to Event Logs

Next we introduce *event logs* and relate them to the *observable* behavior of a DPN.

Definition 8 (Trace, Event Log with Data). *A trace $\sigma \in (\mathcal{U}_A \times \mathcal{U}_{VM} \times \mathcal{U}_{VM})^*$ is a sequence of activities with input and output data. $L \in \mathbb{B}((\mathcal{U}_A \times \mathcal{U}_{VM} \times \mathcal{U}_{VM})^*)$ is an event log with read and write information, i.e., a multiset of traces with data.*

Definition 9 (From Bindings to Traces). *Consider a Petri net with Data with transitions T and labeling function $l \in T \not\rightarrow \mathcal{U}_A$. A binding sequence $\sigma_b \in (T \times \mathcal{U}_{VM} \times \mathcal{U}_{VM})^*$ can be converted into a trace $\sigma_v \in (\mathcal{U}_A \times \mathcal{U}_{VM} \times \mathcal{U}_{VM})^*$ by removing the bindings that correspond to unlabeled transitions and by mapping the labeled transitions onto their corresponding label. $l(\sigma_b)$ denotes the corresponding trace σ_v.*

Note that we overload the labeling function to binding sequences, $\sigma_v = l(\sigma_b)$. This is used to define $\phi(DPN)$: the set of all visible traces.

Definition 10 (Observable Behavior of a Petri net with Data). *Let DPN be a Petri net with Data. $(DPN, (M, s))[\sigma_v \rhd (DPN, (M', s'))$ if and only if there is a sequence σ_b such that $(DPN, (M, s))[\sigma_b\rangle(DPN, (M', s'))$ and $\sigma_v = l(\sigma_b)$. $\phi(DPN) = \{l(\sigma_b) \mid \sigma_b \in \phi_f(DPN)\}$ is the set of* visible *traces starting in $(M_{init}, init)$ and ending in (M_{final}, s) for some data marking s.*

Definition 11 (Perfectly Fitting with Data). *A trace $\sigma \in (\mathcal{U}_A \times \mathcal{U}_{VM} \times \mathcal{U}_{VM})^*$ is perfectly fitting $DPN \in \mathcal{U}_{DPN}$ if $\sigma \in \phi(DPN)$. An event log $L \in \mathbb{B}((\mathcal{U}_A \times \mathcal{U}_{VM} \times \mathcal{U}_{VM})^*)$ is perfectly fitting DPN if all of its traces are perfectly fitting.*

Later, we will need to project binding sequences and traces onto subsets of transitions/activities and variables. Therefore, we introduce a generic projection operator $\Pi_{Y,V}(\sigma)$ that removes transitions/activities not in Y and variables not in V.

Definition 12 (Projection). *Let X be a set of transitions or activities (i.e., $X \subseteq T$ or $X \subseteq \mathcal{U}_A$). Let $Y \subseteq X$ be a subset and $V \subseteq \mathcal{U}_{VN}$ a subset of variable names. Let $\sigma \in (X \times \mathcal{U}_{VM} \times \mathcal{U}_{VM})^*$ be a binding sequence or a trace with data. $\Pi_{Y,V}(\sigma) \in (Y \times (V \not\rightarrow \mathcal{U}_{VV}) \times (V \not\rightarrow \mathcal{U}_{VV}))^*$ is the projection of σ onto transitions/activities Y and variables V. Bindings/events unrelated to transitions/activities in Y are removed completely. Moreover, for the remaining bindings/events all read and write variables not in V are removed. $\Pi_{Y,V}(L) = [\Pi_{Y,V}(\sigma) \mid \sigma \in L]$ lifts the projection operator to the level of logs.*

2.4 Alignments

Conformance checking requires an *alignment* of event log L and process model DPN, that is the alignment of each single trace $\sigma \in L$ and process model DPN.

The events in the event log need to be related to transitions in the model, and vice versa. Such an alignment shows how the event log can be replayed on the process model. Building this alignment is far from trivial, since the log may deviate from the model at an arbitrary number of places. We need to relate "moves" in the log to "moves" in the model in order to establish an alignment between a process model and an event log. It may be that some of the moves in the log cannot be mimicked by the model and vice versa. We denote such "no moves" by $\gg$. An alignment is a sequence of moves:

Table 2: Examples of complete alignments of $\sigma_{example}$ and N. For readability, the read operations are omitted. Of course, read operations for any variable must match the most recent value for that variable. Any move is highlighted with a gray color if it contains deviations, i.e. it is not a move in both without incorrect read/write operations.

(a)

Event-Log Trace	Process
(**a**, {(A,4000)})	(**a**, {(A,4000)})
(**b**, {(I,450),(V,false)})	(**b**, {(I,450),(V,true)})
(**c**, {(D,true)})	(**c**, {(D,true)})
(**e**, ∅)	(**e**, ∅)
(**f**, {(A,5000)})	(**f**, {(A,3000)})
(**b**, {(I,450),(V,false)})	(**b**, {(I,450),(V,false)})
(**d**, {(D,false)})	(**d**, {(D,false)})
(**e**, ∅)	(**e**, ∅)
(**h**, ∅)	≫
≫	(**g**, ∅)

(b)

Event-Log Trace	Process
(**a**, {(A,4000)})	(**a**, {(A,5100)})
(**b**, {(I,450),(V,false)})	(**b**, {(I,511),(V,true)})
(**c**, {(D,true)})	(**c**, {(D,true)})
(**e**, ∅)	(**e**, ∅)
(**f**, {(A,5000)})	(**f**, {(A,5000)})
(**b**, {(I,450),(V,false)})	(**b**, {(I,511),(V,false)})
(**d**, {(D,false)})	(**d**, {(D,false)})
(**e**, ∅)	(**e**, ∅)
(**h**, ∅)	≫
≫	(**g**, ∅)

Definition 13 (Legal alignment moves). *Let* $DPN = (SN, V, val, init, read, write, guard)$ *be a Petri net with Data, with* $SN = (PN, M_{init}, M_{final})$ *and* $PN = (P, T, F, l)$. *Let* $S_L = \mathcal{U}_A \times \mathcal{U}_{VM} \times \mathcal{U}_{VM}$ *be the universe of events. Let* $S_{DPN} = T \times \mathcal{U}_{VM} \times \mathcal{U}_{VM}$ *be the universe of bindings of* DPN. *Let be* $S^{\gg}_{DPN} = S_{DPN} \cup \{\gg\}$ *and* $S^{\gg}_{L} = S_L \cup \{\gg\}$.

A legal move in an alignment *is represented by a pair* $(s_L, s_M) \in (S^{\gg}_{L} \times S^{\gg}_{DPN}) \setminus \{(\gg, \gg)\}$ *such that*

- (s_L, s_M) *is a* move in log *if* $s_L \in S_L$ *and* $s_M = \gg$,
- (s_L, s_M) *is a* move in model *if* $s_L = \gg$ *and* $s_M \in S_{DPN}$,
- (s_L, s_M) *is a* move in both without incorrect read/write operations *if* $s_M = (t, r, w) \in S_{DPN}$ *and* $s_L = (l(t), r, w) \in S_L$,
- (s_L, s_M) *is a* move in both with incorrect read/write operations *if* $s_M = (t, r, w) \in S_{DPN}$ *and* $s_L = (l(t), r', w') \in S_L$, *and* $r \neq r'$ *or* $w \neq w'$.

All other moves are considered as illegal.

Definition 14 (Alignments). *Let* $DPN = (SN, V, val, init, read, write, guard)$ *be a Petri net with Data and* $\sigma \in (S_L)^*$ *be an event-log trace. Let* $\mathcal{A}_{DPN}$ *be the set of legal moves for* DPN. *A* complete alignment *of* σ_L *and* DPN *is a sequence* $\gamma \in {\mathcal{A}_{DPN}}^*$ *such that, ignoring all occurrences of* $\gg$, *the projection on the first element yields* σ_L *and the projection on the second yields a* $\sigma_P \in \phi_f(DPN)$.

Table 2 shows two complete alignments of the process model in Figure 3 and the log trace σ_{ex} from Section 1.

In order to define the severity of a deviation, we introduce a cost function on legal moves: $\kappa \in \mathcal{A}_{DPN} \rightarrow \mathbb{R}^+_0$. This cost function can be used to favor one type of explanation for deviations over others. The cost of each legal move depends on the specific model and process domain and, hence, the cost function κ needs to be defined specifically for each setting. The cost of an alignment γ is the sum of the cost of all individual moves composing it: $\mathcal{K}(\gamma) = \sum_{(s_L, s_M) \in \gamma} \kappa(s_L, s_M)$.

However, we do not aim to find just any complete alignment. Our goal is to find a complete alignment of σ_L and DPN which minimizes the cost: an optimal alignment.

Let $\Gamma_{\sigma_L,N}$ be the (infinite)set of all complete alignments of σ_L and DPN. The alignment $\gamma \in \Gamma_{\sigma_L,DPN}$ is an *optimal alignment* if, for all $\gamma' \in \Gamma_{\sigma_L,N}$, $\mathcal{K}(\gamma) \leq \mathcal{K}(\gamma')$. Note that an optimal alignment does not need to be unique, i.e. multiple complete alignments with the same minimal cost may exist.

Let us consider again our example introduced above. Let us assume to have a cost function κ^s such that $\kappa^s(s_L, s_M) = 1$ if (s_L, s_M) is a visible move in process or a move in log (i.e. $s_L = \gg$ and s_M corresponds to a labeled transition or, conversely, $s_M = \gg$, respectively) or a move in both with incorrect read/write operations and $\kappa^s(s_L, s_M) = 0$ in case of move in both without incorrect read/write operations or a move in model corresponding to an unlabeled transition. The alignment in Table 2a has a cost of 6 whereas the alignment in Table 2b has a cost 8.[7] It follows that the former is a better alignment. As a matter of fact, it is also an optimal alignment, although it is not the only one. For instance, any variation of such an alignment where the move for f is of the form (now including read operations) $((\mathbf{f}, \{(A, 4000)\}, \{(A, 5000)\})$ $(\mathbf{f}, \{(A, 4000)\}, \{(A, x)\})\})$ with $2250 < x < 4000$ corresponds to an optimal alignment, as well.

In Section 1, we have mentioned that the data-aware conformance checking is undecidable in the general case. This is caused by the fact that Petri nets with Data are Turing-complete. Therefore, it is not decidable to verify whether a sequence of valid bindings exists that takes from the initial marking to any final marking (M_{final}, s). As a consequence, for instance, it is not possible to find an alignment of a Petri net with Data and the empty log trace. As mentioned in Section 1, the problem becomes decidable (with an exponential complexity) if guards are restricted to linear (in)equalities.

3 Valid Decomposition of Data-aware Models

In [6] the author defines *valid decomposition* in terms of Petri nets: the overall system net SN is decomposed into a collection of subnets $\{SN^1, SN^2, \ldots, SN^n\}$ such that the union of these subnets yields the original system net. A decomposition is *valid* if the subnets "agree" on the original labeling function (i.e., the same transition always has the same label), each place resides in just one subnet, and also each invisible transition resides in just one subnet. Moreover, if there are multiple transitions with the same label, they should reside in the same subnet. Only unique visible transitions can be shared among different subnets.

Definition 15 (Valid Decomposition for Petri nets [6]). *Let $SN \in \mathcal{U}_{SN}$ be a system net with labeling function l. $D = \{SN^1, SN^2, \ldots, SN^n\} \subseteq \mathcal{U}_{SN}$ is a valid decomposition if and only if:*

- *$SN^i = (N^i, M^i_{init}, M^i_{final})$ is a system net with $N^i = (P^i, T^i, F^i, l^i)$ for all $1 \leq i \leq n$,*
- *$l^i = l|_{T^i}$ for all $1 \leq i \leq n$,*
- *$P^i \cap P^j = \emptyset$ for $1 \leq i < j \leq n$,*
- *$T^i \cap T^j \subseteq T^u_v(SN)$ for $1 \leq i < j \leq n$,*
- *$rng(l^i) \cap rng(l^j) \subseteq T^u_v(SN)$ for $1 \leq i < j \leq n$, and*

[7] They also include a cost of two that is accounted for incorrect read operations, not shown in the alignments, which are caused by incorrect write operations.

– $SN = \bigcup_{1 \leq i \leq n} SN^i$.

$\mathcal{D}(SN)$ is the set of all valid decompositions of SN.

From the definition the following properties follow:

1. each place appears in precisely one of the subnets, i.e., for any $p \in P$: $|\{1 \leq i \leq n \mid p \in P^i\}| = 1$,
2. each invisible transition appears in precisely one of the subnets, i.e., for any $t \in T \setminus T_v(SN)$: $|\{1 \leq i \leq n \mid t \in T^i\}| = 1$,
3. all visible transitions with the same label (i.e. the label is not unique) appear in the same subnet, i.e., for any $a \in A_v(SN) \setminus A_v^u(SN)$: $|\{1 \leq i \leq n \mid \exists t \in T_v(SN) \cap T^i,\ l(t)\}| = 1$,
4. visible transitions having a unique label may appear in multiple subnets, i.e., for any $t \in T_v^u(SN)$: $|\{1 \leq i \leq n \mid t \in T^i\}| \geq 1$, and
5. each edge appears in precisely one of the subnets, i.e., for any $(x, y) \in F$: $|\{1 \leq i \leq n \mid (x, y) \in F^i\}| = 1$.

As shown in [6], these observations imply that conformance checking can be decomposed. Any trace that fits the overall process model can be decomposed into smaller traces that fit the individual model fragments. Moreover, if the smaller traces fit the individual fragments, then they can be composed into a trace that fits into the overall process model. This result is the basis for decomposing process mining problems.

Theorem 1 (Conformance Checking Can be Decomposed [6]). *Let $L \in \mathbb{B}(A^*)$ be an event log with $A \subseteq \mathcal{U}_A$ and let $SN \in \mathcal{U}_{SN}$ be a system net. For any valid decomposition $D = \{SN^1, SN^2, \ldots, SN^n\} \in \mathcal{D}(SN)$: L is perfectly fitting system net SN if and only if for all $1 \leq i \leq n$: the projection of L onto $A_v(SN^i)$ is perfectly fitting SN^i.*

In this paper, the definition of valid decomposition is extended to cover Petri nets with data.

Definition 16 (Valid Decomposition for Petri nets with Data). *Let $DPN \in \mathcal{U}_{DPN}$ be a Petri net with Data. $D = \{DPN^1, DPN^2, \ldots, DPN^n\} \subseteq \mathcal{U}_{DPN}$ is a valid decomposition if and only if:*

– *for all $1 \leq i \leq n$: $DPN^i = (SN^i, V^i, val^i, init^i, read^i, write^i, guard^i)$ is a Petri net with Data, $SN^i = (PN^i, M^i_{init}, M^i_{final}) \in \mathcal{U}_{SN}$ is a system net, and $PN^i = (P^i, T^i, F^i, l^i)$ is a labeled Petri net,*
– *$D' = \{SN^1, SN^2, \ldots, SN^n\} \subseteq \mathcal{U}_{SN}$ is a valid decomposition of $\bigcup_{1 \leq i \leq n} SN^i$,*
– *$V^i \cap V^j = \emptyset$ for $1 \leq i < j \leq n$,*
– *$DPN = \bigcup_{1 \leq i \leq n} DPN^i$.*

$\mathcal{D}(DPN)$ is the set of all valid decompositions of DPN.

Each variable appears in precisely one of the subnets. Therefore, there cannot be two fragments that read and or write the same data variables: $\bigcup_{t \in T^i} read^i(t) \cup write^i(t) \bigcap \bigcup_{t \in T^j} read^j(t) \cup write^j(t) = \emptyset$ for $1 \leq i < j \leq n$. Moreover, two guards in different fragments cannot refer to the same variable. If a transition t appears in multiple fragments, then it needs to have a visible unique label as shown in [6]. Such a uniquely labeled transition t shared among fragments, may use, read, or write different variables in different fragments. Since $DPN = \bigcup_{1 \leq i \leq n} DPN^i$, we know that, for all t in DPN, $guard(t)$ is the product of all $guard^i(t)$ such that $t \in T^i$. Without loss of generality

we can assume that the first k fragments share t. Hence, $guard(t) = guard^1(t) \cdot \ldots \cdot guard^k(t)$. Hence, in a valid decomposition, the guard of a shared transition can only be split if the different parts do not depend on one another. Notice that, the splitting of the data variables is limited by how the variables are used throughout the process, existing a worst-case where all the data variables are used in all the steps of the process.

Based on these observations, we prove that we can decompose conformance checking also for Petri nets with data.

Theorem 2 (Conformance Checking With Data Can be Decomposed). *Let $L \in \mathcal{B}((\mathcal{U}_A \times \mathcal{U}_{VM} \times \mathcal{U}_{VM})^*)$ be an event log with information about reads and writes and let $DPN \in \mathcal{U}_{DPN}$ be a Petri net with Data. For any valid decomposition $D = \{DPN^1, DPN^2, \ldots, DPN^n\} \subseteq \mathcal{U}_{DPN}$: L is perfectly fitting Petri net with Data DPN if and only if for all $1 \leq i \leq n$: $\Pi_{A_v(SN^i),V^i}(L)$ is perfectly fitting DPN^i.*

Proof. Let $DPN = (SN, V, val, init, read, write, guard)$ be a Petri net with Data with $SN = (PN, M_{init}, M_{final})$ and $PN = (P, T, F, l)$. Let $D = \{DPN^1, DPN^2, \ldots DPN^n\}$ be a valid decomposition of DPN with $DPN^i = (SN^i, V^i, val^i, init^i, read^i, write^i, guard^i)$, $SN^i = (PN^i, M^i_{init}, M^i_{final}) \in \mathcal{U}_{SN}$, and $PN^i = (P^i, T^i, F^i, l^i)$.
($\Rightarrow$) Let $\sigma_v \in L$ be such that there exists a data marking s such that $(DPN, (M_{init}, init))[\sigma_v \triangleright (DPN, (M_{final}, s))$. This implies that there exists a corresponding σ_b with $(DPN, (M_{init}, init))[\sigma_b\rangle(DPN, (M_{final}, s))$ and $l(\sigma_b) = \sigma_v$. For all $1 \leq i \leq n$, we need to prove that there is a σ^i_b with $(DPN^i, (M^i_{init}, init^i))[\sigma^i_b\rangle(DPN^i, (M^i_{final}, s^i))$ for some s^i. This follows trivially because DPN^i can mimic any move of DPN with respect to transitions T^i: just take $\sigma^i_b = \Pi_{T^i,V^i}(\sigma_b)$. Note that guards can only become weaker by projection.
($\Leftarrow$) Let $\sigma_v \in L$. For all $1 \leq i \leq n$, let σ^i_b be such that $(DPN^i, (M^i_{init}, init^i))[\sigma^i_b\rangle (DPN^i, (M^i_{final}, s^i))$ and $l^i(\sigma^i_b) = \Pi_{A_v(SN^i),V^i}(\sigma_v)$. The different σ^i_b sequences can be stitched together into an overall σ_b s.t. $(DPN, (M_{init}, init))[\sigma_b\rangle(DPN, (M_{final}, s))$ with $s = s^1 \oplus s^2 \oplus \ldots \oplus s^n$. This is possible because transitions in one subnet can only influence other subnets through unique visible transitions and these can only move synchronously as defined by σ_v. Moreover, guards can only be split in independent parts (see Definition 16). Suppose that t appears in T_i and T_j, then $guard(t) = guard^i(t) \cdot guard^j(t)$. Hence, a read/write in subnet i cannot limit a read/write in subnet j. Therefore, we can construct σ_b and $l(\sigma_b) = \sigma_v$. □

4 SESE-based Strategy for Realizing a Valid Decomposition

In this section we present a concrete strategy to instantiate the valid decomposition definition over a Petri net with data presented in the previous section (cf. Def.16). The proposed strategy decomposes the Petri net with data in a number of Single-Entry Single-Exit (SESE) components, which have recently been shown to create meaningful fragments of a process model [11,8]. SESE decomposition is indicated for well-structured models, whereas for unstructured models some automatic transformation techniques can be considered as a pre-processing step [12].

We will now informally describe the necessary notions for understanding the proposed data-oriented SESE-based valid decomposition strategies described below. For

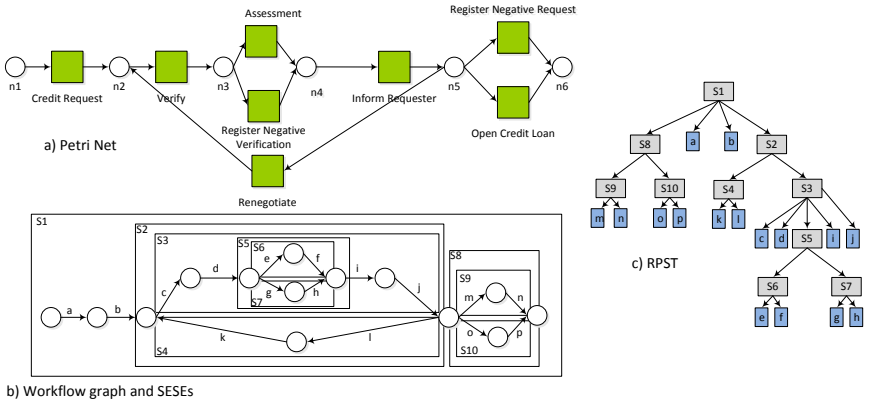

Fig. 4: A Petri net modeling the control-flow of the running example, its workflow graph and the RPST and SESE decomposition.

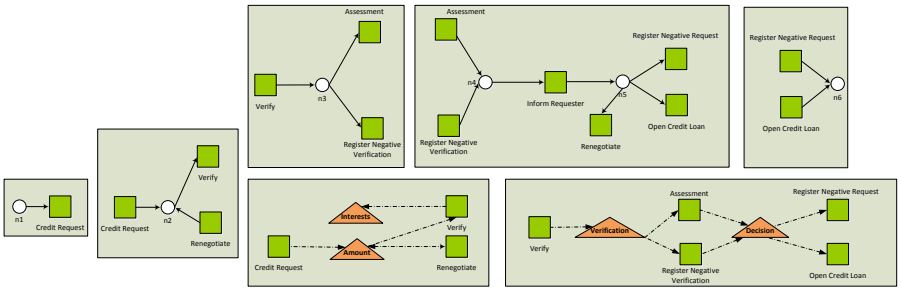

Fig. 5: SESE-based decomposition for the running example, with 2-decomposition.

the sake of clarity, we will focus on the control flow to illustrate the concepts, although the definitions will be extended at the end to also consider data.

Given Petri net $PN = (P, T, F, l)$, its *workflow graph* is the structural graph $WG = (S, E)$ with no distinctions between places and transitions, i.e., $S = P \cup T$ and $E = F$. For instance, Fig. 4(b) shows the workflow graph of the Petri net of Fig. 4(a) (corresponding with the control-flow part of the running example). Given a subset of edges $E' \subseteq E$ of WG, the nodes $S\restriction_{E'} = \{s \in S : \exists s' \in S.\ (s, s') \in E' \vee (s', s) \in E'\}$ can be partitioned into interior and boundary. Interior nodes have no connection with nodes outside $S\restriction_{E'}$, while boundary nodes do. Furthermore, boundary nodes can be partitioned into entry (no incoming edge belongs to E), or exit (no outgoing edge belongs to E'). $E' \subseteq E$ is a *SESE* of WG iff the subnet derived from E has exactly two boundary nodes: one entry and one exit. Fig. 4(b) shows all non-trivial SESEs[8] of the Petri net of Fig. 4(a). For a formal definition we refer to [11].

The decomposition based on SESEs is a well studied problem in the literature, and can be computed in linear time. In [13,14], efficient algorithms for constructing the

[8] Note that by definition, a single edge is a SESE.

Algorithm 1 SESE-based Decomposition

1: Build data workflow graph DWG from F, R, W
2: Compute $RPST$ from DWG
3: Compute SESE decomposition D from the $RPST$
4: Compute and merge subnets if necessary to preserve valid decomposition.
5: **return** valid decomposition where perspectives are decomposed altogether

Refined Process Structure Tree (RPST), i.e., a hierarchical structure containing all the *canonical* SESEs of a model, were presented. Informally, an RPST is a tree where the nodes are canonical SESEs, such that the parent of a SESE S is the smallest SESE that contains S. Fig. 4(c) shows the RPST of the workflow graph depicted in Fig. 4(b). By selecting a particular set of SESEs in the RPST (e.g., k-decomposition [8]), it is possible to obtain a partitioning of the arcs. We refer the reader to the aforementioned work for a formal description of the SESE-based decomposition.

To extend the previous definitions to also account for data, one simply has to incorporate in the workflow graph the variables and read/write arcs, i.e., the *data workflow graph* of a Petri net with Data $(((P, T, F, l), M_{init}, M_{final}), V, val, init, read, write, guard)$ with data arcs $R = \{(v, t)|v \in read(t)\}$ and $W = \{(t, v)|v \in write(t)\}$ is $DWG = (S, E)$ with $S = P \cup T \cup V$ and $E = F \cup R \cup W$. The subsequent definitions after this extension (SESE, RPST) are analogous.

Similar to [8], we propose a SESE decomposition to analyze the conformance of Petri nets with data, but considering data workflow graph instead. Algorithm 1 describes the steps necessary to construct a SESE decomposition. The arcs are partitioned in SESEs by means of creating the RPST from the data workflow graph, and selecting a particular set of SESES over it. Once the partitioning is done, a subnet is created for each part. Subnets contradicting some of the requirements of Def. 16 (e.g. sharing places, invisible or duplicate transitions, variables, or transitions with non-splitting guards) are merged to preserve the valid decomposition definition.

Figure 5 shows the decomposition for the example of Fig.3, where the RPST is partitioned using the 2-decomposition algorithm [8], i.e., SESEs of at most 2 arcs[9]. To ensure a valid decomposition is obtained, step 4 of Algorithm 1 combines multiple SESE fragments into larger fragments, which are not necessarily SESEs anymore.

5 Implementation and Experimental Results

The approach discussed in this paper has been implemented as a plug-in for the open-source *ProM* framework for process mining.[10] Our plug-in requires a Petri Net with Data and an event log as input and returns as many bags of alignments as the number of fragments in which the Petri Net with Data has been decomposed. Each bag refers to a different fragment and shows the alignments of each log trace and that fragment. A second type of output is also produced in which the alignments' information is projected onto the Petri net with Data. Transitions are colored according to the number of

[9] Although the SESEs have at most two arcs, this is not guaranteed for the final subnets, i.e., some subnets are merged to preserve the valid decomposition definition.

[10] `http://www.promtools.org`

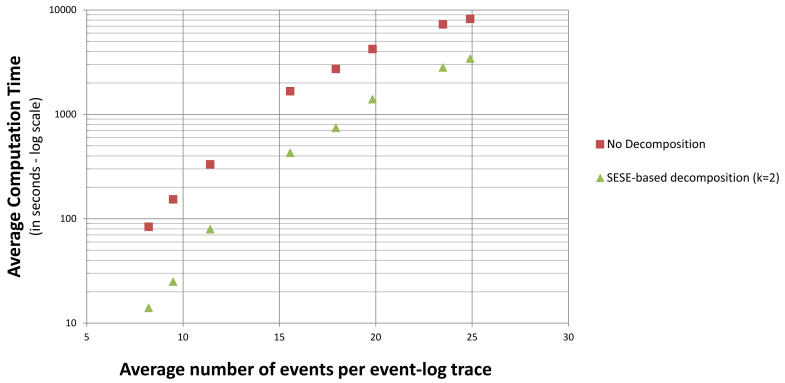

Fig. 6: Computation time for checking the conformance of the Petri net with Data in Figure 3 and event logs of different size. The Y axis is on a logarithmic scale.

deviations: if no deviation occurs for a given transition, the respective box in the model is white-colored. The filling color of a box shades towards red as a larger fraction of deviations occur for the corresponding transition. Something similar is also done for variables: the more incorrect read/write operations occur for a variable, the more the variable is shown with a color close to red. This output is extremely interesting from an end-user viewpoint as it allows for gaining a helicopter view on the main causes of deviations. Space limitations prevent us from giving more details and showing screenshots of the output. Interested readers can refer to [15] for further information.

As previously mentioned, the plug-in has been evaluated using a number of synthetic event logs and also a real-life process. The plug-in has been evaluated using the model in Figure 3 and with a number of event logs that were artificially generated. In particular, we have generated different event logs with the same number of traces, 5000, but increasing number of events, meaning that, on average, traces were of different length. To simulate that, for each simulated process execution, an increasing number of renegotiations was enforced to happen. Traces were also generated so as to contain a number of deviations: the event logs were generated in a way that 25% of transitions fired violating the guards.

Figure 6 shows the results of checking for conformance of the different event logs and the process model, comparing the SESE-based decomposition with $k = 2$ with the case in which no decomposition is made. To check the conformance of each fragment, we used the technique reported in [4]. Each dot in the chart indicates a different event log with traces of different size. The computation time refers to the conformance checking of the whole event logs (i.e., 5000 traces). The decomposed net is the same as in Figure 5. Regarding the cost function, we assign cost 1 to any deviation; however, this could be customized based on domain knowledge. The results show that, for every combination of event log and process model, the decomposition significantly reduces the computation time and the improvement is exponential in the size of the event log.

To assess the practical relevant of the approach, we also performed an evaluation with a Dutch financial institute. The process model was provided by a process analyst of the institute and consists of 21 transitions: 13 transitions with unique labels, 3 activities labels shared between 2 transitions (i.e. 6 transitions in total), plus 3 invisible

transitions. The model contains twelve process variables, which are read and written by the activities when being executed. The process model is omitted for space reasons and shown in [15]. We were also provided with an event log that recorded the execution of 111 real instances of such a process; overall, the 111 log traces contained 3285 events, which means roughly 29.6 events per trace. We checked the conformance of this process model and this event log, comparing the results when the model has or has not been decomposed in small fragments. For conformance checking, here we used the technique reported in [5] since the provided process model breaks the soundness assumptions required by [4]. For this experiment round, the additional optimizations proposed in [5] were deactivated to allow for a fair comparison.

The application of the decomposition approach to this real-life case study has shown tremendous results: the conformance checking has required 52.94 *seconds when the process model was decomposed using the SESE-based technique presented in Section 4; conversely, it required* 52891 *seconds when the model was not decomposed. This indicates that decomposing the process model allowed us to save* 99.999% *of the computation time.* As a matter of fact, we tried for different values of SESE parameter k but we obtained similar results: the computation time did not move away for more than 1 second. The reason of this is related to the fact that every decomposition for any value of k always contained a certain fragment, along with others. Indeed, that fragment could not be decomposed any further than a given extent. Since the computation time was mostly due to constructing alignments with that fragment, no significant difference in computation time could be observed when varying k.

6 Conclusions and Future Work

Conformance checking is becoming more important for two reasons: (1) the volume of event data available for checking normative models is rapidly growing (the topic of "Big Data" is on the radar of all larger organizations) and (2) because of a variety of regulations there is a need to check compliance. Moreover, conformance checking is also used for the evaluation of process discovery algorithms. Also genetic process mining algorithms heavily rely on the efficiency of conformance checking techniques.

Thus far, lion's share of conformance checking techniques has focused on control-flow and relatively small event logs. As shown in this paper, abstracting from other perspectives may lead to misleading conformance results that are too optimistic. Moreover, as process models and event logs grow in size, divide-and-conquer approaches are needed to still be able to check conformance and diagnose problems. Perspectives such as work distribution, resource allocation, quality of service, temporal constraints, etc. can all be encoded as data constraints. Hence, there is an urgent need to support data-aware conformance checking in-the-large.

This paper demonstrates that data-aware decompositions can be used to speed up conformance checking significantly. The evaluation with a real-life case study has shown that real data-aware process models can indeed be decomposed, thus obtaining even tremendous saving of computation time. As future work, we would like to extend our experimental evaluation with real-life process models of larger sizes. Moreover, we would like to explore alternative decomposition strategies using properties of the underlying data, and to analyze the impact of different component sizes. This paper only

focuses on fitness aspect of conformance, namely whether a trace can be replayed on a process model. However, recently, research has also been carried on as regards to different conformance dimensions [16,17], such as whether the model is precise enough to not allow for too much behavior compared with what observed in reality in the event log. We plan to use data-aware decomposition approaches to speed up the assessment of the quality of process models with respect to these other conformance dimensions, as well.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Information System **33**(1) (2008) 64–95
3. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance checking using cost-based fitness analysis. In: Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference, (EDOC 2011), IEEE Computer Society (2011) 55–64
4. de Leoni, M., van der Aalst, W.M.P.: Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In: Proceedings of the 11th International Conference on Business Process Management, (BPM 2013). Volume 8094 of LNCS., Springer (2013) 113–129
5. Mannhardt, F., de Leoni, M., Reijers, H.A. van der Aalst, W.M.P.: Balanced Multi-Perspective Checking of Process Conformance (2014) BPM Center Report BPM-14-07.
6. van der Aalst, W.M.P.: Decomposing Petri nets for process mining: A generic approach. Distributed and Parallel Databases **31**(4) (2013) 471–507
7. van der Aalst, W.M.P.: Decomposing process mining problems using passages. In: Proceedings of the 33rd International Conference on Application and Theory of Petri (PETRI NETS 2012). Volume 7347 of LNCS., Springer (2012) 72–91
8. Munoz-Gama, J., Carmona, J., van der Aalst, W.M.P.: Single-entry single-exit decomposed conformance checking. Information Systems **46** (2014) 102–122
9. Montali, M., Chesani, F., Mello, P., Maggi, F.M.: Towards data-aware constraints in declare. In Shin, S.Y., Maldonado, J.C., eds.: SAC, ACM (2013) 1391–1396
10. Jensen, K., Kristensen, L.: Coloured Petri Nets. Springer Verlag (2009)
11. Polyvyanyy, A.: Structuring process models. PhD thesis, University of Potsdam (2012)
12. Dumas, M., García-Bañuelos, L., Polyvyanyy, A.: Unraveling unstructured process models. In Mendling, J., Weidlich, M., Weske, M., eds.: BPMN. Volume 67 of LNBIP., Springer (2010) 1–7
13. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data Knowl. Eng. **68**(9) (2009) 793–818
14. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. In: 7th International Workshop on Web Services and Formal Methods. Revised Selected Papers. Volume 6551 of LNCS., Springer (2011) 25–41
15. de Leoni, M., Munoz-Gama, J., Carmona, J., van der Aalst, W.M.P.: Decomposing Conformance Checking on Petri Nets with Data. (2014) BPM Center Report BPM-14-06.
16. Munoz-Gama, J., Carmona, J.: A General Framework for Precision Checking. International Journal of Innovative Computing, Information and Control (IJICIC) **8**(7B) (July 2012) 5317–5339
17. De Weerdt, J., De Backer, M., Vanthienen, J., Baesens, B.: A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. Information System **37**(7) (2012) 654–676