Hierarchical Conformance Checking of Process Models Based on Event Logs

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

[1] Universitat Politecnica de Catalunya, Barcelona (Spain)
[2] Eindhoven University of Technology, Eindhoven (The Netherlands)
jmunoz@lsi.upc.edu, jcarmona@lsi.upc.edu , w.m.p.v.d.aalst@tue.nl

Abstract. Process mining techniques aim to extract knowledge from event logs. *Conformance checking* is one of the hard problems in process mining: it aims to diagnose and quantify the mismatch between observed and modeled behavior. Precise conformance checking implies solving complex optimization problems and is therefore computationally challenging for real-life event logs. In this paper a technique to apply hierarchical conformance checking is presented, based on a state-of-the-art algorithm for deriving the subprocesses structure underlying a process model. Hierarchical conformance checking allows us to decompose problems that would otherwise be intractable. Moreover, users can navigate through conformance results and zoom into parts of the model that have a poor conformance. The technique has been implemented as a ProM plugin and an experimental evaluation showing the significance of the approach is provided.

Keywords: Process Mining, Conformance Checking, Process Diagnosis

1 Introduction

Process mining emerged as a crucial discipline for addressing challenges related to Business Process Management (BPM) and "Big Data"[1]. Information systems record an overwhelming amount of data representing the footprints left by process executions. For example, Boeing jet engines can produce 10 terabytes of operational information every 30 minutes, and Walmart logs may store one million customer transactions per hour [2].

Process mining tackles three challenges relating event data (i.e., log files) and process models: *discovery* of a process model from an event log, *conformance checking* given a process model and a log, and *enhancement* of a process model with the information obtained from a log. Process mining research has produced a multitude of algorithms that demonstrated to be of great value for undertaking small or medium-sized problem instances. However, it is well-accepted that most of the existing algorithms are unable to handle problems of industrial size. In our opinion, as has happened in other areas (VLSI, manufacturing, among others) where the size of the input prevents from solving the problem straight away, the next generation of algorithms for process mining must incorporate high-level

techniques that enable the problem decomposition in a divide-and-conquer style. This paper proposes decomposition techniques for conformance checking.

In spite of its significance, few conformance checking algorithms exist in the literature. The seminal work by Rozinat *et al.* [3] was the first in formalizing the problem and enumerating the four dimensions to consider for determining the adequacy of a model in describing a log: *fitness*, *precision*, *generalization* and *simplicity*. In this paper we focus on the first two: fitness, that quantifies the capability of the model in reproducing the traces of the log, and precision, that quantifies how precise is the model in representing the log. In this paper, we will decompose state-of-the-art approaches for conformance checking. On the one hand, we will use *alignment techniques*. These provide a crucial step into relating model and log traces for unfitting models or models with duplicate/invisible transitions [4,5]. On the other hand, a novel precision technique based on superposing log and model behavior and detecting accordingly model's escaping points will allow us to quantify precision [6]. Both approaches have been combined in order to derive a robust estimation of precision [5]. The high-level technique presented in this paper is grounded on the aforementioned work.

To decompose process models, we use a technique to create the so-called *Refined Process Structure Tree* (RPST) originally proposed in [7]. This decomposition computes fragments of the net that have a single-entry and a single-exit node, thus intuitively resembling isolated subprocesses within the general model. Remarkably, it is a structural decomposition that can be computed in linear time on the underlying graph structure. Additionally, a tree-like structure may be derived representing the hierarchy between the computed components. This tree will be used in the strategy presented in this paper for hierarchical conformance checking: each component in the tree is processed in order to satisfy the requirements for conformance checking of [5] (e.g., find an initial and final marking, determine whereas it belongs to a cyclic part of the net), and finally a conformance checking problem instance is solved on the component and the log projected into the participating activities. Unlike the decomposition approaches proposed in [8,9], our approach uses a hierarchy of semantically meaningful process fragments. The RPST structure recursively splits the process into smaller fragments that are understandable for the analyst. Moreover, the hierarchy can be used to navigate to problematic parts of the process.

The approach has been implemented as a ProM plugin, and experiments have been performed on different dimensions, ranging from a comparison between the manual decomposition made by a human and the one presented in this paper, to the application of the technique to a set of benchmarks. Also, experiments illustrating the differences with the passage-based approach [8] are reported.

The paper is organized as follows: Sect. 2 presents the theoretical background of the paper. Sect. 3 presents the main stages of the methodology for hierarchical conformance checking, while Sect. 4 describes how to apply the approach to sound and safe workflow nets (a important class of Petri nets tailored towards business processes). Sect. 5 describes related work. Sect. 6 shows several experi-

ments illustrating the usefulness of the suggested approach. Finally, conclusions and future work are reported in Sect. 7.

2 Preliminaries

The starting point for conformance checking are an *event log* and a *model*. An event log records the execution of all cases (i.e. process instances). Each case is described by a trace, i.e., an activity sequence. Different cases may have exactly the same trace, i.e., an event log is a multiset of traces. A model represents the possible flows of the process. In this approach we use Petri net as a formal model of processes. A Petri Net PN is a tuple (P, T, A, m_i, m_o) where P and T represent finite sets of places and transitions, respectively, with $P \cap T = \emptyset$. Function $A : (P \times T) \cup (T \times P) \to \mathbb{N}$ represents the weighted flow relation. The markings over a Petri net are defined as *multisets* and we use multiset notation accordingly, e.g., $m = [p^2, q]$ or $m = [p, p, q]$ for a marking m with $m(p) = 2$, $m(q) = 1$, and $m(x) = 0$ for all $x \notin \{p, q\}$. The initial and final markings of PN are denoted as m_i and m_o respectively.[3] Given a marking m and a set n, $m_{\downarrow n}$ denotes the projection of m on n, e.g., $[a, a, b, c]_{\downarrow [a,c]} = [a, a, c]$.

3 Methodology for Hierarchical Conformance Checking

In this section we introduce the approach for decomposing a model in a hierarchical manner in order to perform conformance checking. Algorithm 1 presents the approach, which has three stages: the *Decomposition Stage* (Sec. 3.1) where the model is decomposed into components, the *Post-processing Stage* (Sec. 3.2) where the components are processed to enable the analysis, and finally the *Conformance Stage* (Sec. 3.3) where conformance is analyzed for every component.

3.1 Decomposition Stage

The first stage corresponds with the *decomposition* phase: the model is decomposed into hierarchical components, i.e., the decomposition is not a partitioning (which is the case in [8]) but instead for each component different of the net itself there is always another component containing it. This enables the analysis at different degrees of granularity and independence, similar to zooming in and out using online maps, to get a better understanding of the underlying process. In order to be able to navigate thought different layers of the model properly, three ingredients are needed: 1) a subprocesses decomposition, 2) a hierarchy relating these subprocesses, and 3) a mechanism to enhance and propagate additional information about the components within the hierarchy.

[3] In some approaches such as [3,6] only a initial marking is considered. However, in [5] (cornerstone of the conformance checking of this paper) the authors introduce analogously the need of the final marking.

Algorithm 1 Decomposed Conformance algorithm

procedure DECONF(net, log, m_i, m_o)
  $\{c_1 \dots c_n\} \leftarrow$ DECOMPOSE(net) ▷ Decomposition Stage
  $h \leftarrow$ BUILD_HIERARCHY($\{c_1 \dots c_n\}$)
  $h^* \leftarrow$ ENRICH_HIERARCHY(h, net)

  for all subprocess $c \in h^*$ **do**
    $pn_c \leftarrow$ COMPUTE_PETRI_NET(c, net) ▷ Post-processing Stage
    $m_{ic} \leftarrow$ COMPUTE_INITIAL_MARKING(c, net, m_i)
    $m_{oc} \leftarrow$ COMPUTE_FINAL_MARKING(c, net, m_o)
    $log_c \leftarrow$ COMPUTE_LOG(log, pn_c)

    COMPUTE_CONFORMANCE($log_c, pn_c, m_{ic}, m_{oc}$) ▷ Conformance Stage
  end for
end procedure

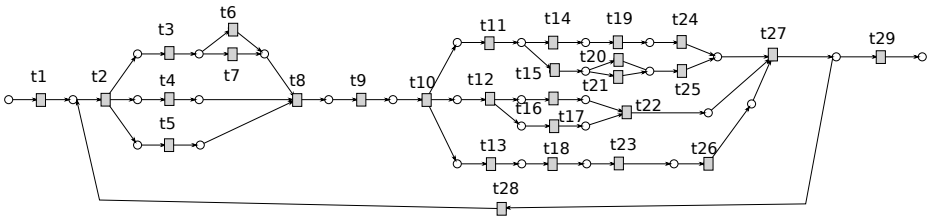

Fig. 1: Illustrative example of process modeled as a Petri net.

Subprocesses Decomposition Given that the final goal of this approach is the conformance analysis and the diagnosis of potential deviations, we propose a decomposition based on the identification of subprocesses within the main process. In particular, these subprocesses that match the *Single-Entry-Single-Exit* pattern, also known as *SESE*s [10], are detected. SESE represents a well-defined part of a general process. Note that, this decomposition refers to the structure of the model (not its behavior). Therefore, the SESE detection is not performed directly over the model, but over the underlying graph of the model, called *workflow graph*.

In our case, given a Petri net $PN = (P, T, A)$ we define its workflow graph simply as graph $G = (V, E)$ with no distinction between places and transitions, i.e., $V = P \cup T$ and $E = \{(x, y) \in V \times V | A(x, y) > 0\}$. For example, considering the example process of Fig. 1 modeled as a Petri net, its corresponding workflow graph is shown in Fig. 2. Similar representations can be obtained for other modeling notations such as BPMN, EPCs, or UML, and therefore, the hierarchical conformance approach presented in this paper is fully generic and can be applied to other notations. However, in this work, we focus on Petri nets.

In the remaining definitions, the following context is assumed: Let G be the workflow graph of a given Petri net, and let $G_F = (V_F, F)$ be a connected subgraph of G formed by a set of edges F and the vertexes V_F induced by F.

Definition 1 (Interior and Boundary nodes [7]). *A node $x \in V_F$ is* interior *with respect to G_F iff it is connected only to nodes in V_F; otherwise x is a* boundary *node of G_F.*

For instance, in the graph of Fig. 2, given the subgraph composed by the edges in D and its induced vertexes, $t2$ and $t8$ are boundary nodes, while the rest are interior nodes. A boundary node is an entry or exit node if additional requirements are satisfied.

Definition 2 (Entry and Exit nodes [7]). *A boundary node x of G_F is an* entry *of G_F iff no incoming edge of x belongs to F or if all outgoing edges of x belong to F. A boundary node x of G_F is an* exit *of G_F iff no outgoing edge of x belongs to F or if all incoming edges of x belong to F.*

Following with the example above, $t2$ would be an entry, and $t8$ an exit. Now we can define a SESE as follows:

Definition 3 (SESE, Trivial SESE and Canonical SESE [7]). *$F \subseteq E$ is a* SESE *(Single-Exit-Single-Entry) of graph $G = (V, E)$ iff G_F has exactly two boundary nodes: one entry and one exit. A SESE is* trivial *if it is composed by a single edge. F is a* canonical *SESE of G if it does not overlap with any other SESE of G, i.e., given any other SESE F' of G, they are nested ($F \subseteq F'$ or $F' \subseteq F$), or they are disjoint ($F \cap F' = \emptyset$).*

For the sake of clarity and unless otherwise is stated, in the rest of the paper we will refer to canonical SESEs simply as SESEs. Note also that the SESEs are defined as a set of edges (not as subgraphs). However, for simplicity we will refer also to the subgraph as SESE when the context is clear. Given that a single edge is a SESE, in the remainder of the paper we will only consider SESEs above a given threshold size[4]. Figure 2 shows the canonical SESEs of the example in Fig. 1 having size greater than 4 nodes (i.e., A,... R).

Component Hierarchical Structure Next we construct the Refined Process Structure Tree (RPST) between canonical SESEs. This tree-like structure will group all non-overlapping siblings at the same level that give rise, in an upper level, to the canonical SESE that includes all of them. This will allow us to navigate through the different levels of the tree thus providing views at different level of granularity.

Definition 4 (Refined Process Structure Tree (RPST) [7]). *The* Refined Process Structure Tree (RPST) *of G is the tree composed by the set of all its canonical SESEs, such that, the parent of a canonical SESE F is the smallest canonical SESE that contains F. The root of the tree is the entire graph, the leaves are the trivial SESEs.*

[4] On Sect. 6.1 we provide an experiment to estimate this threshold size based on manual decompositions.

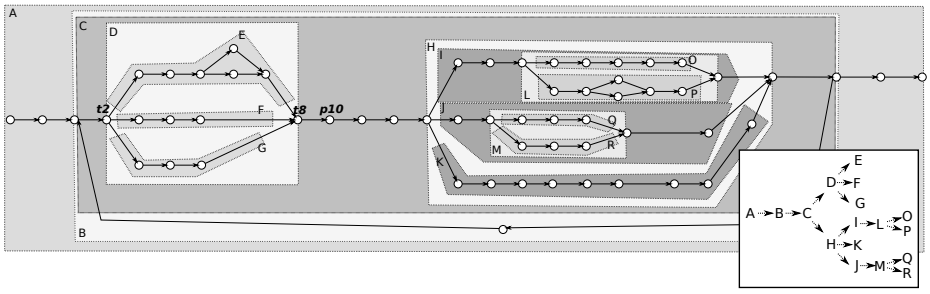

Fig. 2: Workflow graph of the example in Fig. 1, and its canonical SESEs with size greater than 4 nodes. On the bottom-right corner we show the corresponding RPST.

Figure 2 shows the RPST for the SESEs found in example of the same figure. Note that, due to the definition of canonical SESE, all siblings of a tree node will never have overlapping edges. However, it is not required that the union of all the siblings results in the entire parent canonical fragment (i.e., the parent of a SESE may have arcs not included in any of its children). For example, the edge ($t8$,$p10$) belonging to C is neither included on D nor H. The computation of the canonical SESEs and it's corresponding RPST of a workflow graph is a well studied problem in the literature, and can be computed in linear time. In [11], the authors proposed the original algorithm for constructing an RPST. In [7,10], the computation of the RPST is considerably simplified by introducing a pre-processing step that reduces the implementation effort considerably. Besides providing an explicit hierarchy among SESEs, the RPST structure satisfies additional properties useful for our approach: the RPST is *unique* (i.e., same graph will always result in the same RPST) and *modular* (i.e., a local change in the workflow graph only results in a local change of the tree).

Structure Enhancement Next, we enrich the hierarchical structure obtained (the RPST) with additional information that may be used to improve the conformance checking result. In this step we detect when a subprocess can be repeated more than once in a process execution. The idea behind this is that in order to perform the conformance checking analysis correctly, one must determine whether the subprocess reflected in a component can appear more than once within the same trace, i.e., within the same instance of the whole process, this subprocess can be executed several times. Such knowledge is highly relevant for the conformance evaluation.

In order to incorporate this information, first we must determine the cyclic behavior of the model. We will use this information to determine if a component may be iterated to reproduce a trace. Finally, this information is going to be transmitted to the RPST, and propagated through the tree. We will use the structural theory of Petri nets to determine potential iterative behaviors. In par-

ticular, we will use *T-invariants* [12] to determine potential repetitive behavior (see Sect. 4 for further details).

3.2 Component Post-Processing Stage

To perform a multilayer fine-grained conformance checking as the one presented in this paper, we must first derive the log and the net for each one of the subprocesses considered. For the log, the complete log refers to actions of the whole model. Therefore, we must project it only over the actions involved in the component we are analyzing, i.e., to analyze a component x that models only the set of tasks T_x, we must remove from the log all events not in T_x. Besides the net itself, we must determine also the initial and final markings of the Petri net for analyzing conformance. This step, and how it is performed, depends on the type of Petri net considered. In Sect. 4 we will show how to perform this processing for a well-known class of Petri nets used to model business processes.

3.3 Conformance Stage

Finally, the last stage is the actual conformance checking, i.e., ultimate goal of the approach presented in this paper. In process mining, checking the conformance refers to the procedure of analyzing if the model considered is an appropriate and faithful representation of the reality reflected in the log. Literature clearly shows that conformance of a model with respect to a log is a multidimensional property. There is consensus on four orthogonal dimensions: fitness, precision, generalization and simplicity [1,3,13]. In this paper, we will focus on the first two: fitness and precision. However, we would like to stress that the whole approach presented in this paper is not restricted to any particular conformance algorithm, i.e., it can be used in combination with any other conformance technique for the four dimensions.

Fitness measures if the model is able to represent all the behavior in the log. For example, consider the models and event log shown in Fig 3. Model a) fits perfectly the log d) because it is able to reproduce both traces. However, model b) fails to reproduce the second trace. Precision, on the other hand, measures if the model is precise modeling the log. For instance, a) models the log precisely (i.e., no more behavior is allowed than the one reflected), while c) is completely imprecise (it allows much more behavior than the one in the log, and therefore, it underfits the process it is meant to describe).

The difference between the hierarchical conformance checking (as presented in this paper) and existing approaches is that the conformance analysis is computed at different levels (not only in the complete model and log). By combining the conformance results of the different layers, we will be able to navigate and perform a complete diagnosis analysis of our system in both a top-down and bottom-up way. The fitness and precision checking proposed in this paper is based on cost-optimal alignment between traces in the log and possible runs of the model. Such alignments are then used as reference to compute fitness and

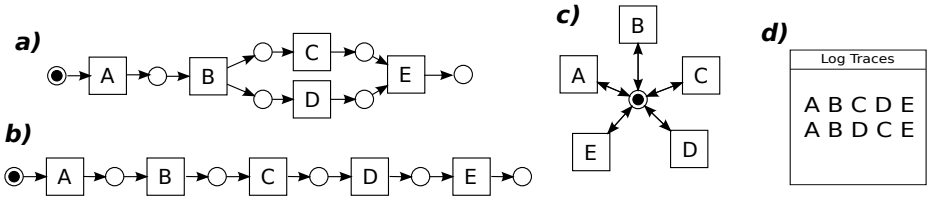

Fig. 3: Illustrative examples of fitness and precision with respect to a log d). Model a) fits better than b), and is more precise than c).

precision (cf. Fig 4). The technique presented in [5] only requires a log and a Petri Net with an initial and final marking.

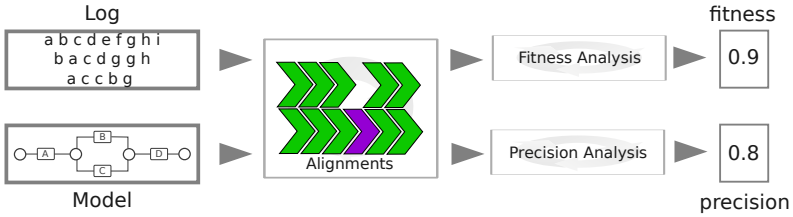

Fig. 4: Conformance checking based on Cost-Optimal Alignment

Cost-Optimal Alignment In order to check conformance, given a trace in the log, we need to find the model execution (trace) that best represents such log trace. This is done by aligning the trace with all possible executions of the model, assigning a cost to each alignment, and choosing the optimal one. An alignment is defined as sequence of pairs (called *moves*), where each pair can be: a) a 'move' in the model b) a 'move' in the log, and c) a 'move' both in the model and the log. For example, here are two possible alignments between the only two traces of the model in Fig.3a (ABCDE and ABDCE) and the log trace ABCDE:

$$\gamma_1 = \begin{vmatrix} A|B|C|D|E \\ A|B|C|D|E \end{vmatrix} \gamma_2 = \begin{vmatrix} A|B|C|D|\perp|E \\ A|B|\perp|D|C|E \end{vmatrix}$$

For each alignment γ_1 and γ_2, the upper row represents moves in the log, and the lower row represents moves in the model. The alignment between the trace and the execution ABCDE (shown as γ_1) is composed of synchronized moves (i.e., moves in both model and log), whereas the other alignment (γ_2) contains both a move on the log $(C, \perp)$ and a move on the model $(\perp, C)$ to be able to align the model execution and the trace. To measure the cost of an alignment, we define a cost function. For instance, we define a function that, for each pair, assigns a cost of '0' when it is a synchronized pair, and '1' otherwise (model and log disagree). The total cost of the alignment is the sum of the costs of individual

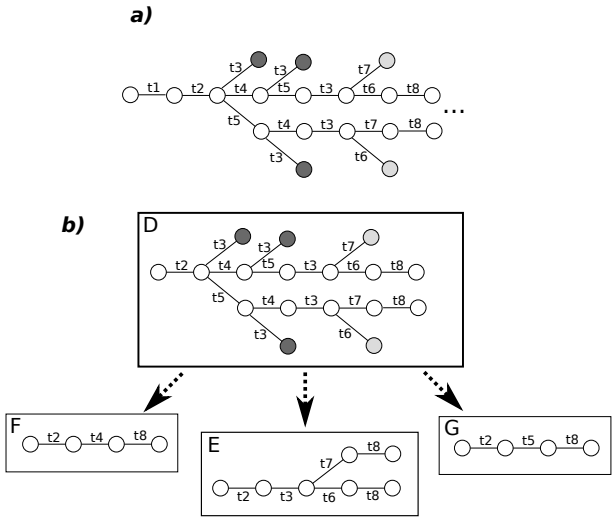

Fig. 5: Imprecisions for the example: a) Prefix automaton for the model of Fig. 1, b) Prefix automata for SESEs F, E and G.

moves[5]. Given such function, the costs of the alignments in the example above are 0 for γ_1 and 2 for γ_2. In [4,14], the authors propose various approaches to efficiently compute an optimal alignment for a given log trace.

Precision In order to compute precision from the optimal alignments, the bottom rows of all alignments are used to build a prefix automaton describing the modeled behavior observed in reality. Note that the alignments help to squeeze observed behavior into the model even in case of deviations. Then, this prefix automaton describing the actual observed behavior is compared with the modeled behavior, leading to the detection of the so called *imprecisions*, i.e., points in the process where the model allows for more behavior that actually observed in the event log. Due to space constraints, we refer to [5] for further details. The main difference between the approach in the literature, and the one proposed here, is that precision checking is performed for every subprocess (i.e., SESE), not only for the complete system.

The differences are illustrated using the following example: imagine the initial model of Fig.1, and consider a log composed of two traces: $t_1t_2t_4t_5t_3t_6t_8[\dots]$ and $t_1t_2t_5t_4t_3t_7t_8[\dots]$ and the model of Fig.1 (for sake of simplicity in this example we focus only on the part of the model between t_2 and t_8). The corresponding prefix automaton shown Fig. 5 reflects two main imprecisions: those nodes marked in dark gray represent the possibility in the model of executing concurrently the three branches (the branch starting at t_4, the one at t_5 and the one at t_3). The other imprecision (in light gray) is derived from the choice between t_6 and t_7. The process has three concurrent branches (notice that the log only reflects the concurrency among the $t4$ and $t5$ ones), and all the behavior modeled

[5] For the sake of simplicity, we use the default unit cost function. However, arbitrary complex user-defined cost functions can be used [4].

in each individual branch is reflected in the log (including the the choice between t_6 and t_7 were both options appear in the log). And this is precisely what we can see in a precision analysis using the hierarchical approach: the conformance analysis of SESE D (cf. Fig. 2) is similar to the previous approach, i.e., will derive a similar prefix automaton. However, in the hierarchical approach the analysis will also be done for the interior SESEs E, F and G, which will reflect a perfect precision, e.g., E precisely represents the projected log for (traces $t_2t_3t_6t_8[\dots]$ and $t_2t_3t_7t_8[\dots]$). A process-analyst can see that the conformance problems in D are not within any of its interior subprocesses but instead is a problem related to the sequencing of the subprocesses.

Fitness Once the optimal alignment for each trace is found, the non-sync moves are used to detect the fitness anomalies, i.e., points where the model does not reflect the log, or points where the log does not reflect the model. Various fitness metrics have been proposed, penalizing such anomalies. Due to space constraints, we refer to [4] for further details. Similar to the precision case, our hierarchical helps to diagnose of the process: one can navigate through the hierarchy, discarding subprocesses that are perfectly fitting, and focusing the analysis only on those that have fitness problems. For example, given long and repetitive traces such as $t_1t_2t_3t_4t_5\mathbf{t_6}\mathbf{t_7}t_8$ $[\dots]$ $t_{28}t_2t_3t_4t_5\mathbf{t_6}\mathbf{t_7}t_8$ $[\dots]$ $t_{28}t_2t_3t_4t_5\mathbf{t_6}\mathbf{t_7}t_8$ $[\dots]$ t_{29}, note that t_6 and t_7 are sequential in this trace. Looking at component E, one may clearly diagnose the fitness problem i.e., t_6 and t_7 are in conflict in the model.

4 The Case of Business Processes

In this section we apply the approach on a particular subclass of Petri nets tailored towards business processes: sound and safe workflow nets [15].

Workflow nets are a well studied model in the Business Process Management literature, and therefore, they are the target of various conformance approaches, (e.g., [3]). This section instantiates the methodology presented in the previous section to this class of nets. In the remainder we consider Petri nets satisfying the following seven conditions. The process models used for conformance checking must be a *workflow* net: 1) there is a single source place *in*, i.e., $\{p \in P | {}^\bullet p = \emptyset\} = \{in\}$, 2) there is a single sink place *out* , i.e, $\{p \in P | p^\bullet = \emptyset\} = \{out\}$, 3) every node is on a path from *in* to *out*. The workflow net must be *sound*: 4)have the option to complete, i.e., starting from the initial marking (just a token in place *in*), it is always possible to reach the marking with one token in place *out* (marking $[out]$), 5) have proper completion, i.e., at the moment a token is put in place *out*, all the other places should be empty, 6) there are no dead transitions starting from the initial state $[in]$. And finally, the sound workflow net must be *safe*: 7) the number of tokens of any place at any time must be at most one.

We now present the details of the post-processing stage for safe and sound workflow nets. In particular, given $G_F = (V_F, F)$, a SESE obtained during the decomposition phase, we describe how to build a SESE-Net SN^* from it, and

how to determine its initial and final markings. In other words, we aim to build a Petri net $SN^* = (P^*, T^*, A^*, i^*, o^*, p_i^*, p_o^*, m_i^*, m_o^*)$, where P^*, T^*, A^* define the net, p_i^*, p_o^* are the source and sink places of the net, m_i^*, m_o^* are its initial and final markings, and o^*, i^* define the nodes of the Petri net (transition or place) representing the single-entry and the single-exit of the SESE.

In the remainder of this section the following context is assumed: Let $WF = (P, T, A, p_i, p_o, m_i, m_o)$ be the workflow net to be analyzed, and $G = (V, E)$ its corresponding workflow graph. Let $m_V : V \to (P \cup T)$ and $m_E : E \to A$ be the relations between elements on the workflow graph and their corresponding elements in the workflow net. Let $G_F = (V_F, F)$ define a canonical SESE of G, and let $i, o \in V_F$ be the entry and exit nodes of G_F, respectively.

Given that the SESE decomposition is performed over the workflow graph, the first step is to determine which nodes of the original net WF are included in SN^*. In other words, for all $x \in V_F : m_V(x) \in P \implies m_V(x) \in P^*$ and $m_V(x) \in T \implies m_V(x) \in T^*$. Similar, the projection over the arcs is done, i.e., if $x \in F$ then $m_E(x) \in A^*$. For example, given the SESE D of the running example, shown in Fig. 6a, $t_2 \dots t_8$ and $p_3 \dots p_9$ are the transitions and places of the original net included in this SESE-net, respectively.

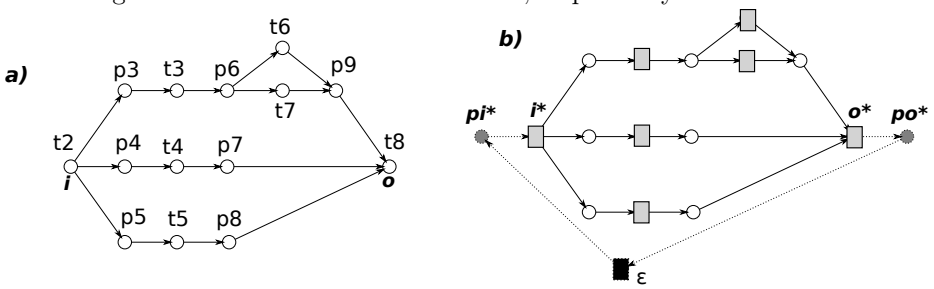

Fig. 6: a) SESE and b) its corresponding SESE-Net

Note that, by definition, G_F contains a single start and end node: i and o. Therefore, we define i^*, o^* as $i^* = m_V(i)$ and $o^* = m_V(o)$. The existence of such single-entry and single-exit is an appropriate property in order to determine the source and sink places of the SN^*. Since the SESE decomposition is performed over the workflow graph (where there is no distinction between places and transitions), it is not guaranteed that i^* and o^* are places (they may be transitions). This is the case of the component D of Fig. 6. In such cases where the entry (or the exit) node is a transition, a pre-processing of the SN^* is required, i.e., an artificial place is created and linked with an artificial arc to such transition. The transformation is informally illustrated in Fig. 7. An example is shown in Fig. 6a, where both the entry and the exit nodes are transitions. Therefore, two artificial places are created (and its corresponding arcs) to represent the p_i^* and p_o^* of the SESE-net, as it is shown in Fig. 6b.

As discussed in the previous section, a net can contain cyclic behavior, i.e., subprocesses that are repeated several times within the main process execution. In the decomposition phase, the cyclic behavior is detected by determining those

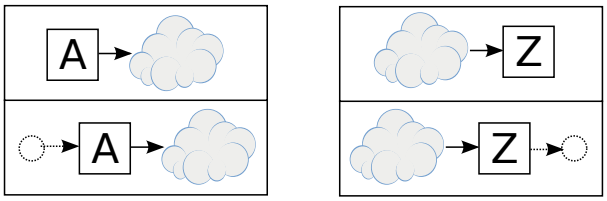

Fig. 7: Pre-processing of the SESE-net to guarantee a place-bordered net.

SESEs that are covered by T-invariants (i.e., if all transitions in a SESE belong to some T-invariant, we assume that the component can be repeated) and the hierarchical structure is enhanced with this information. Formally, a T-invariant is a vector $\mathbf{x} : T \to \mathbb{Q}$ such that $\mathbf{C} \cdot \mathbf{x} = 0$, where $\mathbf{C}$ is the *incidence matrix* of the net [12]. Intuitively, a T-invariant is a set of transitions such that the marking reached after executing them is the same as the one before its execution. Clearly, T-invariants provide only an approximation of the real repetitive behavior, but we have seen in practice that this heuristically approximation works fine. Moreover, if repetition is possible, a corresponding T-invariant exists.

Potential cyclic behavior has to be transferred to the subprocess level, i.e., SN^* has to reflect the possibility of being executed more than once. In order to do that, the SN^* is *short-circuited* using a silent transition ϵ, i.e., when the execution of the subprocess reaches the end, there is the possibility to re-start, though the firing of a transition that leaves no track on the log (i.e., it is *silent*). Formally, if G_F is detected as cyclic, then $\epsilon \in T^*$ and $(p_o^*, \epsilon), (\epsilon, p_i^*) \in A^*$, where $\epsilon \notin T$. This is the case for the subprocess in Fig. 6a: the two minimal T-invariants $\{\{t2, t3, t4, t5, t6, t8, t9, t10, t11, t12, t13, t15, t16, t17, t18, t20, t22, t23, t25, t26, t27, t28\}, \{t2, t3, t4, t5, t7, t8, t9, t10, t11, t12, t13, t15, t16, t17, t18, t20, t22, t23, t25, t26, t27, t28\}\}$ of the initial net (Fig.1) include the transitions of the SESE and therefore it is tagged as cyclic. Hence, a silent transition short-circuit is created as it is shown in Fig. 6b.

Next, we need to determine the initial marking m_i^* of the subprocess. Given the workflow net nature of the subprocess, the initial marking is obvious: a single token on the initial place in order to start the execution of the process. However, in order to preserve the initial state of the whole process, it is also required to project such state into the elements of the component. Formally, $m_i^* = m_{i \downarrow P^*} \cup [p_i^*]$. The initial marking of the net in Fig. 6b would be $[p_i^*]$. The the final marking of the subprocess can be obtained in a similar manner. The final marking of the net in Fig. 6b would be $[p_o^*]$.

After post-processing we obtain a new Petri net SN^* modeling the subprocess behavior. When iteration is detected, due to the type of nets we are restricting in this section (safe and sound workflow nets), the modification done to the original SESE-net enables the iterative behavior in the modified SESE-net. The final SESE-net (with or without iteration), is used to do conformance analysis.

Likewise it is done in [9] for fitness, it should be possible to establish formal guarantees relating both fitness and precision metrics evaluated in the subpro-

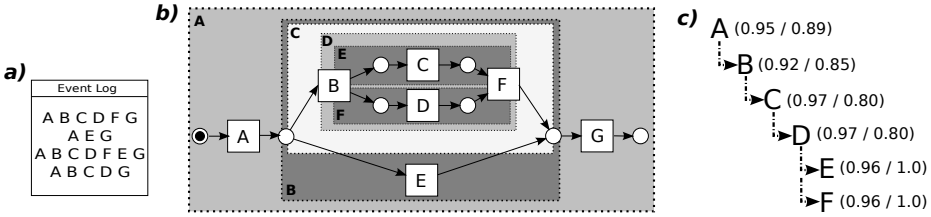

Fig. 8: The hierarchical approach is non-monotonic with respect to subprocess inclusion. Each node in the hierarchy c) of the SESEs detected in b) is annotated with a pair of real numbers, representing the fitness and precision of the corresponding SESE-net for the log obtained after projecting the event log of a) for the participating activities. For instance, the fitness (first number in the nodes) for nodes A (0.95), B (0.92) and C (0.97) is neither increasing nor decreasing.

cesses with respect to the ones of the original net. However, since neither fitness nor precision are monotonic with respect to subprocess inclusion, the weakest conditions under which a formal guarantee relating model and subprocesses fitness/precision can be given will be investigated in the future. The example in Fig. 8 illustrates the non-monotonic nature of the hierarchical approach.

5 Related Work

This section provides the relation with the only one approach in the literature for decomposed conformance checking. In [8] the authors propose a non-hierarchical decomposition based on passages, also meant to be applied to the same class of Petri nets as the one considered in this paper. A passage is a pair of two non-empty sets of transitions (X, Y) such that the set of direct successors of X is Y, whilst the set of direct predecessors of Y is X. Any graph can be decomposed into minimal passages representing a partitioning of its transition set, and in [8] authors demonstrate how both discovery and conformance problems can be decomposed using passages.

There are significant differences between the passage decomposition approach and the one presented in this paper. We enumerate here the most important ones:

1. The passages approach computes a 1-level partitioning, whereas our approach derives a hierarchy of components. Note that once 1-level passages are obtained, they may be united to form higher-level passages, since the union of passages is a passage. However, algorithms for this high-level post-process to form a hierarchy have not been proposed nor implemented.
2. The components in the passages approach represent causality fragments between two sets of transitions, whilst components in our approach denote parts of the model which interface with the rest of the system only through two boundary nodes. In other words, components derived by each one of the approaches are incomparable, i.e., a typical passage cannot be obtained

through our approach whilst a typical SESE (like the one shown in Fig. 6a) is not a passage.

In the next section an empirical comparison for a set of benchmarks is provided that witness the previous claims.

6 Experimental Results

In this section we present the experimental results supporting the claims made in earlier sections. Most of the models have been generated with the `PLG` tool [16]. Two main type of experiments are provided in two subsections: Sect. 6.1 is structured in two parts: in part I we compare the hierarchical approach presented in this paper with manual decompositions performed by some persons, and in part II we test the approach through a set of experiments, comparing it with the approaches in the literature and especially with the one presented in [8]. Sect. 6.2 describes an experiment illustrating how the approach can be applied for a large example, where it is not possible to handle large subprocesses and therefore only nodes of the RPST that have a size less than a certain value can be analyzed in practice. An implementation of the approach presented in this paper can be found as a plugin within the ProM Framework[6].

6.1 Empirical evaluation

Part I: Similarities with human-made model decomposition
The first set of experiments is designed to study one of the strong points of the approach proposed: the process-like decomposition (based on Single Entry Single Exit components) and its intuitive relation with the mental schema of process analyst. A set of process models (*man01* to *man06*) has been prepared in order to be manually decomposed by possible actors of the approach, and the resulting components have been compared with the ones obtained by the approach of this paper. The set of benchmarks is composed by six Petri nets modeling plausible processes. Each model contains the most common patterns found on process modeling: choice, concurrency, sequencing, invisible tasks, and loops. The manual decomposition has been performed by 7 persons, all them with complete liberty to decide their own decomposition. The results of the experiments are shown in Table 1[7]. The table shows, the number of transitions and places of the initial Petri net, whether all persons used hierarchy in the decomposition (hierarchy ?), the (average of the 7 persons) maximum number of levels in the hierarchy (*h- max*), and the corresponding maximum number of levels in the hierarchy computed by our approach (with a 4-nodes threshold). Also, the number of components computed by all the persons is shown ($|C|$), and the percentage of components having less than 5 nodes reported ($|Size(C) < 5|$).

[6] Under the package *JorgeMunozGama*.
[7] In the link `http://www.lsi.upc.edu/~jmunoz/files/PN2013benchmarks.zip` the reader can inspect the manual decompositions.

Table 1: *Manual decomposition compared with the hierarchical approach.*

	$\|T\|$	$\|P\|$	hierarchy ?	h-max	h-SESE	$\|C\|$	$\|Size(C) < 5\|$	% SESE-like
man01	44	45	Yes	3.4	8	63	14%	91%
man02	16	16	Yes	2.4	5	28	7%	97%
man03	31	34	Yes	3.3	6	50	16%	89%
man04	31	31	Yes	3.3	7	51	16%	96%
man05	48	50	Yes	3.7	7	79	14%	92%
man06	45	51	Yes	3.6	8	71	3%	94%

Finally, we provide the percentage of the components that satisfy the SESE property (column % SESE-like).

The first conclusion raised from the experiment is related with the intuitive use of hierarchy for decomposition. This is one of the main differences of the approach proposed in this paper compared with other approaches such as [8]. The results show that in all benchmarks, all persons use hierarchy. Second, the average number of levels in the hierarchy used by the 7 persons is smaller than the one computed by our approach. The main reasons for that is that the persons have not considered components that differ from each other of very few nodes which is the case for a parent and sibling having a high degree of overlapping. For instance, in Fig. 2, the only difference between SESE A and SESE B is the four edges in A not in B. This suggests to refine the SESE detection algorithm to collapse in the RPST nodes that differ on very few edges, in order to remove this type of redundancy, e.g., collapse nodes A and B as one single SESE in the example.

Note that persons prefer larger and process-like components instead of small and oversimple ones. For example, from column $|C|$, the percentage of small components ranges from 3% to 16% (and in most of the cases they correspond with components of size four: a choice of two transitions). This motivates the use of a threshold in the approach proposed in this paper. And finally, there is a direct relation between the components suggested manually and the ones provided by the approach, i.e., the vast majority of the components correspond with SESEs, as it is shown in *% SESE-like* column.

Part II: Evaluation through a set of benchmarks

This second set of experiments focuses on the actual hierarchical conformance checking. The experiment is composed by six models of different sizes, and their corresponding logs. For each of the benchmark model and log combinations, the decomposition is obtained and the conformance analysis is performed per component. The results are shown in Table 2 (comparison with respect to the passage approach), and Table 3, where the results for approaches in the literature [4,5,6] are provided. In Table 2 the number of places and transitions per benchmark are reported, and for each one of the decomposition approaches (passages or hierarchical), the number of components obtained (column $|C|$) is reported. The table also provides the average fitness/precision with respect to the number of

Table 2: *Conformance Results: passages vs. hierarchical*

			Passages					*Hierarchical*								
				Fitness					Fitness				Precision			
	$\|T\|$	$\|P\|$	$\|C\|$	comp	node	max	min	$\|C\|$	comp	node	max	min	comp	node	max	min
lu01	81	74	50	1	1	1	1	48	1	1	1	1	.958	.761	1	.293
lu02	45	51	29	.897	.898	.942	.84	34	.926	.912	1	.883	.978	.899	1	.649
lu03	86	80	50	1	1	1	1	55	1	1	1	1	.918	.719	1	.327
lu04	43	38	23	1	1	1	1	27	1	1	1	1	.837	.710	1	.343
lu05	91	82	50	.976	.977	1	.939	59	.914	.915	1	.480	.955	.789	1	.362
lu06	59	54	36	.988	.988	1	.955	28	.992	.989	1	.98	.92	.689	1	.272

Table 3: *Conformance Results: results for non-decomposition approaches in the literature.*

	Fitness	Precision	
	[4]	[5]	[6]
lu01	1	.301	.301
lu02	.884	.65	.72
lu03	1	.335	.335
lu04	1	.35	.35
lu05	.965	.22	.22
lu06	.988	.283	.295

components (*comp*), and with respect to the size of the components (*node*), i.e., in *node* larger components have larger weight on the average fitness/precision computation. The table provides the maximum and minimum fitness/precision value obtained for each benchmark (max and min)[8]. Finally, Table 3 reports the conformance analysis (fitness and precision) by other approaches in the literature.

Regarding the comparison between the passages approach and the hierarchical, it should be noted that there is a tendency for the passages approach to derive less components, which in fact are considerably small. On the fitness reported, both approaches report perfect fitness for the fitting models *lu01*, *lu03* and *lu04*, and some differences for the rest. Interestingly, in *lu05* a component with a fitness value of 0.480 is detected, representing the diagnosis of a fitness problem in a particular subprocess of the model.

By comparing the values in Table 3 of the non-decomposition approaches in the literature with the ones in Table 2, a clear tendency of the decomposition techniques to provide higher average values (both in the comp and the node columns) is detected. This is especially manifested in the precision dimension, where although the precision problems are still detected and equally evaluated as in the non-decomposition approaches (see column min for the hierarchical

[8] Only the fitness calculation is implemented in the approach [8].

precision), they are reported at the level of the subprocess, thus identifying the real portion of the model that represents the conformance problem.

Hence, the results for the hierarchical approach, divided into components, provide a useful tool for diagnosis. This is complemented by the tree visualization provided by the implementation, where the conformance results of each component are displayed over its corresponding node in the hierarchy using a intuitive code of colors, aiming at localizing problems within the process (see Fig. 9). As an example, in some processes such as *lu03* (shown in Fig. 9), the differences in precision between comp and node is considerable, reflecting that while the vast majority of components have a high precision, the larger components have a low one. This is confirmed in the conformance results obtained for each particular component: while the two largest components have a precision of 0.33 and 0.32 respectively (shown in the figure as nodes in red in the RPST), the rest of components (of significant smaller size) have precisions close to or exactly 1.0. An introspective view of this conformance problem shows that a big component allows the concurrent execution of three subprocesses, each one having no conformance problems. However, in the log there is a real sequencing on these three components, and therefore a precision problem has been identified between these three subprocesses.

6.2 Handling a large conformance problem

The hierarchical approach presented in this paper can be used to limit the complexity of the conformance analysis by bounding the size of the conformance instances to solve. This may be done by selecting a partitioning in the RPST selecting those SESEs whose size do not exceed a given threshold size. This section illustrates how this strategy can be used to perform conformance analysis for problems of industrial size.

With the `PLG` tool, we have created a model of 693 nodes, depicted in Fig. 10, and its corresponding log[9]. Then we have used the approach in [4] to estimate the fitness of the model with respect to the log, establishing a time limit of 1 hour. Within this limit, the aforementioned approach did not completed the fitness computation. If instead the hierarchical approach is applied together with a restriction on the size of subprocesses to consider (we have set 200 nodes as maximum SESE size), we have been able to analyze conformance for the subprocesses forming the partitioning shown in Fig. 10 in less than 2 minutes. This conformance analysis revealed, for instance, that the subprocess in green (lefmost) has a poor precision (0.318) and therefore it may be inspected in isolation to determine the causes of the conformance problem, while the process in pink has a perfect precision of 1.0.

[9] To ease subprocess identification, we have used colors in Fig. 10 which will only be visible in the electronic version or in a colored printed version of the paper.

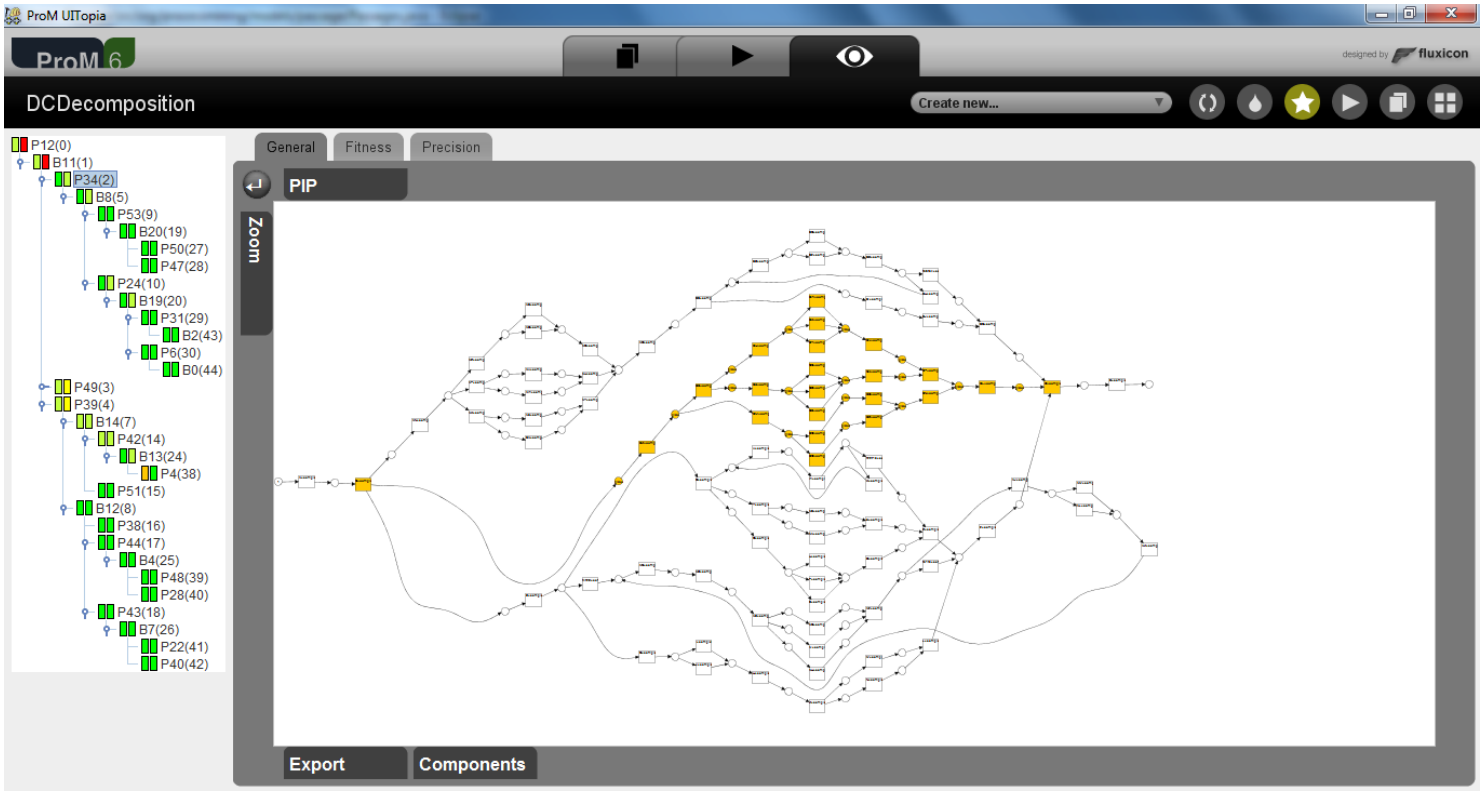

Fig. 9: The hierarchical view of conformance checking: the tree-view (left panel) provides the RPST together with the conformance results. Fitness and precision metrics for each SESE (node in the RPST) are provided as two colored squares, following a semaphore code: green (100-90%), yellow (89-75%), orange (74-50%) and red (49-0 %). By selecting a particular node in the RPST, the corresponding SESE is highlighted in the right panel.

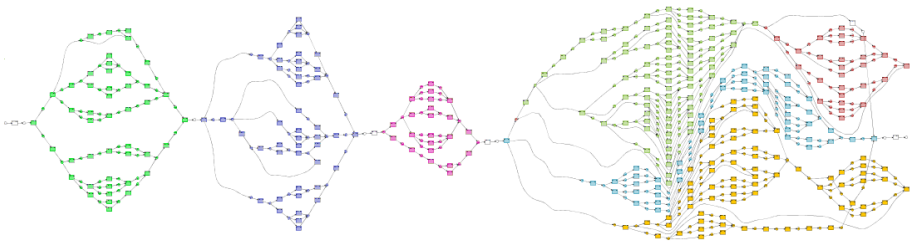

Fig. 10: Partitioning based on SESEs for conformance analysis: we have highlighted in colors the SESEs for which a conformance analysis has been done.

7 Conclusions and Future Work

In this paper we presented a hierarchical approach for conformance checking that enables process analysts to investigate deviations between modeled and observed behavior. The technique combines previous work on conformance checking and model decomposition techniques in order to identify subprocesses within a given model for which an isolated conformance checking can be done, offering a hierarchical structure that can be used to navigate through the conformance results. The experiments show both the usefulness of the approach and the difference with related techniques.

The future work aims to extend the result in various directions. We plan to investigate the theoretical guarantees and extend the proposed technique to a larger class of Petri nets. Regarding algorithms, we plan to study new algorithms to improve both the quality and the performance of the proposed methodology, e.g., proposing a reduction of the RPST to avoid too much overlapping between SESEs and finding ways to propagate in a bottom-up manner the conformance results. The latter is very important to be able for real-life conformance checking where performance is an issue.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (May 2011)
2. Rogers, S.: Big data is scaling BI and analytics-data growth is about to accelerate exponentially. Information and Management - Brookfield **21**(5) (2011) 14
3. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1) (2008) 64–95
4. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance checking using cost-based fitness analysis. In: EDOC, IEEE Computer Society (2011) 55–64
5. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Alignment based precision checking. In: Business Process Management Workshops. (2012)

6. 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
7. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. In Bravetti, M., Bultan, T., eds.: WS-FM. Volume 6551 of Lecture Notes in Computer Science., Springer (2010) 25–41
8. van der Aalst, W.M.P.: Decomposing process mining problems using passages. In Haddad, S., Pomello, L., eds.: Petri Nets. Volume 7347 of Lecture Notes in Computer Science., Springer (2012) 72–91
9. van der Aalst, W.M.P.: Decomposing Petri nets for process mining: A generic approach. Technical Report BPM-12-20, BPM Center (September 2012)
10. Polyvyanyy, A.: Structuring process models. PhD thesis, University of Potsdam (2012)
11. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. In Dumas, M., Reichert, M., Shan, M.C., eds.: BPM. Volume 5240 of Lecture Notes in Computer Science., Springer (2008) 100–115
12. Silva, M., Teruel, E., Colom, J.M.: Linear algebraic and linear programming techniques for the analysis of place or transition net systems. In Reisig, W., Rozenberg, G., eds.: Petri Nets. Volume 1491 of Lecture Notes in Computer Science., Springer (1996) 309–373
13. Rozinat, A., de Medeiros, A.K.A., Günther, C.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The need for a process mining evaluation framework in research and practice. In ter Hofstede, A.H.M., Benatallah, B., Paik, H.Y., eds.: Business Process Management Workshops. Volume 4928 of Lecture Notes in Computer Science., Springer (2007) 84–89
14. Adriansyah, A., Sidorova, N., van Dongen, B.F.: Cost-based fitness in conformance checking. In Caillaud, B., Carmona, J., Hiraishi, K., eds.: ACSD, IEEE (2011) 57–66
15. van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. Formal Asp. Comput. **23**(3) (2011) 333–363
16. Burattin, A., Sperduti, A.: Plg: A framework for the generation of business process models and their execution logs. In zur Muehlen, M., Su, J., eds.: Business Process Management Workshops. Volume 66 of Lecture Notes in Business Information Processing., Springer (2010) 214–219