Information Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data

Massimiliano de Leoni [a,*], Fabrizio M. Maggi [b], Wil M.P. van der Aalst [a]

[a] Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
[b] Institute of Computer Science, University of Tartu, Estonia

ARTICLE INFO

Keywords:
Process mining
Declare
LTL
Conformance checking
Event-log preprocessing

ABSTRACT

Process mining can be seen as the "missing link" between data mining and business process management. The lion's share of process mining research has been devoted to the discovery of procedural process models from event logs. However, often there are predefined constraints that (partially) describe the normative or expected process, e.g., "activity *A* should be followed by *B*" or "activities *A* and *B* should never be both executed". A collection of such constraints is called a *declarative process model*. Although it is possible to discover such models based on event data, this paper focuses on *aligning* event logs and predefined declarative process models. Discrepancies between log and model are mediated such that observed log traces are related to paths in the model. The resulting alignments provide sophisticated *diagnostics* that pinpoint *where deviations occur and how severe they are*. Moreover, selected parts of the declarative process model can be used to *clean* and *repair* the event log before applying other process mining techniques. Our alignment-based approach for preprocessing and conformance checking using declarative process models has been implemented in ProM and has been evaluated using both synthetic logs and real-life logs from a Dutch hospital.

1. Introduction

Traditional Workflow Management (WFM) and Business Process Management (BPM) systems are based on the idea that processes can be described by procedural languages where the completion of one task may enable the execution of other tasks, i.e., procedural models are used to "drive" operational processes. While such a high degree of support and guidance is certainly an advantage when processes are repeatedly executed in the same way, in dynamic and less structured settings (e.g., healthcare) these systems are often considered to be too restrictive. Users need to react to exceptional situations and execute the process in the most appropriate manner. It is difficult, if not impossible, to encode this human flexibility and decision making in procedural models.

Declarative process models acknowledge this and aim at providing freedom without unnecessarily restricting users in their actions. Procedural process models take an "inside-to-outside" approach, i.e., all execution alternatives need to be explicitly specified and new alternatives need to be incorporated in the model. Declarative models use an "outside-to-inside" approach: anything is possible unless explicitly forbidden. Hence, a declarative process model can be viewed as a set of constraints rather than as a procedure.

WFM and BPM systems tend to force people to work in a particular way. When using a declarative WFM and BPM system, more freedom can be offered. However, in most dynamic and less structured settings no system is enforcing users to work in a particular way. This may result in

* Corresponding author.
E-mail addresses: m.d.leoni@tue.nl (M. de Leoni), f.m.maggi@ut.ee (F.M. Maggi), w.m.p.v.d.aalst@tue.nl (W.M.P. van der Aalst).

0306-4379/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.is.2013.12.005

undesirable deviations and inefficiencies. Sometimes there may be good reasons to do things differently. Consider the "breaking the glass" functionality in many systems as a means to deal with exceptions, e.g., using the emergency breaks in case of an accident, unauthorized access to private patient data in case of an emergency and bypassing an administrative check to help an important customer.

Even though process models are typically not enforced, many events are recorded by today's information systems. As information systems are becoming more and more intertwined with the operational processes they support, "torrents of event data" become available. Therefore, it is interesting to compare observed behavior with modeled behavior.

This paper proposes the implementation of a framework for the analysis of the execution of declarative processes. It is based on the principle of creating an *alignment* of an event log and a process model. Each trace in the event log is related to a possible path in the process model. Ideally, every event in the log trace corresponds to the execution of an activity in the model. However, it may be the case that the log trace does not fit completely. Therefore, there may be "moves" in the event log that are not followed by "moves" in the model or vice versa.

The alignment concept has successfully been used in the context of procedural models (e.g., [1–3]); here, we adapt it for declarative models. Similarly to what has been proposed for procedural models, in our approach, events in the log are mapped to executions of activities in the process model. A cost/weight is assigned to every potential deviation. We use the A* algorithm [4] to find, for each trace in the event log, an optimal alignment, i.e., an alignment that minimizes the cost of the deviations. The application of the A* algorithm is more challenging for declarative models than for procedural models. This is due to the fact that, since in a declarative model everything is allowed unless constrained otherwise, the set of admissible behaviors is generally far larger than the set of behaviors allowed by procedural models. This implies that the search space to find an optimal alignment of a log and a declarative model is much larger. Therefore, for this type of models, it is essential to avoid exploring search-space portions that certainly lead to non-optimal solutions.

The log-model alignment can be the main input of a wide range of techniques for the analysis of declarative processes. On this concern, Section 3 shows the three main use cases that are considered in this paper. The first use case is concerned with cleaning the event logs by removing log traces that should not be used for further analysis (e.g., incomplete traces). The second use case is about checking the conformance of the event logs against a given declarative model, which can be regarded and measured from diverse dimensions, highlighting where deviations occur. The third and last use case concerns repairing event logs to make sure that the essential constraints are satisfied before further analysis. These use cases are supported by functionalities that are available in ProM, a generic open-source framework for implementing process mining tools in a standard environment [5].

In this paper, we use *Declare* as an example of declarative language. Section 2 introduces the basic aspects of the *Declare* language along with the working example that is used throughout the paper, while Section 4 introduces some background knowledge. Section 5 describes the notion of log–model alignment and some diagnostics that can be computed using alignments. Section 6 describes the application of the A* algorithm to find an optimal alignment. Here, we also introduce an optimization of the algorithm to prune large irrelevant portions of the search space that certainly lead to non-optimal solutions. Section 7 discusses the second use case in detail, i.e., how the alignments can be used to check the conformance of an event log with respect to a *Declare* model. Section 8 focuses on the first and third use case, i.e., how event logs can be cleaned and repaired. Section 9 reports an evaluation of the different techniques, which is based on synthetic and real-life logs. Section 10 discusses related work, whereas Section 11 concludes the paper and highlights potential future work.

2. Declare and basic notation

Declare is a declarative language with an intuitive graphical representation to describe constraints and activities [6–8]. Its formal semantics is based on Linear Temporal Logic (LTL) for finite traces [9] where each constraint is defined through an LTL formula.[1] The *Declare* toolset includes a graphical designer, a workflow engine, a worklist handler and various analysis tools [10].[2] A detailed description of *Declare* is out of the scope of this paper. The most relevant *Declare* features are introduced here through an illustrative example. Interested readers are referred to [11] for a detailed coverage of the *Declare* language.

Example 1. A travel agency has enacted a process to handle health-related insurance claims. The process to handle these claims is illustrated in Fig. 1. The model includes eight activities (depicted as rectangles, e.g., *Contact Hospital*) and six constraints (shown as connectors between activities). Depending on the claimed amount, a claim can be classified as high or low. For low claims, tasks *Low Insurance Check* and *Low Medical History* need to be executed. The *co-existence* constraint indicates that these activities are always executed together (in any order). If a claim is classified as low, no activities referring to high claims can be executed and vice versa. The *not co-existence* constraint indicates that *Low Insurance Check* and *High Insurance Check* can never coexist in the same process instance. Moreover, in case of high claims, the medical history check (*High Medical History*) can only be executed together with the insurance check (*High Insurance Check*), even though they can be executed in any order. Nevertheless, it is possible to execute *High Insurance Check* without executing *High Medical History*. All this is enforced by the *responded existence* constraint. For every claim, it is also possible to contact the doctor/hospital for verification. However, in case of high claims, this cannot be done before the insurance check. This is defined by the *not succession* constraint: *Contact Hospital* cannot be followed in the same process instance by *High Insurance Check*. A

[1] For compactness, in the following we will use the LTL acronym to denote LTL for finite traces.

[2] Declare web site – http://www.win.tue.nl/declare/.

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005

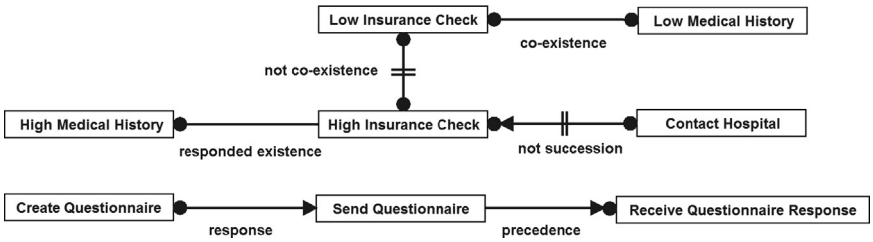

Fig. 1. *Declare* model used as working example. It consists of six constraints and eight activities.

questionnaire may be created and eventually sent to the applicant (modeled by the *response* constraint); only after the questionnaire is sent, the applicant can possibly decide whether to fill it out or not (*precedence* constraint).

The *response* constraint in Fig. 1 can be formally represented using LTL as $\Box$ (*Create Questionnaire* $\rightarrow \Diamond$ *Send Questionnaire*). This means that "whenever activity *Create Questionnaire* is executed, eventually activity *Send Questionnaire* is executed". In the following, we refer to a *Declare model* $\mathcal{D} = (A, \Pi)$, where A is a set of activities and Π is a set of *Declare* constraints defined over activities in A.

We conclude this section by introducing the basic notation that is used throughout the paper. A multiset $M = (Y, \Lambda)$ consists of a set Y and a function $\Lambda : Y \rightarrow \mathbb{N}$ that associates each element $a \in Y$ with its multiplicity $\Lambda(a)$, i.e., the number of its occurrences in the multiset. $a \in Y$ if and only if $a \in M$. We indicate the number of elements in the multiset with $|M| : |M| = \sum_{y \in Y} \Lambda(y)$. Let X be a set; $|X|$ denotes the number of elements in X. We indicate the set of all finite sequences over X with X^* and the set of all multi-sets over X with $\mathbb{B}(X)$. Moreover, we use angle brackets to denote sequences $\sigma \in X^*$, i.e., $\sigma = \langle s_1, \ldots, s_n \rangle$. Given two arbitrary sequences x and y, $|x|$ denotes the length of x and $x \oplus y$ denotes the sequence obtained by concatenating y to x. Moreover, prefix(x) denotes the set of all prefixes of x, i.e., a sequence $z \in$ prefix(x) if there exists a sequence w such that $z \oplus w = x$. From the definition above, for any sequence x, both x and the empty sequence $\langle\rangle$ always belongs to prefix(x).

3. A framework based on log–model alignment for the analysis of declarative processes

This paper proposes a framework for the analysis of declarative processes that is based on three *use cases*. All three use cases are supported by new functionalities added to ProM. The application of the use cases heavily relies on the computation of log–model alignments, which is also part of this paper.[3] The starting point is a raw event log $\mathcal{L}_{raw}$ and a so-called *whole model* $\mathcal{D}_{whole} = (A_{whole}, \Pi_{whole})$.

A first use case is based on the definition of a *core model* $\mathcal{D}_{core} = (A_{core}, \Pi_{core})$, where $\Pi_{core} \subseteq \Pi_{whole}$ and $A_{core} \subseteq A_{whole}$. This model is used to clean the event log, i.e., if the optimal alignment of a log trace reveals a violation of any constraint in Π_{core}, the trace is simply discarded. For example, if the *not co-existence* constraint in Fig. 1 is in Π_{core}, then all log traces containing both *Low Insurance Check* and *High Insurance Check* are removed from the log. The core model $\mathcal{D}_{core} = (A_{core}, \Pi_{core})$ can be used to filter out traces that are obviously incomplete or deviating too much from the expected behavior to be used for further analysis. Indeed, without the log cleaning, the results of some analysis can be far misleading.

The cleaned event log $\mathcal{L}_{clean}$ and the whole model $\mathcal{D}_{whole} = (A_{whole}, \Pi_{whole})$ can be used to diagnose deviations. In particular, the analyst may use the new functionalities added to ProM to inspect the optimal alignment for each log trace and to interpret the deviations. In our toolset, we also provide the process analyst with a summary that gives a helicopter view of the conformance of the model with respect to the entire log. In particular, we aggregate the information associated with the optimal alignments and visualize the deviations on the process model. In fact, we generate a "map" thus highlighting on the *Declare* model in which constraints are more often violated during the execution and in which activities are mostly involved in the violations.

We also introduce a *repair model* $\mathcal{D}_{repair} = (A_{repair}, \Pi_{repair})$ such that $\Pi_{core} \subseteq \Pi_{repair} \subseteq \Pi_{whole}$ and $A_{core} \subseteq A_{repair} \subseteq A_{whole}$. The cleaned event log $\mathcal{L}_{clean}$ and the *repair model* $\mathcal{D}_{repair} = (A_{repair}, \Pi_{repair})$ can be used to adjust the event log to prepare it for further analysis. Depending on the computed optimal alignment, some events may be discarded whereas other events may be artificially created and inserted in $\mathcal{L}_{clean}$. Most process mining techniques for performance analysis [12] and operational support (e.g., predictions and recommendations) *only function properly if the event log indeed "fits" the model used for analysis*. These process mining techniques often assume a perfect fitness and need to ignore deviating log traces.

However, in real-life scenarios a large fraction of cases may not fit perfectly (especially if the corresponding log traces are relatively long). Generally, it is not acceptable to discard large amounts of event data. Moreover, deviating log traces may have different characteristics (e.g., longer flow times). Hence, leaving them out may lead to misleading results. Therefore, the log is repaired before analysis: $\mathcal{L}_{repair}$ contains repaired traces satisfying all constraints in Π_{repair} and do not contain deviations for the activities in A_{repair}. Note that, unlike when cleaning an event log, in this case no traces are removed from the log.

When repairing the event log, the original behavior is modified, i.e., non-fitting traces are "squeezed" into the

[3] ProM, including the new functionalities described in this paper, can be downloaded from www.processmining.org.

repair model $\mathcal{D}_{repair} = (A_{repair}, \Pi_{repair})$. This should be taken into consideration when analyzing $\mathcal{L}_{repair}$ using other process mining techniques.

Note that, given a whole model, a clean and repair model can be usually defined in collaboration with process experts. Indeed, process experts have the proper expertise to determine which characteristics a log trace needs to have not to be discarded, i.e., which activities and constraints need to be part of the core model. For example, if all events in a certain time frame are extracted from a database and, then, grouped in traces based on the process instance identifier, some traces may be incomplete. In these traces, for example, the first or the last event can be missing since they occurred outside the considered time frame. Certainly, these traces should be discarded as they would affect the correctness of the outcomes of various process-mining techniques. Regarding the repair model, process experts can identify the constraints that should not be violated. The violations of these constraints are not as severe as the violations of the constraints belonging to the core model. Hence, the violating log traces are still worth to be considered for further analysis but they need to be "adjusted".

All three use cases described in Fig. 2 heavily rely on the computation of alignments. The main contribution of this paper is, then, a new approach for efficient computing alignments of logs and declarative process models.

4. Checking declare constraints

To identify potential deviations of a log from a reference *Declare* model, we need to map each event in the log to an activity in the model. Each process instance in a log follows a trace (a sequence) of events and different instances may follow the same trace. Therefore, an event log can be seen as a multi-set of traces, i.e., $\mathcal{L} \in \mathbb{B}(A_L^*)$, where A_L is the set of activities in the log. Since *Declare* is an "open" language, it allows for the execution of any activity which is not in the model. Therefore, the set of

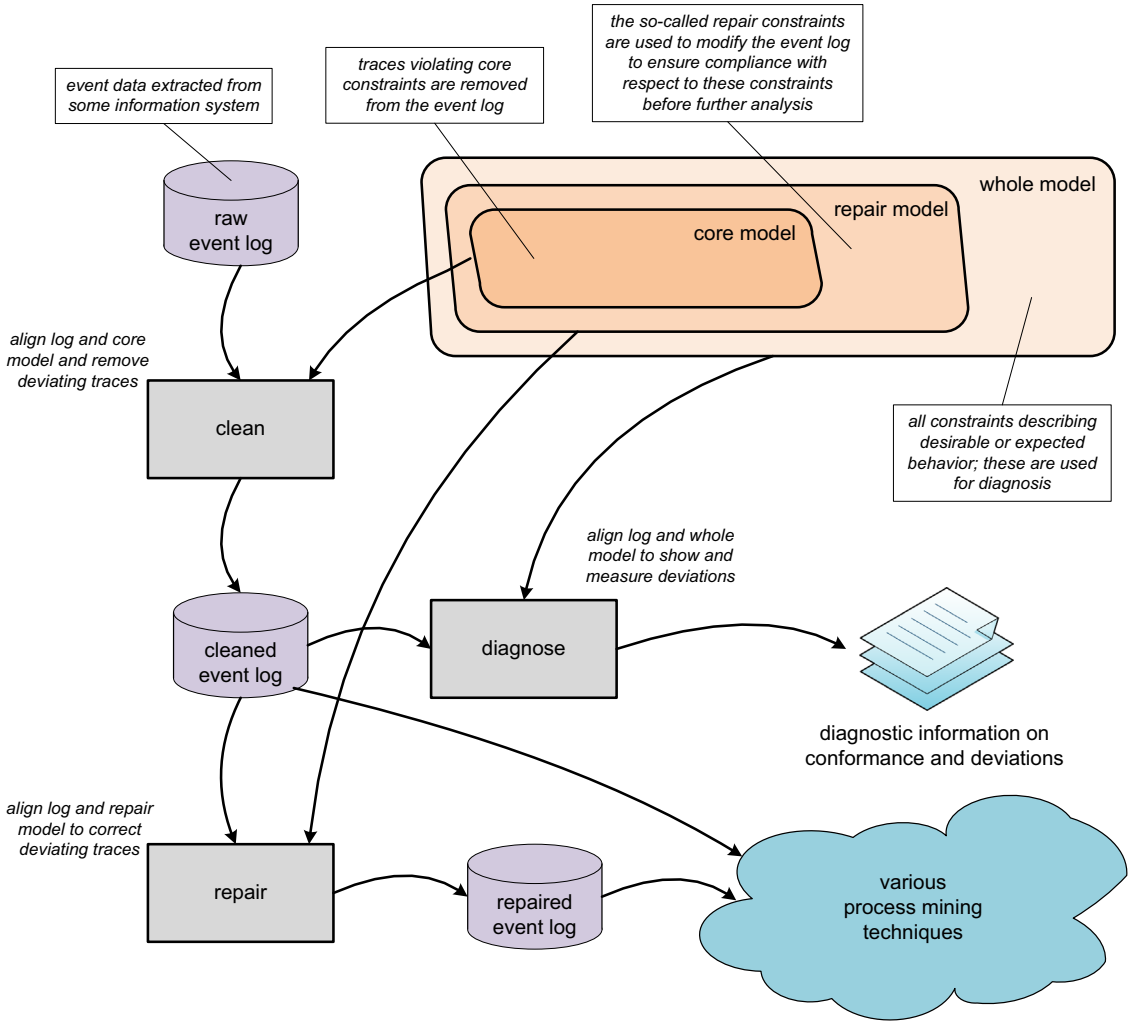

Fig. 2. Overview showing the *three main use cases* considered in this paper: (i) *cleaning event logs* to remove log traces that should not be used for further analysis, (ii) *diagnosing event logs* to check conformance, show deviations and measure compliance scores and (iii) *repairing event logs* to make sure that the essential constraints are satisfied before further analysis.

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005

activities in the log may include all activities in the model and more, i.e., $A \subseteq A_L$.

For conformance checking, we do not need to distinguish the activities in the set $A_L \backslash A$ as they do not appear in the model. This allows us to reduce the space of the allowed behaviors. Note that we cannot completely abstract from the activities in $A_L \backslash A$ because some constraints use LTL's next operator (e.g., the chain response and chain precedence constraints). Therefore, we map all events referring to activities in $A_L \backslash A$ to ✓. Then, the log $\mathcal{L} \in \mathbb{B}(A_L^*)$ is converted into $\mathcal{L}' \in \mathbb{B}(\Sigma^*)$ with $\Sigma = A_L \cup \{✓\}$. Function $\chi : A_L \to \Sigma$ is used to map each activity in A_L to Σ. $\forall a \in A$, $\chi(a) = a$ and $\forall a \in (A_L \backslash A)$, $\chi(a) = ✓$, i.e., every event in a log trace referring to an activity that is also defined in the model is mapped to the activity itself, whereas the other events are mapped to ✓.

Example 1 (cont.). The set of activities of the Declare model in Fig. 1 is

$A = \langle$Low Insurance Check, Low Medical History,
High Insurance Check, High Medical History,
Contact Hospital, Create Questionnaire,
Send Questionnaire, Receive Questionnaire Response$\rangle$

Let us assume to have a log that contains the following trace:

$\sigma_L = \langle$Register, Low Insurance Check, Create Questionnaire,
Prepare Notification Content, Create Questionnaire,
Send Notification by e−mail,
Send Notification by Post, Archive$\rangle$

Using the mapping function χ, σ_L can be transformed into

$\sigma_L{}' = \langle ✓,$ Low Insurance Check, Create Questionnaire, ✓,
Create Questionnaire, ✓, ✓, ✓$\rangle$

Note that Register, Send Notification by e-mail, Send Notification by Post, Archive are mapped to ✓.

In the remainder, when aligning *Declare* models and event logs, we only consider event logs after mapping unconstrained activities to ✓, i.e., $\mathcal{L} \in \mathbb{B}(\Sigma^*)$. To check whether a log trace $\sigma_L \in \mathcal{L}$ is compliant with a *Declare* constraint $\pi \in \Pi$, we use the technique described in [13] and translate the LTL formula corresponding to π into a finite state automaton that accepts all traces that do not violate π.

Definition 1 (*Constraint automaton*). Let $\mathcal{D} = (A, \Pi)$ be a Declare model, $\pi \in \Pi$ and $\Sigma = A \cup \{✓\}$. The *constraint automaton* $\mathcal{A}_\pi = (\Sigma, \Psi_\pi, \psi_{0_\pi}, \delta_\pi, F_\pi)$ is the finite state automaton that accepts exactly those traces $\sigma \in \Sigma^*$ satisfying π, where:

- $\Sigma = A \cup \{✓\}$ is the input alphabet;
- Ψ_π is a finite, non-empty set of states;
- $\psi_{0_\pi} \in \Psi_\pi$ is an initial state;
- $\delta_\pi : \Psi_\pi \times \Sigma \to \Psi_\pi$ is the state-transition function;
- $F_\pi \subseteq \Psi_\pi$ is the set of final states.

In the remainder, given a state-transition function δ and a sequence of symbols $\sigma = \langle s_1, \ldots, s_n \rangle$, $\delta^*(\psi_0, \sigma)$ denotes the recursive application of the state-transition function over σ starting from state ψ_0, i.e., $\delta^*(\psi_0, \sigma) = \psi_n$ where, for all $0 < i \leq n$, ψ_i is recursively defined as $\psi_i = \delta(\psi_{i-1}, s_i)$.

A constraint automaton $\mathcal{A}_\pi$ is associated with every constraint $\pi \in \Pi$. For each constraint automaton, we use a general convention according to which δ_π is allowed to be a partial function, i.e., $\delta_\pi(\psi, s)$ does not have to be defined for every combination of $\psi \in \Psi_\pi$ and $s \in \Sigma$. If a constraint automaton is in a state ψ, the next symbol is s and $\delta_\pi(\psi, s)$ is not defined, then the sequence is not accepted. A sequence $\sigma \in \Sigma^*$ is accepted by a constraint automaton $\mathcal{A}_\pi$, if $\delta_\pi^*(\psi_0, \sigma) \in F_\pi$, i.e., after reading the sequence σ, the automaton $\mathcal{A}_\pi$ is in a final state.

These automata can be used to check the conformance of a log trace with respect to each constraint in $\mathcal{D}$.

Example 1 (cont.). For the *co-existence* constraint and the *precedence* constraint in Fig. 1, we obtain the automata depicted in Fig. 3(a) and (b). In both cases, state 0 is the initial state and final states are indicated using a double outline. A transition is labeled with the set of the activities triggering it (we use the initial letters to denote an activity, e.g., we use *LIC* to indicate *Low Insurance Check*). This indicates that we can follow the transition for any event included in the set (e.g., we can execute event *High Insurance Check* from state 0 of the *precedence* automaton and remain in the same state).

The *process behavior set* $\mathcal{P}_\mathcal{D} \subseteq \Sigma^*$ of a *Declare* model $\mathcal{D} = (A, \Pi)$ is the set of traces that are accepted by all automata $\mathcal{A}_\pi$ with $\pi \in \Pi$, i.e., all process executions that comply with the model $\mathcal{D}$. We call these traces *model traces*.

Note that the cardinality of the process behavior set is infinite in most of the cases. For example, consider the

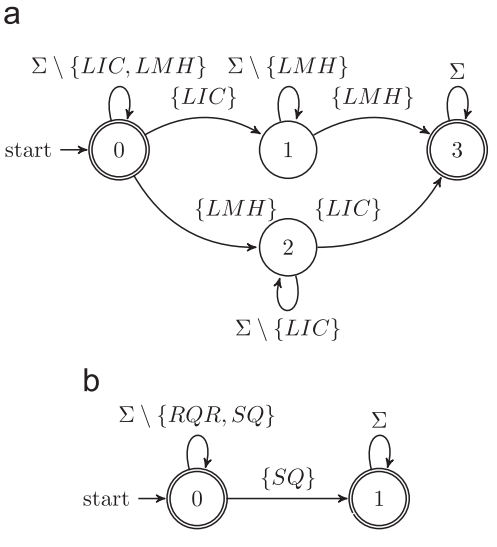

Fig. 3. Constraint automata for two *Declare* constraints in Fig. 1. (a) Constraint automaton for the co-existence constraint. (b) Constraint automaton for the precedence constraint.

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005

Declare model that only contains the *precedence* constraint in Fig. 3(b) and the activities shown in Fig. 2. The process behavior set of this model contains the traces accepted by the automaton in Fig. 3(b), i.e., the traces of the form $(\Sigma\backslash\{RQR, SQ\})^*\ SQ\ (\Sigma)^*$. This set has an infinite number of traces.

5. Alignment of event logs and Declare models

To check the conformance of an event log $\mathcal{L}$ with respect to a *Declare* model $\mathcal{D}$, we adopt an approach where we search for an *alignment* of the log and the model. An alignment relates *moves in log* and *moves in model* as explained in the following definition. Here, we explicitly indicate *no move* with $\gg$. We indicate set $\Sigma \cup \{\gg\}$ with $\Sigma_\gg$, where Σ denotes the input alphabet of each constraint automaton in $\mathcal{D}$.

Definition 2 (*Alignment and complete alignment*). A pair $(s', s'') \in (\Sigma_\gg \times \Sigma_\gg)\backslash\{(\gg, \gg)\}$ is

- a *move in log* if $s' \in \Sigma$ and $s'' = \gg$;
- a *move in model* if $s' = \gg$ and $s'' \in \Sigma$;
- a *move in both* if $s' \in \Sigma$, $s'' \in \Sigma$ and $s' = s''$.

Let $\Sigma_A = (\Sigma_\gg \times \Sigma_\gg)\backslash\{(\gg, \gg)\}$ be the set of the *legal moves*. The *alignment* of two traces $\sigma', \sigma'' \in \Sigma^*$ is a sequence $\gamma = \langle (s'_1, s''_1), \ldots, (s'_n, s''_n)\rangle \in \Sigma_A^*$ such that $\langle s'_1, \ldots, s'_n\rangle$ is σ' and $\langle s''_1, \ldots, s''_n\rangle$ is σ'' (ignoring $\gg$).

In particular, if $\sigma' = \sigma_L \in \mathcal{L}$ and $\sigma'' \in \mathcal{P}_\mathcal{D}$, we refer to the alignment γ as a *complete alignment* of σ_L and $\mathcal{D}$.

Given $\sigma_L \in \mathcal{L}$ and $\sigma_M \in \mathcal{P}_\mathcal{D}$, let us suppose to have an alignment $\gamma = \langle (s'_1, s''_1), \ldots, (s'_n, s''_n)\rangle \in \Sigma_A^*$ such that $\sigma'_L = \langle s'_1, \ldots, s'_n\rangle \in \text{prefix}(\sigma_L)$ and $\sigma'_M = \langle s''_1, \ldots, s''_n\rangle \in \text{prefix}(\sigma_L)$, ignoring $\gg$. We indicate σ'_L with $\gamma\#_L$ and σ'_M with $\gamma\#_M$.

An alignment of the event log $\mathcal{L}$ and the model $\mathcal{D}$ is a multi-set $\mathcal{A} \in \mathbb{B}(\Sigma_A^*)$ of alignments such that, for each log trace σ_L, there exists an alignment $\gamma \in \mathcal{A}$ of σ_L and $\mathcal{D}$. The definition of $\mathcal{A}$ as a multi-set is motivated by the fact that an event log may contain the same log trace σ_L multiple times and, hence, the same alignment can be associated with all its occurrences.

Example 1 (cont.). Given the log trace $\sigma_L = \langle \checkmark, LIC, CQ, \checkmark, CQ, \checkmark, \checkmark, \checkmark\rangle$, there are many possible complete alignments of σ_L and the model in Fig. 1. For example, the alignments $\gamma_1, \ldots, \gamma_4$ in Fig. 4 are examples of complete alignments. Conversely, alignment γ_0 is not a complete alignment since $\gamma\#_M$ is not in the process behavior set of the model. Indeed, the *co-existence* constraint is violated, because *Low Insurance Check* occurs in $\gamma\#_M$ but *Low Medical History* does not. Moreover, two occurrences of *Create Questionnaire* are not followed by any occurrence of *Send Questionnaire*, as the *response* constraint would prescribe.

In order to quantify the severity of a deviation, we introduce a cost function on the legal moves $\kappa : \Sigma_A \to \mathbb{R}_0^+$. One can use a standard cost function with unit costs for moves in log or in model. However, the costs may also depend on the specific characteristics of the process, e.g., it may be more costly to skip an insurance check for high claims than for low claims. Therefore, in some cases, a different cost function κ needs to be defined. The cost of an alignment γ is defined as the sum of the costs of the individual moves in the alignment, $\mathcal{K}(\gamma) = \sum_{(s', s'') \in \gamma} \kappa(s', s'')$.

γ_0 =

L:	✓	LIC	CQ	✓	CQ	✓	✓	✓
M:	✓	LIC	CQ	✓	CQ	✓	✓	✓

γ_1 =

L:	✓	LIC	CQ	✓	CQ	$\gg$	✓	✓	✓
M:	✓	$\gg$	CQ	✓	CQ	SQ	✓	✓	✓

γ_2 =

L:	✓	LIC	$\gg$	CQ	✓	CQ	✓	✓	✓
M:	✓	LIC	LMH	$\gg$	✓	$\gg$	✓	✓	✓

γ_3 =

L:	✓	LIC	$\gg$	CQ	✓	CQ	$\gg$	✓	✓	✓
M:	✓	LIC	LMH	CQ	✓	CQ	SQ	✓	✓	✓

γ_4 =

L:	✓	LIC	CQ	✓	CQ	✓	✓	✓
M:	✓	$\gg$	$\gg$	✓	$\gg$	✓	✓	✓

Fig. 4. Some examples of alignments of σ_L and the model in Fig. 1. Apart from γ_0, they are all complete alignments.

Given a log trace $\sigma_L \in \mathcal{L}$, our goal is to find a complete alignment of σ_L and a model trace $\sigma_M \in \mathcal{P}_\mathcal{D}$ that minimizes the cost with respect to all $\sigma'_M \in \mathcal{P}_\mathcal{D}$. This complete alignment is referred to as an *optimal alignment*.

Definition 3 (*Optimal alignment*). Let $\sigma_L \in \mathcal{L}$ be a log trace and $\mathcal{D}$ a Declare model. Let $\Gamma_{(\sigma_L, \mathcal{D})}$ be a set of the complete alignments of σ_L and $\mathcal{D}$. A complete alignment $\gamma \in \Gamma_{(\sigma_L, \mathcal{D})}$ is an *optimal alignment* of σ_L and $\mathcal{D}$ if and only if $\forall \gamma' \in \Gamma_{(\sigma_L, \mathcal{D})}.\ \mathcal{K}(\gamma') \geq \mathcal{K}(\gamma)$.

Example 1 (cont.). In our example, we can suppose that deviations for activity *Send Questionnaire* are less severe than those referring to the other activities, since this activity is automatically performed by a system. Moreover, moves associated with ✓ can be weighted less than any other, since they refer to activities that are not in the model. Therefore, a reasonable cost function on legal moves can be defined as follows:

$$\kappa(a', a'') = \begin{cases} 0 & \text{if } a' = a'' \\ 1 & \text{if } a' = \checkmark \wedge a'' = \gg \\ 1 & \text{if } a' = \gg \wedge\ a'' = \checkmark \\ 2 & \text{if } a' = \textit{Send Questionnaire} \wedge a'' = \gg \\ 2 & \text{if } a' = \gg \wedge\ a'' = \textit{Send Questionnaire} \\ 4 & \text{if } a' \neq \textit{Send Questionnaire} \wedge a'' = \gg \\ 4 & \text{if } a' = \gg \wedge\ a'' \neq \textit{Send Questionnaire} \\ \infty & \text{otherwise} \end{cases}$$

Using this cost function, alignments γ_1, γ_2, γ_3 and γ_4 have the following costs: $\mathcal{K}(\gamma_1) = 6$, $\mathcal{K}(\gamma_2) = 12$, $\mathcal{K}(\gamma_3) = 6$ and $\mathcal{K}(\gamma_4) = 12$. Then, γ_1 and γ_3 are better complete alignments; in fact, they are optimal alignments, as well.

In Section 6, we discuss a technique to create an optimal alignment with respect to a custom cost function κ. The approach is based on the A* algorithm that is intended to find the path with the lowest overall cost between two nodes in a direct graph with costs associated with nodes.

In our alignment-based technique, *activities are promoted as first-class citizens*, whereas the literature proposes approaches that focus on constraint violations. In

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005

several scenarios, companies may be interested in knowing which actual executions or missing executions of activities have triggered the deviations. Section 5.1 discusses how the application of existing techniques would tend to overestimate the number of activities whose execution deviates from the prescribed behavior.

The provision of a set of alignments does not allow process analysts to gain a quick insight into the most common deviations. Therefore, we also provide a summary to determine which activities are mostly involved in deviations and which constraints are more often violated. On this concern, we provide a metrics to measure the *degree of conformance* of single activities and constraints in a *Declare* model with respect to a log.

In order to measure the degree of conformance, we first need to compute which constraints each move in model or in log contributes to solve, i.e., why a certain move in model/log is needed. Section 5.2 illustrates how to identify which moves are needed to solve constraint deviations and Section 5.3 details how to measure the degree of conformance.

5.1. Promoting activities as first-class citizens: why existing techniques tend to overestimate the activities involved in deviations?

There is a body of techniques that check (running) process instances based on some temporal logic, including LTL (e.g., [14–16]). These approaches classify process instances as deviating or not and diagnose the constraints that cause deviations. Unfortunately, these approaches tend to overestimate the activities whose execution deviates from the prescribed behavior. Therefore, when aggregated diagnostics are shown to the user, activities tend to be considered as involved in a large number of deviations when executed, far more than the real number of deviations. This due to the fact that different constraints and activities may interact in various ways. That motivates why optimal alignments are necessary.

Consider, for example, the *Declare* model in Fig. 5 and trace $\sigma = \langle a, b, c \rangle$. According to the model, the not-coexistence constraints prescribe that activities a and b cannot occur in the same trace, like so activities c and b. Existing techniques would diagnose that, indeed, the two constraints are violated during the execution of σ. Therefore, they would derive that the executions of the three activities are deviating from the prescribed behavior. Nonetheless, our technique would return the following optimal alignment:

L:	a	b	c
M:	a	$\gg$	c.

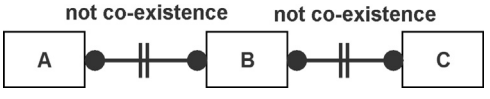

Fig. 5. An example to illustrate the limitation of existing techniques: the number of activities that are involved in deviated executions is overestimated.

This shows that, in fact, only the execution of activity b is deviating. Therefore, generally speaking, if an optimal alignment is not built, e.g., through the technique proposed in this paper, the diagnosis will report several deviations which, actually, are not so.

5.2. Why a log or model move is needed?

Let $\gamma = \langle (a_1^L, a_1^M), \ldots, (a_n^L, a_n^M) \rangle$ be an optimal alignment of σ_L and $\mathcal{D} = (A, \Pi)$. Let $\mathcal{A}_\pi$ be the constraint automaton for a constraint $\pi \in \Pi$. Each move $(a_i^L, a_i^M) \in \gamma$ in log or in model of an alignment (i.e., such that either $a_i^L = \gg$ or $a_i^M = \gg$), is introduced to contribute to solve violation(s) of constraint (s) in the *Declare* model. For diagnosis purpose, it is important to identify the violation(s) that moves contribute to solve (every move in log or in model always solves one violation, at least). For this purpose, a trace σ_i is constructed by removing every $\gg$ occurrence from the trace $\langle a_1^M, \ldots, a_{i-1}^M, a_i^L, a_{i+1}^M, \ldots, a_n^M \rangle$. Then, for each constraint $\pi \in \Pi$, we check whether σ_i is accepted by $\mathcal{A}_\pi$. If it is not accepted, (a_i^L, a_i^M) has been introduced to solve a violation in π.

Example 1 (cont.). Let us consider again the optimal alignment γ_1 (see Fig. 4). It contains two moves in log or in model: $(LIC, \gg)$ and $(\gg, SQ)$. For $(LIC, \gg)$, we build the trace $\sigma_1 = \langle \checkmark, LIC, CQ, \checkmark, CQ, \checkmark, \checkmark, \checkmark \rangle$. This sequence is accepted by all the constraint automata in the Declare model, besides the constraint automaton for the *co-existence* constraint (see Fig. 3(a)). Similarly, for $(\gg, SQ)$, $\sigma_5 = \langle \checkmark, CQ, \checkmark, CQ, \checkmark, \checkmark, \checkmark \rangle$ is accepted by all the constraint automata, besides the constraint automaton for the *precedence* constraint (shown in Fig. 3(b)). Therefore, $(LIC, \gg)$ has been introduced to solve a violation in the *co-existence* constraint and $(\gg, SQ)$ has been introduced to solve a violation in the *precedence* constraint.

5.3. Degree of conformance

In order to quantify the degree of conformance, we introduce metrics $MM_\gamma(a)$, $ML_\gamma(a)$ and $MB_\gamma(a)$. For $a \in \Sigma$, we denote with $MM_\gamma(a)$ the number of moves in the model involving a, with $ML_\gamma(a)$ being the number of moves in log involving a and with $MB_\gamma(a)$ the number of moves in both model and log involving a.

For reliability, we average over all optimal alignments. Let $\Gamma = (\{\gamma_1, \ldots, \gamma_n\}, \Lambda)$ be a multi-set of optimal alignments of a log $\mathcal{L}$ and a model $\mathcal{D}$. The *degree of conformance of an activity* $a \in \Sigma$ with respect to Γ is defined as follows:

$$\mathrm{DConf}_\Gamma(a) = 1 - \frac{1}{n} \cdot \sum_{\gamma \in \Gamma} \Lambda(\gamma) \frac{MM_\gamma(a) + ML_\gamma(a)}{MM_\gamma(a) + ML_\gamma(a) + MB_\gamma(a)}.$$

$\mathrm{DConf}_\Gamma(a) = 1$, if the moves that involve a are only moves in both (i.e., there are no deviations related to a). $\mathrm{DConf}_\Gamma(a)$ decreases when the fraction of moves in model or in log involving a increases. $\mathrm{DConf}_\Gamma(a) = 0$, if all moves that involve a are only moves in the log or moves in the model.

In the remainder, given an alignment γ and a constraint π, we denote with $MC_\gamma(\pi)$ the metric representing the number of moves in the model and in the log in γ that

contributes to solve a violation of π. The moves that contribute to solve a violation of a constraint are computed according to the technique described in Section 5.2. The *degree of conformance of a constraint* $\pi \in \Pi$ with respect to Γ is defined as follows:

$$\mathrm{DConf}_{\Gamma}(\pi) = 1 - \frac{1}{n} \cdot \sum_{\gamma \in \Gamma} \Lambda(\gamma) \frac{MC_{\gamma}(\pi)}{|\gamma|}.$$

$\mathrm{DConf}_{\Gamma}(\pi) = 1$, if π is never violated. $\mathrm{DConf}_{\Gamma}(\pi)$ gets closer to 0 when the fraction of moves in the model or in the log needed to solve violations of π increases.

6. The A* algorithm for computing log–model alignments

Let us suppose to have a graph V with costs associated with arcs. The A* algorithm, initially proposed in [4], is a pathfinding search in V. It starts at a given *source* node $v_0 \in V$ and explores adjacent nodes until one node of a given *target set* $V_{Trg} \subset V$ is reached, aiming at finding the path with the overall lowest cost.

Every node $v \in V$ is also associated with a cost, which is determined by an *evaluation* function $f(v) = g(v) + h(v)$, where

- $g : V \to \mathbb{R}_0^+$ is a function that returns the smallest path cost from v_0 to v;
- $h : V \to \mathbb{R}_0^+$ is a heuristic function that estimates the smallest path cost from v to any target node $v' \in V_{Trg}$.

Function h is said to be *admissible*, if it never overestimates the smallest path cost to reach any target node: for each node $v \in V$ and for each target node $v' \in V_{Trg}$ reachable from v, $h(v) \leq g(v')$. Technical results in [4] show that if h is admissible, A* finds a path that is guaranteed to have the overall lowest cost.

The A* algorithm keeps a priority queue of nodes to be visited: nodes with lower costs are assigned higher priority, thus guaranteeing that these are going to be visited earlier. The algorithm works iteratively: at each step, the node v with lowest cost is taken from the priority queue. If v belongs to the target set, the algorithm ends returning v. Otherwise, v is expanded: every successor v' of v is added to the priority queue with cost $f(v')$.

We use A* to find any of the optimal alignments of a log trace $\sigma_L \in \Sigma^*$ and a *Declare* model $\mathcal{D} = (A, \Pi)$. In order to be able to apply A*, an opportune search space V needs to be defined.

The search space for finding an optimal alignment between σ_L and $\mathcal{D}$ is a directed graph (V,E) where

- V contains prefixes of some complete alignment of σ_L and $\mathcal{D}$: $V = \{\gamma \in \Sigma_A^* : \gamma\#_L \in \mathrm{prefix}(\sigma_L)$ and $\exists \sigma_M \in \mathcal{P}_{\mathcal{D}}, \gamma\#_M \in \mathrm{prefix}(\sigma_M)\}$;
- E contains all edges $(\gamma', \gamma'') \in V \times V$, where γ'' is obtained concatenating a legal move to γ': $E = \{(\gamma', \gamma'') \in V \times V \ : \exists (s', s'') \in \Sigma_A$ s.t. $\gamma'' = \gamma' \oplus (s', s'')\}$.

The set of target nodes $V_{trg} \subset V$ contains all complete alignments of σ_L and $\mathcal{D}$.

It is easy to see that, if the directions on the edges are ignored, the path from a certain $\gamma' \in V$ to a certain $\gamma'' \in V$ never contains the same node twice and, therefore, the search space is a directed tree. The root of the tree is the empty alignment $\gamma_0 = \langle\rangle$, which is the source node to start the application of the A* algorithm from. Since a different alignment is also associated with every node and vice versa, later on, we use the alignment to refer to the associated node.

Let us consider a custom cost function κ and denote with $\kappa^{\min}$ the smallest value greater than 0 in the codomain of κ, i.e., $\kappa^{\min} = \min_{(s',s'') \in \Sigma_A^* : \kappa(s',s'') \gg 0} \kappa(s', s'')$.

We define the cost of a path from the initial node to $\gamma \in V$ as

$$g(\gamma) = \kappa^{\min} \cdot |\gamma\#_L| + \mathcal{K}(\gamma),$$

This definition of $g(\gamma)$ guarantees that the optimal solution computed by the A* algorithm is an optimal alignment.

Theorem 1. *Let (V,E) be the search space of the A* algorithm to find an optimal alignment between a log trace σ_L and a Declare model $\mathcal{D}$. Let $V_{Trg} \subseteq V$ be the set of the target nodes of the search space, i.e., the set of all complete alignments of σ_L and $\mathcal{D}$. If $\gamma_O \in V_{Trg}$ is an optimal alignment, then, for each $\gamma \in V_{Trg}$, $g(\gamma) \geq g(\gamma_O)$.*

Proof. Let us consider any complete alignment $\gamma' \in V_{Trg} \backslash \{\gamma_O\}$. Let us suppose by contradiction that $g(\gamma') < g(\gamma_O)$. It holds that $g(\gamma') = \kappa^{\min} \cdot |\gamma'\#_L| + \mathcal{K}(\gamma')$ and $g(\gamma_O) = \kappa^{\min} \cdot |\gamma_O\#_L| + \mathcal{K}(\gamma_O)$. Since γ' and γ_O are complete alignments, $\gamma'\#_L = \gamma_O\#_L = \sigma_L$. Therefore, $\kappa^{\min} \cdot |\gamma'\#_L| + \mathcal{K}(\gamma') < \kappa^{\min} \cdot |\gamma_O\#_L| + \mathcal{K}(\gamma_O)$, i.e., $\mathcal{K}(\gamma') < \mathcal{K}(\gamma_O)$. Therefore, it is not true that γ_O is an optimal alignment. □

The application of the A* algorithm to find optimal alignments employs the following heuristics:

$$h(\gamma) = \kappa^{\min} \cdot (|\sigma_L| - |\gamma\#_L|)$$

As the next theorem shows, this heuristics is admissible and, hence, guarantees the optimality of the solution computed by the A* algorithm.

Theorem 2. *Let (V,E) be the search space of the A* algorithm to find an optimal alignment between a log trace σ_L and a Declare model $\mathcal{D}$. Let $V_{Trg} \subseteq V$ be the set of the target nodes of the search space. The heuristic $h(\gamma) = \kappa^{\min} \cdot (|\sigma_L| - |\gamma\#_L|)$ is admissible, i.e., for each $\gamma' \in V_{Trg}$ such that $\gamma \in \mathrm{prefix}(\gamma')$, $h(\gamma) \leq g(\gamma')$.*

Proof. Let $\gamma \in V$ be an alignment. If $\gamma \in V_{Trg}$, $|\sigma_L| = |\gamma\#_L|$ and, hence, $h(\gamma) = 0 \leq g(\gamma)$ is trivially true. If $\gamma \notin V_{Trg}$, let us take any $\gamma' \in V_{Trg}$ such that $\gamma \in \mathrm{prefix}(\gamma')$. Let us assume $g(\gamma') < h(\gamma)$ by contradiction. Therefore, $g(\gamma') = \kappa^{\min} \cdot |\sigma_L| + \mathcal{K}(\gamma') < h(\gamma) = \kappa^{\min} \cdot (|\sigma_L| - |\gamma\#_L|)$. This implies that $\mathcal{K}(\gamma') < -\kappa^{\min} \cdot |\gamma\#_L| < 0$, which is not possible. Hence, $g(\gamma') < h(\gamma)$ cannot be true since $\mathcal{K}(\cdot)$ can never be negative (it is a sum of non-negative costs). □

The set of any *Declare* process behavior is infinite and, hence, the set of complete alignments for a given log trace is also infinite, like so the search space. In principle, the same may also hold for procedural models (e.g., in the presence of cycles). Nonetheless, *Declare* models allow for more flexibility and, hence, a prefix of a model trace can be extended with a very large number of activities, if compared with procedural

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005

models. This implies that every search-space node has a very large number of successors, far more than for computing the alignment of a trace and a procedural model.

Therefore, it is important to prune the search-space tree so as not to visit nodes that are equivalent to nodes which have already been visited. We say that two alignments (i.e., search-space nodes) are equivalent, if they can be extended with the same moves. Space limitations prevent us from giving further details on the technique we have devised for search-space pruning. We refer interested readers to [17] for further information on it.

7. Conformance checking using alignments

There are four basic quality dimensions for checking the conformance of event logs with respect to process models: (a) *fitness*, (b) *precision*, (c) *generalization* and (d) *simplicity* [18]. A model with good *fitness* allows for most of the behavior seen in the event log. A model has a perfect fitness if all traces in the log can be replayed by the model from the beginning to the end.

A model is *precise* if it does not allow for "too much" behavior. A model that is not precise is "underfitting", i.e., it is too generic.

A model that does not *generalize* is "overfitting". An overfitting model is too specific, i.e., it explains a particular sample log but it is unlikely that another sample log of the same process can be explained well by the same model.

The fourth quality dimension, *simplicity*, is related to Occam's Razor which states that "one should not increase, beyond what is necessary, the number of entities required to explain anything". Following this principle, we look for the "simplest process model" that can explain what is observed in the event log. In this paper, we do not take the simplicity dimension into account and we characterize the other three dimensions.

Although these quality dimensions were originally devised for procedural models, they are also important in the declarative case. It is also worth highlighting that the precision and generalization dimensions of conformance do not hinder the nature of declarative models. Also for declarative models, a process analyst can be interested to verify whether the model is too precise, i.e., the model's constraints limit too much the allowed process behavior, or, conversely, the model is too general, i.e., it has too few constraints and, hence, too much behavior is allowed. The following example discusses the importance of precision and generalization for declarative models.

Example 2. Let us suppose to have the Declare model $\mathcal{D}_a$ in Fig. 6(a). By executing this model, we can generate an event log $\tilde{\mathcal{L}}$ with the following 5 traces: $\langle a,b,c,d\rangle$, $\langle a,b,a,b,c,d\rangle$, $\langle a,b,c,c,d\rangle$, $\langle c,c,a,b,d\rangle$, $\langle c,a,b,d\rangle$. Using the Declare miner [19] with certain settings, we can generate the models $\mathcal{D}_b$ and $\mathcal{D}_c$ in Fig. 6(b) and (c) from $\tilde{\mathcal{L}}$. These models have a perfect fitness with respect to $\tilde{\mathcal{L}}$. Nonetheless, $\mathcal{D}_b$ contains no constraints and its precision is lower than the precision of $\mathcal{D}_a$. Conversely, the generalization of $\mathcal{D}_c$ is lower than the generalization of $\mathcal{D}_a$, since an additional constraint is attached to D, which prescribes D to be executed exactly once. Moreover, the *response* constraint between A and B in $\mathcal{D}_a$ becomes a *chain response* in $\mathcal{D}_c$. This means that A must be immediately followed by B. However, the reason why $\mathcal{D}_c$ contains these constraints is related to the fact that, only by coincidence, in all traces of $\tilde{\mathcal{L}}$, A is always immediately followed by B and D always occurs only once.

Clearly, the model $\mathcal{D}_a$ of Example 2 is the "ideal model" that the Declare miner should discover. Unfortunately, this model is not known since only the behaviors recorded in the event logs are observed: unfortunately, the event log with 5 traces is not *complete*, i.e., it does not contain examples of all possible behavior. As a consequence, it is highly probable that any discovered model will score low for some quality dimensions of conformance (e.g., $\mathcal{D}_b$ and $\mathcal{D}_c$ have low precision and generalization respectively). However, one wants to have an estimation of how much a mined model moves away from the ideal one. This motivates why the four dimensions of conformance are important. It is out of scope of this paper to have a thorough discussion about the comparison between the ideal model and the mined models with respect to the conformance dimensions. For further details about this, we refer the reader to [20].

7.1. The fitness dimension: are the observed behaviors allowed by the model?

When focusing on the fitness dimension of conformance, we are interested in quantifying the fitness score of a log trace σ_L with respect to a *Declare* model $\mathcal{D}$. Therefore, we introduce a *fitness function* $\textsc{Fitness}(\sigma_L,\mathcal{D})$, which returns a value between 0 and 1. $\textsc{Fitness}(\sigma_L,\mathcal{D})=1$ if there is an (optimal) alignment between σ_L and $\mathcal{D}$ with cost 0, i.e., σ_L is perfectly compliant with $\mathcal{D}$. Conversely, $\textsc{Fitness}(\sigma_L,\mathcal{D})=0$ denotes a very poor fitness.

Cost function $\mathcal{K}(\cdot)$, introduced in Section 5, is a good candidate but it cannot be used as it is, since the score needs to be a number between 0 and 1. The normalization

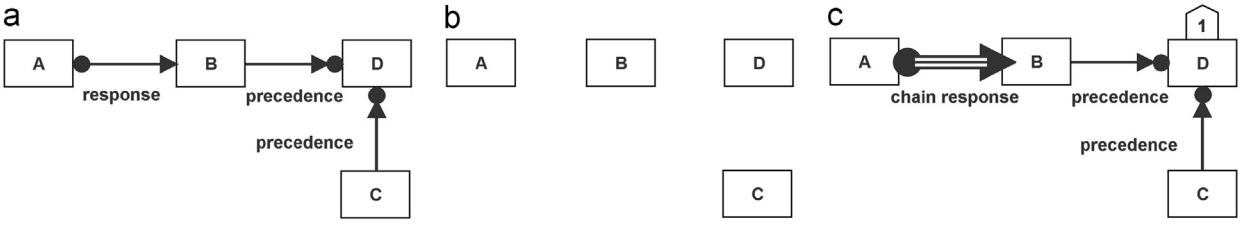

Fig. 6. Three alternative *Declare* models derived from the same event log. Model (a) adequately balances the precision and generalization dimensions of conformance, whereas (b) is not a precise *Declare* model because it contains no constraints and, hence, too much behavior is allowed. Finally, model (c) does not generalize sufficiently: the number of constraints and their type disallow a lot of behavior that is potentially admissible.

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005

between 0 and 1 can be done in several ways. In our approach, we divide the cost of the optimal alignment by the maximal alignment cost. Typically, the greatest cost of an alignment of a log trace $\sigma_L = \langle a_1^L, \ldots, a_n^L \rangle$ and a model trace $\sigma_M = \langle a_1^M, \ldots, a_m^M \rangle \in \mathcal{P}_\mathcal{D}$ is obtained for the *reference alignment*, i.e., the alignment in which there are only moves in the model and moves in the log:

$$\gamma^{\text{ref}}_{(\sigma_L,\sigma_M)} = \begin{array}{|c|c|c|c|c|c|c|} \hline \textbf{L:} & a_1^L & \ldots & a_n^L & \gg & \gg & \gg \\ \hline \textbf{M:} & \gg & \gg & \gg & a_1^M & \ldots & a_m^M \\ \hline \end{array}.$$

Definition 4 (*Fitness*). Let $\mathcal{D} = (A, \Pi)$ be a Declare model and σ_L be a log trace. Let $\gamma \in \Sigma_A^*$ be an optimal alignment of σ_L and $\mathcal{D}$. Let $\sigma_M \gamma \#_M$ be the aligned model trace. Let $\gamma^{\text{ref}}_{(\sigma_L,\sigma_M)} \in \Sigma_A^*$ be the reference alignment of σ_L and $\mathcal{D}$. The *fitness* score of σ_L with respect to $\mathcal{D}$ is defined as follows:

$$\textsc{Fitness}(\sigma_L, \mathcal{D}) = 1 - \frac{\mathcal{K}(\gamma_{\sigma_L})}{\mathcal{K}(\gamma^{\text{ref}}_{(\sigma_L,\sigma_M)})}.$$

Therefore, $\textsc{Fitness}(\sigma_L, \mathcal{D}) = 1$, if in the alignment there are only moves in both, i.e., there are no deviations. $\textsc{Fitness}(\sigma_L, \mathcal{D}) = 0$, if the optimal alignment only contains moves in the log or in the model. Note that the above-defined fitness always scores between 0 and 1. Indeed, if $\gamma_O \in \Sigma_A^*$ is an optimal alignment, any other alignment, including $\gamma^{\text{ref}}_{(\sigma_L,\sigma_M)}$, must have the same or a higher cost.

Example 1 (cont.). The optimal alignment γ_1 is associated with a cost 3, whereas the cost of the relative reference alignment is 32. Therefore, $\textsc{Fitness}(\sigma_L, \mathcal{D}) = 1 - 3/32 = 0.90625$.

7.2. Precision: does the model allow for too much behavior?

In [21], a technique is proposed to compute the precision of procedural model. Here, this technique is adapted and extended to deal with declarative models. The technique proposed relies on the construction of the so-called *alignment automaton*.

Definition 5 (*Alignment automaton*). Let $\mathcal{L}$ and $\mathcal{D} = (A, \Pi)$ be an event log and a Declare model respectively. Let $\Gamma = (\{\gamma_1, \ldots, \gamma_n\}, \Lambda_\Gamma) \in \mathbb{B}(\Sigma_A^*)$ be the multi-set of optimal alignments of $\mathcal{L}$ and $\mathcal{D}$. Let P be the multi-set $(\{\sigma_1, \ldots, \sigma_n\}, \Lambda_P) \in \mathbb{B}(A \cup \checkmark)$ of "aligned" model trace, i.e., for each $1 \le i \le n$, $\sigma_i = \gamma_i \#_M$ and $\Lambda_P(\sigma_i) = \Lambda_\Gamma(\gamma_i)$.

The *alignment automaton* is a finite state automaton $(\Sigma, S, \sigma_0, \delta, F)$, where

- $\Sigma = A \cup \{\checkmark\}$ is the alphabet of the automaton;
- $S \in \Sigma^*$ is the set of all prefixes of sequences in P, i.e., $S = \{\sigma : \exists \sigma' \in P,\ \sigma \in \text{prefix}(\sigma')\}$;
- σ_0 is the initial state, which is the empty sequence;
- $\delta : S \times \Sigma \to S$ is a function such that, if $\delta \oplus \langle s \rangle \in S$, $\delta(\sigma, s)$ is defined and equal to $\delta \oplus \langle s \rangle$;
- $F \subseteq S$ is the set of final states. A prefix $\sigma \in S$ belongs to F if there exists $s \in \Sigma$ for which $\delta(\sigma, s)$ is defined.

Moreover, each alignment automaton is associated with a function $\Upsilon : S \to \mathbb{N}$. For each $\sigma \in S$, $\Upsilon(\sigma)$ returns the number of sequences in multi-set P that have σ as prefix: $\Upsilon(\sigma) = \sum_{\sigma_P \in P : \sigma \in \text{prefix}(\sigma_P)} \Lambda_P(\sigma_P)$.

Once the alignment automaton is built, the precision can be scored as follows.

Definition 6 (*Precision*). Let $(\Sigma, S, \sigma_0, \delta, F)$ be the alignment automaton for a Declare model $\mathcal{D}$ and an event log $\mathcal{L}$. For each activity sequence $\sigma \in S$, let $av_\mathcal{D}(\sigma)$ be the set of all activities that are enabled for execution: $av_\mathcal{D}(\sigma) = \{s \in \Sigma : \sigma \oplus \langle s \rangle \in \mathcal{P}_\mathcal{D}\}$. Let $ex_\mathcal{L}(\sigma) \subseteq av_\mathcal{D}(\sigma)$ be the set of all activities that have been executed after executing σ, $ex_\mathcal{L}(\sigma) = \{s \in \Sigma : \delta(\sigma, s) \text{ is defined}\}$. The *precision* score of $\mathcal{D}$ with respect to $\mathcal{L}$ is defined as follows:

$$\textsc{Precision}(\mathcal{D}, \mathcal{L}) = \frac{\sum_{\sigma \in S \setminus F} (\Lambda_P(\sigma) \cdot |ex_\mathcal{L}(\sigma)|)}{\sum_{\sigma \in S \setminus F} (\Lambda_P(\sigma) \cdot |av_\mathcal{D}(\sigma)|)}.$$

The precision is always a value between 0 and 1. If all behavior allowed by the model $\mathcal{D}$ is actually observed in the event log $\mathcal{L}$, then $\textsc{Precision}(\mathcal{D}, \mathcal{L}) = 1$. In general, a precision of $x/100$ identifies a situation in which $x\%$ of possible behaviors are observed. Therefore, a model that allows for too much behavior has a precision score close to 0.

Example 2 (cont.). Fig. 7 shows the alignment automaton for the four traces in $\tilde{\mathcal{L}}$. Since all log traces are perfectly fitting, $P = \tilde{\mathcal{L}}$. Let us consider two states of the alignment automaton: $\tilde{\sigma_1} = \langle\rangle$ and $\tilde{\sigma_2} = \langle a, b \rangle$. According to the alignment automaton, as first activity (i.e., after the empty sequence $\tilde{\sigma_1}$), activities a and c have been performed, i.e., $|ex_{\tilde{\mathcal{L}}}(\tilde{\sigma_1})| = 2$. Activities a and c have also been performed after sequence $\tilde{\sigma_2}$, i.e., $|ex_{\tilde{\mathcal{L}}}(\tilde{\sigma_1})| = 2$. Let us consider the three Declare models $\mathcal{D}_a$, $\mathcal{D}_b$ and $\mathcal{D}_c$ in Fig. 6. After $\tilde{\sigma_1}$, models $\mathcal{D}_a$ and $\mathcal{D}_c$ allow for activities a, b, c as well as any other activity not in the model, i.e., $\checkmark$. In addition, model $\mathcal{D}_b$ also allows for d. It follows that $|av_{\mathcal{D}_a}(\sigma_1)| = 4$, $|av_{\mathcal{D}_b}(\sigma_1)| = 5$ and $|av_{\mathcal{D}_c}(\sigma_1)| = 4$. Similarly, $|av_{\mathcal{D}_a}(\sigma_2)| = 4$, $|av_{\mathcal{D}_b}(\sigma_2)| = 5$ and $|av_{\mathcal{D}_c}(\sigma_2)| = 4$. To measure the precision of each $\mathcal{D} \in \{\mathcal{D}_a, \mathcal{D}_b, \mathcal{D}_c\}$, $|ex_{\tilde{\mathcal{L}}}(\sigma)|$ and $|av_\mathcal{D}(\sigma)|$ need to be computed for each state σ of the alignment automaton. For this example, we have the following scores: $\textsc{Precision}(\mathcal{D}_a, \tilde{\mathcal{L}}) \simeq 0.19$, $\textsc{Precision}(\mathcal{D}_b, \tilde{\mathcal{L}}) \simeq 0.17$ and $\textsc{Precision}(\mathcal{D}_c, \tilde{\mathcal{L}}) \simeq 0.21$. As expected, models $\mathcal{D}_b$ and $\mathcal{D}_c$ are the least and the most precise respectively. We will show that, however, the generalization of $\mathcal{D}_c$ is low. The precision score is relatively low for all the three models in Fig. 6.

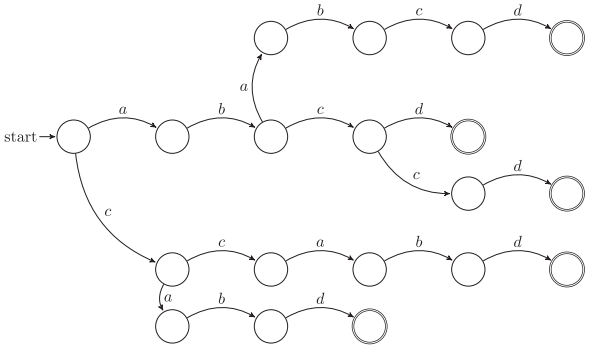

Fig. 7. The alignment automaton for traces in Example 2.

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005

Precision of a model is defined with respect to a given event log. If the event log is sufficiently large (e.g., spanning over a period of multiple years), due to the *strong law of large numbers*, the event log is likely being complete. Clearly, log completeness is related to precision: if the event log is complete, all possible behavior is observed and, hence, the precision score is reliable.

The precision score of many *Declare* models can be relatively low, since *Declare* models allow for high flexibility. However, the absolute score of precision of a model is only partly interesting. In many situations, one is interested to compare multiple models (e.g., the models mined from a given event log) to choose the one with the highest precision.

7.3. Generalization: does the model allow for too little behavior?

It is difficult to reason about generalization because it refers to unseen behavior. Our approach is based on probability estimators, similar to what has been proposed in [1] for procedural models.

We again exploit the alignment automaton $(\Sigma, S, \sigma_0, \delta, F)$. Let $\mathcal{D} = (A, \Pi)$ be a *Declare* model. For each $\pi \in \Pi$, let $\mathcal{A}_\pi$ be the constraint automaton for π. Let $\mathcal{A}_\mathcal{D} = (\Sigma, \Psi_\mathcal{D}, \psi_{0\mathcal{D}}, \delta_\mathcal{D}, F_\mathcal{D})$ be the so-called *global* automaton obtained through the cartesian (intersection) of every automaton $\mathcal{A}_\pi$ with $\pi \in \Pi$.

The set of non-final states $X = S \backslash F$ can be partitioned in equivalent classes using an equivalence relation $\sim_\mathcal{D}$ such that, for each $\sigma', \sigma'' \in X$, $\sigma' \sim_\mathcal{D} \sigma''$ iff $\delta^*_\mathcal{D}(\psi_{0\mathcal{D}}, \sigma') = \delta^*_\mathcal{D}(\psi_{0\mathcal{D}}, \sigma'')$.[4] In the remainder, the set of all equivalence classes in X is denoted as follows: $X/\sim_\mathcal{D} = \{\mathfrak{X}_1, \ldots, \mathfrak{X}_n\}$.

Equivalence class $\mathfrak{X}_i \in X/\sim_\mathcal{D}$ comprises all sequences that reach a certain state of the global automaton. Let $\omega_\mathcal{D}(\mathfrak{X}_i)$ be the number of times that state is reached: $\omega_\mathcal{D}(\mathfrak{X}_i) = \sum_{\sigma' \in \mathfrak{X}_i} Y(\sigma')$.

Let $\Delta_\mathcal{D}(\mathfrak{X}_i)$ be the set of activities performed when being in that state: $\Delta_\mathcal{D}(\mathfrak{X}_i) = \{s \in \Sigma : \exists \sigma' \in [\sigma]_\mathcal{D},\ \delta(\sigma', s) \text{ is defined }\}$.

For any $\mathfrak{X}_i \in X/\sim_\mathcal{D}$, if $|\Delta_\mathcal{D}(\mathfrak{X}_i)| \simeq \omega_\mathcal{D}(\mathfrak{X}_i)$, the execution supposedly proceeds with an activity different from that already observed. Therefore, it is likely that a new event refers to a new unseen activity. Conversely, if $|\Delta_\mathcal{D}(\mathfrak{X}_i)| \gg \omega_\mathcal{D}(\mathfrak{X}_i)$, the execution can be reasonably assumed to proceed with an activity that has already been seen.

Definition 7 (*Generalization*). Let $(\Sigma, S, \sigma_0, \delta, F)$ be the alignment automaton for a Declare model $\mathcal{D}$ and an event log $\mathcal{L}$. Let $X/\sim_\mathcal{D}$ the set of all equivalence classes of $X = S \backslash F$. Let $pnew(n, m)$ be an estimator of the probability of observing a new activity in the $(m+1)$-th observation, when n different activities are observed in the first m observations [1]:

$$pnew(n,m) = \begin{cases} \dfrac{n(n+1)}{m(m-1)} & \text{if } m \geq n+2 \\ 1 & \text{otherwise} \end{cases}$$

The *generalization* of $\mathcal{D}$ with respect to $\mathcal{L}$ is defined as follows:

$$\textsc{Generalization}(\mathcal{D}, \mathcal{L}) = 1 - \frac{1}{|X/\sim_\mathcal{D}|} \sum_{\mathfrak{X}_i \in X/\sim_\mathcal{D}} pnew(|\Delta_\mathcal{D}(\mathfrak{X}_i)|, \omega_\mathcal{D}(\mathfrak{X}_i)).$$

Estimator $pnew(n, m)$ is the transition probability to observe a new activity, under the assumption that its distribution is multinomial. Generalization can be defined as the probability that the next trace, not yet observed, can be replayed by the process model.

Example 2 (cont.). Let $(\Sigma, S, \sigma_0, \delta, F)$ be the alignment automaton in Fig. 7 and let be $X = S \backslash F$. For Declare model $\mathcal{D}_b$, there exist no constraint automata. As a consequence, $X/\sim_{\mathcal{D}_b} = \{X\}$ with $\omega_{\mathcal{D}_b}(X) = 16$ and $|\Delta_{\mathcal{D}_b}(X)| = 4$. Therefore, $\textsc{Generalization}(\mathcal{D}_b, \tilde{\mathcal{L}}) = 1 - pnew(16, 4) = 0.95$.

Consider $\tilde{\sigma_3} = \langle a, b \rangle \in S$, $\tilde{\sigma_4} = \langle a, b, a, b \rangle \in S$. For the Declare model $\mathcal{D}_c$, after performing $\tilde{\sigma_4}$, (i) the chain-response constraint automaton is back to the initial state, (ii) the automaton of the precedence constraint between b and c is in the same state as after performing $\tilde{\sigma_3}$ and (iii)the automata for the other two constraints have not moved. Consequently, one of the partitions of X is $\mathfrak{X}' = \{\tilde{\sigma_3}, \tilde{\sigma_4}\}$. In this case, $\tilde{\sigma_3}$ is a prefix of three traces and $\tilde{\sigma_4}$ is a prefix of one trace; hence, $\omega_{\mathcal{D}_b}(\mathfrak{X}') = 4$. Moreover, after performing $\tilde{\sigma_3}$ and $\tilde{\sigma_4}$, executions of activities a, c are observed; hence, $|\Delta_{\mathcal{D}_b}(\mathfrak{X}')| = 2$. Therefore, when the global automaton is the state as after performing $\tilde{\sigma_3}$ and $\tilde{\sigma_4}$, the probability of observing an activity different from what observed is $pnew(3, 2) = 1$. The computation of this probability needs to be repeated for any other partition of X in order to score the generalization of $\mathcal{D}_c$: $\textsc{Generalization}(\mathcal{D}_c, \tilde{\mathcal{L}}) = 0.316$. Space limitations prevent us from discussing the computation of the generalization score of $\mathcal{D}_a$, which is $\textsc{Generalization}(\mathcal{D}_a, \tilde{\mathcal{L}}) = 0.45$. As it could be expected, the generalization score of $\mathcal{D}_b$ is higher than of $\mathcal{D}_a$, which, in turn, has a higher score than $\mathcal{D}_c$.

If an event log contains relatively few traces, the generalization will score low, since the next log trace is likely to be different from those already seen and, probably, it cannot be replayed by the model. Conversely, if the event log is complete and many traces appear multiple times, the generalization will be high: even if a log trace has not been observed yet, the model will be able to replay it.

7.4. Complexity analysis for the computation of fitness, precision and generalization

After computing the optimal alignment for each trace in the event log, the scores of fitness, precision and generalization can be computed in polynomial time with respect to the number and length of the log traces.

[4] The concept of global automaton is only introduced to simplify the discussion. Concretely, the equivalence classes can be easily computed by using an equivalence notion where two sequences/states of the alignment automaton are equivalent iff all constraint automata are in the same state.

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005

Let n be the number of traces of an event log $\mathcal{L}$ and m be the order of magnitude of the number of events in each trace.

Without additional information, we assume that each event has the same probability of being involved in a move in both or in log. Moreover, we assume that, for each move in the log (or model), one move in the model (or log, respectively) is also necessary. Therefore, the number of moves in each alignment γ has the order of magnitude of m and like so the length of every "aligned" model trace $\gamma\#_M$.

To compute the fitness score, each alignment needs to be entirely iterated once. Therefore, the complexity for computing the fitness of all traces in $\mathcal{L}$ is $O(m \cdot n)$. For constructing the alignment automaton $(\Sigma, S, \sigma_0, \delta)$, every "aligned" model trace, whose length has the order of magnitude of m, needs to be iterated once. Hence, the complexity for constructing the automaton is $O(m \cdot n)$. The number $|S|$ of states of the alignment automaton is also $O(m \cdot n)$. While iterating over the "aligned" model traces, the set of equivalence classes can be constructed on the fly, like so $ex_{\mathcal{L}}(\sigma)$, $av_{\mathcal{D}}(\sigma)$ can be computed for each $\sigma \in S$. Therefore, The precision and generalization scores can be computed in $O(m \cdot n)$.

8. Usage of the optimal alignments to clean and repair event logs

This section discusses the first and third use case of the framework described in Section 3, i.e., how to clean and repair event logs. Let $\mathcal{L} = (L, \Lambda)$ be an event log and let $\mathcal{D}_{whole} = (A_{whole}, \Pi_{whole})$ be a *Declare* (whole) model. For each trace $\sigma \in \mathcal{L}$, let γ_σ be an optimal alignment of σ and a core model $\mathcal{D}_{core} = (A_{core}, \Pi_{core})$ such that $\Pi_{core} \subseteq \Pi_{whole}$.

A *cleaned log* $\mathcal{L}_{clean}$ of $\mathcal{L}$ with respect to $\mathcal{D}_{core}$ is an event log (L_c, Λ_c), where L_c only contains the log traces $\sigma \in L$ such that γ_σ contains no move in the log or the model for any activity $a \in A_{core}$ as well as σ does not violate any constraint $\pi \in \Pi_{core}$. The cleaned event log $\mathcal{L}_{clean}$ only contains the log traces that are worthwhile to be considered for further analysis.

As mentioned in Section 3, many analysis techniques require that the event log perfectly fits a process model and, hence, results lose (part of) their significance if there are deviations. Therefore, non-fitting event logs need adjustments to remove deviations. The resulting log is called *repaired* event log.

Let $\mathcal{L} = (\{\sigma_1, \ldots, \sigma_n\}, \Lambda) \in \mathbb{B}(A_L^*)$ be a (cleaned) log. Let $\mathcal{D}_{repair} = (A_{repair}, \Pi_{repair})$ be a repair model. Let $\overline{\gamma_{\sigma_i}}$ be an optimal alignment of σ_i and $\mathcal{D}_{repair}$.

A *repaired log* $\mathcal{L}_{repair}$ of $\mathcal{L}$ with respect to $\mathcal{D}_{repair}$ is an event log $\mathcal{L}_{repair} = (\{\sigma_1^R, \ldots, \sigma_n^R\}, \Lambda_R) \in \mathbb{B}((A_L \cup \checkmark)^*)$ where, for $1 \le i \le n$, $\sigma_i^R = \overline{\gamma_{\sigma_i}}\#_M$ and $\Lambda_R(\sigma_i^R) = \Lambda(\sigma_i)$.

When an event is created because of a move in the model, there are two aspects to consider. Moves in model might be associated with ✓. As a result, an event ✓ could be inserted. Events ✓ do not affect many diagnostics when using the repaired log as input. Nonetheless, if log traces can only be repaired by adding ✓ events, i.e., events referring to activities not belonging to the model, that should be an alert of the fact that possibly the model is not designed well.

For the sake of simplicity, each event has been abstracted as the activity name only, and an event trace is a sequence of activity names. Of course, in reality, each event is also associated with a timestamp indicating the time when the event occurred. When inserting an event, this event needs to be associated with a timestamp, as well. The timestamp is necessary when one wants to execute some performance analysis. The inserted event is associated with a timestamp which is the average between the timestamps of the events that precede and follow it.

Example 1 (cont.). Let us suppose that $\mathcal{D}_{core} = (A_{core}, \Pi_{core})$ where $A_{core} = \{CQ\}$ and set Π_{core} contains all constraints in Fig. 1 apart from the *response* and the *not-succession* constraint. Furthermore, let us suppose that $\mathcal{D}_{repair}$ is the entire *Declare* model in Fig. 1, i.e., traces are repaired with respect to all the constraints in the model in Fig. 1. Trace σ_L is repaired as follows:

$$\sigma_L^R = \langle \text{Register}, \text{Create Questionnaire}, \text{Prepare Notification Content}, \text{Create Questionnaire}, \text{Send Questionnaire}, \text{Send Notification by e}-\text{mail}, \text{Send Notification by Post}, \text{Archive} \rangle$$

which is part of the repaired event log $\mathcal{L}_{repair}$.

9. Implementation and evaluation

In order to evaluate the three use cases described in Section 3, we have implemented a series of plug-ins of ProM, a generic open-source framework for implementing process mining functionalities [5]:

Declare Replayer It takes a *Declare* model and an event log as input and, using the algorithm described in Section 6, finds a multi-set of optimal alignments, i.e., one alignment for each trace in the event log. This plug-in is also in charge of computing the score of the different dimensions of conformance (fitness, precision and generalization).

Declare Diagnoser It takes as input the multi-set of optimal alignments and projects them on the model. Activities and constraints are colored according to their degree of conformance (see Section 5.3).

Log Cleaner It takes a (core) *Declare* model and an event log and returns an event log in which traces in the original log violating at least one constraint in the model are discarded. Internally, it invokes the *Declare Replayer* plug-in to find the multi-set of optimal alignments.

Log Repairer It takes a (repair) *Declare* model and, using the algorithm in Section 8, adjusts the log traces by removing and inserting events, thus producing a perfectly fitting event log. Internally, it invokes the *Declare Replayer* plug-in to find the multi-set of optimal alignments.

To assess the viability of the proposed techniques, we performed validations along two main directions. Firstly, the scalability was evaluated with *Declare* models and event

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005

logs of increasing sizes (one real-life and some synthetic event logs were used) and with increasing numbers of deviations. Secondly, we aimed at validating the usefulness of the approach and the effectiveness of the results obtained. In particular, we applied all of the three use cases described in Section 3 on a real case study from a Dutch hospital. Section 9.2 discusses the validation, highlighting the possible benefits for potential stakeholders.

9.1. Performance evaluation

The scalability of the proposed techniques has been evaluated through two sets of experiments. In a first set of experiments, one single *Declare* model was used and synthetic event logs were generated with different sizes and including different numbers of deviations. In a second set, the size of the *Declare* model was varied. Models were generated with different numbers of constraints and number of activities.

For the first set of experiments, several synthetic logs has been generated by modeling the process described in Example 1 in CPN Tools[5] and by simulating the model. Specifically, we generated event logs of different sizes (i.e., containing from 250 up to 750 traces) and with different probabilities of violating a certain constraint within a log trace (i.e., from 0% up to 90%). In other words, a probability of violation of $x\%$ means that the probability that a certain constraint is violated in a trace is $x/100$. Note that multiple constraints can be violated at the same time in a log trace: for instance, if the probability of violating a constraint is p, the probability of violating two constraints is p^2.

In Fig. 8, the results are shown. For each combination of log size (250, 500 or 750 traces) and probability of violation, the execution time is averaged over 5 runs. A trend line is also shown to highlight that the execution time grows linearly with increasing probability of violations in the event logs.

Also, *with respect to a single probability of violation, the execution time is linear in the number of traces of the event log*. It was expected that the alignment of a trace of the event log and the *Declare* model is computed without considering the other traces of the event log. Therefore, if the number of log traces is multiplied by n, the execution time would be n times greater.

In order to compare the optimized approach with the situations in which no pruning is made and also no heuristics is used, an event log was used with 250 traces and a degree of non-fitness of 90%. Table 1 shows the results of this comparison. If we do not prune the search-space, the *Declare Replayer* has to visit 457% of extra nodes and, consequently, the execution time increases by 500%. Table 1 also shows the reduction in terms of time and number of nodes to visit when using the heuristics described in Section 6. When no heuristics is used, the A* algorithm coincides with Dijkstra.

In a second set of experiments, a single event log was used with *Declare* models of different sizes. At the beginning, a *Declare* model was generated through the *Declare*

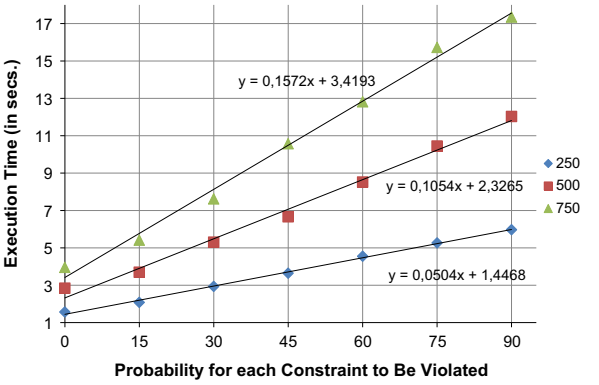

Fig. 8. Average execution time for different combinations of log sizes and probabilities of violation. Each combination has been run 5 times with a variance which is never higher than 0.04 s.

Table 1
Comparison of the execution time of the *Declare Replayer* when all optimizations are enabled with respect to the cases when they are selectively turned off. The results refer to a synthetic log from Example 1 with 250 traces and a probability of violation of 90%.

Employed Technique	Visited Nodes	Tree Size	Execution Time
Optimized A*	67	218	6 s
Without pruning	373	3310	36 s
Without heuristics	324,119	3,402,799	ca. 12 h

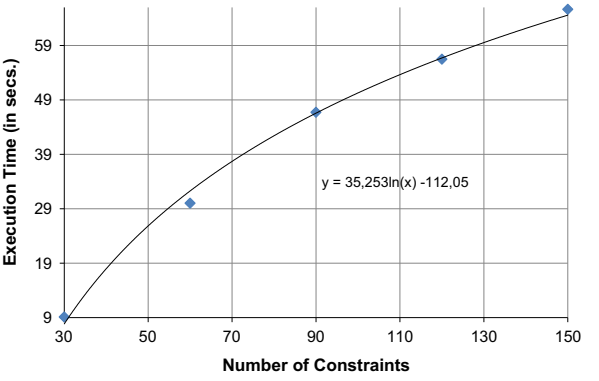

Fig. 9. Results of the experiments conducted with different *Declare* models with 28 activities and different numbers of constraints. The variance is negligible as it is never more than 0.07 s.

Miner plug-in in ProM [19]. The input was a real-life event log concerned with a process enacted in a Dutch hospital (see Section 9.2 for more details). The event log contained 289 log traces, 10,215 events and 152 activities and the mined *Declare* model consisted of 157 constraints and 28 activities (the least frequent activities were abstracted out). In order to obtain models with fewer constraints, a percentage of constraints was randomly removed while ensuring each activity to be connected to one constraint, at least. To conduct the experiments, these models were checked for conformance against the event log. Fig. 9 shows the execution time when varying the number of constraints. For

[5] http://cpntools.org

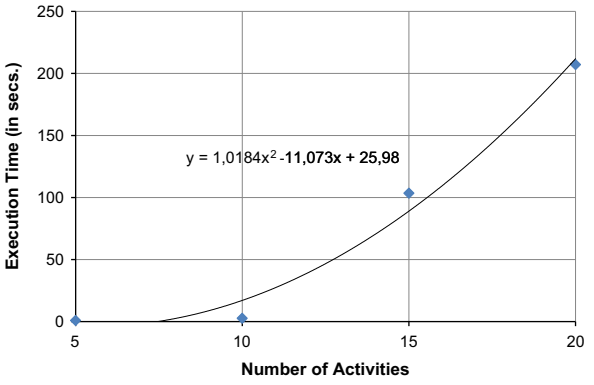

Fig. 10. Results of the experiments conducted including a balanced number of constraints of type precedence, responded existence and response and different numbers of activities. The execution time trend is polynomial when the number of activities is equal to or bigger than 10; the execution time is very small and almost constant for smaller numbers.

reliability, given a number of constraints, different models were considered with that number of constraints. The trend line illustrates that the execution time increases with the logarithm of the number of constraints.

Later, the experiments aimed to evaluating how the execution time is affected when varying the number of activities in models including different types of constraints. Different *Declare* models were generated with different numbers of activities (5, 10, 15 and 20) with nearly two constraints per activity. The types of constraints were precedence, responded existence and response. The models were generated so as to balance the number of constraints of these three types (i.e., the models contain the same number of constraints of the three types) and, also, to have similar fitness scores (around 0.85). The experimental results are shown in Fig. 10. The trend line illustrates that the execution time increases polynomially with the number of activities; for models with less than 10 activities, the execution time is very small and almost constant.

9.2. Validation using a case study

In the previous section, the approach feasibility has been validated. This section goes beyond and aims at assessing its relevance and usefulness in a real scenario. For this purpose, the same event log relative to a Dutch hospital has been used. The event log concerns the treatment of bladder cancer. Bladder cancer is a complex disease and each patient must be treated in a different way, depending on the patient's characteristics (e.g., age, gender), the histology, grade and depth of invasion. Based on these parameters, there are several possible tests and treatments the doctors can decide to apply to patients affected by bladder cancer. Due to the high flexibility of the therapy, the use of a declarative process model is highly indicated.

Fig. 11 shows a *Declare* model that has been drawn by hand in compliance to the guidelines published by the European association of Urology for treatment of bladder cancer.[6] According to the log–model alignment framework, we have identified the core and repair model. The core model is the union of two parts delimited by the solid green line, whereas the repair model is delimited by the red dotted line. The repair model comprises the constraints that, according to the medical guidelines, must be always respected for a correct treatment of the cancer.

The core model contains two constraints: the *init* constraint associated with *First patient visit*, according to which this activity has to be the first of every process execution, as well as the 0..1 constraint associated with *Dbc Registration*, which states that this activity cannot be executed more than once in a process instance. Every log trace that violates these constraints should be discarded as certainly incomplete: the activities for the hospital registration and the first visit are not logged. In real scenarios, it is not infrequent that there are logging problems and that a number of activity executions are not stored in the event logs. These traces should certainly be left out since they are biased and would negatively affect the efficacy of process mining techniques.

This is also shown in this event log: out of the 289 log traces, only 68 traces were kept after log cleaning. The cleaned log and the whole *Declare* model were used as input for the *Declare Replayer*.

Fig. 12 illustrates the output produced by the *Declare Replayer* plug-in. Each sequence of triangles refers to an alignment of a trace with respect to the model. Each triangle is a different legal move; the color of the move depends on its type: moves in the log, moves in the model or moves in both are colored in yellow, purple and green respectively. Each sequence is also associated with a number that identifies the fitness score of the specific trace.

When clicking the *Detail* button associated with each trace, the alignment is shown at the bottom (e.g., the screenshot shows the details when selecting the trace numbered 1000181554). Each rectangle represents a different move and is annotated with the activity involved in the move. The color of the rectangle is again associated with the type of move, using the color scheme mentioned before. When passing with the mouse over any rectangle, the *Declare Replayer* shows which constraint violations that move contributes to solve (see Section 5.3).

In the screenshot in Fig. 12, it is shown the 17th move is in the log for activity *Transurethral Resection*. When passing over the corresponding rectangle with the mouse, the *Declare Replayer* lists the constraints that the move contributes to solve. Two precedence constraints are solved by the move in the log: every occurrence of *Transurethral Resection* should be preceded by *Cysto-urethescopy* and *Preoperative Screening*. Since occurrences of these two activities were not present in the trace before *Transurethral Resection*, the latter cannot be executed, which motivates the move in the log.

Immediately above the enumeration of the traces with their optimal alignments, the graphical user interface shows the average fitness score with respect to all event

[6] See http://www.uroweb.org/guidelines/online-guidelines/. It is worth saying that this model is none of the models used for the scalability evaluation in the previous section.

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005

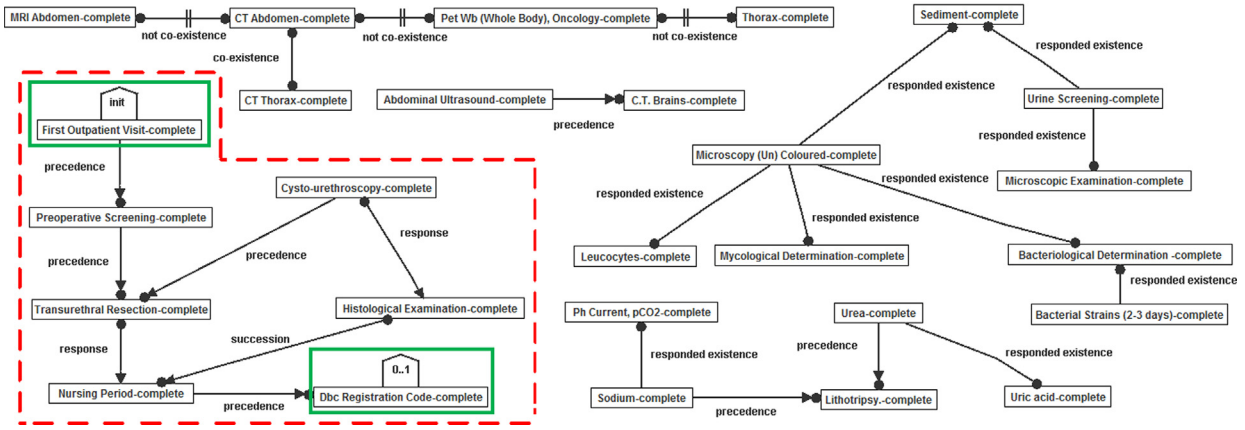

Fig. 11. The core, repair and whole models to describe the case study from a Dutch hospital. The core constraints are those in the two areas delimited by the solid green line, whereas the repair constraints are those included in the area delimited by the red dotted line. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

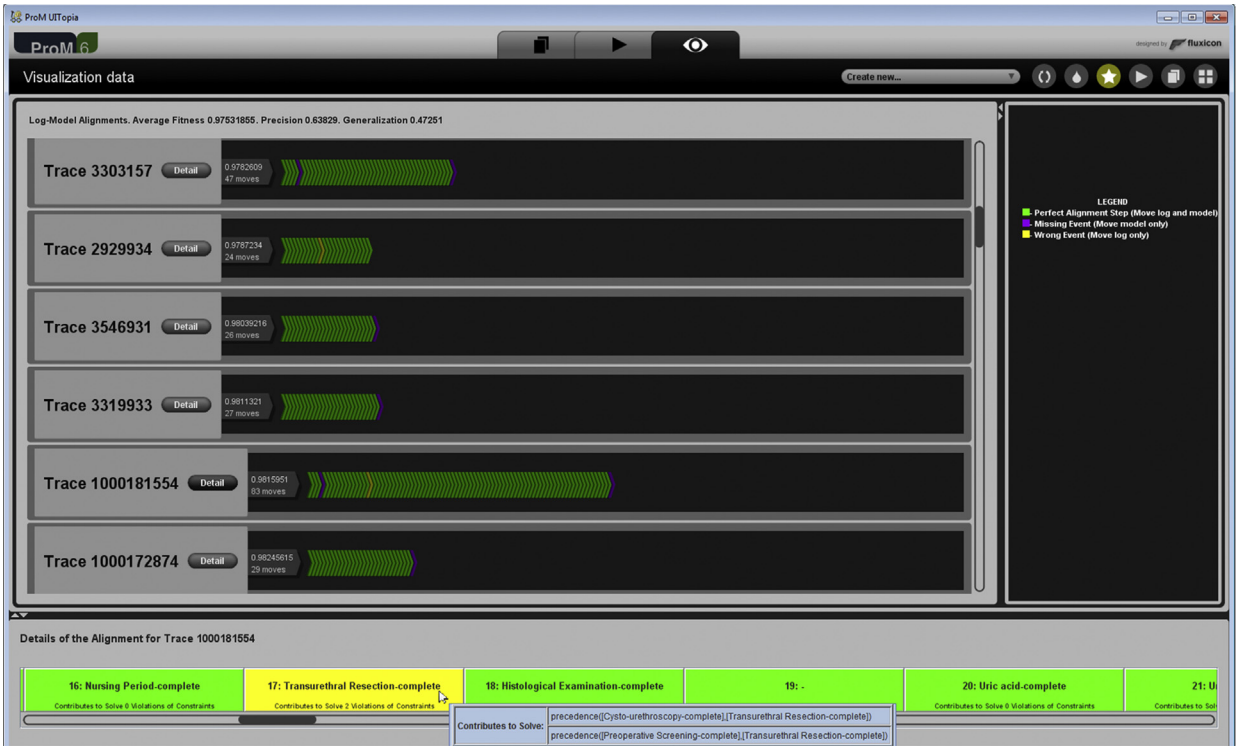

Fig. 12. A screenshot showing the trace-model alignments as visualized in the *Declare Replayer* plug-in of ProM. The average fitness is also shown with respect to all optimal alignments, along with the values of precision and generalization of the model. (For interpretation of the references to colors in this figure caption, the reader is referred to the web version of this paper.)

traces as well as the score of the precision and generalization dimensions of conformance.

Fig. 13 shows a screenshot of the *Declare Diagnoser* plug-in where moves in the log and in the model are projected on the *Declare* model. The figure refers to the same set of alignments as in Fig. 12, i.e., using the clean event log and the whole model as input. Each activity is associated with a number that represents the degree of conformance (see Section 5.3). Each constraint is associated with two numbers, the first representing the degree of conformance and the second the percentage of process instances in which a certain constraint is violated. Moreover, the degree of conformance is also reflected in the colors of activities to make the visualization more effective. Green and red nodes and arcs indicate a degree 1 or 0 of conformance respectively. Intermediate shades between

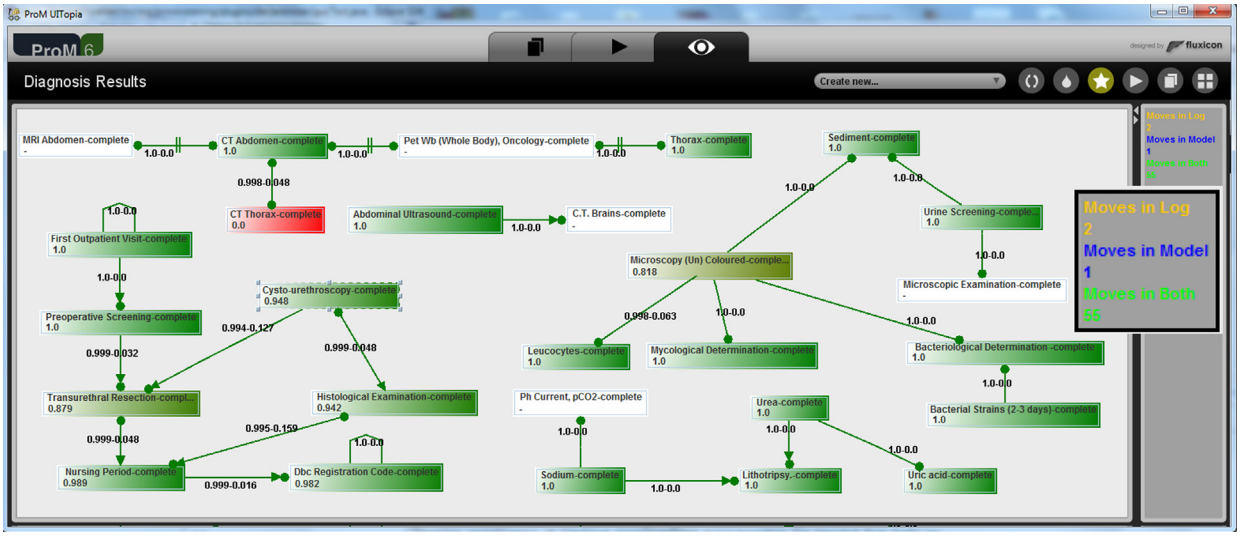

Fig. 13. A screenshot showing the output of the *Declare Diagnoser* plug-in, which gives a helicopter view of the activities and constraints that are mostly involved in deviations. Red color is associated with activities which are often involved in moves in the log or model as well as with constraints that are violated more frequently. Green color indicates the opposite situations, with intermediate shades indicating intermediate situations. White color is given to activities that have never occurred in the event log. (For interpretation of the references to colors in this figure caption, the reader is referred to the web version of this paper.)

green and red reflect values in-between these two extremes. White color is associated with activities that do not occur in the log traces.

In our case study, the clean log only contains traces that do not violate the core constraints; therefore, the degree of conformance of the core constraints is equal to 1. The presence of white-colored activities is motivated by the fact that some activities are so infrequent that they do not occur in the event log resulting from the cleaning phase. The figure also brings out the red color for activity *CT Thorax*: its degree of conformance is equal to 0. This means that it is always associated with moves in the log or in the model, never with moves in both. This means that the activity is never executed correctly. Either it is performed when it should not be or is not performed when it should be.

When selecting an activity, on the left-hand side of the screen, a summary is provided with the number of moves in which the activity is involved; in the figure, activity *Cysto-urethescopy* is selected, which is involved in two moves in the log, one move in the model and 27 moves in both. *It is worth noting that the outcome of Declare Diagnoser is also very insightful for* repairing the *Declare* model. *The model can be repaired by removing the constraints for which the degree of conformance is below a certain threshold.*

In order to meaningfully apply additional process mining techniques off the shelf, the event log has to entirely comply with the model. Therefore, we need to apply the use case (iii) in Fig. 2 and repair the event log using the repair model delimited by the red dotted line in Fig. 11.

Fig. 14 shows the results of using the repaired event log as input for some process mining techniques implemented as plug-ins of ProM. Fig. 14(a) refers to the prediction technique described in [12]. The underlying model is a transition system. Nodes correspond to the states which some process instances went through. Respectively, ingoing and outgoing arcs of a state identify the activities that led to that state and that were performed when being in that state. Without repairing and cleaning the event log, the transition system is much more chaotic. Indeed, the number of states would blow up since it should incorporate those states that refer to non-conforming behavior. Obviously, those states should not be part of the transition system and, hence, the predictions for those states are meaningless and also affect the predictions for the other states. The predictions are also more inaccurate due to the fact that the technique is based on replaying the log traces and shows issues when replaying non-conforming traces.

Fig. 14(b) depicts a social network that describes the hand over of work between resources (see [22]): each oval identifies a different resource and the horizontal and vertical diameters grow with the amount of work that is delegated or received respectively. If we compare this hand over social network with the same obtained from the event log before repairing, the ovals' size tends to reduce with respect to both the horizontal and vertical diameters. In general, this is normal, since, in the repaired log, some events are removed and, hence, there is less work that is handed over. Nonetheless, the comparison shows that the length of the diameters is mostly reduced for only some ovals, whereas the diameters stay unchanged for others. This means that always the same resources are used to misbehave.

Fig. 14(c) illustrates a bar diagram obtained by the throughput analyzer in ProM (see [18]). Each bar is associated with a different process instance. The *y*-axis indicates the execution time of each process instance in hours. Apart from few outliers, it is clear that the entire treatment of a bladder cancer is usually concluded within 1000 h. If we use the event log before repairing, the resulting bar diagram does not differ significantly. Conversely, if we do

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005

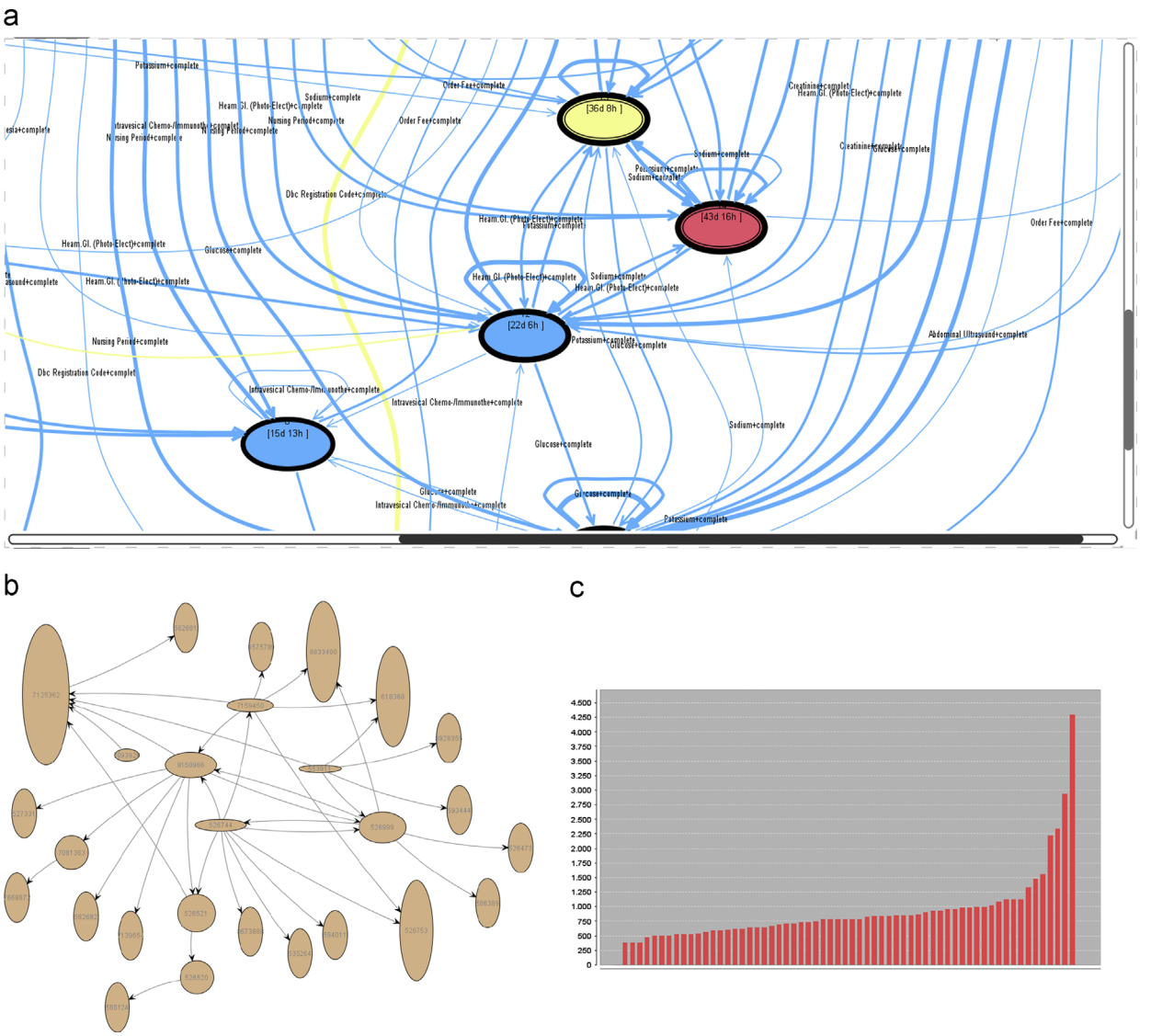

Fig. 14. The application of diverse process mining techniques using the repaired or cleaned event log of the case study stemming from a Dutch hospital. The figures refer to screenshots of the graphical user interfaces of the respective plug-in implementations in ProM. (a) Execution time prediction: each oval is a state which some process execution has gone through. Each arc indicates the activity that causes the state change. Each state is also annotated with the predicted execution time to complete a process instance, when the execution is in that state. (b) Social network: each oval represents a different resource and the arcs highlight the hand over of work between resources. (c) Execution time analysis: each bar is a different process instance. The *y*-axis indicates the execution time of the each process instance and is measured in hours.

not clean the event logs, the bar diagrams show 15% more outliers characterized by very long execution time. This suggests that the traces with logging problems are also characterized by long execution time. Possibly, this is caused by errors in the logging system.

10. Related work

Over the last decade *process mining* evolved into an active research field relating data analysis (data mining, machine learning, etc.) and process analysis. See [18] for an introduction to process mining. We refer to the recently released Process Mining Manifesto [23] for the main challenges in process mining. The focus of this paper is *on the analysis of the execution of Declare models*; therefore we do not elaborate on process discovery techniques.

Conformance checking is highly relevant for auditing [24] and risk analysis [25]. However, the literature on auditing and risk analysis [24,25] does not suggest concrete techniques to systematically compare observed behavior and modeled behavior. Fortunately, a variety of conformance checking techniques has been developed in the context of process mining.

Cook and Wolf [26] were among the first to quantify the relationship between an event log and a process model. They compare event streams generated from the model with event steams generated from the event log.

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005

Several authors proposing process discovery algorithms also provide a quality metric (often related to fitness). For example, in [27] the authors define a fitness function for searching for the optimal model using a genetic approach. In [28] a "process mining evaluation framework" for benchmarking process discovery algorithms is proposed.

The first comprehensive approach to conformance analysis was proposed in [29] by Rozinat and van der Aalst. Two different types of metrics are proposed: (a) *fitness metrics*, i.e., the extent to which the log traces can be associated with valid execution paths specified by the process model and (b) *appropriateness metrics*, i.e., the degree of accuracy in which the process model describes the observed behavior, combined with the degree of clarity in which it is represented. Fitness in [29] is measured by "replaying the event log" and counting the number of missing and remaining tokens. This typically results in rather high fitness values as also pointed out in [2,30]. In [29], four appropriateness metrics are defined. *Simple behavioral appropriateness* looks at the average number of enabled transitions. If most transitions are continuously enabled, the model is likely to lack precision (i.e., underfitting). *Advanced behavioral appropriateness* compares the "footprint" of the log (follows and precedes relationships) to the "footprint" of the model. *Simple structural appropriateness* and *advanced structural appropriateness* quantify the complexity of the model.

One of the drawbacks of the approach in [29] and most other approaches that "play the token game" is that fitness is typically overestimated. When a model and log do not fit well together, replay will overload the process model with superfluous tokens. As a result, the model will allow for too much behavior. In fact, most conformance techniques give up after the first non-fitting event or simply "guess" the corresponding path in the model. Therefore, Adriansyah et al. formulated conformance checking problems as an optimization problem [2]. This is the approach we also use in this paper. Besides the four appropriateness notions in [29], Muñoz-Gama and Carmona quantified additional precision notions [31,32].

It is difficult to use classical quality notions such as *precision* and *recall* for process mining. The main reason is that event logs only contain positive examples, i.e., one can see what "did happen" but not what "could not happen". Therefore, some authors suggest inserting artificially generated "negative events" [33,34]. Goedertier et al. proposed such events for both process discovery and conformance checking [33]. Weerdt et al. defined a so-called F-measure based on artificially generated negative events [34]. The authors of the latter paper also conducted a comparative analysis of several conformance metrics [30]. However, their study did not consider the more advanced alignment-based approaches discussed in this paper.

In [35], a completeness metric and a soundness metric are defined. These metrics compare the model traces with the log traces. This approach suffers from several drawbacks. First of all, only complete traces are compared. Second, it is assumed that the model's behavior can be enumerated. Finally, it is assumed that the log contains all possible traces.

There are also efficient algorithms to perform sequence alignments (e.g., the algorithms of Needleman–Wunsch and Smith–Waterman). Similarly, in process mining, Jagadeesh Chandra Bose and van der Aalst [36] have proposed techniques to efficiently align pairs of log traces. Unfortunately, they cannot be applied to find an alignment between a log trace and a process model. In our setting, we do not know a priori the model trace to align with the log trace; conversely, the model trace needs to be chosen, thus minimizing the severity of the deviations.

This paper extends [17] along the following directions:

1. It proposes a framework for analysis of *Declare* models that is based on three *use cases* which are supported by ProM and are based on log–model alignment. In these use cases, conformance is evaluated through three dimensions: fitness, precision and generalization. Paper [17] was only concerned with computing fitness for diagnostics purposes, which is one single aspect of one of the three use cases.
2. The performance evaluation of the approach has been strengthened. We provide a detailed analysis of how the approach scales with models of increasing sizes, proving that it is applicable to realistic scenarios.
3. The approach has also been used to evaluate a real-life process enacted in a Dutch hospital. We have applied all three use cases defined in the analysis framework. This has allowed us to put in evidence which kind of stakeholders in process-aware information systems would benefit from the approach presented in this paper.
4. The implementation in ProM is more efficient and outperforms the one presented in [17]. Comparing Fig. 8 with Fig. 3 in [17], the improvements are evident: the execution is almost 90% faster with respect to the old implementation. Of course, the new experiments and those reported in [17] were carried out using the same computer. Also, the same algorithm was used. The reason of this drastic improvement is only related to the implementation. Specifically, in the implementation reported in [17], the list of already visited search-space nodes/alignments γ with their cost $g(\gamma)$ was kept in a linked list without indexing. Therefore, the cost of checking whether an equivalent alignment was already visited was $O(n)$ where n is the length of the list. For the new implementation, the linked list was replaced by a hash table with nodes/alignments as key and cost as value. Therefore, the cost to search for equivalent alignments has decreased from $O(n)$ to $O(1)$. Moreover, the implementation reported in [17] stored the priority queue for the A* algorithm in a list sorted by cost, with insertion cost $O(m)$ and removal cost $O(1)$ where m is the length of the list. In the new implementation, a self-balancing binary search tree is used and insertion and removal operations take $O(\log m)$ time. Since the number of insertions are far more than the removals, search trees perform better than sorted lists.

11. Conclusion

This paper presents a novel log preprocessing and conformance checking approach tailored towards declarative models. Most conformance checking techniques defined for procedural models (e.g., Petri nets) are not directly applicable to declarative models since they are based on playing the "token game" while counting missing and remaining tokens. Moreover, these

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005

techniques tend to provide poor diagnostics, e.g., just reporting the fraction of fitting cases. We adapted alignment-based approaches to be able to deal with the large search spaces induced by the inherent flexibility of declarative models.

As shown in this paper, alignments provide a very powerful tool when relating observed behavior with modeled behavior. We described three possible applications of our alignment-based approach: (i) *cleaning event logs* to remove traces that should not be used for further analysis, (ii) *diagnosing event logs* to check conformance, show deviations and measure compliance scores and (iii) *repairing event logs* to make sure that the essential constraints are satisfied before further analysis.

The presented approach has been implemented in ProM. Our implementation provides novel diagnostics, at the trace level, showing why events need to be inserted/removed in a trace and at the model level, coloring constraints and activities in the model based on their degree of conformance. As future work, we plan to extend our approach in order to incorporate other perspectives (data, resources, risks, costs, etc.) in our analysis. For example, alignments can also be used to diagnose problems such as activities that are executed too late or by the wrong person.

References

[1] W.M.P. van der Aalst, A. Adriansyah, B.F. van Dongen, Replaying history on process models for conformance checking and performance analysis, Wiley Interdiscip. Rev.: Data Mining Knowl. Discov. 2 (2) (2012) 182–192.

[2] A. Adriansyah, B. van Dongen, W. van der Aalst, Conformance checking using cost-based fitness analysis, in: IEEE International Enterprise Distributed Object Computing Conference (EDOC '11), IEEE Computer Society, 2011, pp. 55–64.

[3] M. de Leoni, W.M.P. van der Aalst, B.F. van Dongen, Data- and resource-aware conformance checking of business processes, in: 15th International Conference on Business Information Systems, Lecture Notes in Business Information Processing, vol. 117, Springer Verlag, 2012, pp. 48–59.

[4] R. Dechter, J. Pearl, Generalized best-first search strategies and the optimality of A*, J. ACM (JACM) 32 (1985) 505–536.

[5] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, W.M.P. van der Aalst, XES, XESame, and ProM 6, in: Proceedings of Information Systems Evolution (CAiSE Forum 2010), Lecture Notes in Business Information Processing, vol. 72, 2011, pp. 60–75.

[6] F.M. Maggi, A.J. Mooij, W.M.P. van der Aalst, Analyzing vessel behavior using process mining, in: Situation Awareness with Systems of Systems, Springer, New York, 2013, pp. 133–148.

[7] S. Zugal, J. Pinggera, B. Weber, The impact of testcases on the maintainability of declarative process models, in: Enterprise, Business-Process and Information Systems Modeling (BMMDS/EMMSAD 2011), Lecture Notes in Business Information Processing, vol. 81, 2011, pp. 163–177.

[8] P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, H.A. Reijers, Imperative versus declarative process modeling languages: an empirical investigation, in: Proceedings of Business Process Management Workshops 2010, Lecture Notes in Business Information Processing, vol. 99, 2011, pp. 383–394.

[9] D. Giannakopoulou, K. Havelund, Automata-based verification of temporal properties on running programs, in: Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE'01), IEEE Computer Society Press, Providence, 2001, pp. 412–416.

[10] M. Westergaard, F.M. Maggi, Declare: a tool suite for declarative workflow modeling and enactment, in: Proceedings of the Demo Track of the 9th Conference on Business Process Management 2011, CEUR Workshop Proceedings, vol. 820, CEUR-WS.org, 2011.

[11] W.M.P. van der Aalst, M. Pesic, H. Schonenberg, Declarative workflows: balancing between flexibility and support, Comput. Sci.—Res. Dev. 23 (2) (2009) 99–113.

[12] W.M.P. van der Aalst, M. Schonenberg, M. Song, Time prediction based on process mining, Inf. Syst. 36 (2) (2011) 450–475.

[13] M. Westergaard, Better algorithms for analyzing and enacting declarative workflow languages using LTL, in: Proceedings of the 9th Business Process Management Conference (BPM'11), Lecture Notes in Computer Science, vol. 6896, Springer, Berlin, 2011, pp. 83–98.

[14] A. Bauer, M. Leucker, C. Schallhart, Runtime verification for LTL and TLTL, ACM Trans. Softw. Eng. Methodol. 20 (4) (2011).

[15] A. Awad, G. Decker, M. Weske, Efficient compliance checking using BPMN-Q and temporal logic, in: Proceedings of the 6th International Conference on Business Process Management, Lecture Notes in Computer Science, vol. 5240, Springer, Berlin, 2008, pp. 326–341.

[16] F.M. Maggi, M. Montali, M. Westergaard, W.M.P. van der Aalst, Monitoring business constraints with linear temporal logic: an approach based on colored automata, in: Proceedings of the 9th International Conference on Business Process Management (BPM'11), Lecture Notes in Computer Science, vol. 6896, Springer-Verlag, 2011, pp. 132–147.

[17] M. de Leoni, F. Maggi, W.M.P. van der Aalst, Aligning event logs and declarative process models for conformance checking, in: Proceedings of the 10th International Conference on Business Process Management, Lecture Notes in Computer Science, vol. 7481, Springer, Berlin, Heidelberg, 2012, pp. 82–97.

[18] W.M.P. van der Aalst, Process Mining—Discovery, Conformance and Enhancement of Business Processes, Springer, Berlin, 2011.

[19] F.M. Maggi, R.P.J.C. Bose, W.M.P. van der Aalst, Efficient discovery of understandable declarative process models from event logs, in: Proceedings of the 24th International Conference on Advanced Information Systems Engineering (CAISE 2012), Lecture Notes in Computer Science, vol. 7328, 2012, pp. 270–285.

[20] W.M. van der Aalst, Mediating between modeled and observed behavior: the quest for the "right" process, in: Proceedings of the IEEE International Conference on Research Challenges in Information Science (RCIS 2013), IEEE Computing Society, Paris, 2013, pp. 31–43.

[21] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. van Dongen, W. van der Aalst, Alignment based precision checking, in: Proceedings of Business Process Management Workshops 2012, Lecture Notes in Business Information Processing, vol. 132, Springer Verlag, Berlin, 2013, pp. 137–149.

[22] M. Song, W.M.P. van der Aalst, Towards comprehensive support for organizational mining, Decis. Support Syst. 46 (1) (2008) 300–317.

[23] W.M.P. van der Aalst et al., Process mining manifesto, in: Proceedings of Business Process Management Workshops 2011, Lecture Notes in Business Information Processing, vol. 99, Springer Verlag, 2012, pp. 169–194.

[24] M. Vasarhelyi, M. Alles, A. Kogan, Principles of analytic monitoring for continuous assurance, J. Emerg. Technol. Account. 1 (1) (2004) 1–21.

[25] J. Hulstijn, J. Gordijn, Risk analysis for inter-organizational controls, in: Proceedings of the 12th International Conference on Enterprise Information Systems (ICEIS 2010), SciTePress, 2010, pp. 314–320.

[26] J. Cook, A. Wolf, Software process validation: quantitatively measuring the correspondence of a process to a model, ACM Trans. Softw. Eng. Methodol. 8 (2) (1999) 147–176.

[27] A. Medeiros, A. Weijters, W.M.P. van der Aalst, Genetic process mining: an experimental evaluation, Data Min. Knowl. Discov. 14 (2) (2007) 245–304.

[28] A. Rozinat, A. Medeiros, C. Günther, A. Weijters, W.M.P. van der Aalst, The need for a process mining evaluation framework in research and practice, in: Proceedings of Business Process Management Workshops 2007, Lecture Notes in Computer Science, vol. 4928, Springer-Verlag, Berlin, 2008, pp. 84–89.

[29] A. Rozinat, W.M.P. van der Aalst, Conformance checking of processes based on monitoring real behavior, Inf. Syst. 33 (1) (2008) 64–95.

[30] J. Weerdt, M. De Backer, J. Vanthienen, B. Baesens, A critical evaluation of model–log metrics in process discovery, in: Proceedings of Business Process Management Workshop 2010, Lecture Notes in Business Information Processing, vol. 66, Springer-Verlag, Berlin, 2011, pp. 158–169.

[31] J. Muñoz-Gama, J. Carmona, A fresh look at precision in process conformance, in: Proceedings of the 8th International Conference on Business Process Management (BPM'10), Lecture Notes in Computer Science, Springer-Verlag, 2010, pp. 211–226.

[32] J. Muñoz-Gama, J. Carmona, Enhancing precision in process conformance: stability, confidence and severity, in: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011), IEEE, 2011, pp. 184–191.

[33] S. Goedertier, D. Martens, J. Vanthienen, B. Baesens, Robust process discovery with artificial negative events, J. Mach. Learn. Res. 10 (2009) 1305–1340.

[34] J. Weerdt, M. De Backer, J. Vanthienen, B. Baesens, A robust F-measure for evaluating discovered process models, in: Proceedings

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005

of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011), IEEE, Paris, France, 2011, pp. 148–155.

[35] G. Greco, A. Guzzo, L. Pontieri, D. Sacca, Discovering expressive process models by clustering log traces, IEEE Trans. Knowl. Data Eng. 18 (8) (2006) 1010–1027.

[36] R.P. Jagadeesh Chandra Bose, W.M.P. van der Aalst, Process diagnostics using trace alignment: opportunities, issues, and challenges, Inf. Syst. 37 (2) (2012) 117–141.

Please cite this article as: M. de Leoni, et al., An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014), http://dx.doi.org/10.1016/j.is.2013.12.005