Information Systems 47 (2015) 258–277

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

Available online 17 January 2014

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/.