Declarative Process Discovery with MINERful in ProM

Claudio Di Ciccio[1], Mitchel H. M. Schouten[2], Massimiliano de Leoni[2], and Jan Mendling[1]

[1] Vienna University of Economics and Business, Austria
claudio.di.ciccio@wu.ac.at, jan.mendling@wu.ac.at
[2] Eindhoven University of Technology, The Netherlands
m.h.m.schouten@student.tue.nl, m.d.leoni@tue.nl

Abstract. Declarative process models consist of a set of constraints exerted over the execution of process activities. DECLARE is a declarative process modelling language that specifies a set of constraint templates along with their graphical notation. The automated discovery of DECLARE models aims at finding those constraints that are verified throughout a given event log. In this paper, we present a fast scalable tool for mining DECLARE models in ProM. Its usage is described with its application on a use case, based on a publicly available real-life benchmark.

Keywords: Process Mining; Process Discovery; Declarative Processes

1 Overview

Process Mining [1] is the area of research embracing the automated discovery, conformance checking and enhancement of business process models. All involved techniques are evidence-based, as the input always comprises a collection of computer-recorded information that track the executions of process instances, namely *event logs*. Indeed, process discovery pertains to the inference of process models stemming from event logs.

Over the last years, the declarative process modelling approach has flanked the classical procedural one [5]. Declarative approaches only depict the behavioural constraints under which a process instance can unfold in its execution: as long as the constraints are not violated, the process instance is considered as valid. The declarative approach is a complementary strategy to the procedural models, which specify what are the next allowed activities at each stage of the process execution. Declarative process models are effective in a context of high flexibility for business processes [2]. The reason intuitively lies in the fact that fewer constraints allow for more possible executions. On the contrary, more flexibility implies a higher number of alternative paths to depict in the procedural models.

DECLARE [2] is a declarative process modelling language. It specifies an extensible set of constraint templates that are parametric with respect to the process activities. A list of constraint templates used in the remainder of the paper are listed in Table 1, where a, b, and c are example activities. Examples of DECLARE constraints are $\mathit{Init}(\mathsf{a})$, and $\mathit{Response}(\mathsf{b},\mathsf{c})$. The first one states that every instance must start with the execution

Constraints	Description
$Init$(a)	a should be the first activity in a trace
$AtMostOne$(a)	a should be executed at most once
$CoExistence$(a, b)	If one of the activities a or b is executed, the other one also has to be executed
$Response$(a, b)	When a is executed, b has to be executed after a
$AlternateResponse$(a, b)	When a is executed, b has to be executed after a and no other a can be executed in between
$Precedence$(a, b)	b has to be preceded by a
$AlternatePrecedence$(a, b)	b has to be preceded by a and another b cannot be executed between a and b
$AlternateSuccession$(a, b)	Combination of $AlternateResponse$(a, b) and $AlternatePrecedence$(a, b)
$ChainSuccession$(a, b)	a is immediately followed by b
$NotChainSuccession$(a, b)	a is not allowed to be immediately followed by b

Table 1: Table listing the constraints mentioned in this paper.

of activity a. The second constraint imposes that if activity b is performed, then c must be performed eventually in the future. *Init* is named existence constraint template as it constrains the execution of one activity in process instances. *Response* is named relation constraint template instead, because it constrains the interplay of two activities. Among the pair of constrained activities, there always are at least an *activation* and a *target*. The activation is an event whose occurrence constrains the possibility of other events (targets) to occur before or afterwards. For example, for the constraint "every request is eventually acknowledged", each request is an activation. This activation is eventually associated with either a *fulfilment* or a *violation*, depending on whether or not the activation is matched with a target event that satisfies the constraint. Using again the example of requests that need acknowledgements, if the request occurs, this activation is associated with a fulfilment if the acknowledgement event is later observed; otherwise, the activation is associated with a violation. For *Response*(b, c), b is the activation and c is the target. The full list of DECLARE constraint templates can be found in [2].

This paper reports on the implementation of MINERful, a technique to mine DECLARE process models from an existing event log [3]. Compared with other existing techniques, MINERful has shown the best scalability with respect to the input size, in terms of number of traces, length of traces and activities of the process. Readers are referred to[3] for more details about this comparison. In particular, the implementation presented in this paper has been realised in ProM,[3] an extensible framework that provides support to develop and exploit a wide variety of process mining techniques in a standardised environment. To use MINERful with ProM, it is necessary to download the ProM Nightly build[4] and, subsequently, install the *DeclareMinerFul* package through the ProM's Package Manager.

[3] http://www.processmining.org/tools/prom
[4] http://www.promtools.org/prom6/nightly

2 Usage of the Tool on a Use Case

In this paper we will demonstrate the functionalities of MINERful using the publicly available real-life event log Road Traffic Fine Management Process.[5] The event log records executions of instances of the process enacted in an Italian local police office for managing fines for road traffic violations. It contains 150,370 traces and 561,470 events for 11 different process activities.

2.1 Parameters

The MINERful plug-in uses an event log as input. In the remainder, we will adopt the following example event log: $\{\langle \mathsf{a}, \mathsf{b}, \mathsf{a}, \mathsf{c}\rangle, \langle \mathsf{a}, \mathsf{b}, \mathsf{b}, \mathsf{a}, \mathsf{c}, \mathsf{b}, \mathsf{a}\rangle, \langle \mathsf{a}, \mathsf{c}, \mathsf{c}\rangle, \langle \mathsf{a}, \mathsf{b}, \mathsf{c}\rangle\}$.
The application of the MINERful plug-in for the DECLARE-model discovery can be customised through four parameters, namely:

Support. It is the number of fulfilments divided by either *(i)* the number of traces in the log, in the case of existence constraints like $\mathit{Init}(\mathsf{a})$, or *(ii)* the number of occurrences of the activations (in the case of relation constraints like $\mathit{Response}(\mathsf{b}, \mathsf{c})$). In the example log, the support of $\mathit{Init}(\mathsf{a})$ is 1.0, because all traces start with a, whereas the support of $\mathit{Response}(\mathsf{b}, \mathsf{c})$ is 0.8, as 4 b's out of 5 fulfil the constraint.

Confidence. It is the product of the support and the fraction of traces in the log where either *(i)* the constrained activity occurs (existence constraints), or *(ii)* the activation occurs (relation constraints). The confidence of $\mathit{Init}(\mathsf{a})$ is $1.0 \cdot 1.0 = 1.0$ and the confidence of $\mathit{Response}(\mathsf{b}, \mathsf{c})$ is $0.8 \cdot 0.75 = 0.6$, since b occurs in 3 traces out of 4.

Interest Factor. It is the product of confidence and the fraction of traces in the log where either *(i)* the constrained activity occurs (existence constraints), or *(ii)* the target occurs (relation constraints). The interest factor of $\mathit{Init}(\mathsf{a})$ is $1.0 \cdot 1.0 \cdot 1.0 = 1.0$, and the interest factor of $\mathit{Response}(\mathsf{b}, \mathsf{c})$ is $0.8 \cdot 0.75 \cdot 1.0 = 0.6$, since c occurs in all traces.

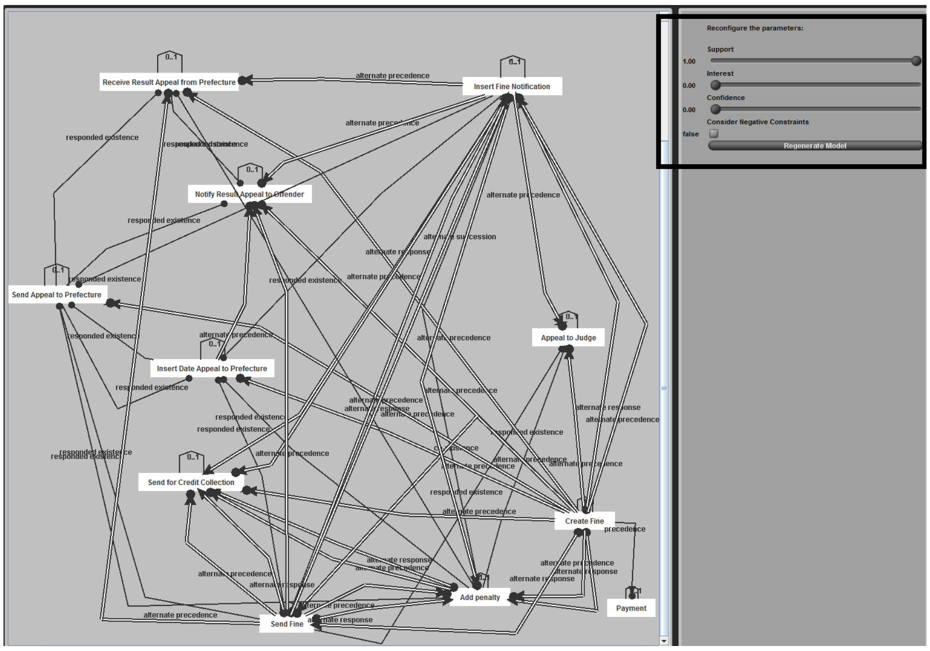
Skip Negative Constraints. When the process is characterised by parts with a rigid structure, the discovered model may blow up in term of presence of negative constraints. Therefore, analysts are provided with an option to not considering negative constraints, thus increasing the readability of the discovered models.

2.2 Output

Initially, the MINERful plug-in was executed skipping the negative constraints and using the following values for the other parameters: *(i)* support $= 0.50$, *(ii)* confidence $= 0.00$, *(iii)* interest factor $= 0.00$. The resulting declarative process model can be seen in Fig. 1.

The output view consists of two panels. The panel on the left-hand side contains the mined declarative process model. The user is free to relocate activities and constraints to manually improve the readability. The panel on the right-hand side allows the user to adjust the four parameters mentioned in Section 2.1 (see the area delimited by a black

[5] `http://dx.doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5`



Reconfigure the parameters:
Support 1.00
Interest 0.00
Confidence 0.00
Consider Negative Constraints false
Regenerate Model

Fig. 1: The resulting output screen from MINERful's where the model has been discovered by setting support to 0.5 and setting confidence and interest factor to 0.

rectangle in the figure). After the adjustment, the user can click on button *Regenerate Model* to mine a new model with the new values set for those parameters.

The model in Fig. 1 has been obtained by assigning value 0 to all parameters, except for support. This configuration has produced a cluttered declarative process model with many constraints, i.e. the model is probably overfitting the event log. The increase of the value of any parameter would generate a model with fewer constraints, thus probably reducing the overfitting problems and, also, improving the readability of the declarative process model. Of course, an excessive increase of any parameter may have a detrimental effect on the precision of the discovered model: the model may underfit the event log, allowing for too much behaviour. The declarative process model shown in Fig. 2 derives from the application of the following parameters: *(i)* support $= 0.70$, *(ii)* confidence $= 0.30$, *(iii)* interest factor $= 0.00$. A screencast illustrating the functioning of the MINERful plug-in is available at `https://svn.win.tue.nl/repos/prom/Documentation/DeclareMinerFul/screencast.mp4`.

2.3 Tool Maturity

The process models were discovered using a laptop equipped with an Intel Core i3 with 4GB of RAM. With this modest hardware, the MINERful plug-in was able to mine the model in less than 30 seconds using a real-size event log with 561,470 events belonging to 150,370 traces. This indicates that the MINERful plug-in has reached a

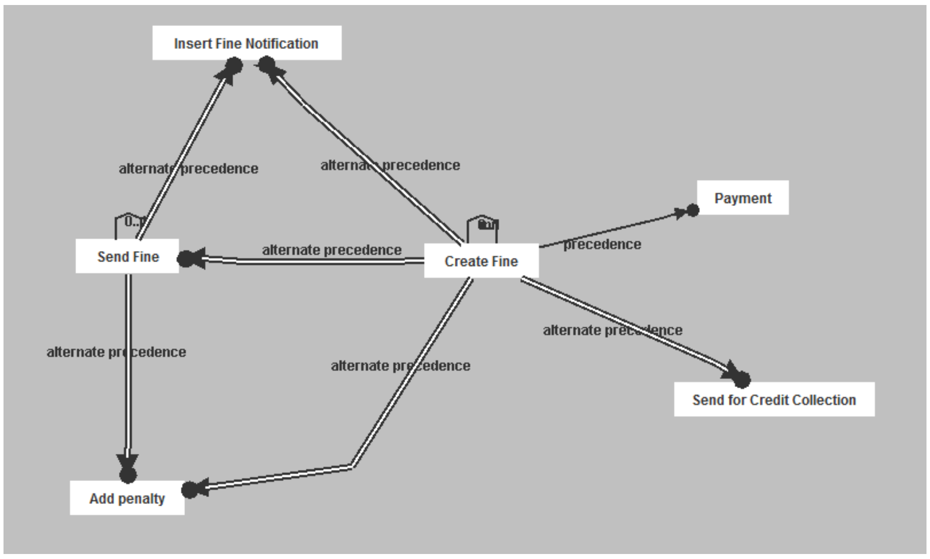


Fig. 2: The resulting output screen from MINERful's where the model has been discovered by setting support, confidence and interesting factor to 0.7, 0.3 and 0, respectively. The resulting model is certainly simpler but may be underfitting, hence not very precise.

large degree of maturity as it performs extremely well in terms of scalability. Also, the plug-in is integrated with the entire repertoire of techniques that are already available in ProM (see, e.g., [4]): the mined model can thus be later used for conformance checking, bottleneck analysis, improvement, and more.

Acknowledgements. The work of Dr. Di Ciccio and Dr. de Leoni has received funding from the EU Seventh Framework Programme under grant agreement 318275 (GET Service) and grant agreement 603993 (CORE), respectively.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: WS-FM. pp. 1–23 (2006)
3. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Trans. Manage. Inf. Syst. 5(4), 24:1–24:37 (2015)
4. Maggi, F.M.: Declarative process mining with the Declare component of ProM. In: BPM Demos. CEUR Workshop Proceedings (2013)
5. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative versus declarative process modeling languages: An empirical investigation. In: BPM Workshops. pp. 383–394 (2011)