Declarative Process Mining with MINERful, Reloaded

Cecilia Iacometta[1], Claudio Di Ciccio[1,*]

[1]Utrecht University, Utrecht, The Netherlands

Abstract

Declarative process specifications are defined by a set of behavioural constraints exerted over the execution of activities. These constraints are rooted in Linear Temporal Logic over finite traces. Declarative process mining encompasses a collection of techniques based on recorded process executions (event logs), with the objective of extracting such specifications and enhancing them. This paper presents MINERful, an open-source tool for declarative process mining in its new, reloaded version. It offers a range of functionalities revolving around the mining cycle of a process specification: its automated discovery from event logs, its simplification to remove redundancies and inconsistencies, its simulation to generate synthetic datasets, and fitness checking to gauge its level of conformance with logs. We showcase its usage with a real-world event log in the healthcare domain.

Keywords

Specification mining, Linear Temporal Logic over finite traces, Simulation, Automated reasoning

Metadata description	Value
Tool name	MINERful
Current version	2.0
Legal code license	Apache 2.0
Languages, tools and services used	Java
Supported operating environment	GNU/Linux, MacOS, Microsoft Windows
Download/Demo URL	github.com/process-in-chains/MINERful.git
Documentation URL	github.com/process-in-chains/MINERful/wiki
Source code repository	github.com/process-in-chains/MINERful/tree/master
Screencast video	youtu.be/a6jEWdgS_yY

1. Introduction

Process mining is a discipline that lies between data mining and computational intelligence, mainly focused on the discovery, monitoring, and checking of system behaviour based on recorded process executions [1]. Process mining serves as the basis for Business Process Management and beyond to understand the actual execution of a process, discover inefficiencies, and suggest improvements [2]. Over the last two decades, a novel approach to process management and mining emerged, namely the *declarative* approach [3]. Declarative process specifications consist of rules, also known as *constraints*, each restricting the behaviour of the process [4]: any run of the process is allowed, as long as none of the constraints is violated. Constraints regulate the temporal unfolding of a process without explicitly specifying the routing of process instances to meet those constraints. Declarative process mining aims at inferring and enhancing constraints constituting process specifications based on process data.

MINERful is a command-line tool for declarative process mining. A preliminary prototype was presented more than a decade ago with respect to the time of writing [5]. It has vastly evolved over the years, and new features have been added alongside updates on the core engine. This paper is the first one focussing on its fundamental functionalities, up to date with the most recent development.

Next, we provide preliminary information on declarative process specifications (Sect. 2). Equipped with these notions, we detail the four core functionalities offered by MINERful (Sect. 3). Finally, we draw conclusions by discussing the toolkit's maturity and availability (Sect. 4).

Proceedings of the Best BPM Dissertation Award, Doctoral Consortium, and Demonstrations & Resources Forum co-located with 23rd International Conference on Business Process Management (BPM 2025), Seville, Spain, August 31st to September 5th, 2025.

*Corresponding author. The authors contributed equally.

c.iacometta@students.uu.nl (C. Iacometta); c.diciccio@uu.nl (C. D. Ciccio)
0009-0007-4778-2950 (C. Iacometta); 0000-0001-5570-0475 (C. D. Ciccio)

2. Declarative Process Specifications in Brief

Declarative process specification languages like Declare [7] provide repertoires of constraint templates. A typical example of constraint template is Response, which is exerted on two activities, say a and c, and dictates that if a is executed, then c will eventually follow. As exemplified in Fig. 1, Declare constraints are formally grounded in Linear Temporal Logic on Finite Traces (LTL$_f$) [8], which can be turned into accepting finite state automata (FSAs) [9] and are hence legitimately expressible in the form of regular expressions [6]. Aside from the common set operators and propositional logic connectives, we use the following denotations in the figure: $\Sigma \supseteq \{a, c\}$ is the alphabet of propositional symbols; $\mathbf{G}$, $\mathbf{F}$, $\mathbf{X}$ are the LTL$_f$ temporal modalities "globally", "eventually", and "next", respectively; `.`, `[^a]`, and `*` indicate the occurrence of a symbol $x \in \Sigma$, an occurrence of $x \in \Sigma \backslash \{a\}$, and the Kleene-star operator to generate strings of any length, respectively. The detailed explanation of these expressions transcends the scope of this paper. We refer the interested reader to [6, 4] for further information.

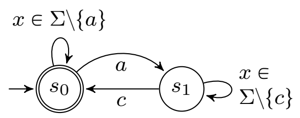
Finite state automaton	 $x \in \Sigma\backslash\{a\}$ a c s_0 s_1 $x \in \Sigma\backslash\{c\}$
LTL$_f$ formula	$\mathbf{G}\ (a \rightarrow \mathbf{F}\ \mathbf{X}\ c)$
Regular expr.	`[^a]*(a.*c)*[^a]*`

Figure 1: Response(a, c) formalised [6, 4]

Following the reactive-constraint form proposed in [10], any constraint κ can be written in an if-then form. What triggers the constraint, i.e., if(κ), is named *activation* [11]. For example, the occurrence of a in the trace is the activation of Response(a, c). The condition to be checked upon the satisfaction of the activation, i.e., then(κ), is the *target* of the constraint (e.g., the eventual occurrence of c in the trace for Response(a, c)). Table 1 provides a list of Declare constraints, alongside their template and a natural-language description of their activation and target.

Constraints are verified against finite *traces*, i.e., runs of a process. Each trace is typically abstracted as a finite sequence representing a run of the system's process (e.g., $\langle a, b, c, b, d\rangle$). Each occurrence of a symbol in the sequence (e.g., a) represents the execution of an activity (*event*) identified by that symbol. Event logs (or logs, for short) collect multiple runs of a process, and are usually abstracted as multi-sets of *traces*. The following is an example of log: $L \doteq \{\langle a, b, c, b, d\rangle^{10}, \langle c, a, b, d, b\rangle^{5}, \langle c, c, c, d\rangle^{1}, \langle a, b, c, a, c, a\rangle^{4}\}$. In L, e.g., trace $\langle a, b, c, b, d\rangle$ occurs 10 times. Overall, it consists of 20 traces with 111 events.

We can interchangeably interpret Declare constraints as *(i)* behavioural relationships among activities in a process specification, from a modelling perspective, or *(ii)* rules governing the occurrence of events in event logs' traces, from a mining perspective. Identifying rules that define the most relevant boundaries of the process behaviour registered in event logs' traces is the core objective of declarative process mining. To quantify the interestingness of those rules, a number of measures have been introduced, inspired by the literature in association rule mining [12, 13]. Here we specifically focus on three of those. For their computation, we resort to two possible computation schemes [4]: the *event-based* interpretation considers the event as the probabilistic case to gauge satisfaction; the *trace-based* interpretation considers the whole trace as such, in a more coarse-granular fashion. Table 2 provides formulas for their computation. In the table, we denote with $\#_{\mathrm{e}}$ a function that takes an event log (e.g., L) and a statement (e.g., if(κ), the activation of constraint κ, or $\top$, i.e., the logical *true* value), and returns the count of events that satisfy the statement in the formula. We use $\#_{\mathrm{t}}$ to indicate a function that also takes a log and a statement as an input but returns the traces that satisfy the statement in the log. The computations can be summarised as follows: (1) *support* is the proportion of cases that satisfy the activation and the target in the log; (2) *coverage* is the proportion of cases that satisfy the activation in the log; (3) *confidence* is the proportion of cases that satisfy the activation and the target over the cases that satisfy the activation. Table 3 shows their computation for Response(a, c) given the above event log L. For example, the trace-based support is 0.5 because in 10 cases out of 20 all occurrences of a are eventually followed by c in the trace. The denominator of the trace-based confidence is 19 because a never occurs in a trace. The event-based support considers instead that in 18 events out of 111, a occurs and is eventually followed by c. These notions serve as the basis for the next section, elucidating the core features offered by MINERful for declarative process mining.

Table 1: A selection of Declare constraints

Constraint κ	**Template**	**Activation** $\textsc{if}(\kappa)$	**Target** $\textsc{then}(\kappa)$
$\textsc{Init}(x)$	Init	The trace starts	x is the first activity in the trace
$\textsc{AtMostOne}(x)$	AtMostOne	The trace starts	x occurs at most once in the trace
$\textsc{Response}(x,y)$	Response	x occurs	y eventually follows a
$\textsc{ChainResponse}(x,y)$	ChainResponse	x occurs	y immediately follows x
$\textsc{Precedence}(x,y)$	Precedence	y occurs	x precedes at least once y in the trace
$\textsc{AlternatePrecedence}(x,y)$	AlternatePrecedence	y occurs	x precedes y in the trace and x cannot recur between x and y
$\textsc{ChainPrecedence}(x,y)$	ChainPrecedence	y occurs	x immediately precedes y
$\textsc{CoExistence}(x,y)$	CoExistence	x or y occur	x and y occur in the trace

Table 2: Computation of interestingness measures for a declarative constraint κ given an event log L

Based on	**Confidence**	**Coverage**	**Support**
Events	$\frac{\#_{\mathrm{e}}(L,\textsc{if}(\kappa)\wedge\textsc{then}(\kappa))}{\max\{1,\#_{\mathrm{e}}(L,\textsc{if}(\kappa))\}}$	$\frac{\#_{\mathrm{e}}(L,\textsc{if}(\kappa))}{\max\{1,\#_{\mathrm{e}}(L,\top)\}}$	$\frac{\#_{\mathrm{e}}(L,\textsc{if}(\kappa)\wedge\textsc{then}(\kappa))}{\max\{1,\#_{\mathrm{e}}(L,\top)\}}$
Traces	$\frac{\#_{\mathrm{t}}(L,\textsc{if}(\kappa)\wedge\textsc{then}(\kappa))}{\max\{1,\#_{\mathrm{t}}(L,\textsc{if}(\kappa))\}}$	$\frac{\#_{\mathrm{t}}(L,\textsc{if}(\kappa))}{\max\{1,\#_{\mathrm{t}}(L,\top)\}}$	$\frac{\#_{\mathrm{t}}(L,\textsc{if}(\kappa)\wedge\textsc{then}(\kappa))}{\max\{1,\#_{\mathrm{t}}(L,\top)\}}$

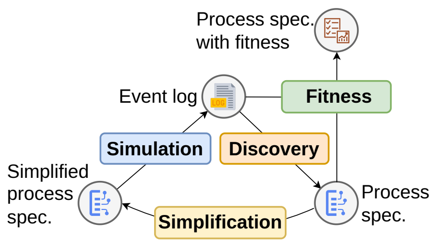


Figure 2: MINERful functionalities

Table 3: $\textsc{Response}(a,c)$ measures on $L \doteq \{\langle a,b,c,b,d\rangle^{10},\langle c,a,b,d,b\rangle^{5},\langle c,c,c,d\rangle^{1},\langle a,b,c,a,c,a\rangle^{4}\}$

	Confidence	**Coverage**	**Support**
Event based	$\frac{1\times10+2\times4}{1\times10+1\times5+3\times4}=\frac{18}{27}\approx0.67$	$\frac{1\times10+1\times5+3\times4}{5\times10+5\times5+4\times1+8\times4}=\frac{27}{111}\approx0.24$	$\frac{1\times10+2\times4}{5\times10+5\times5+4\times1+8\times4}=\frac{18}{111}\approx0.16$
Trace based	$\frac{10}{10+5+4}=\frac{10}{19}\approx0.53$	$\frac{10+5+4}{10+5+1+4}=\frac{19}{20}=0.95$	$\frac{10}{10+5+1+4}=\frac{10}{20}=0.5$

3. The MINERful Toolkit

Figure 2 displays a schematic overview of the core functionalities offered by MINERful: discovery, simplification, fitness checking, and simulation. For all of those, MINERful provides dedicated APIs, intended for its usage as a library, and executable scripts, designed for client- or server-side experiments. Next, we outline how MINERful realises those functionalities, and showcase them with a real-world log.

Automated discovery. Process discovery consists of extracting a Declare specification that effectively represents the traces found in an event log. To cater for scalability, MINERful applies a two-staged approach. In the first phase, it builds a knowledge-base extracting the quantitative statistics about co-occurrences of activities. In the second phase, it queries that knowledge base to compute the aforementioned interestingness measures related to candidate constraints, and filters out from the results the constraints having those measures below user-defined thresholds. MINERful accepts as an input event logs stored as eXtensible Event Stream (XES), Mining eXtensible Markup Language (MXML) or text files (useful for rapid testing). In addition to the standard process discovery, MINERful allows the miner to run on consecutive sub-logs, processed in a shifting-window fashion. The obtained results can be saved in tabular formats (CSV), and tree-like structures (JSON). MINERful can print the output in the form of finite state automata (saved as Graphviz DOT files), POSIX regular expressions, and NuSMV LTL$_f$ formulae (see Fig. 1).

Simplification. Simplification of a process specification means reducing the constraints therein to purge redundancies and inconsistencies while keeping the overall behaviour intact. MINERful supports this operation by applying the method described in [6], including the sorting heuristics for constraint fetching, and the single- or double-pass mode (as a trade-off between computation time and accuracy).

Simulation. Starting from a given process specification, MINERful enables the creation of synthetic event logs (in XES, MXML, and text formats, to allow for a subsequent re-descovery task). MINERful allows the user to indicate whether some constraints should be violated, and in which percentage, to cater for a what-if analysis of partial non-conformity of runs with the input specification.

Table 4: A sample of the process specification discovered by MINERful from the Sepsis event log [16]

Constraint	**Event-based measures**			**Trace-based measures**		
	Confidence	**Coverage**	**Support**	**Confidence**	**Coverage**	**Support**
Init(ER Registration)	0.948	0.069	0.065	0.948	1.000	0.948
ChainResponse(ER Registration, ER Triage)	0.925	0.069	0.064	0.925	1.000	0.925
ChainPrecedence(ER Triage, ER Sepsis Triage)	0.863	0.069	0.059	0.863	0.999	0.862
AtMostOne(IV Antibiotics)	1.000	0.069	0.069	1.000	1.000	1.000
CoExistence(IV Antibiotics, IV Liquid)	0.956	0.104	0.099	0.915	0.784	0.717
Precedence(IV Antibiotics, Admission NC)	0.874	0.078	0.068	0.859	0.762	0.654
AlternatePrecedence(Admission NC, Release A)	0.999	0.044	0.044	0.999	0.639	0.638
Thresholds	0.850	0.040	0.040	0.850	0.125	0.125

Fitness checking. To evaluate how closely an event log matches a given process specification, MINERful lets the user compare the log and process behaviour by computing the fitness of the latter with the former. We consider fitness as defined in [14] while our computation follows the technique described in [15], to distinguish constraint satisfactions between vacuous and relevant [11, 10], whereby the former occur when the constraint's activation is not satisfied.

Showcase with a real-world event log. We showcase the usage of MINERful with a real-world event log in the healthcare domain: Sepsis [16]. Table 4 shows the results attained by running its automated discovery and simplification engines, discovered using a laptop equipped with an Apple M1 with 8GB of RAM in 726 msec. We set the thresholds as indicated in the table, and applied the simplification post-processing with double-pass. The constraints in Tab. 4 show the consistency of MINERful discovered process specification with the Sepsis process model described in [17]. The process begins with the patient's registration (ER Registration), which is immediately followed by patient's triage (ER Triage). The patient's Sepsis triage (ER Sepsis Triage) requires that step immediately before. Moreover, antibiotics are administered (IV Antibiotics) at most once, together with liquids (IV Liquid) before the admission to normal care ward (Admission NC). Every time a patient is discharged (Release A), a previous admission to the normal care is necessary. The average fitness of the specification is 0.93. Finally, we created an event log including 10 % of traces violating Init(ER Registration), to simulate runs that fully comply with the discovered specification, except a few traces in which the patients' data were not recorded at the start due to a rush dictated by extreme emergency. The log can be used for variant analysis and is provided in our code repository, described next.

4. Maturity and Availability

MINERful has been adopted in process mining research both as a stand-alone tool and as a library. Originally developed as the mining core of a tool to extract control flows from knowledge-workers' email [18], it was later released as a separate software prototype, and integrated in the open-source process mining platform ProM [19]. Following further refinements and functionality extensions [20, 6], its APIs were developed to include it in the rule mining RuM suite [21]. Thereafter, tools began including it as a third-party library. An example is offered by the Visual Drift Detection system VDD [22], for which the aforementioned window-shifting analysis of sub-logs was initially developed. Also, MINERful offered the codebase forked by another LTL$_f$ specification mining toolkit, namely Janus [10]. The various integrations with different tools and research studies provide evidence of MINERful's versatility.

To date, the description of the different functionalities offered by MINERful was scattered among different publications, and recent advancements and updates of the tool went undocumented. This demo paper (bundled with the additional online material illustrated next) is intended to provide a point of reference for researchers and practitioners who wish to utilise MINERful for their process analytic tasks. Further functionalities and extensions are in preparation at the time of writing, and will be included in the project code repository available at github.com/process-in-chains/MINERful.

The MINERful toolkit's source code can be downloaded from github.com/process-in-chains/MINERful/

tree/master. Development builds are available in the dev branch. The Wiki of the repository guides the installation and usage of MINERful's functionalities described in Sect. 3. The video demonstration of MINERful is included in the repository's read-me file, and can be watched at youtu.be/a6jEWdgS_yY.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] W. M. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, Springer, 2022.
[2] M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers, Fundamentals of Business Process Management, Second Edition, Springer, 2018.
[3] M. Pesic, H. Schonenberg, W. M. P. van der Aalst, DECLARE: Full support for loosely-structured processes, in: EDOC, 2007, pp. 287–300.
[4] C. Di Ciccio, M. Montali, Declarative process specifications: reasoning, discovery, monitoring, in: [1], 2022, pp. 108–152.
[5] C. Di Ciccio, M. Mecella, Mining constraints for artful processes, in: BIS, Springer, 2012, pp. 11–23.
[6] C. Di Ciccio, F. M. Maggi, M. Montali, J. Mendling, Resolving inconsistencies and redundancies in declarative process models, Inf. Syst. 64 (2017) 425–446.
[7] W. M. Van Der Aalst, M. Pesic, DecSerFlow: Towards a truly declarative service flow language, in: WS-FM, Springer, 2006, pp. 1–23.
[8] G. De Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: IJCAI, 2013, pp. 854–860.
[9] G. De Giacomo, R. De Masellis, M. Montali, Reasoning on LTL on finite traces: Insensitivity to infiniteness, in: AAAI, 2014, pp. 1027–1033.
[10] A. Cecconi, C. Di Ciccio, G. De Giacomo, J. Mendling, Interestingness of traces in declarative process mining: The Janus LTLp$_f$ approach, in: BPM, Springer, 2018, pp. 121–138.
[11] F. M. Maggi, A. J. Mooij, W. M. P. van der Aalst, User-guided discovery of declarative process models, in: CIDM, IEEE, 2011, pp. 192–199.
[12] L. Geng, H. J. Hamilton, Interestingness measures for data mining: A survey, ACM Comput. Surv. 38 (2006) 9.
[13] A. Cecconi, G. De Giacomo, C. Di Ciccio, F. M. Maggi, J. Mendling, Measuring the interestingness of temporal logic behavioral specifications in process mining, Inf. Syst. 107 (2022) 101920.
[14] M. De Leoni, F. M. Maggi, W. M. van der Aalst, Aligning event logs and declarative process models for conformance checking, in: BPM, Springer, 2012, pp. 82–97.
[15] C. Di Ciccio, F. M. Maggi, M. Montali, J. Mendling, On the relevance of a business constraint to an event log, Inf. Syst. 78 (2018) 144–161.
[16] F. Mannhardt, Sepsis cases, 2016. doi:10.4121/UUID:915D2BFB-7E84-49AD-A286-DC35F063A460.
[17] F. Mannhardt, D. Blinde, Analyzing the trajectories of patients with sepsis using process mining, in: RADAR+ EMISA 2017, CEUR-ws. org, 2017, pp. 72–80.
[18] C. Di Ciccio, M. Mecella, Mining artful processes from knowledge workers' emails, IEEE Internet Computing 17 (2013) 10–20.
[19] C. Di Ciccio, M. H. Schouten, M. de Leoni, J. Mendling, Declarative process discovery with MINERful in ProM, in: BPM (Demos), CEUR-WS.org, 2015, pp. 60–64.
[20] C. Di Ciccio, M. L. Bernardi, M. Cimitile, F. M. Maggi, Generating event logs through the simulation of Declare models, in: EOMAS, Springer, 2015, pp. 20–36.
[21] A. Alman, C. D. Ciccio, D. Haas, F. M. Maggi, A. Nolte, Rule mining with RuM, in: ICPM, IEEE, 2020, pp. 121–128.
[22] A. Yeshchenko, C. D. Ciccio, J. Mendling, A. Polyvyanyy, Visual drift detection for event sequence data of business processes, IEEE Trans. Vis. Comput. Graph. 28 (2022) 3050–3068.