

# Declare4Py: A Python Library for Declarative Process Mining

Ivan Donadello<sup>1,\*</sup>, Ph.D., Francesco Riva<sup>1</sup>, M.S., Fabrizio Maria Maggi<sup>1</sup>, Ph.D. and Aladdin Shikhizada<sup>2</sup>, M.S.

<sup>1</sup>Free University of Bozen-Bolzano, Bolzano, Italy

<sup>2</sup>Visioncraft OÜ, Tallin, Estonia

## Abstract

In process mining, procedural process models can be difficult to manage when the process is unpredictable and characterized by many possible exceptions since they can easily become unreadable. Declarative languages, instead, model the process by imposing logical constraints on the process behavior and are suitable to represent variable processes in a compact way. Declare is the reference declarative language in the BPM community. Although several Java tools are available for process analysis based on Declare, a library implementing process mining tasks with Declare in Python is still missing. Therefore, in this paper, we present Declare4Py, the first Python package that offers support for declarative process mining. Declare4Py includes methods for conformance checking, process discovery and query checking.

## Keywords

Declare, Conformance Checking, Process Discovery, Query Checking, Python API, Declarative Process Mining

## 1. Introduction

Process mining [1] focuses on the analysis of business processes based on event logs that contain information about the process executions. A key component in process mining is a process model that is a formal representation of the process in a standard format. Procedural models require to define the whole control-flow of the process step-by-step thus making procedural process mining not suitable for processes with a high number of different branches and exceptions. Declarative process models are, instead, easier to manage as they just encode a set of constraints that the process should follow. While procedural models can be designed and analyzed using several available commercial and academic process mining tools<sup>12345</sup>, this variety is lacking for


---


*The BPM 2022 Demos & Resources Forum. 20<sup>th</sup> Business Process Management Conference, Münster, Germany, September 11-16, 2022*

\*Corresponding author.

✉ ivan.donadello@unibz.it (I. Donadello); Francesco.Riva@unibz.it (F. Riva); maggi@inf.unibz.it (F. M. Maggi); aladdin.shikhizada@gmail.com (A. Shikhizada)

🆔 0000-0002-0701-5729 (I. Donadello); 0000-0002-9089-6896 (F. M. Maggi)

 © 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

<sup>1</sup>apromore.com/

<sup>2</sup>celonis.com/

<sup>3</sup>fluxicon.com/disco/

<sup>4</sup>processmining.org

<sup>5</sup>pm4py.fit.fraunhofer.de/

declarative models where only few (Java-based) tools and libraries are available<sup>6</sup> [2, 3].

In this paper, we present Declare4Py, a novel and easy-to-use Python library that implements a set of APIs covering the main declarative process mining tasks based on Declare [4]. In particular, the language employed is MP-Declare [5], the multi-perspective extension of Declare that supports data- and time-aware constraints along with control-flow constraints. Our contribution to the BPM community is the first Python library with APIs for conformance checking, process discovery and query checking based on Declare models. Being a Python library, Declare4Py can be easily integrated with the main Machine Learning frameworks such as SKlearn, Tensorflow and Pytorch, and, as a library, can be easily invoked via code to conduct large experimentations. We also stress the fact that the query checking functionality provided by Declare4Py is novel and not available in the existing tools for declarative process mining. We compared the Declare4Py performance with RuM [2], a Java-based tool for declarative process mining on the core task of conformance checking, achieving better computational times.

This first release of Declare4Py is online in a GitHub repository available at <https://github.com/francxx96/declare4py>. The repository contains the code and some tutorials in Jupyter notebooks (<https://github.com/francxx96/declare4py/tree/main/tutorials>) showing how to use Declare4Py using the well-known Sepsis cases log.<sup>7</sup> A video that overviews the package is available at <https://youtu.be/hJhgoqFLM7s>.

## 2. Overview of the Declare4Py Features

Declare4Py has been designed to analyze event logs using declarative, constraint-based process models. It relies on well-known standards for input and output files, such as XES [6] for event logs and decl [7] for the Declare models. This ensures its interoperability with other libraries and tools.

We briefly recall here some preliminary definitions. A *trace*  $\sigma$  is an execution of a business process. A trace contains a sequence of events where each event is related to the execution of an activity  $a \in A$  (with  $A$  the set of all possible activities), performed at time  $t$  with a (possible) set of other attributes a.k.a. the payload of the event. A Declare *model*  $\mathcal{M} = \{\varphi_1, \varphi_2, \dots\}$  is a set of Declare constraints instantiation of parameterized templates [4]. We indicate the set of Declare templates with  $\mathcal{A}$ . A trace *satisfies* a Declare model ( $\sigma \models \mathcal{M}$ ), when the trace satisfies each constraint  $\varphi \in \mathcal{M}$ , i.e.,  $\forall \varphi \in \mathcal{M}, \sigma \models \varphi$ . A *log*  $L$  is a multi-set of traces.

**Conformance Checking.** Given a log  $L$  of traces  $\sigma_i$  and an MP-Declare model  $\mathcal{M}$ , the conformance checking task checks, for all the traces  $\sigma_i \in L$ , whether, for all constraints  $\varphi \in \mathcal{M}$ ,  $\sigma_i \models \varphi$  holds. Declare4Py implements the conformance checking task using the approach presented in [5] that takes an MP-Declare model and a log as inputs and returns the number of activations, fulfillments, and violations for each constraint in the input model and for each trace in the input log. These results are listed in a Python data structure indexed by trace identifier. Therefore, the user can easily query such data structure to retrieve or aggregate information.

---

<sup>6</sup>[rulemining.org](http://rulemining.org)

<sup>7</sup>[https://data.4tu.nl/articles/dataset/Sepsis\\_Cases\\_-\\_Event\\_Log/12707639](https://data.4tu.nl/articles/dataset/Sepsis_Cases_-_Event_Log/12707639)

**Process Discovery.** Given a log  $L$  of traces  $\sigma_i$  and a support threshold  $th_s$ , the process discovery task returns a Declare<sup>8</sup> model  $\mathcal{M}$  of constraints satisfied by a percentage of traces in  $L$  higher than or equal to  $th_s$ . More formally:

$$\mathcal{M} = \{\varphi : |\{\sigma \in L : \sigma \models \varphi\}|/|L| \geq th_s\}. \quad (1)$$

Declare4Py implements the approach presented in [8] that consists of two steps, i.e., (1) the discovery of frequent (pairs of) activities from  $L$ ; (2) the construction of  $\mathcal{M}$  from this set. The set of frequent (pairs of) activities from  $L$  is built with the Apriori algorithm [9] by computing the frequent itemsets of activities of length 1 and 2. These itemsets are used to build a set of candidate Declare constraints  $\mathcal{C}$  obtained by instantiating the templates in  $\mathcal{A}$  with the activities belonging to each itemset.  $\mathcal{M} \subseteq \mathcal{C}$  is then computed by selecting the constraints  $\varphi \in \mathcal{C}$  such that  $|\{\sigma \in L : \sigma \models \varphi\}|/|L| \geq th_s$ . The results are returned in a Python data structure containing, for each constraint in  $\mathcal{M}$ , the traces that satisfy it. A Declare4Py function allows the user to filter such data structure to retrieve the most relevant (i.e., the most frequently satisfied) constraints. The discovered model can be exported as a `decl` file.

**Query Checking.** This task takes as input a log  $L$  of traces  $\sigma_i$ , a support threshold  $th_s$ , and an MP-Declare query  $q$ , i.e., an MP-Declare constraint in which the activation and/or the target activity are unspecified. For example, constraint `Response(?A, ER Triage)`<sup>9</sup> contains a placeholder for the activation activity, whereas `Response(?A, ?T)` contains placeholders for both activation and target. Let  $Vars$  be the set of placeholders of a Declare query and  $\lambda : Vars \rightarrow A$  be an assignment function that assigns placeholders to activities. The query checking task returns the set of assignments  $\Lambda = \{\lambda_1, \lambda_2, \dots\}$  such that the input query  $q$  instantiated using the assignments in  $\Lambda$  is satisfied by a percentage of traces in  $L$  higher than or equal to  $th_s$ . More formally:

$$\Lambda = \{\lambda_i : |\{\sigma \in L : \sigma \models q[\lambda_i]\}|/|L| \geq th_s\}. \quad (2)$$

Declare4Py returns a data structure containing the assignments.

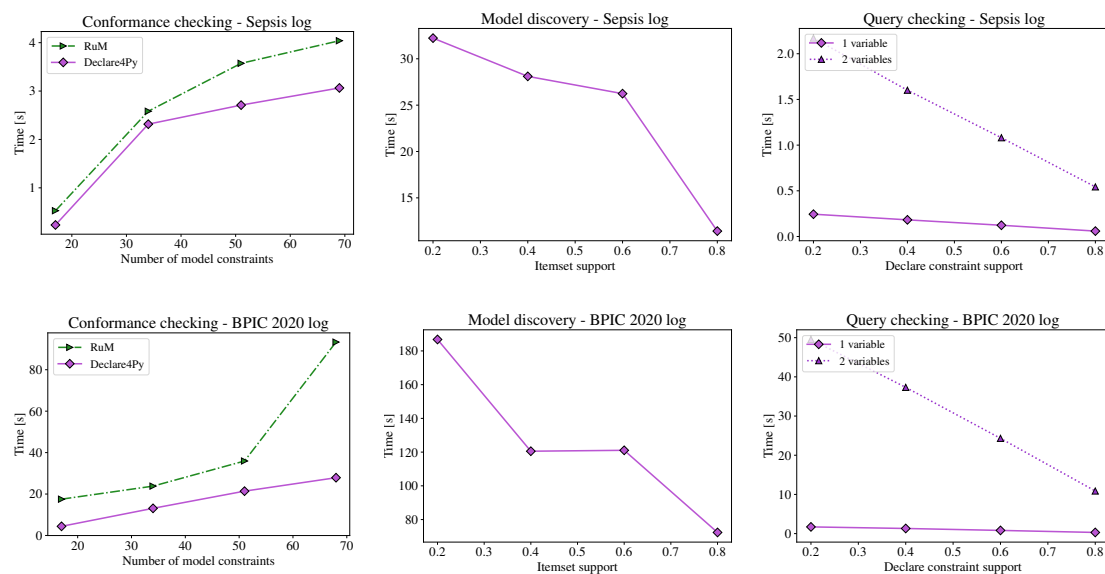
### 3. Performance

We tested the computational time performance of Declare4Py on the above tasks under different conditions using the Sepsis cases log<sup>7</sup> and the log provided for the annual Business Process Intelligence Challenge (BPIC) in 2020<sup>10</sup>. The core task is conformance checking as process discovery and query checking are built on top of it. Therefore, we compared the conformance checking task (based on [5]) implemented both in Declare4Py and in RuM [2], increasing the number of Declare constraints in the input model. The performance of the discovery and the query checking tasks is, instead, computed for different support values ranging in the set

<sup>8</sup>The process discovery functionality, differently from the conformance checking and the query checking tasks, is data-agnostic.

<sup>9</sup>For simplicity, we do not define data and time conditions in this example. However, fully defined data and time conditions can be specified in the query.

<sup>10</sup><http://icpmconference.org/2020/wp-content/uploads/sites/4/2020/03/InternationalDeclarations.xes.gz>



**Figure 1:** Declare4Py shows better performance than RuM for conformance checking for both the Sepsis log (above) and the BPIC 2020 log (below).

$\{0.2, 0.4, 0.6, 0.8\}$ .<sup>11</sup> For query checking, we used the Chain Response template and performed two tests. In the first one, we fixed the activation activity leaving the target unspecified; in the second test, we left both the activation and the target activities unspecified.

The results of our experiments are reported in Figure 1. Declare4Py presents slightly lower computational times with respect to RuM for conformance checking on small models. However, as the model grows in the number of constraints, the computational times diverge. This is particularly evident in the BPIC 2020 case. The computational time for the discovery and the query checking tasks decreases when the support increases, since a higher support implies less candidates to check. The computational time for the query checking task is obviously higher when two placeholders have to be assigned.

## 4. Maturity and Future Remarks

Declare4Py has been used for deviance mining in [11] and as a tool for a new feature encoding for business process analysis using Machine Learning methods [12]. This first release can be improved both in terms of performance and number of functionalities. As future work, we plan to increase the Declare4Py performance by implementing optimization techniques, such as multi-threading, and by using the Numba library,<sup>12</sup> which translates, at runtime, Python code into optimized machine code by using the industry-standard LLVM [13]. The Declare4Py functionalities will be improved by including state-of-the-art process mining algorithms. For

<sup>11</sup>In this case, a comparison with RuM would not be fair as this tool implements the optimization technique based on multi-threading, presented in [10], which is currently not developed in Declare4Py.

<sup>12</sup><https://numba.pydata.org/>

conformance checking, we plan to include the techniques presented in [14] and in [15], while, for process discovery, we will implement the techniques introduced in [16] and [17].

## Acknowledgments

The work of Francesco Riva is supported by the UNIBZ project PRISMA.

## References

- [1] W. M. P. van der Aalst, *Process Mining - Data Science in Action*, Springer, 2016.
- [2] A. Alman, C. Di Ciccio, D. Haas, F. M. Maggi, A. Nolte, Rule mining with RuM, in: 2nd International Conference on Process Mining, ICPM 2020, 2020, pp. 121–128.
- [3] C. Di Ciccio, M. Mecella, On the discovery of declarative control flows for artful processes, *ACM Trans. Manag. Inf. Syst.* 5 (2015) 24:1–24:37.
- [4] M. Pesic, H. Schonenberg, W. M. P. van der Aalst, DECLARE: full support for loosely-structured processes, in: *EDOC*, IEEE Computer Society, 2007, pp. 287–300.
- [5] A. Burattin, F. M. Maggi, A. Sperduti, Conformance checking based on multi-perspective declarative process models, *Expert Syst. Appl.* 65 (2016) 194–211.
- [6] C. W. Gunther, H. Verbeek, *XES-standard definition* (2014).
- [7] V. Skydanienco, C. Di Francescomarino, C. Ghidini, F. M. Maggi, A tool for generating event logs from multi-perspective declare models, in: *BPM (Dissertation/Demos/Industry)*, volume 2196 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2018, pp. 111–115.
- [8] 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: *CAiSE*, 2012, pp. 270–285.
- [9] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: *VLDB*, 1994, pp. 487–499.
- [10] F. M. Maggi, C. Di Ciccio, C. Di Francescomarino, T. Kala, Parallel algorithms for the automated discovery of declarative process models, *Inf. Syst.* 74 (2018) 136–152.
- [11] G. Bergami, C. Di Francescomarino, C. Ghidini, F. M. Maggi, J. Puura, Exploring business process deviance with sequential and declarative patterns, *CoRR abs/2111.12454* (2021).
- [12] C. Di Francescomarino, I. Donadello, C. Ghidini, F. M. Maggi, W. Rizzi, Making sense of temporal data: the DECLARE encoding, in: *PMAI@IJCAI*, CEUR-WS.org, 2022.
- [13] J. Lee, C. Hur, R. Jung, Z. Liu, J. Regehr, N. P. Lopes, Reconciling high-level optimizations and low-level code in LLVM, *Proc. ACM Program. Lang.* 2 (2018) 125:1–125:28.
- [14] M. de Leoni, F. M. Maggi, W. M. P. van der Aalst, Aligning event logs and declarative process models for conformance checking, in: *BPM*, 2012, pp. 82–97.
- [15] G. Bergami, F. M. Maggi, A. Marrella, M. Montali, Aligning data-aware declarative process models and event logs, in: *BPM*, Springer, 2021, pp. 235–251.
- [16] C. Di Ciccio, F. M. Maggi, J. Mendling, Efficient discovery of target-branched Declare constraints, *Inf. Syst.* 56 (2016) 258–283.
- [17] V. Leno, M. Dumas, F. M. Maggi, M. La Rosa, A. Polyvyanyy, Automated discovery of declarative process models with correlated data conditions, *Inf. Syst.* 89 (2020) 101482.