I-DiDD: A Tool for the Incremental Discovery of Data-Aware Declare Models

Tedi Ibershimi[1,*], Fabrizio Maria Maggi[2] and Marco Comuzzi[3]

[1]Landesverband der Tourismusorganisationen Südtirols, Bolzano, 39100, Italy
[2]Free University of Bozen-Bolzano, Bolzano, 39100, Italy
[3]Ulsan National Institute of Science and Technology, Ulsan, 44919, South Korea

Abstract

Incremental Process Discovery is considered crucial in various safety-critical real-life scenarios, where discovering process models using logs collected over a long period of time may pose unacceptable risks. In such contexts, real-time insights are essential to enable appropriate decisions and interventions. To support this need, we present *I-DiDD*, a tool designed for the incremental discovery of Data-Aware Declare models using a novel approach based on finite state automata, equipped with a dynamic dashboard that provides real-time feedback to the user on the discovery results. The dashboard displays event log statistics and the discovered models, in both a model-based and a tabular view.

Keywords

Declarative Process Mining, Data-Aware Declare, Process Discovery, Event Streams

1. Introduction

A frequently performed activity in process mining is *Process Discovery*, which aims to create a representative model of all observed process executions. In recent years, process discovery has become increasingly important in the context of *incremental process mining*. Discovering process models in real-time is useful in a variety of real-life scenarios where an immediate understanding of ongoing processes is required. In such organizational contexts, immediate intervention is essential, and delays in understanding processes can pose unacceptable risks or lead to grave consequences.

Many approaches have been proposed in the literature for incremental process model discovery, spanning both the declarative and procedural paradigms of process mining. For declarative process models, Declare [1] is considered the most widely adopted language among the proposed solutions. It expresses process behavior as a set of constraints that process executions must not violate. Recently, extensions of Declare have also been proposed that go beyond control-flow constraints, incorporating additional perspectives such as resources, time, and data.

This work focuses on Data-Aware Declare [2], an extension of the Declare modeling language that enriches control-flow constraints with data extracted from process executions. In contrast to previous works [3], which addressed only standard Declare constraints using ad hoc and non-formal verification mechanisms, the proposed application adopts an algorithmic approach based on finite state automata to formally verify all Data-Aware Declare constraints[1]. Specifically, it includes three core algorithms, each responsible for discovering constraints based on the type of knowledge required to check *constraint activations*: past, future, and past+future knowledge.

The discovered model includes the full Data-Aware Declare catalog introduced in [2], which natively encompasses all original Declare constraints extended with data at the activations. In particular, the discovered models include the following constraints: *Response*, *Chain Response*, *Alternate Response*,

Proceedings of the BPM 2026 Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Forum (BPM 2026) , Toronto, Canada, September 27th to October 2nd, 2026.

*Corresponding author.
tedi.ibershimi@lts.it (T. Ibershimi); maggi@inf.unibz.it (F. M. Maggi); mcomuzzi@unist.ac.kr (M. Comuzzi)
0009-0005-5601-1942 (T. Ibershimi); 0000-0002-9089-6896 (F. M. Maggi); 0000-0002-6944-4705 (M. Comuzzi)

[1]A first attempt to implement an approach for the online discovery of Data-Aware Declare was presented in [4], but only two templates were addressed in that work, each with a dedicated ad hoc algorithm.

Precedence, *Chain Precedence*, *Alternate Precedence*, *Not Response*, *Not Chain Response*, *Not Precedence*, *Not Chain Precedence*, *Responded Existence*, and *Not Responded Existence*.

The discovery procedure is implemented and integrated into a GUI in the application presented in this paper, named *I-DiDD* (Incremental DIscovery of Data Declare). This tool is the first and only one that provides full support for discovering Data-Aware Declare models in a streaming setting. The source code is available on GitHub[2], and a video demonstration of the application is available online[3].

2. Tool description

Figure 1 illustrates the overall structure of *I-DiDD*, highlighting the flow of data, the relationships among components, and the external dependencies (represented by dashed lines). Each component of the system is responsible for a specific aspect of the application, ranging from message serialization and deserialization to data source replay, model discovery, and final display. This modular design supports the independent development and maintenance of system components.

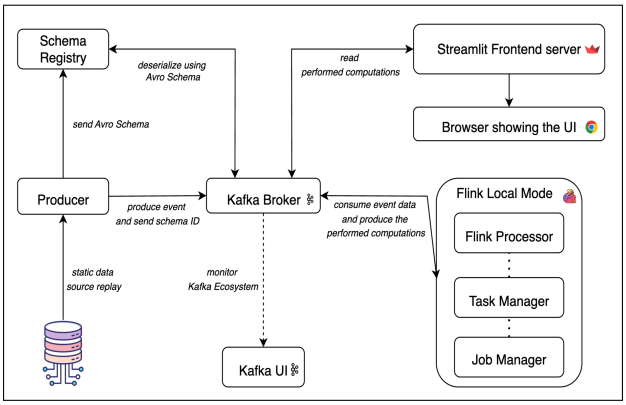

Figure 1: System Architecture Overview.

The entry point of the system is an *Apache Kafka event producer*, which reads and sorts events from the input data source by their timestamps to simulate a live stream. The sorted events are sent at fixed intervals in the *Avro format* to a *Kafka Topic* responsible for handling the *raw events*. A *Schema Registry Server* hosts the Avro schema used for serializing and deserializing event messages, which is customized for XES[4] event logs. The core processing is performed by an *Apache Flink processor*, which consumes the raw events, computes statistics, and performs Data-Aware Declare discovery. These results are then published to a Kafka Topic different from the one containing the raw events. However, both topics are managed by the same *Kafka broker*. Finally, a *Streamlit-based frontend server* retrieves the data through a background Kafka consumer and displays them in the dashboard. The frontend is solely responsible for presenting the data to the user, while all processing steps are handled by the processor.

Discovering *Data-Aware Declare* process models in a streaming environment introduces significantly greater challenges than those encountered in an offline environment. Transitioning from the offline to the online scenario requires re-engineering the existing pipeline phases to make them suitable for online processing, as well as introducing new components. The final proposed pipeline consists of the following phases (see Figure 2): (1) stream mining using *Lossy Counting*, (2) *Apriori*-based candidate generation, (3) control-flow discovery, (4) data condition learning, (5) rule extraction, and (6) presentation of the discovered models to the end user.

Each event is passed to the stream mining procedure using *Lossy Counting* [5] at the *trace level*. Specifically, it tracks the number of times each trace identifier occurs in the event stream and performs Data-Aware Declare model discovery only for events belonging to the most frequent traces. As soon

[2]https://github.com/tedib04/I-DIDD
[3]https://www.youtube.com/watch?v=clAVO2i8nzI
[4]https://www.xes-standard.org/

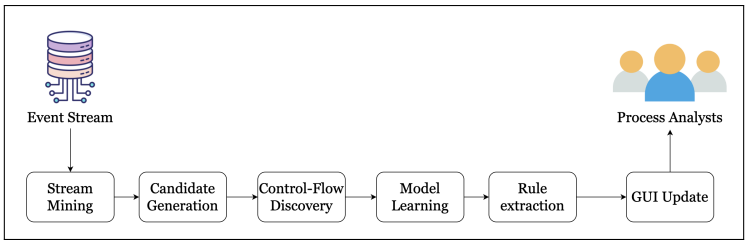

Figure 2: Online Data-Aware Declare discovery pipeline.

as two different activities appear in the stream, the *Apriori-based candidate generation* component [6] produces the corresponding binary constraint instances for all DECLARE patterns. These candidates are immediately sent to an *activation token discovery* component, where, for each activation of a candidate, an activation token is created and replayed over a finite state automaton to classify it as either a *fulfillment* or a *violation*. For each activation, the payload and classification label are fed into a *Hoeffding Adaptive Tree* [7]. When a branch of the tree leads to a *fulfillment* leaf node and the tree is balanced, the underlying decision rules are extracted from the path that runs from the root to the purest *fulfillment* leaf node. Finally, the discovered data-aware constraints are presented to the user.

Figure 3: Application GUI.

I-DiDD provides users with a dashboard that, in addition to general log statistics, allows them to explore the results of model discovery through both a tabular and model-based view. As new events are ingested from the stream, all views are updated incrementally, so that the dashboard consistently reflects the information derived from the portion of the stream processed thus far. The dashboard is shown in Figure 3, and it consists of three main components: (1) the general statistics view, (2) the activity bar chart view, and (3) the view of the Data-Aware Declare discovery results. In view (1), users are provided with information about key dimensions of the log, including the number of observed events, the number of observed traces, the timestamp of the first observed event, and the timestamp of the last observed event. In view (2), users are presented with two bar charts: one displaying the frequency of each activity in the log, and the other showing the number of distinct traces in which each activity appears. The data shown in these charts can be downloaded locally as a CSV file. Lastly, in view (3), users are provided with a table containing all information about the discovered Data-Aware Declare models, along with a visual representation of the last ten models discovered. The models are visually presented using the `easyDeclare` notation [8] and can be downloaded locally by the user. The table-based view includes all discovered constraints, along with the corresponding number of activations and fulfillment ratio (the ratio of fulfillments to the total number of activations) for both the data-aware and control-flow perspectives. Additionally, the table can be sorted based on user-defined

criteria, such as constraint name, data conditions, number of activations, and fulfillment. The models can be downloaded as *.decl* files, and the table-based view can be exported as CSV for offline analysis.

3. Tool Maturity

We validated the application by evaluating the effectiveness of the discovery procedure using two event logs: *Sepsis* [9] and *WABO*[5]. The *Sepsis* event log records medical data related to sepsis treatment in a hospital, while the *WABO* event log captures the process of handling environmental permit applications. All experiments were performed on a MacBook Pro equipped with an Apple M3 Pro chip and 36 GB of RAM, and the results are publicly available on GitHub.[6]

For each event log, Data-Aware Declare model discovery was performed using various *Apriori* support thresholds for candidate generation. It is important to configure the *Apriori* support appropriately based on the number of distinct activities in the process. In processes with many distinct activities, a low support threshold may result in too many candidate constraints being generated, making the discovery task inefficient. On the other hand, in processes with only a few distinct activities, a high support threshold may lead to too few candidate constraints, which could be insufficient to capture the process from a data-aware perspective. We accounted for this consideration in our experiments by using two different *Apriori* support values for the two event logs, as they differ in activity cardinality: 16 distinct activities for *Sepsis* and 27 for *WABO*. After replaying each log with *I-DiDD*, the difference in the fulfillment ratio between the control-flow and data-aware constraints was computed for each discovered constraint.

Table 1
Summary statistics for the Sepsis event log.

Metric	Sepsis (Apriori Support=0.4)	Sepsis (Apriori Support=0.6)
Number of Constraints	89	56
Higher Data-Aware Fulfillment Ratio	74.2% (66)	69.6% (39)
Higher Control-Flow Fulfillment Ratio	25.8% (23)	30.4% (17)
Equal Fulfillment Ratio	-	-
Processing Time	4.092 ms	3.161 ms
Memory Consumption	2.809 GB	2.217 GB

In our experiments, we computed the following metrics:

- **Number of Constraints**: The total number of constraints discovered.
- **Higher Data-Aware Fulfillment Ratio**: The number of discovered constraints for which the fulfillment ratio of the data-aware constraint is greater than that of the corresponding control-flow constraint.
- **Higher Control-Flow Fulfillment Ratio**: The number of discovered constraints for which the fulfillment ratio of the control-flow constraint is greater than that of the corresponding data-aware constraint.
- **Equal Fulfillment Ratio**: The number of discovered constraints for which both the data-aware and control-flow versions have equal fulfillment ratios.
- **Processing Time**: The time required to process a single event.
- **Memory Consumption**: An indicator of the memory usage during the entire discovery process, measured in gigabytes (GB). It is obtained by summing the *Bytes Sent* reported for each operator on the Apache Flink dashboard, i.e., the total amount of data emitted by all operators to their downstream operators over the course of the run.

Table 1 presents the results of the experiments performed on the *Sepsis* event log, using *Apriori* support thresholds of 0.4 and 0.6. As expected, increasing the support threshold led to a 37.1% reduction in the number of discovered constraints. Data-aware constraints generally exhibited a higher fulfillment ratio compared to the control-flow ones at both thresholds (74.2% and 69.6%, respectively). We also analyzed the constraints where the fulfillment ratio was higher for the control-flow version and observed

[5]https://promtools.org/event-logs-used-in-this-course/
[6]https://github.com/tedib04/I-DIDD/tree/main/experiments/results

that these constraints already had a very high control-flow fulfillment ratio. In such cases, enriching the constraint with data may not be beneficial, especially if the decision tree fails to effectively discriminate between fulfillments and violations. The time required to process a single event was 4.092 ms. Moreover, increasing the support threshold resulted in lower memory consumption, decreasing from about 2.8 GB to 2.2 GB, due to fewer candidates being generated.

Table 2
Summary statistics for the WABO event log.

Metric	WABO (Apriori Support=0.2)	WABO (Apriori Support=0.4)
Number of Constraints	71	45
Higher Data-Aware Fulfillment Ratio	61.9% (44)	57.8% (26)
Higher Control-Flow Fulfillment Ratio	32.3% (23)	33.3% (15)
Equal Fulfillment Ratio	5.8% (4)	8.9% (4)
Processing Time	0.45 ms	0.41 ms
Memory Consumption	1.76 GB	0.91 GB

Table 2 shows the results for the *WABO* event log using *Apriori* support thresholds of 0.2 and 0.4. As the threshold increased, the number of discovered constraints dropped from 71 to 45, while the majority still showed better fulfillment in the data-aware version (61.9% and 57.8%, respectively). The time required to process a single event was 0.45 ms, with memory usage decreasing from 1.8 GB to 0.9 GB due to fewer generated candidates.

Acknowledgments

The work was partially supported by the Autonomous Province of Bolzano–Bozen (Provincia autonoma di Bolzano–Alto Adige) under the Research Alto Adige (2024) funding programme.

Declaration on Generative AI

ChatGPT was used for final English proofreading of the paper.

References

[1] M. Pesic, H. Schonenberg, W. M. P. van der Aalst, Declare: Full support for loosely-structured processes, in: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, 2007, pp. 287–300.
[2] F. M. Maggi, M. Dumas, L. García-Bañuelos, M. Montali, Discovering data-aware declarative process models from event logs, in: BPM, volume 8094, 2013, pp. 81–96.
[3] A. Burattin, M. Cimitile, F. M. Maggi, A. Sperduti, Online discovery of declarative process models from event streams, IEEE Transactions on Services Computing (2015) 833–846.
[4] N. Navarin, M. Cambiaso, A. Burattin, F. M. Maggi, L. Oneto, A. Sperduti, Towards online discovery of data-aware declarative process models from event streams, in: International Joint Conference on Neural Networks, 2020, pp. 1–8.
[5] G. D. S. Martino, N. Navarin, A. Sperduti, A lossy counting based approach for learning on streams of graphs on a budget, in: IJCAI, 2013, pp. 1294–1301.
[6] 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: Advanced Information Systems Engineering, 2012, pp. 270–285.
[7] A. Bifet, R. Gavaldà, Adaptive learning from evolving data streams, in: IDA, volume 5772 of *Lecture Notes in Computer Science*, 2009, pp. 249–260.
[8] G. Blasilli, L. S. Ferro, S. Lenti, F. M. Maggi, A. Marrella, T. Catarci, Improving the understandability of declarative process discovery results using easydeclare, Information Systems 138 (2026) 102667.
[9] F. Mannhardt, Sepsis cases - event log, 2016. URL: https://data.4tu.nl/articles/_/12707639/1.