Multi-perspective log generator for declarative models

Bruna Alves Wanderley de Siqueira[1], Ricardo Massa Ferreira Lima[2] and Katiane Oliveira Alpes da Silva[3]

Abstract— **Background: The generation of synthetic event logs is fundamental to the advancement and evaluation of process mining techniques. However, synthetic log generation in the context of multi-perspective declarative modeling faces challenges due to its flexible nature, where any behavior that is not constrained by rules is allowed. In addition, the synthetic event log needs to simultaneously meet process, data access and privacy constraints.**

Objective: This study presents a general-purpose, configurable, multi-perspective log generator capable of simultaneously ensuring compliance with the rules of the flow, data access and privacy perspectives in a flexible context such as that found in declarative processes.

Method: A multi-perspective synthetic log generator was developed, capable of integrating declarative process models, data access and organizational structure. The tool was structured in three stages: processing the input models, generating the event log with Declare4Py, and generating the access log respecting the restrictions.

Results: Two experiments were carried out with 1,000 and 2,000 cases respectively. The logs generated respected all the rules of conformity between activities, accesses and permissions. There was a linear increase in file size and a moderate increase in execution time, compatible with the complexity of the models.

Conclusions: The proposed tool proved effective in generating coherent synthetic logs with multiple perspectives, supporting research into process mining and the evaluation of compliance algorithms. It is a relevant solution for contexts where real data is scarce or sensitive.

1. INTRODUCTION

Process mining is a research area that aims to analyze business processes using process models and event logs, which record the actual flow of executed activities [1]. In recent years, significant advances have occurred in this area, including new algorithms, tools, and case studies. These advances provide relevant insights, contributing to improving business processes across various sectors.

The wide availability of data, enabled by computerized systems, decisively drives process mining. Event logs with diverse characteristics are essential to evaluate algorithms in real or simulated scenarios [2]. However, security and privacy concerns often restrict access to these records.

Van der Aalst [1] emphasizes the central role of event logs, referring to them as "first-class citizens" in process mining. However, acquiring and preparing these logs remains a major challenge for applying process mining techniques and developing new algorithms. Accorsi and Lebherz [3] report that generating event logs can consume up to 80% of the total effort of a project. Likewise, Weerdt and Wynn [4] estimate that preprocessing alone may account for as much as 90% of the workload, highlighting the complexity and significance of this stage.

Given the complexity of generating and preprocessing event logs, Skydanienko et al. [2] emphasize the need for configurable synthetic log generators. These generators enable the evaluation of new process mining techniques and the application of algorithms in complex or privacy-restricted scenarios.

Synthetic data offers a promising alternative to overcome privacy concerns and the inherent complexity of real-world data. Synthetic event logs facilitate developing, validating, and disseminating novel process mining algorithms, fostering greater transparency in research methods and results. Moreover, making these datasets publicly available promotes reproducibility and enables independent validation of proposed approaches.

Skydanienko et al. [2] stress the importance of collective efforts in developing synthetic log generators. The authors note the lack of research on generating synthetic logs from declarative process models. Due to the flexible nature of this approach, event logs can exhibit any behavior not explicitly restricted by the model, making their generation more challenging. Additionally, existing log generators mainly focus on process flow and resource attributes, neglecting other relevant aspects. Therefore, the authors propose developing a multi-perspective declarative synthetic log generator that considers tasks, resources, and other event attributes, such as time, customer type, and contextual characteristics.

Evaluating the multi-perspective compliance analysis proposed by Mehr et al. [5] required generating event logs that accurately reflect multiple dimensions of the process. Their procedural modeling approach integrates control-flow, data access, and privacy perspectives. To support this analysis, the authors used a synthetic log generator capable of producing both event logs and data access logs. Because of the multi-perspective nature of the model, the generation process combined inputs from process, data access, and organizational models, ensuring coherent and realistic representations of business behavior.

Generating multi-perspective synthetic logs for declarative modeling presents challenges similar to those identified by Mehr et al. [5], but with greater complexity due to the flexible nature of this modeling approach, where all actions not explicitly restricted are allowed. This characteristic increases the difficulty of creating synthetic logs that faithfully represent system interactions. Furthermore, evaluating the

[1]Bruna Alves is with Centro de Informática of Universidade Federal de Pernambuco, Recife, Pernambuco, Brazil baws@cin.ufpe.br

[2]Ricardo Massa is with Centro de Informática of Universidade Federal de Pernambuco, Recife, Pernambuco, Brazil rmfl@cin.ufpe.br

[3]Katiane Alpes is with Centro de Informática of Universidade Federal de Pernambuco, Recife, Pernambuco, Brazil koas@cin.ufpe.br

flow, data access, and privacy perspectives together requires hierarchical validation of process rules. In other words, the synthetic log must ensure compliance with flow constraints between tasks, data access rules, and authorship of task execution and data access at the start and end of each activity, maintaining alignment with the models for process, data access, and privacy.

Considering this context, this study presents a general-purpose synthetic log generator designed to address a declarative process flow model, data access, and privacy perspectives. The proposed generator can produce customizable event and data access logs for diverse contexts. Experiments validate the tool's effectiveness, demonstrating accurate log generation according to the rules.

The remainder of this paper is organized as follows: Section II presents related works, Section III introduces the context definition, Section IV presents the log generator itself, Section V evaluates the solution and presents the results, Section VI concludes the paper with directions for future work.

2. Related Works

Chiariello et al. [6] introduce the application of Answer Set Programming (ASP) to solve three process mining problems: log generation, query checking, and conformance checking. Their main focus is the theoretical formalization and modeling of these problems in ASP, employing a solver to achieve viable solutions. The principal contribution of Chiariello et al. [6] to this article is the theoretical formalization underlying log generation from declarative models, implemented in Python by Donadello et al. [7]. The Declare4Py library allows the manipulation of declarative models and event logs, offering ASP-based parameterized log generation. Results presented by Donadello et al. [7] demonstrate the maturity of the continuously improving tool, justifying its selection for one of the stages in this study.

In contrast to ASP-based approaches, Bergami [8] proposes an alternative technique for log generation, emphasizing the proceduralization of declarative models to utilize graph traversal algorithms. This synthetic log generation technique achieves greater efficiency through faster algorithms by converting declarative models into procedural equivalents. Bergami's [8] approach also supports generating additional attributes beyond activities and timestamps by employing random parameter generation according to possible values. This random parameter generation strategy, crucial for realistic modeling, will also be utilized in the proposed solution of this research, not only in the generation of attributes of a log, but also for generating the multi-perspective aspects described in the context.

Andrews et al. [9] introduce a semi-automated log generation technique utilizing databases with timestamp columns to produce event logs. The primary contribution of their work is handling tables with foreign keys and processing entity relationships to create logs with minimal manual intervention. Their multidimensional approach integrates various data types into the log generation process. In the multi-perspective context of this research, Andrews et al.'s solution [9] relates to data integration from different types, necessitating a unique identifier similar to database foreign keys. This work also deals with different data types, related by unique keys to connect all perspectives, nevertheless beyond what Andrews et al. proposed, this is in a multi-perspective context, that require generating two distinct logs, with no previous data available, having only process, data object access and privacy models.

Grüger et al. [10] present SAMPLE, an innovative method for generating multi-perspective event logs. Using procedural techniques, SAMPLE introduces contextual variables into log generation, enabling more semantically representative logs. Grüger et al. [10] consider the generation of event log attributes beyond process analysis as an additional perspective. Although employing the same "multi-perspective" definition, in the present study the perspectives focus on process flow, privacy, and data object access within business processes across various contexts. For instance, Grüger et al.'s study [10] generated event logs in healthcare, ensuring data realism, such as assigning "prostate cancer" to male gender patients. Regarding data access and privacy dimensions, it would be possible in medical contexts to verify if a resource performing a procedure had the proper permissions or accessed patient records appropriately. Thus, data dimensions and the perspectives introduced in this work significantly contribute to generating high-quality synthetic logs.

3. Preliminaries

This section introduces the main concepts and notations for representing the log structure and the parameters for the log generation.

Our modeling strategy uses a declarative framework that specifies activities according to particular constraints, supported by a data schema that sets guidelines for accessing data elements and an organizational structure embedding privacy considerations. The system's behavior is captured by logging both the execution of activities and the interactions involving data access.

A. Process perspective

The declarative process model will be specified using the Declare modeling language [11], which enables the definition of flexible workflows through constraints rather than rigid sequences. This model consists of a collection of activities and constraints governing the relationships among these activities.

Definition 3.1. [Declarative Process Model] Let $A = \{a_1, a_2, \ldots, a_n\}$ be a finite set of activities, and let $A_M \subseteq A$ be the subset of activities present in the model (Declare allows activities not present in the process model to be performed within the process). Define $\text{TEMP} = \{temp_1, temp_2, \ldots, temp_m\}$ as the set of Declare constraint templates, each applied over activities in A_M. A declarative process model is then $P = A_M \cup \text{TEMP}$.

In the example shown in Figure 1, the model recognizes the activities Functionality Maintenance and Update Requirements, which appear in the Existence and Response rules. The Existence rule ensures the activity appears in the log, while the Response rule verifies that the second activity must follow if the first activity occurs.

activity Functionality Maintenance
activity Update Requirements
Existence[Update Requirements] | |
Response[Update Requirements, Functionality Maintenance] | | |

Fig. 1. Declare model representation with two activities and the rules Existence and Response.

B. Data perspective

Relationships between activities and data object operations define the data access model. Access to data objects can be mandatory, optional, or prohibited for each activity. Possible data operations include reading (r), updating (u), deleting (d), and creating (c). Thus, the data access model consists of activities from the process model combined with rules governing these data operations.

Definition 3.2. [Data Access Model] Let $A = \{a_1, a_2, \ldots, a_n\}$ be a finite set of activities, where each a_i represents a specific task within a given context. Let $OBJ = \{obj_1, obj_2, \ldots, obj_m\}$ be a finite set of data objects, where each obj_j represents a distinct entity subject to data operations.

Let $OT = \{c, r, u, d\}$ be a finite set of operation types allowed on any data object $obj_j \in OBJ$, defined as:

- *create (c)*: adding new data to the data object,
- *read (r)*: retrieving or viewing data within the data object,
- *update (u)*: modifying existing data in the data object,
- *delete (d)*: removing data from the data object.

Let $CT = \{md, op, p\}$ be a finite set of constraints on the data operations, where:

- *mandatory (md)*: the operation must be performed during the corresponding activity,
- *optional (op)*: the operation may or may not be performed during the corresponding activity,
- *prohibited (p)*: the operation must not be performed during the corresponding activity.

A *data access model* is then defined as:

$$AD \subseteq A \times OBJ \times OT \times CT,$$

where each element $(a,\, obj,\, ot,\, ct) \in AD$ specifies a relation between an activity $a \in A$, a data object $obj \in OBJ$, an allowed operation $ot \in OT$, and a constraint $ct \in CT$.

Table I shows an example of a data access model used in this study. This model specifies mandatory, optional, or prohibited CRUD operations (creation, reading, updating, deletion) on data objects during agile team activities. Capital letters represent mandatory operations; lowercase letters denote optional ones; the absence of a letter indicates a prohibited operation. Each activity requires updating at least the **Management** object; the matrix explicitly defines restrictions for other operations.

TABLE I
DATA ACCESS MODEL EXAMPLE IN AGILE SOFTWARE DEVELOPMENT

Activity	Data Object			
	Management	Requirements	Code	Test
Upd. Req.	crUd	crUd	r	r
Funct. Maint.	crUd	r	crUd	r

C. Organizational perspective

The organizational model allocates resources to profiles and assigns these profiles to specific activities. In this context, privacy is managed by restricting data object access, ensuring only authorized profiles can access data associated with a particular activity. Privacy is enforced through access control rules, safeguarding sensitive data involved in the execution of activities.

Definition 3.3. [Organizational Model] Let $PR = \{pr_1, pr_2, \ldots, pr_p\}$ be a finite set of profiles, where each pr_i denotes a particular profile. Let $RE = \{re_1, re_2, \ldots, re_q\}$ be a finite set of resources, where each re_j represents a distinct agent. Let $A = \{a_1, a_2, \ldots, a_r\}$ be a finite set of activity, where each a_k corresponds to a particular activity. The representation of an *organizational model* is a set $O \subseteq PR \times A \times RE$, where each tuple $(pr_i, a_k, re_j) \in O$ signifies that resource re_j holds profile pr_i and this profile is assigned to execute task a_k.

An example of an organizational model in the context of agile software development conforming to the definition above appears in Table II. The company specifies privacy rules for associate profiles, activities, and resources in this scenario. Concretely, profile e_1 (team 1) is assigned to activities of a given software demand ad_1, and encompasses resources $\{re_1, re_2, re_3, re_4\}$. Likewise, profile e_2 (team 2) is assigned to activities of another software demand ad_2, and includes resources $\{re_5, re_6, re_7\}$.

TABLE II
EXAMPLE OF AN ORGANIZATIONAL MODEL

Profile	Activity	Resource
e_1	ad_1	re_1, re_2, re_3 e re_4
e_2	ad_2	re_5, re_6 e re_7

D. Event Log Structure

We now present the event log structure produced in this work, which combines the flow, data, and privacy perspectives.

Definition 3.4. [Event Log] Let EL_{s} be a set of events $\{\, e \mid e \in EL_{\text{s}}\}$, where each event e is a tuple

$$e = \big(c_{\text{id}}, ev_{\text{id}}, insta,\, a,\, re,\, t,\, trans\big),$$

comprising the following elements:

- $c_{\text{id}} \in C_{\text{id}}$: a unique case identifier, where C_{id} is a finite set that distinguishes each case in the log;
- $ev_{\text{id}} \in EV_{\text{id}}$: a unique event identifier, where EV_{id} is a finite set that distinguishes each event in the log;
- $insta \in INSTA$: a unique activity identifier, where $INSTA$ is a finite set that distinguishes each activity in a case;
- $a \in A$: the activity associated with the event, where A is a finite set of activities;
- $re \in RE$: the resource responsible for performing the activity, where RE is a finite set of agents;
- $trans \in TRANS$: the transaction type associated with the event, where $TRANS$ is a finite set of possible transactions. The valid values are:
  - *start*: denotes the beginning of the transaction's lifecycle;
  - *complete*: denotes the end of the transaction's lifecycle;
- $t \in T$: the timestamp of the event, where T is a finite set of time instants indicating when the event occurred.

The mapping function

$$\tau : EL_{\text{s}} \longrightarrow C_{\text{id}} \times EV_{\text{id}} \times INSTA \times A \times RE \times TRANS \times T$$

ensures that each event $e \in EL_{\text{s}}$ is uniquely identified by its case identifier c_{id}, event identifier ev_{id}, instance of activity $insta$, activity a, resource re, transaction type $trans$, and timestamp t.

The *transaction* attribute defines an activity's temporal boundaries (start and complete), facilitating the evaluation of data access rules at the activity-level granularity. For instance, if accessing data object d_1 is prohibited during the execution of activity A, the log must explicitly capture both the start and complete timestamps of A. These timestamps are considered in Declare constraints, thus ensuring that, in this example, we flag as non-compliant any access to d_1 occurring between the initiation and termination of activity A.

Table III provides an illustrative example of an event log that conforms to the previously presented definition.

TABLE III
EXAMPLE OF AN EVENT LOG STRUCTURE

case	event	inst	activity	res.	trans	timestamp
C1	ev1	1	a_1	re1	start	29.12.24:10.00
C1	ev2	1	a_1	re1	complete	29.12.24:14.30
C1	ev3	2	a_2	re3	start	30.12.24:18.22
C1	ev4	2	a_2	re3	complete	30.12.24:19.22
C1	ev5	3	a_3	re1	start	31.12.24:12.02
C1	ev6	3	a_3	re1	complete	14.01.25:10.10
C2	ev7	1	a_4	re6	start	14.01.25:14.00
C2	ev8	1	a_4	re6	complete	14.01.25:18.00
C2	ev9	2	a_5	re5	start	14.01.25:19.20
C2	ev10	2	a_5	re5	complete	15.01.25:08.00
C2	ev11	3	a_6	re2	start	15.01.25:10.22
C2	ev12	3	a_6	re2	complete	20.01.25:10.22

E. Data Access Log Structure

We now introduce the structure of the data access log generated by our proposed approach.

Definition 3.5. [Data Access Log] Let DL be the set of data access entries $\{ da \mid da \in DL\}$. Each entry da is formally defined by the tuple

$$da = (c_{\text{id}}, insta, obj, ot, re, trans, t),$$

where:

- $c_{\text{id}} \in C_{\text{id}}$: a unique case identifier, where C_{id} is a finite set that distinguishes each case in the process log;
- $insta \in INSTA$: a unique activity identifier, where $INSTA$ is a finite set that distinguishes each activity in a case of the process log;
- obj indicates the accessed data object;
- $ot \in OT$, where OT is a finite set of data operation types;
- $re \in RE$, where RE is a finite set of agents;
- $trans \in TRANS$: the transaction type associated with the operation, always set to *start*, indicating that the operation is atomic;
- $t \in T$, with T representing a finite set of timestamps specifying when the data access operation occurred.

The mapping function

$$\tau : DL \rightarrow C_{\text{id}} \times INSTA \times OBJ \times OT \times RE \times TRANS \times T$$

ensures that each data access entry $da \in DL$ is uniquely identified by its case identifier c_{id}, instance of activity $insta$, data object obj, operation type ot, agent re, transaction $trans$, and timestamp t.

This definition highlights the importance of the attributes *case identifier* and *instance of activity*, which link the process model to the data access model. Table IV shows an example of a data access log that conforms to this definition. In case C1, for instance 2, resource *re3* performed an *update* operation on the data object *Code* at timestamp *30.12.24:18.45*. The same resource later executed a *create* operation on *Requirements* within the same activity instance.

TABLE IV
EXAMPLE OF A DATA ACCESS LOG FOR AGILE SOFTWARE TEAMS.

case	inst	data obj.	oper.	res.	trans.	timestamp
C1	2	Code	u	re3	start	30.12.24:18.45
C1	2	Requirem.	c	re3	start	30.12.24:19.00
C1	3	Code	u	re2	start	13.01.25:12.22
C2	1	Code	d	re6	start	14.01.25:17.22
C2	2	Requirem.	u	re2	start	21.01.25:16.10

4. MULTI-PERSPECTIVE LOG GENERATOR

The proposed solution consists of three main stages: (i) input parameter processing, (ii) event log generation, and (iii) data access log generation.

A. Input Parameter Processing

The log generator requires the following input parameters:

- a *declarative process model*, containing activities and constraints in `.decl` format (see Section 3.1);
- a *data access model*, specifying access restrictions for each activity in `.csv` format (see Section 3.2);
- an *organizational model*, detailing access profiles for activities in `.csv` format (see Section 3.3);

- the *number of cases*;
- the *minimum number of events per case*;
- the *maximum number of events per case*;
- (Optional) For each activity, an integer value indicates its *duration in hours*.

The system parses the declarative process model using the `Declare4Py` library [7], supporting process log generation in the next stage. At this point, the system validates the model's format and syntax. It processes the data access and organizational models, converting them into structured `Data Frames`. Moreover, the system groups activity durations into an activity-duration dictionary for later use.

B. Event Log Generation

The system uses `Declare4Py` [7] to generate process log events. Specifically, it leverages the `AspGenerator` [6], which receives the process model, number of cases, and the minimum and maximum number of events per case as input. It generates execution traces that comply with the declarative model's constraints and include basic attributes. Each generated activity is atomic, meaning it has a single event with the transaction attribute set to `'complete'`.

Algorithm 1 takes as input the basic event log generated by Declare4Py and enriches it with additional attributes to support a multi-perspective analysis. It begins (lines 1–7) by defining the necessary data structures: *orgModel* (which maps each activity to a set of compatible resources), *activities* (which associates each activity with its duration), *BasicLog* (which represents the basic log generated by Declare4Py), the variables $id\_instance$ and *case*, which track the activity instance index and the currently processed case, respectively, and the creation of the columns $resource$ and $instance$ with the initial values ' '(empty string) and 0, respectively.

Next (lines 8–15), the algorithm iterates over each row in *BasicLog*. If *case* is empty (or if the case value in the row differs from the stored one), a new case has begun: *case* is updated, and $id\_instance$ is reset to 1. This mechanism ensures that each new case starts with a fresh activity instance counter.

In lines 16–18, resource assignment is performed. The *RandomChoice* function employs the method proposed by Matsumoto and Nishimura [12]. It randomly selects a valid resource from the *orgModel* for the given activity. This selected resource is then stored in the *resource* column at the corresponding row index. Simultaneously, the activity instance number is recorded in the *instance* column, allowing for the identification of repeated occurrences of the same activity within the same case.

Then (lines 19–22), the algorithm calculates *timestamp_complete* by retrieving the typical duration of the activity from the *activities* dictionary and adding it to the initial timestamp *row[timestamp]*. The timestamp of the row is updated with *timestamp_complete* so that this occurence of the activity refers to the transaction = 'complete', as created in the BasicLog.

Finally (lines 23–26), the algorithm inserts a new event into the log to explicitly indicate the activity's start, with transaction = 'begin'. This new row retains the values for *case*, *resource*, *instance*, and *timestamp_begin*, and increments *id_instance* for upcoming activities.

Algorithm 1 Processing basic log to add privacy perspective and transaction's lifecycle

```
1:  Initialize:
2:      orgModel ← mapping activities to profiles and resources
3:      activites ← mapping activities to durations
4:      BasicLog ← Predefined basic log
5:  Initialize id_instance ← 1, case ← ' '
6:  Initialize row[resource] ← ' '
7:  Initialize row[instance] ← 0
8:  for all index, row in BasicLog log do
9:      if case = ' ' then
10:         case ← row[case]
11:     end if
12:     if case ≠ row[case] then
13:         case ← row[case]
14:         id_instance ← 1
15:     end if
16:     resource ← RandomChoice(orgModel[row[activity]])
17:     log[index, resource] ← resource
18:     log[index, instance] ← id_instance
19:     duration ← activities[row[activity]]
20:     timestamp_complete ← row[timestamp] + duration
21:     timestamp_begin ← row[timestamp]
22:     log[index, timestamp] ← timestamp_complete
23:     Append to log: [row[activity], 'begin', timestamp_begin,
24:     row[case], resource, id_instance]
25:     id_instance ← id_instance + 1
26: end for
```

After this step, the log contains new columns *resource* and *instance*, populated according to Definition 3.4.

C. Data Access Log Generation

Data Access log generation relates hierarchically to the event log, as each activity in the workflow may involve multiple tool accesses and operations. This approach maintains the relationship between activities and their accesses by uniquely identifying each entry by utilizing the case identifier and activity instance. Table V maps the attributes and their corresponding names in the generated log.

TABLE V
MAPPING OF ATTRIBUTES TO DATA ACCESS LOG

Attribute	Log attribute
Case ID	case:concept:name
Activity instance	concept:instance
Data Object	concept:dataobj
Resource	concept:resource
Timestamp	time:timestamp
Transition Lifecycle	lifecycle:transition

The algorithm presented in *Algorithm 2* aims to generate an *Data Access Log* by examining activities within a predefined *Process Log* and applying access rules for each activity. In lines 2–4, the algorithm sets up the necessary data structures: it declares *EventLog* as a predefined event log containing activities, initializes an empty *AccessLog* for storing the outputs, and defines *AccessModel* with rules that

specify which operations and data objects each activity might access.

Line 6 sorts the *Log* by *[case, instance, transition]*, ensuring that all events for a given case and instance follow the correct chronological order. Beginning at line 8, the algorithm iterates through every *row* in the *Log*. Check whether the current transition is equal to *'begin'*, which means that an activity has just started.

At line 10, for each pair $(data\_object, value)$ in *access*, the algorithm examines all operations in *value*, labeled *operat*. In line 12, it verifies whether *operat* is uppercase or if *RandomInt(1, 10)* yields an even number, that means if an access is mandatory or if the random condition for the generation of optional access is attended. If either condition is true, the algorithm proceeds at line 14 to compute a *random_time* between the current event's time and the next event's time $\big(Log[\,index+1,\ time]\big)$. This simulates an access happening randomly between the start of the current activity and its end, represented by the next event, with lifecycle 'complete'.

Line 16 then appends a record to *AccessLog*, storing the case identifier, the data object, the chosen operation, the random timestamp, the resource responsible for operating, the activity instance number and the 'start' for atomic lifecycle. The final lines (18, 20, and 22) close the *if*, *for*, and *forAll* constructs, respectively.

Algorithm 2 Generating Data Access Log

```
    Initialize:
2:  EventLog ← Predefined event log with activities
    AccessLog ← Empty log for access records
4:  AccessModel ← Predefined access model with rules for each activity
    Sort EventLog by [case, instance, transition]
6:  for all index, row in EventLog do
        if row[transition] = 'begin' then
8:          activity ← row[activity]
            access ← AccessModel[activity]
10:         for all (data_object, value) in access do
                for all operat in value do
12:                 if operat is uppercase or RandomInt(1,10) is even then
                        random_time ← Random(row[time],
14:                                  EventLog[index + 1, time])
                        Append [row[case], data_object, operat,
16:                             random_time, row[resource],
                                row[instance]],' start' to AccessLog
18:                 end if
                end for
20:         end for
        end if
22: end for
```

At the end of execution, the algorithm generates the access log according to Definition 3.5, respecting the data access model.

To illustrate the combined use of flow, data, and privacy constraints, consider the scenario of updating requirements in a software development company. As soon as the activity *Update Requirements* begins (an event with `transaction = begin`), the algorithm checks which data can be created, deleted, read or updated for this activity and which user profiles are authorized to do so. For example, suppose that a resource with profile *Analyst of team 2* performs the activity *Update Requirements* in a software demand *dem1* but lacks permission to delete records in the *Requirements* object of that demand. In that case, the log generator blocks any delete operation, ensuring compliance with privacy rules and randomly generates only create, update or read operations.

Next, when moving on to the activity *Functionality Maintenance*, the log generator creates new reads and updates operations on the *Code* object, as long as they respect the permissions of the resource acting. This way, the system records the process log (start and end of each activity) and the detailed data access log. These records demonstrate how the system enforces each constraint (flow, data, and privacy) at every step.

D. Graphical User Interface

Given the complexity of generating a multi-perspective log and the importance of making this tool accessible to other researchers, a graphical user interface was developed. This interactive web application enables users to easily input parameters, as Figure 2 shows. After processing, the generated logs are available for download on the platform.

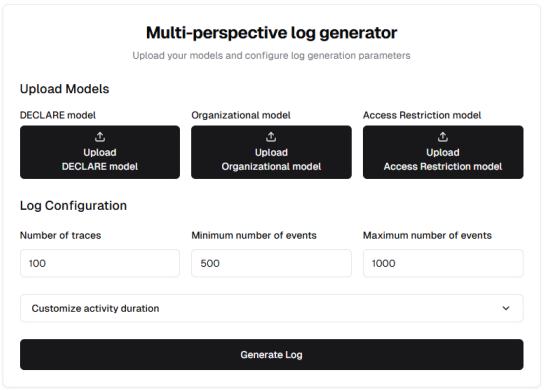

Fig. 2. Multi-perspective log generator web application screen

5. Experiment

A. Complete Example

An experiment was conducted with simple models to demonstrate the complete multi-perspective log generation flow. A description of the input parameters is as follows:

- *Declarative process model* containing 3 activities and 3 rules (Figure 3)
- *Data access model* for the 3 activities in the process model and 3 data objects (Table VI)
- *Organizational model* considering 3 profiles (Table VII)
- *Number of cases* = 3
- *Minimum number of events* = 3
- *Maximum number of events* = 7
- *Duration in hours* for each of the 3 activities in the process model (Table VIII)

After executing the steps, the algorithm generates the event and data access logs. Table IX shows an excerpt of the

activity Management
activity Update Requirements
activity Functionality Maintenance
Existence[Management] | |
Existence[Update Requirements] | |
Response[Update Requirements, Functionality Maintenance] | | |

Fig. 3. Declare model representation with three activities and three rules.

TABLE VI
DATA ACCESS MODEL

Activities	Data Objects		
	Management	Code	Requirements
Functionality Maintenance	c, r, U, d	r, U, d	r
Update Requirements	c, r, U, d	r	r, U, d
Management	c, r, U, d		

TABLE VII
ORGANIZATIONAL MODEL

Activity	Profile	Resources
All of case_0	e2	re2, re4, re5
All of case_1	e5	re12, re13
All of case_2	e3	re5, re6, re7, re8, re9

TABLE VIII
ACTIVITY DURATIONS IN HOURS

Activity	Duration (hours)
Functionality Maintenance	4
Update Requirements	2
Management	1

generated event log, displaying only one case; the system generated 3 cases and 20 events. This example satisfies the declared rules: the required activities Management and Update Requirements each appear at least once, and each Update Requirements activity comes before a Functionality Maintenance activity. Functionality Maintenance activities occurring without a preceding Update Requirements activity do not violate the rules, as the rule activates only upon the occurrence of Update Requirements.

TABLE IX
CASE 0 OF THE EVENT LOG

Timestamp	Trans.	Activity	Case	Res.	Inst.
2025-03-29 02:21:28	begin	Management	case_0	re2	1
2025-03-29 03:21:28	complete	Management	case_0	re2	1
2025-03-29 02:57:36	begin	Upd. Req.	case_0	re5	2
2025-03-29 04:57:36	complete	Upd. Req.	case_0	re5	2
2025-03-29 03:56:21	begin	Funct. Maint.	case_0	re2	3
2025-03-29 07:56:21	complete	Funct. Maint.	case_0	re2	3
2025-03-29 05:12:49	begin	Funct. Maint.	case_0	re4	4
2025-03-29 09:12:49	complete	Funct. Maint.	case_0	re4	4
2025-03-29 05:47:07	begin	Funct. Maint.	case_0	re5	5
2025-03-29 09:47:07	complete	Funct. Maint.	case_0	re5	5
2025-03-29 07:24:13	begin	Funct. Maint.	case_0	re2	6
2025-03-29 11:24:13	complete	Funct. Maint.	case_0	re2	6

The data access log generated for this example contains 96 events: 33 for Case 0, 27 for Case 1, and 36 for Case 2. Table X presents an excerpt of the accesses generated for Case 0. All mandatory accesses occur correctly; for instance, activity instance 1 (Management) includes a mandatory update operation on the Management data object and optional create and delete operations on the same object. Additionally, this activity correctly respects the prohibition rules by preventing access to other data objects, aligning with the data access model in Table VI.

TABLE X
PART OF DATA ACCESS LOG GENERATED FOR CASE 0

Case	Data Ob.	Op.	Timestamp	Res.	Inst.	Trans.
case_0	Manage.	c	2025-03-29 03:10:37	re2	1	start
case_0	Manage.	u	2025-03-29 02:58:21	re2	1	start
case_0	Manage.	d	2025-03-29 02:54:19	re2	1	start
case_0	Manage.	c	2025-03-29 03:32:24	re5	2	start
case_0	Manage.	u	2025-03-29 04:14:45	re5	2	start
case_0	Code	r	2025-03-29 03:46:32	re5	2	start
case_0	Requir.	r	2025-03-29 04:32:52	re5	2	start
case_0	Requir.	u	2025-03-29 03:01:47	re5	2	start
case_0	Requir.	d	2025-03-29 04:18:12	re5	2	start
case_0	Manage.	c	2025-03-29 06:09:38	re2	3	start
case_0	Manage.	r	2025-03-29 04:35:38	re2	3	start

B. Case Study

With the multi-perspective log generator in a stable version, experiments in a more robust scenario were done as part of a case study within an agile software development company. Input models reflect the evaluated real-world context. Input parameters were chose aiming to approximate existing values regarding data volume.

The experiments in this case study aim to validate the generated logs against the context, performance assessment, and computational resource usage evaluation.

The following parameters were used:

- *Declarative process model* containing 12 activities and 18 rules
- *Data access model* for the 12 activities in the process model and 5 data objects
- *Organizational model* linking one case to a profile (team), where all team members can perform any task within that case
- *Number of cases* = {1000, 2000}
- *Minimum number of events* = 20
- *Maximum number of events* = 50
- *Duration in hours* for each of the 12 activities in the process model

Two experiments were performed with 30 runs each, differing only in the number of cases generated: the first experiment generated 1000 cases, and the second 2000 cases. The algorithms ran on a Windows 11 machine with a 12th Gen Intel(R) Core(TM) i7-1255U, featuring 10 cores and 12 logical processors. Tables XI and XII summarize results regarding execution time and log sizes for each experiment. Consider E1 as the experiment generating 1000 cases, E2 as the experiment generating 2000 cases, Step 1 as the basic event log generation using Declare4Py [7], Step 2 as the complete event log generation, and Step 3 as the data access log generation.

According to these results, it is possible to conclude that the log size increases linearly with the number of cases,

TABLE XI

MEAN, MODE AND MEDIAN FOR THE DURATION OF EACH STEP OF THE LOGS GENERATION FOR EACH EXECUTION

Step	E1			E2		
	Mean	Mode	Median	Mean	Mode	Median
Step 1	3m58s	3m46s	3m57s	8m15s	8m36s	8m7s
Step 2	2m11s	2m22s	2m11s	10m14s	9m55s	9m55s
Step 3	13m29s	14m16s	13m28s	54m53s	55m32s	54m53s

TABLE XII

MEAN, MODE AND MEDIAN FOR THE FILES SIZES IN MB OF EACH STEP OF THE LOGS GENERATION FOR EACH EXECUTION

Step	E1			E2		
	Mean	Mode	Median	Mean	Mode	Median
Step 1	16.8	16.6	16.8	33.6	33.7	33.6
Step 2	46	45.6	45.9	91.9	91.4	92
Step 3	109.3	108.7	109.2	218.7	219.5	218.6

allowing estimation of the generated log size according to the desired number of cases. Predicting the logs size is important for preparing future data analysis stages and assessing their feasibility. Regarding execution time, the basic event log generation step using AspGenerator [6] doubled its runtime proportionally to the increase in the number of traces; however, the other steps did not follow this pattern. This discrepancy occurs because the algorithms from the previous section define additional processing. That processing requires repeatedly adding new rows to the log, a more costly operation involving memory and data structure modifications.

6. CONCLUSIONS

Generating synthetic logs is essential in developing and validating process mining techniques. However, applying synthetic log generation to multi-perspective declarative modeling poses significant challenges due to its inherent flexibility, allowing any event that explicit rules do not explicitly restrict. Moreover, in this context, synthetic records must simultaneously comply with constraints from multiple model perspectives, including process rules, data access controls, and privacy requirements. Given these challenges, research in this field explores strategies to enable the creation of representative synthetic logs compatible with the multiple perspectives of declarative modeling, thus enhancing process mining techniques.

The results of this work confirm that the developed tool meets the proposed objectives, demonstrating effectiveness in generating multi-perspective synthetic logs that simultaneously respect rules regarding process flow, data access, and privacy. The experiments demonstrate the tool's robustness in generating logs at realistic scales, complying with established constraints. Thus, the presented solution significantly contributes to process mining research applied across various contexts and the evaluation of algorithms, especially in scenarios where acquiring real data from different perspectives is challenging.

Our findings reinforce that the generation of multi-perspective synthetic logs, together with the simultaneous fulfillment of process rules, data access requirements, and privacy constraints, not only brings theoretical advances to process mining research but also opens new avenues for practical application. This convergence of perspectives points to a promising future, where well-structured synthetic logs can enable more transparent, efficient solutions that align with the real demands of modern organizations.

Unlike previous approaches, which generally address only activity flows, or resource attributes, the solution proposed here stands out by simultaneously integrating declarative process rules, data access constraints, and privacy regulations. This multi-perspective focus in declarative scenarios is the novel contribution that differentiates our tool, as it enables a unified analysis of compliance and violations that only emerge when one considers all these dimensions together.

Future work could enable the customization of additional parameters to make generated logs even more flexible parameter settings and support more realistic context attributes.

REFERENCES

[1] W. V. der Aalst, *Process mining: Data science in action*. Springer Berlin Heidelberg, 1 2016.

[2] V. Skydanienko, C. D. Francescomarino, C. Ghidini, and F. M. Maggi, "A tool for generating event logs from multi-perspective declare models," in *International Conference on Business Process Management*, 2018.

[3] R. Accorsi and J. Lebherz, *A Practitioner's View on Process Mining Adoption, Event Log Engineering and Data Challenges*, pp. 212–240. 06 2022.

[4] J. Weerdt and M. Wynn, *Foundations of Process Event Data*, pp. 193–211. 06 2022.

[5] A. S. Mozafari Mehr, R. M. de Carvalho, and B. van Dongen, "Detecting privacy, data and control-flow deviations in business processes," in *International Conference on Advanced Information Systems Engineering*, pp. 82–91, Springer, 2021.

[6] F. Chiariello, F. M. Maggi, and F. Patrizi, "Asp-based declarative process mining," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 5539–5547, Jun. 2022.

[7] I. Donadello, F. M. Maggi, F. Riva, and M. Singh, "Asp-based log generation with purposes in declare4py," in *Doctoral Consortium and Demo Track 2023 at the International Conference on Process Mining: ICPM-DCDT 2023*, vol. 3648, (Rome, Italy), pp. 1–4, 5th International Conference on Process Mining, ICPM 2023, October 23–27, 2023 2023.

[8] G. Bergami, "Fast synthetic data-aware log generation for temporal declarative models," in *Proceedings of the 6th Joint Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, GRADES-NDA '23, (New York, NY, USA), Association for Computing Machinery, 2023.

[9] R. Andrews, C. van Dun, M. Wynn, W. Kratsch, M. Röglinger, and A. ter Hofstede, "Quality-informed semi-automated event log generation for process mining," *Decision Support Systems*, vol. 132, p. 113265, 2020.

[10] J. Grüger, T. Geyer, D. Jilg, and R. Bergmann, "Sample: A semantic approach for multi-perspective event log generation," in *Process Mining Workshops* (M. Montali, A. Senderovich, and M. Weidlich, eds.), (Cham), pp. 328–340, Springer Nature Switzerland, 2023.

[11] W. Aalst, M. Pesic, and H. Schonenberg, "Declarative workflows: Balancing between flexibility and support," *Computer Science - Research and Development*, vol. 23, pp. 99–113, 05 2009.

[12] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, p. 3–30, Jan. 1998.