Characterizing Process Variants with Natural Examples

Carlos Capitán-Agudo[1,2], Claudio Di Ciccio[3], Cristina Cabanillas[1,2], and Manuel Resinas[1,2]

[1] SCORE Lab, Universidad de Sevilla, Spain
[2] I3US Institute, Universidad de Sevilla, Spain
{ccagudo,cristinacabanillas,resinas}@us.es
[3] Utrecht University, Utrecht, The Netherlands, c.diciccio@uu.nl

Abstract. To improve business processes, it is crucial to analyze why processes differ across process variants. To support this analysis, prior research has often relied on machine learning models trained to classify process variants and used model explanations to distinguish them. User studies in explainable artificial intelligence show that these explanations can be expanded through the inclusion of examples. To this end, we introduce an approach that uses declarative behavioural constraints to support variant classification tasks. Building on these constraints, we perform a post-hoc probability analysis and derive a set of multi-feature measures to characterize cases that are representative of each process variant (prototypes) alongside cases that are close to the classification boundaries between process variants (boundary cases). We assess the explanatory power of those natural examples with two real-world event logs. The results demonstrate that our approach shows the distinctive behavioural traits of variants effectively and intuitively.

Keywords: Process mining · Variant analysis · Explainability · Natural examples · Declarative specifications · Explainable Artificial Intelligence

1 Introduction and motivation

The analysis of *process variants* has emerged as a relevant approach for gaining more fine-grained insights into process behavior. A process variant refers to a subset of cases that share a distinguishing characteristic such as a specific activity sequence, outcome, attribute value, or a combination of them, thereby differentiating them from other cases in the log [25]. Analyzing differences between such variants enables a deeper understanding of the process and potential improvement opportunities. For example, consider an event log related to the management of traffic fines [14]. Its activities include 'Send Fine', 'Payment', and 'Appeal to Judge'. Considering process variants that discriminate whether the payment is completed on time or with a delay, the differences between those variants should show what led to the differing outcomes (e.g., offenders appeal the fine rather than paying it directly).

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-37777-7_19

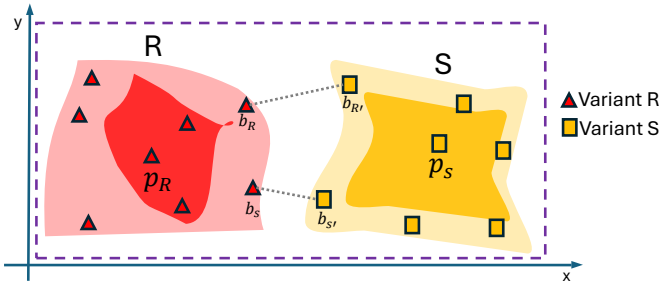


Fig. 1: Bidimensional representation of case examples for two process variants: R and S. Red triangles and yellow squares represent cases of their respective variants. The dashed purple rectangle represents the feature space of the entire log. Color intensity reflects the probability of belonging to a variant. p_R and p_S represent the prototypes and b_n highlight the case of a pair of boundary cases. X and Y are feature projections.

A common approach to identify these differences between process variants is based on training machine learning (ML) models to classify cases into the different process variants considered [25]. From this model, a set of constraints can be extracted to understand what features characterize each process variant. These constraints are typically expressed in an "if X then Y" format, where X is a logical combination of features, and Y indicates a specific process variant. For instance, in the traffic fine scenario, a possible constraint might be: *"If 'Appeal to Judge' occurs in a case, then it belongs to the delayed process variant"*.

While such constraints provide concise and interpretable summaries, they remain abstractions of the underlying process behavior. In parallel, user studies in the field of explainable artificial intelligence (XAI) suggest that examples can complement rule-based explanations by illustrating how model behavior manifests in concrete instances [21]. This finding has motivated the development of example-based explainability methods, which aim to communicate model behavior through representative or contrastive samples, either drawn from the data or synthetically generated [20]. However, the use of examples as explanatory artifacts has received little attention in process variant analysis.

In this paper, we introduce an example-based approach for explaining differences between process variants, inspired by methods from XAI [10]. The approach is grounded in the use of natural examples, i.e., real cases extracted from event logs, as explanatory artifacts. Unlike synthetic examples, natural cases are inherently plausible, require no additional feasibility checks, and avoid dependencies on external generation or correction mechanisms. Our approach leverages an ML model that predicts the process variant of a case based on measurements of declarative behavioral constraints. Rather than treating the model as an end in itself, we use it to structure the behavioral space of cases and to retrieve informative examples. In particular, we extract two complementary types of cases:

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-37777-7_19

prototypes, which are highly representative of each process variant and thereby highlight the most salient differences between variants, and *boundary cases*, which are pairs of correctly classified cases that lie close to the model's decision frontier, thus exposing subtle behavioral divergences between variants that prototypes alone cannot capture. Figure 1 visually sketches the underlying idea. To support interpretation, we further relate these examples to the behavioral patterns that drive the model's predictions using SHAP [15].

The contribution of our work is threefold: *(i)* We introduce an example-based perspective on process variant explanation that treats real cases as first-class explanatory artifacts; *(ii)* We present a method for retrieving informative examples from event logs, including representative (prototype) and discriminative (boundary) cases derived from a behavioral feature space; *(iii)* We offer an open-source implementation of our approach, to test it with real-world event logs.

To evaluate our approach, we first quantitatively assess the quality of the retrieved examples by comparing them against baseline selection strategies with respect to properties such as intra- and inter-class distance. Secondly, we demonstrate through two use cases how the extracted examples complement rule-based explanations by providing concrete, case-level insights into variant differences.

The remainder of this paper is organized as follows. Section 2 presents the background and related work. Section 3 details our proposed approach. Section 4 presents the setting and results of the evaluation. Finally, Sect. 5 offers conclusions, discusses limitations, and outlines possible future directions.

2 Background and related work

This section reviews prior work related to our approach along three directions.

2.1 Process variant analysis

Process variant analysis has received increasing attention in the last years, leading to the development of a variety of approaches. A common way to characterize these approaches is based on the output they provide to describe how process variant differ [25]. A prominent class of approaches focuses on extracting sets of rules that provide interpretable differences that characterize each variant. These approches differ in the way used to identify the rules.

Some approaches rely on process models to identify them, which represent how a process is executed. For example, Swinnen et al. [24] compare process variants against a normative model to identify deviations and extract rules that explain how process variants differ. Statistical techniques have been applied to obtain such rules [2,5]. Among others, Cecconi et al. [2] use statistcal techniques to detect DECLARE constraints that differ significantly in the variants.

Other approaches encode cases as feature vectors labeled with the corresponding process variant. Then, a classifier is trained using these multidimensional representations of process executions. The resulting model is analyzed to extract explanatory differences. Bose and van der Aalst [1] extract features related to

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-37777-7_19

individual activities (e.g., repetitions of activity A) and their sequences (e.g., how often activity A is immediately followed by B). Each case is mapped to a feature vector, and association rules and decision tree algorithms are used to generate conditional statements that describe each variant. Cuzzocrea et al. [6] extend this idea by training an ensemble of classification models to distinguish case classes incorporating features derived from data attributes to the previous ones. Moreover, some approaches have enhanced case encodings by adding declarative features [9,22]. Di Francescomarino et al. [9] encode cases using declarative, activity and attribute features. They use DECLARE to describe their declarative behavior with the following mapping based on constraint activations: -1 if the constraint was violated, 0 if it was not activated, and n if it was fulfilled n times.

Closer to our work, Grigore et al [12] define an agnostic framework to explain process variants globally through the use of prototypes and specific cases through its comparison with prototypes. The variants are automatically identified with unsupervised techniques. Particularly, they defined prototypes as the most central case of a process variant (i.e., the most frequent behaviour). They encoded cases at event and activity levels. We complement this work by including user-defined process variants, using a different case encoding, different prototype definition, and by using boundary cases to show subtle differences between variants.

2.2 Natural example-based explainability methods

Natural example-based explainability methods use instances from a dataset to explain model behavior or characterize specific classes [20]. They are categorized into different approaches depending on the semantic meaning of the examples.

A first line of work focuses on retrieving representative instances of a dataset, referred to as *prototypes*, which summarize the typical characteristics of a class or dataset. Prototypes are commonly used to provide *post-hoc interpretability* of ML models, as they reflect the most important features associated with a given class. In the context of process mining, prototypes have been explored to reduce the complexity of process discovery [11].

A second line of work considers *natural contrastive explanations*, which rely on examples to illustrate how differences in features lead to different model predictions. Among the various types of contrastive explanations, *natural counterfactual examples* identify, for a given instance, the closest instance in the dataset whose model prediction is different [20]. Such pairs highlight minimal changes that alter the model's decision, thereby revealing the boundaries between classes.

In this work, we build on these two perspectives by combining representative and contrastive examples to explain differences between process variants. In particular, we adopt the notion of prototypes to capture typical variant behavior and use pairs of nearby cases with different predictions, referred to as boundary cases, to expose subtle distinctions between variants.

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-37777-7_19

2.3 Declarative behavioral constraints and measures

Our approach relies on declarative behavioral constraints to represent process executions in a feature space used to train a ML model. These constraints describe permissible order relationships between activities without prescribing a fixed execution order [8]. The DECLARE language is a widely used formalism for specifying such constraints, offering templates to express temporal relations between activities [19]. For example, the DECLARE constraint RESPONSE(A, B) states that whenever activity A occurs, activity B must eventually follow.

Declarative constraints can be evaluated on individual cases to quantify their degree of satisfaction. A commonly used measure is *confidence* CONF(X), defined as the ratio of fulfilments over activations of a constraint X within a case, with values in $[0, 1]$ when at least one activation occurs [3]. When a constraint is not activated in a case, its confidence is undefined, corresponding to a vacuous satisfaction [16]. This distinction is relevant when encoding cases, as both activated and non-activated constraints contribute to characterizing behavioral patterns. Encodings based on DECLARE constraints have been used in prior work to support tasks such as classification and deviance detection [9,22]. In this work, we build on these representations by using declarative behavioral features to structure the space of cases, enabling the retrieval of representative and discriminative examples for explaining differences between process variants.

3 The approach

Figure 2 illustrates our approach for retrieving *natural examples* that highlight differences between process variants. To illustrate it, consider the customer appeal process (cf. Section 2) is composed of activities such as *Send Appeal* (*SA*), *Evaluate Appeal* (*EA*), *Request information* (*RI*), *Take Decision* (*TD*), *Accept* (*A*) and *Reject* (*R*). Given the event log and its variants, our approach returns two types of samples that explain their differences: (i) prototypes and (ii) boundary cases. These samples are derived from the output of an ML classifier trained on declarative behavioral patterns. We also report the SHAP values of each sample, which identify the most influential features in the classifier's decisions, to improve their interpretability.

Step 1: Feature extraction. We extract features that represent each case of the input event log. In this paper, we focus on control-flow-based features. This choice ensures broad applicability, as control-flow information, i.e., sequences of activities, is the minimal requirement of event logs. Specifically, we use features derived from declarative behavioral contraints, due to their strong predictive performance in prior work [9]. Each case is described by the confidence values CONF(X) of the discovered DECLARE constraints X, obtained using standard declarative process discovery techniques [16,7,3]. When a constraint is not activated in a case, its value is set to *none*, reflecting vacuous satisfaction. For example, suppose that the travel declaration log contains cases

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-37777-7_19

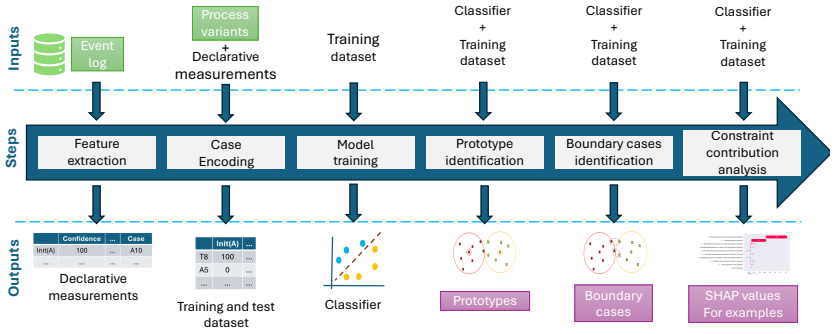


Fig. 2: Overview of our approach. Inputs highlighted in green are given as an external parameter, and outputs highlighted in purple constitute the final outcome.

$c_1 = \langle SA, NCR \rangle$, $c_2 = \langle NC, SA, RA, TDR \rangle$, and $c_3 = \langle RA, RI, A \rangle$. The constraint RESPONSE(SA, NC) has a confidence CONF(RESPONSE(SA, NC)) of 1 for c_1 (fulfilled), 0 for c_2 (violated), and *none* for c_3 (because SA does not occur).

Step 2: Case encoding. Each case is encoded as a feature vector, where each dimension corresponds to a declarative feature obtained in the previous step. These vectors form a matrix X, whose rows represent cases and columns represent features, with each entry storing the corresponding confidence value (see Tab. 1). A label vector Y is also constructed to encode the process variant of each case. Finally, X and Y are randomly split into training and test sets (X_{train}, X_{test} and Y_{train}, Y_{test}), preserving the distribution of process variants.

Step 3: Model training. We train a classifier M on X_{train} and Y_{train} to predict, for a given case c, the probability of belonging to each process variant based on its feature vector. Once trained, M can classify unseen cases using their DECLARE-constraint features. Importantly, M must be well-calibrated and output probability distributions over all variants rather than a single label.

Step 4: Prototype identification. For each variant, we identify a prototype by selecting a case from X_{train}, following standard practice in example-based methods [20]. We define the prototype of a variant V as the training case c that maximizes the predicted probability of belonging to V. Such case exhibits the

Table 1: Excerpt of a matrix containing the feature vectors of cases

	CONF(INIT(SA))	CONF(RESPONSE(SA, NC))	CONF(RESPONSE(RA, RI))	...
c_1	1	1	*none*	...
c_2	1	0	*none*	...
c_3	*none*	*none*	1	...
...	...	...	...	...

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-37777-7_19

most distinctive features of its variant. In case of ties, we prefer shorter cases to favor more concise explanations, breaking remaining ties at random. For example, let the *time-compliant* variant's prototype be $c_{\text{compl}} = \langle SA, EA, TD, NR, A \rangle$, and the prototype of the *non-time-compliant* variant be $c_{\text{notcompl}} = \langle SA, TD, NR, R \rangle$. These cases exhibit distinct behavioral constraints. In the former, the appeal is evaluated (EA) before a decision is made (TD), reflecting that the appeal was performed on time, was reviewed and finally was accepted. Instead, in the latter, the decision (TD) follows immediately the submission because the appeal arrived too late, so it is rejected (R).

Step 5: Boundary case identification. *Boundary cases* are pairs of cases from different process variants that lie close to the decision frontier. They reveal subtle behavioral differences that may explain why a case belongs to one particular variant rather than another [20]. To identify such cases, we rely on the prediction of the trained classifier. Inspired by the *nearest unlike neighbors* concept [10], we search for pairs of cases in X_{train} that (i) are correctly classified in their actual process variant and (ii) are close to another variant according to the classifier output. We quantify this closeness as follows. Let c be a case whose true variant is V, and let $M(c)[V]$ and $M(c)[V']$ denote the probabilities assigned to c by classifier M to variants V and $V' \neq V$, respectively. We define the *Inter-Variant Boundary Proximity* (IVBP) as:

$$\text{IVBP}\big(M, c, V, V'\big) = \frac{1}{1 + |M(c)[V] - M(c)[V']|}$$

A higher IVBP indicates that the two probabilities are closer, and thus that the case lies nearer to the decision boundary between V and V'. For each pair of variants, we select the top-k cases (e.g., $k = 10$) with the highest IVBP scores. Boundary pairs are then formed by choosing the two cases from different variants with the smallest Euclidean distance between their feature vectors. Since some confidence $\text{CONF}(x)$ may be *none* due to vacuous satisfaction, we map them to a large negative value (e.g., -100) when computing the Euclidean distance. This ensures that non-activated constraints are treated as behaviorally distinct from constraints with observed evidence.

Step 6: Constraint contribution analysis. To support the interpretation of prototypes and boundary cases, we analyze the contribution of each DECLARE-constraint feature to the classifier's predictions using SHAP (SHapley Additive exPlanations) values [15]. SHAP quantifies the impact of each feature by measuring the effect of including or excluding it from the model. Positive values indicate the constraint supports the assignment of a case to its predicted variant, while negative values indicate a shift toward alternative variants. For instance, in the *time-compliant* prototype c_{compl}, the constraint $\text{RESPONSE}(SA, EA)$ with confidence 1.0 decisively contributes with a SHAP value of 3.16 toward the prediction of the *timecompliant* class. In contrast, in a boundary case of the same variant where this constraint is not fulfilled (confidence 0.0), its SHAP value of

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-37777-7_19

−0.41 weakens the evidence for the variant and moves the prediction closer to the decision boundary.

4 Evaluation

For the evaluation of our approach, we have implemented an open-source prototype, available at doi.org/10.5281/zenodo.20596012. We applied it to two real-world event logs. Our evaluation is threefold. First, we compare the retrieved samples against several baselines to assess their quality and interpretability. Second, we study how our approach behaves with different configurations to show the implementation-independence of our approach. Third, we illustrate how the extracted samples complement rule-based explanations. All experiments were run on a machine with a 12$^{\text{th}}$-gen Intel Core i7-1265U CPU and 16 GB of RAM.

4.1 Datasets and implementation

We use two event logs commonly used for explainability-oriented process mining [18]: *Road Traffic Fines* [14], and *Sepsis* [17]. The Road Traffic Fines (RTFM) log contains 150,370 cases describing the management of traffic fines in Italy. Each case starts with a fine creation, including an amount attribute. The offender is notified within 90 days, and may pay at any time. A penalty is added if payment is not made on time. Offenders may appeal to a judge or prefecture, and they will be notified of the outcome. If the fine is unpaid, an agency collects it.

Following Völzer et al. [26], we consider four outcome-based variants:
Fully paid: The fine is paid, including expenses and penalties (60,286 cases).
Dismissed: The fine is waived due to a successful appeal (2060 cases).
Collected: A collection agency recovers the unpaid amount (59,010 cases).
Unresolved: Fine is neither fully paid, dismissed, nor collected (29,010 cases).
Four cases related to multiple outcomes were removed, resulting in 150,366 cases.

The Sepsis log contains 1050 cases describing clinical activities in sepsis treatment. Most cases start with a patient being registered and diagnosed. Afterwards, various measurements are performed (e.g., leukocytes), and patients may be transferred to an unit (e.g., intensive care). Finally, they are discharged, but some may return. In this case, variants are defined by *Systemic Inflammatory Response Syndrome* (SIRS) criterion, indicated by a boolean attribute, which is one of the partition criteria suggested by Rubensson et al. [23]:
SIRS–True: SIRS conditions are met, and sepsis is diagnosed (853 cases).
SIRS–False: SIRS conditions are not met, and sepsis is not diagnosed (197).

These event logs were selected because they exhibit substantially different characteristics, as summarized in Tab. 2, including the number of variants and average case length. They also represent two distinct ML settings: binary classification in the Sepsis log, and multiclass classification in the RTFM log. Furthermore, the distribution of cases across process variants differ significantly; while the RTFM log contains three predominant classes (fully paid, collected, and

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-37777-7_19

dismissed), the Sepsis log exhibits a stronger imbalance between its two variants. These differences make the logs complementary for evaluating our approach.

For feature extraction, we used MINERful [13], due to its high performance on large event logs [7]. MINERful discovers DECLARE constraints and can compute, for each case, the confidence CONF of each constraint, with values in the range $[0, 100]$. This resulted in 2189 constraints for the RTFM log and 4704 for the Sepsis log, which we used as classification features. For classification, we employed XGBoost,[4] a gradient boosting framework based on ensembles of decision trees. We chose XGBoost because it outputs probability distributions over variants, supports missing values natively (which is beneficial when DECLARE constraints are vacuously satisfied), and demonstrated strong performance and scalability [4].

4.2 Configuration and performance evaluation of the classifier

We trained the XGBoost classifier using default parameters with early stopping (patience of five rounds) to prevent overfitting. Each dataset was split into 80 % training–validation and 20 % test sets. Performance was evaluated on the test set using the per-class F1-score, which balances precision and recall and is suitable for imbalanced datasets like ours. With the RTFM log, the classifier achieved F1-scores of 1.00 (*collected*), 0.97 (*fully paid*), 0.91 (*unresolved*), and 0.77 (*dismissed*). With the Sepsis log, the F1-scores were 0.96 for *SIRS–true* and 0.82 for *SIRS–false*. These results indicate strong overall performance, with lower scores observed for less frequent or behaviorally overlapping variants. Figure 3 shows the confusion matrices. With the RTFM log, some *unresolved* cases are misclassified as *fully paid*. *Dismissed* and *unresolved* cases are rarely confused. In Sepsis log, almost all *SIRS–true* cases are classified correctly, whereas some *SIRS–false* cases are misclassified.

4.3 Comparison with baselines

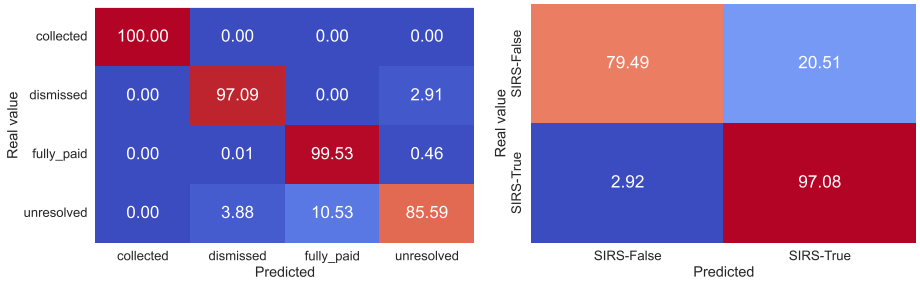
To evaluate the retrieved prototypes and boundary cases, we define a baseline approach for each task. For prototype identification, we use a length-based baseline: the prototype of a process variant is selected as the case whose trace length is closest to the mean length of that variant. In case of ties, one is chosen at random. The baseline is applied to training cases only, mirroring our approach. To assess the representativeness of the discovered prototypes, we compute two

Table 2: Summary of log characteristics

Log	No. cases	No. activities	No. variants	Avg. case length
RTFM	150,366	11	230	3.73
Sepsis	1050	16	846	14.48

[4] https://xgboost.readthedocs.io/en/stable/. Accessed: 31/05/2026.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-37777-7_19



(a) Road Traffic Fines (RTFM) (b) Sepsis

Fig. 3: Prediction distribution for the process variants of each log on the test set. Rows correspond to actual classes, and columns to predicted classes.

metrics based on DECLARE constraints confidence values: (i) *intra-class distance*, i.e., the average Euclidean distance between the prototype and cases of the same variant, and (ii) *inter-class distance*, i.e., the distance to cases from other variants. In both cases, *none* values are mapped to -100 for consistency.

For the identification of boundary cases, we also define a baseline based on length: Let V and V' be two process variants with $V \neq V'$. A boundary case is a case with label V whose trace length is the closest to the mean trace length of V'. Ties for closest trace length are broken randomly.

4.3.1 Prototype. Table 3 reports the prototypes identified by each approach, together with their intra- and inter-class distances. Lower intra-class distance indicates higher representativeness, while higher inter-class distance reflects better separation from other variants. For the *collected* variant, both approaches select the same prototype, likely because the variant is strongly characterized by the presence of the *Send for Credit Collection* (*SC*) activity[5]. For *fully paid*, *unresolved*, and *SIRS–false* variants, our prototypes exhibit lower intra-class distances and higher inter-class distances than the baseline prototypes, indicating improved representativeness and distinctiveness. The opposite trend emerges for the *SIRS–true* and *dismissed* variants, which may be influenced more by trace length than by distinctive behavioral patterns. It is important to note that Euclidean distance may not fully capture the quality of our prototypes. Our approach selects prototypes based on the classifier's probability space, which reflects learned decision boundaries rather than raw distances in an Euclidean space. Thus, a prototype based on probability might be representative for the classifier, but it could not be the best one in terms of Euclidean distance.

Beyond quantitative measures, our prototypes provide intuitive insights into variant behavior. For example, the *unresolved* prototype reflects the absence of *Payment* (*PYM*) and *SC*. Similarly, in the Sepsis log, the *SIRS–false* prototype

[5]Full activity names and their abbreviations are listed in the caption of Tab. 3.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-37777-7_19

shows no activities following *TGS*, indicating that no further clinical actions were required, consistently with the absence of a sepsis diagnosis.

By checking SHAP values, we can analyze the DECLARE-constraint features that contribute most positively to our prototypes. For instance, in the RTFM log, $\text{CONF}(\text{END}(AJ)) = 100$ strongly characterizes the *dismissed* prototype, indicating that the case ends in *AJ*. Conversely, $\text{CONF}(\text{ALTERNATERESPONSE}(PYM, AP)) = 0$ is highly characteristic of the *fully paid* prototype, indicating that *PYM* occurs but without a subsequent *AP*. These insights allow us to pinpoint and confirm the key features of our prototypes.

4.3.2 Boundary cases. Figure 3 shows that some variant pairs are naturally well separated. For instance, *collected* cases consistently receive predicted probabilities close to 99 %, reflecting strong separability driven by the presence of *SC*, an activity absent from other variants. A similar pattern is exhibited by the *dismissed/fully paid* pair. As shown in Fig. 3(a), these variants are rarely confused by the classifier due to fundamentally different behavior captured by their DECLARE features: appeal handling versus payment completion. Fig. 3(b) shows a similar observation for the Sepsis variants. This observation aligns with prior evidence indicating that identifying natural counterfactuals is inherently difficult due to the limited availability of cases that exhibit similar features yet belong to different classes [20].

Table 4 reports the boundary cases for the remaining variant pairs. The baseline boundary cases for *SIRS–false-SIRS–true* show some differences at the beginning (e.g., *CRP* instead of *LC*). However, the size dissimilarity makes it more difficult to figure out the frontiers. The same occurs for the baseline boundary cases of *dismissed-unresolved* process variants. On the other hand, the baseline boundary cases for *fully paid-unresolved* present similar sizes and subtle differences in *SF* and *PYM*. Nevertheless, their sequences match the baseline prototype sequences shown in Table 3. Thus, they are not adequate boundary

Table 3: Prototypes per variant, with intra- and inter-class distances. The distances highlighted in bold represent the best results for a process variant.

Approach	Log	Variant	Intraclass	Interclass	Prototype
Ours	Road Traffic Fines	Collected	107.98	4195.61	⟨ *CF, SF, IF, AP, SC* ⟩
		Dismissed	3860.16	3782.72	⟨ *CF, SF, IF, AP, AJ* ⟩
		Fully paid	**799.06**	**4303.50**	⟨ *CF, PYM* ⟩
		Unresolved	**1126.27**	**3449.54**	⟨ *CF, SF* ⟩
	Sepsis	SIRS-false	**3390.54**	**6947.33**	⟨ *RG, TG, TGS* ⟩
		SIRS-true	5412.00	4404.40	⟨ *RG, TG, TGS, CRP, LC, AB* ⟩
Case length	Road Traffic Fines	Collected	107.98	4195.61	⟨ *CF, SF, IF, AP, SC* ⟩
		Dismissed	**436.59**	**4362.17**	⟨ *CF, SF, IF, ID, AP, SA* ⟩
		Fully paid	2238.75	3736.05	⟨ *CF, SF, PYM* ⟩
		Unresolved	2510.67	3309.89	⟨ *CF, PYM, SF* ⟩
	Sepsis	SIRS-false	4956.33	6131.45	⟨ RG, TG, TGS, CRP, NC, NC, RA, RE ⟩
		SIRS-true	**2790.41**	**5728.74**	⟨ *RG, TG, TGS, CRP, LA, LC, LQ, AB, NC, . . . , RA* ⟩

Activity name abbreviations. (RTFM log) *CF*: Create Fine; *SF*: Send Fine; *IF*: Insert Fine Notification; *AP*: Add Penalty; *PYM*: Payment; *ID*: Insert Date Appeal to Prefecture; *SA*: Send Appeal to Prefecture; *RP*: Receive Result Appeal from Prefecture; *NR*: Notify Result Appeal to Offender; *AJ*: Appeal to Judge; *SC*: Send for Credit Collection. (Sepsis log) *RG*: ER registration; *TG*: ER Triage; *TGS*: ER Sepsis Triage; *LC*: Leucocytes; *AB*: IV Antibiotics; *CRP*: CRP; *LA*: LacticAcid; *LQ*: IV Liquid; *NC*: Admission NC; *RA*: Release A; *RE*: Return ER; *IC*: Admission IC.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-37777-7_19

cases, because they fail to reveal the subtle decision frontier between variants by considering the contrary prototype as a boundary case itself. In contrast, the boundary cases identified by our approach show more subtle divergences reflecting more accordingly the frontiers between the process variants. In the Sepsis log, the *SIRS–false-SIRS–true* pair initially follows similar treatment paths, but diverges after *NC*. The *SIRS–true* case includes additional diagnostic tests (such as *CRP* and *LC*) before release (*RA*). In the RTFM log, the *dismissed-unresolved* pair develops similarly but slightly differs in the final activities. In the *dismissed* case, *SA* is the final activity, whereas in the *unresolved* case finishes in *AJ*. Likewise, in the *fully paid-unresolved* pair, the cases share an identical prefix up to *IF*, but diverge afterward: *SA* occurs after *IF* in the *fully paid* case, but before *IF* in the *unresolved* one. Also, the *unresolved* case contains *PYM* at the end, suggesting inconsistent resolution patterns compared to *fully paid* cases.

Further insights emerge from SHAP explanations of the DECLARE-constraint features contributing to these predictions. For example, in the *fully paid-unresolved* pair, $\text{CONF}(\text{ALTERNATERESPONSE}(PYM, AP)) = 0$ (i.e., *AP* does not occur after every single occurrence of *PYM*) contributes negatively toward the *fully paid* class, which is consistent with the prototype, where no occurrences of *AP* precede *PYM*. Moreover, in the *unresolved-dismissed* pair, $\text{CONF}(\text{END}(AJ)) = 100$ in the *unresolved* case contributes negatively toward its variant, because it contradicts the *dismissed* prototype, which is characterized by ending in *SF* rather than *AJ*.

We notice that the differences showed by our boundary cases might not suffice to give a full account of the discriminants, because the differentiating features are control-flow centric. Other perspectives, such as the data attributes, could be required. We envisage this exploration as a crucial part of future work.

4.4 Configuration study

To demonstrate implementation-independence, we have configured two steps differently and analyzed their results. Due to space constraints, we only report a

Table 4: Boundary cases for the selected pairs of process variants. Activities are abbreviated as in Tab. 3.

Approach	Variant	Boundary cases
Ours	Dismissed	⟨ *CF, SF, IF, AP, ID, SA* ⟩
	Unresolved	⟨ *CF, SF, IF, AP, ID, SA, RP, NR, AJ* ⟩
	Fully paid	⟨ *CF, SF, IF, ID, SA, AP, PYM* ⟩
	Unresolved	⟨ *CF, SF, ID, SA, IF, AP, PYM* ⟩
	SIRS-false	⟨ *RG, TG, TGS, CRP, LC, NC, LC, CRP, LC, LC, CRP, CRP, LC, RA, CRP, LC* ⟩
	SIRS-true	⟨ *RG, TG, TGS, CRP, LC, NC, LC, CRP, CRP, LC, CRP, LC, LC, CRP, CRP, LC, RA* ⟩
Mean length	Dismissed	⟨ *CF, SF, AJ* ⟩
	Unresolved	⟨ *CF, SF, IF, ID, AP, SA* ⟩
	Fully Paid	⟨ *CF, SF, PYM* ⟩
	Unresolved	⟨ *CF, PYM, SF* ⟩
	SIRS-false	⟨ *RG, TG, CRP, LA, LC, TGS, IC, LA, LC, CRP, LA, LC, CRP, LA, NC, RA, RE* ⟩
	SIRS-true	⟨ *RG, TG, TGS, LC, LA, CRP, LQ, AB* ⟩

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-37777-7_19

summary here. Further details are available in our repository.[6] Firstly, we changed Step 1 by encoding cases using a bag of activities (BOA), that is representing a case with sole activity frequencies. We identified a difference for the *Unresolved* variant, where BOA enconding has a worse F1-score (0.83) compared to the DECLARE encoding's F1-score (0.91). Analyzing SHAP values with BOA encoding, we have observed that it provides less descriptive explanations compared to the DECLARE-based ones. For example, the most contributing feature of the *Dismissed* prototype with the BOA encoding is $AJ = 1$ (i.e., an appeal to the judge occurs) while in the DECLARE encoding, it is $\text{CONF}(\text{END}(AJ)) = 100$, clarifying that the prototype *concludes* with the appeal, hence providing a richer explanation.

Secondly, we changed Step 3 by replacing XGBoost with a Histogram Gradient Boosting Classifier (HIST).[7] No differences in F1-scores emerged. Prototype quality varied instead, as HIST achieved better intra- and inter-class distances with the *SIRS–False* variant, while XGBoost achieved better intra-class distances with the *collected*, *fully paid* and *dismissed* variants. HIST boundary cases for the Sepsis log are shorter, hence more intuitive. XGBoost boundary cases for the RTFM log represent frontiers better, since their differences are more subtle (see Tab. 5). Thus, our approach does not rely on a concrete classifier to find good examples. The classifier in use is a parameter to configure based on the desired explainability trade-off (e.g., intuitive or subtle boundary case differences).

4.5 Comparison with rules

To illustrate how our approach complements rule-based explanations, we apply RIPPER, an algorihm of rule extraction, on X_{train}, replacing *none* values with -100. RIPPER has been previously used in process mining to distinguish variants [9]. Table 6 reports the extracted rules for each process variant, expressed as logic predicates over the confidence of DECLARE constraints. For readability, we express rules in disjunctive normal form. For example, three conjunctive clauses characterize *SIRS-false*. If any of those holds (e.g., $\text{CONF}(\text{ABSENCE}(IV\ Antibiotics)) = 100$), then the variant is *SIRS-false*.

Comparison with RTFM rules. To extract rules for the RTFM log, we applied RIPPER on a stratified 20 % sample of X_{train} due to scalability constraints. We limited the maximum number of conditions of the rules to five. The resulting

Table 5: Example of boundary cases by an alternative classifier for the Road Traffic Fines log. Activities are abbreviated as in Tab. 3.

Variant	Boundary cases
Fully paid	⟨ *CF, SF, IF, AP, AJ, PYM, PYM, PYM, PYM* ⟩
Unresolved	⟨ *CF, SF, IF, AJ, AP, PYM* ⟩

[6] Available at https://doi.org/10.5281/zenodo.20425194

[7] scikit-learn/modules/HistGradientBoostingClassifier. Accessed: 31/05/2026.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-37777-7_19

rules clearly characterize variants such as *collected* and *dismissed*. However, for *unresolved* and *fully paid*, the rule sets become harder to interpret: they involve multiple clauses, repeated predicates, and overlapping conditions (e.g., constraints involving *AddPenalty* and *SendForCreditCollection*). As a result, understanding how these variants unfold requires combining several predicates, which is cognitively demanding.

Our prototypes complement these rules by providing compact, concrete representations of variant behavior. For instance, while the rules for *fully paid* involve multiple constraints on penalties and ordering, the prototype directly captures the core behavior as a simple payment completion trace ($\langle CF, PYM \rangle$). SHAP values further align these two views by linking the classifier's decisions on concrete cases to the underlying declarative constraint features. In particular, for the *fully paid* prototype, the constraint AlternateResponse(*Payment*, *AddPenalty*) with confidence 0 (i.e., no penalty following payment) is the strongest contributor to the prediction (1.21). This observation makes it explicit that the absence of penalties after payment is a key distinguishing factor. This insight is only implicitly distributed across multiple clauses. In this way, SHAP acts as a bridge between abstract rule-based explanations and concrete examples, enabling a coherent interpretation across representations.

Table 6: Rules obtained for each process variant of each log. Rules span over different lines, each representing a conjunctive clause. When multiple clauses are listed for a variant, their logical disjunction entails that variant.

Log	Variant	Rules
RTFM	Collected	conf(Absence(*SendforCreditCollection*)) = 0
	Dismissed	conf(End(*Appealtojudge*)) = 100
		conf(CoExistence(*Payment*, *InsertDateAppealtoPrefecture*)) = 0 ∧ conf(Absence(*ReceiveResultAppealfromPrefecture*)) = 100
	Unresolved	conf(AlternateResponse(*SendFine*, *SendForCreditCollection*)) = 0 ∧ conf(End(*SendFine*)) = 100
		conf(AlternateResponse(*AddPenalty*, *SendForCreditCollection*)) = 0 ∧ conf(Alt.Prec.(*Payment*, *AddPenalty*)) = 100
		conf(Alt.Resp.(*AddPenalty*, *SendForCreditCollection*)) = 0 ∧ conf(AtLeast2(*Payment*)) = 0 ∧ conf(Alt.Prec.(*ReceiveResultAppealfromPrefecture*, *InsertDateAppealtoPrefecture*)) = 0 ∧ conf(CoExistence(*ReceiveResultAppealfromPrefecture*, *SendAppealtoPrefecture*)) = 100 ∧ conf(End(*AppealToJudge*)) = 0
	Fully paid	conf(Alt.Resp.(*Payment*, *AddPenalty*)) = 0 ∧ conf(Absence(*SendFine*)) = 100 ∧ conf(Alt.Prec.(*CreateFine*, *Payment*)) = 50
		conf(Alt.Resp.(*Payment*, *AddPenalty*)) = 0 ∧ conf(Absence(*SendFine*)) = 100
		conf(Alt.Resp.(*Payment*, *AddPenalty*)) = 0 ∧ conf(End(*Payment*)) = 100 ∧ conf(Alt.Prec.(*CreateFine*, *Payment*)) = 50 ∧ conf(Absence(*AppealtoJudge*)) = 100 ∧ conf(Absence(*AddPenalty*)) = 100
		conf(Alt.Resp.(*Payment*, *AddPenalty*) = 0) ∧ conf(End(*Payment*)) = 100 ∧ conf(Alt.Prec.(*AddPenalty*, *Payment*)) = 50
		conf(Alt.Resp.(*Payment*, *AddPenalty*)) = 0 ∧ conf(End(*Payment*)) = 100 ∧ conf(Absence(*AddPenalty*)) = 100
		conf(NotPrecedence(*AddPenalty*, *Payment*)) = 0 ∧ conf(Absence(*SendForCreditCollection*)) = 100 ∧ conf(ChainPrecedence(*InsertFineNotification*, *AddPenalty*)) = 100
		conf(Alt.Resp.(*Payment*, *SendForCreditCollection*)) = 0 ∧ conf(Alt.Prec.(*Payment*, *AddPenalty*)) = 0 ∧ conf(Alt.Resp.(*AddPenalty*, *ReceiveResultAppealFromPrefecture*)) = 0 ∧ conf(Alt.Resp.(*SendAppealtoPrefecture*, *ReceiveResultAppealfromPrecture*)) = 0
Sepsis	SIRS-false	conf(Absence(*IVAntibiotics*)) = 100 ∧ conf(NotPrecedence(*AdmissionNC*, *CRP*)) ⩽ 14.29
		conf(Absence(*IVAntibiotics*)) = 100
		conf(ChainResponse(*Leucocytes*, *ReleaseC*)) = 33.33
	SIRS-true	conf(Absence(*IVAntibiotics*)) = 0

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-37777-7_19

The complementarity is also particularly evident when analyzing decision boundaries. For the *dismissed–unresolved* pair, boundary cases reveal that real executions are more nuanced than what rule-based conditions suggest. Although the rules for the *dismissed* variant indicates that cases ending with *AppealToJudge* ($\textsc{conf}(\textsc{End}(Appealtojudge)) = 100$) are classified as *dismissed*, the boundary case $\langle CF, SF, IF, AP, ID, SA, RP, NR, AJ \rangle$ belongs to the *unresolved* variant despite satisfying this condition. This shows that individual rule conditions are not always sufficient to determine the variant, as the broader execution context also plays a role. Boundary cases complement rules by grounding them in concrete executions and revealing where rule-based abstractions break down.

Comparison with Sepsis rules. The extracted rules provide discriminative explanations suggesting that the distinction between *SIRS–false* and *SIRS–true* primarily depends on the presence of *IVAntibiotics*. While this offers a clear characterization, it reduces the difference between variants to a single condition and provides limited insight into how the process unfolds. Our prototypes further clarify how the classifier characterizes each variant. The shared prefix ($\langle$ *RG*, *TG*, *TGS* $\rangle$) appears in both prototypes, indicating that these initial steps are not discriminative for the model. Instead, the distinction emerges afterward: the *SIRS–false* prototype is characterized by the absence of subsequent activities, whereas the *SIRS–true* prototype includes additional diagnostics and treatment, most notably *IVAntibiotics*. Importantly, this distinction is not limited to a single activity. SHAP values show that the classification of *SIRS–false* is driven by the vacuous satisfaction of several declarative constraint features. Therefore, the model does not rely solely on the absence of *IVAntibiotics*, but on a broader lack of continued clinical actions, which remains implicit in the rule-based explanation.

The complementarity is further highlighted when analyzing decision boundaries. Boundary cases for the two variants exhibit highly similar sequences, including repeated diagnostic tests, yet differ in their assigned variant. Notably, both cases lack *IVAntibiotics*, though that is the key rule-based discriminator. This indicates that the rule condition alone is not sufficient to fully explain the classification, as additional contextual factors influence the outcome. Rules identify a dominant distinguishing feature, while prototypes and boundary cases contextualize that feature revealing where its explanatory power is insufficient.

5 Conclusions

In this work, we presented a new approach to analyze process variants using *natural examples*. Our results show that examples can complement rules to this end by grounding abstract conditions in concrete executions and by exposing how variant differences manifest in practice.

However, several limitations should be acknowledged. First, our approach inherits the typical challenges of classification algorithms. One of these is multicollinearity, which arises when two or more features are highly correlated. This can hinder interpretation, as the model may rely on one variable while the actual

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-37777-7_19

contribution comes from another. While not the focus of this work, our use of examples partially mitigates this issue by providing observable evidence of the model's behavior. Second, the evaluation does not involve domain experts, limiting our ability to assess the practical interpretability and usefulness of the explanations. Third, we observed that identifying boundary cases is not always feasible in real-world logs, particularly when variants are well separated. Finally, understanding DECLARE constraints and their associated confidence values may be challenging for users unfamiliar with declarative modeling.

Beyond these limitations, our findings also highlight opportunities. In particular, boundary cases not only explain differences but can reveal when variant distinctions cannot be fully captured by the chosen feature space or by rule-based abstractions. This observation suggests that example-based explanations can act as a diagnostic tool, indicating when additional perspectives (e.g., data attributes) are needed. Future work will explore several directions. We plan to investigate feature selection techniques to reduce multicollinearity and improve interpretability, as well as alternative modeling choices and parameter optimization strategies to enhance robustness. We also aim to incorporate data attributes alongside control-flow features to better capture variant differences that are not behaviorally explicit. Furthermore, we will study mechanisms to ensure the plausibility of boundary cases with respect to the original process behavior. Finally, we intend to conduct comprehensive user studies with process analysts to evaluate the practical usefulness and cognitive effectiveness of example-based explanations.

Acknowledgments. This paper is part of the projects PID2021-126227NB-C21, PID2022-140221NB-I00, and PID2024-156482NB-I00 funded by MICIU/AEI/10.13039/501100011033/FEDER, UE. This work was also funded by the Junta de Andalucía (Consejería de Universidad, Investigación e Innovación) and co-funded by the EU through the FEDER Andalucía 2021–2027 Programme under project DGP_PIDI_2024_01609. C. Capitán-Agudo is supported by the Spanish Ministry of Education under the FPU national plan (FPU21/03631).

References

1. Bose, R.J.C., van der Aalst, W.M.: Discovering signature patterns from event logs. In: CIDM. pp. 111–118 (2013)
2. Cecconi, A., Augusto, A., Di Ciccio, C.: Detection of Statistically Significant Differences Between Process Variants Through Declarative Rules. In: BPM Forum. pp. 73–91 (2021)
3. Cecconi, A., De Giacomo, G., Di Ciccio, C., Maggi, F.M., Mendling, J.: Measuring the interestingness of temporal logic behavioral specifications in process mining. Inf. Syst. **107**, 101920 (2022)
4. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: SIGKDD'22. p. 785–794. Association for Computing Machinery (2016)
5. Cremerius, J., Patzlaff, H., Rahn, V., Leopold, H.: Data-Based Process Variant Analysis. In: HICSS (1 2023)

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-032-37777-7_19

6. Cuzzocrea, A., Folino, F., Guarascio, M., Pontieri, L.: A Robust and Versatile Multi-View Learning Framework for the Detection of Deviant Business Process Instances. Int. J. Coop. Inf. Syst. **25**(04), 1740003 (2016)
7. Di Ciccio, C., Mecella, M.: On the Discovery of Declarative Control Flows for Artful Processes. ACM Trans. Manage. Inf. Syst. **5**(4) (2015)
8. Di Ciccio, C., Montali, M.: Declarative Process Specifications: Reasoning, Discovery, Monitoring. In: Process Mining Handbook, pp. 108–152. Springer (2022)
9. Di Francescomarino, C., Donadello, I., Ghidini, C., Maggi, F.M., Puura, J.: Business Process Deviance Mining with Sequential and Declarative Patterns. Business & Information Systems Engineering **67**, 877–894 (2025)
10. Doyle, D., Cunningham, P., Bridge, D., Rahman, Y.: Explanation Oriented Retrieval. In: Advances in Case-Based Reasoning. pp. 157–168 (2004)
11. Fani Sani, M., Boltenhagen, M., van der Aalst, W.: Prototype selection using clustering and conformance metrics for process discovery. In: BPM Workshops. pp. 281–294 (2020)
12. Grigore, I.M., Marques Tavares, G., Pasquadibisceglie, V., Seidl, T., Barbon Junior, S.: Towards trace variant explainability. In: ADBIS. pp. 95–110 (2026)
13. Iacometta, C., Di Ciccio, C.: Declarative process mining with MINERful, reloaded. In: BPM Demos & Resources. pp. 328–335 (2025), open access
14. de Leoni, M.M., Mannhardt, F.: Road Traffic Fine Management Process (2015). https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5
15. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: NeurIPS. p. 4768–4777 (2017)
16. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: CAiSE. pp. 270–285 (2012)
17. Mannhardt, F.: Sepsis cases - event log (2016). https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460
18. Nannini, L., Catala Bolos, A., Lama, M., Barro, S.: Explainable AI in Process Mining: A Systematic Literature Review (2025)
19. Pesic, M., Schonenberg, H., van der Aalst, W.M.: DECLARE: Full Support for Loosely-Structured Processes. In: EDOC. pp. 287–287 (2007)
20. Poché, A., Hervier, L., Bakkay, M.C.: Natural Example-Based Explainability: A Survey. In: XAI World Conference. pp. 24–47 (2023)
21. Rabold, J., Siebers, M., Schmid, U.: Generating contrastive explanations for inductive logic programming based on a near miss approach. Machine Learning **111**, 1799–1820 (2022)
22. Richetti, P.H.P., Jazbik, L.S., Baião, F.A., Campos, M.L.M.: Deviance mining with treatment learning and declare-based encoding of event logs. Expert Systems with Applications **187**, 115962 (2022)
23. Rubensson, C., Mendling, J., Weidlich, M.: Variants of variants: Context-based variant analysis for process mining. In: CAiSE. pp. 387–402 (2024)
24. Swinnen, J., Depaire, B., Jans, M.J., Vanhoof, K.: A Process Deviation Analysis – A Case Study. In: BPM Workshops. pp. 87–98 (2012)
25. Taymouri, F., La Rosa, M., Dumas, M., Maggi, F.M.: Business process variant analysis: Survey and classification. Knowledge-Based Systems **211**, 106557 (2021)
26. Völzer, H., Zerbato, F., Sulzer, T., Weber, B.: A Fresh Approach to Analyze Process Outcomes. In: ICPM. pp. 97–104 (2023)

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-032-37777-7_19

This document is a pre-print copy of the manuscript
(Capitán-Agudo et al. 2026)
published by Springer (available at link.springer.com).

The final version of the paper is identified by DOI: 10.1007/978-3-032-37777-7_19

References

Capitán-Agudo, Carlos, Claudio Di Ciccio, Cristina Cabanillas, and Manuel Resinas (2026). "Characterizing Process Variants with Natural Examples". In: *BPM*. Ed. by Fabrizio Maria Maggi, Henrik Leopold, Flávia Maria Santoro, and Avigdor Gal. Vol. 16936. Lecture Notes in Computer Science. Springer, pp. 295–312. ISBN: 978-3-032-37776-0. DOI: 10.1007/978-3-032-37777-7_19.

BibTeX

```
@InProceedings{   CapitanAgudo.etal/BPM2026:CharacterizingVariantsNaturalExamples,
  author        = {Capit{\'{a}}n{-}Agudo, Carlos and Di Ciccio, Claudio and
                   Cabanillas, Cristina and Resinas, Manuel},
  booktitle     = {BPM},
  title         = {Characterizing Process Variants with Natural Examples},
  year          = {2026},
  pages         = {295--312},
  crossref      = {BPM2026},
  doi           = {10.1007/978-3-032-37777-7_19},
  keywords      = {Process mining; Variant analysis; Explainability; Natural
                   examples; Declarative specifications; Explainable
                   Artificial Intelligence}
}
@Proceedings{     BPM2026,
  title         = {Business Process Management - 24th International
                   Conference, {BPM} 2026, Toronto, ON, Canada, September
                   27-October 2, 2026, Proceedings},
  year          = {2027},
  editor        = {Fabrizio Maria Maggi and Henrik Leopold and Fl{\'{a}}via
                   Maria Santoro and Avigdor Gal},
  isbn          = {978-3-032-37776-0},
  publisher     = {Springer},
  series        = {Lecture Notes in Computer Science},
  volume        = {16936},
  doi           = {10.1007/978-3-032-37777-7}
}
```