# From Text to Performance Measurement: Automatically Computing Process Performance using Textual Descriptions and Event Logs*

Manuel Resinas[1][0000−0003−1575−406X], Adela del-Río-Ortega[1][0000−0003−3089−4431], Han van der Aa[2][0000−0002−4200−4937]

[1] SCORE Lab, I3US, Universidad de Sevilla, Seville, Spain
resinas@us.es,adeladelrio@us.es
[2] University of Mannheim, Mannheim, Germany
han.van.der.aa@uni-mannheim.de

**Abstract.** Process performance measurement assesses how well a process is running, covering various dimensions such as time, cost, and quality. This task involves the definition of measurable Process Performance Indicators (PPIs), which in many cases are calculated based on data recorded in an event log. An inhibitor of effective performance analysis is that establishing PPI definitions measurable from event logs is highly complex, because it requires process analytical expertise, as well as in-depth knowledge about the structure and contents of the available event data. Given that managers typically do not have such knowledge, this means that those stakeholders that are generally most interested in measuring process performance cannot do so in a convenient manner. Recognizing this, we bridge this gap by proposing an approach for the measurement of process performance based on textual descriptions and event logs, which combines state-of-the-art natural language processing techniques with matching strategies that are tailored to the task at hand. Evaluation experiments using textual descriptions provided by both industry and academic users demonstrate the accuracy of our approach.

**Keywords:** Process performance measurement · process mining · natural language processing · matching

## 1 Introduction

Process Performance Measurement is the practice of evaluating various dimensions of business processes, such as time, cost, and quality, to determine if business processes are achieving strategic and operational goals, and to assist in their optimization. It includes the definition, collection, visualization and analysis of

* This work has been partially supported by projects PID2021-126227NB-C21/ AEI/10.13039/501100011033/ FEDER, UE; TED2021-131023B-C22/ AEI/10.13039/501100011033/ Unión Europea NextGenerationEU/PRTR, and US-1381595 (Junta de Andalucía/FEDER, UE)

Process Performance Indicators (PPIs), which are quantifiable metrics used to evaluate the efficiency and effectiveness of one or more business processes [4]. PPIs can be calculated from different sources, with recorded execution data, stored event logs, being among the main ones [15] and the focus of this paper.

The definition of PPIs calculated from event logs consists of two primary parts: (1) establishing a formal definition of the manner in which process performance should be measured, according to a certain metamodel for PPIs [13,16,21], and (2) linking that definition to the data structure and contents of a specific event log. For example, to measure the "*average time until reimbursement*" in a travel reimbursement process, part (1) involves the recognition that this corresponds to a PPI that should compute the average (an aggregate measure) time (a measure type) between *receiving a request* (start moment) and the *reimbursement being paid* (end moment). Part (2), in turn, requires one to recognize that this measure should be linked to the time between "*receive request*" and "*payment handled*" activities in a particular event log.

The problem here is that both parts of this task involve expertise from users regarding process performance measurement and process mining (how to properly define PPIs, how event logs work, etc.), as well as in-depth knowledge about the data in an event log (which activity or attribute value corresponds to an occurrence of interest, e.g., that the moment of reimbursement, non-straightforwardly, corresponds to a "*payment handled*" activity). Given that such expertise and domain knowledge are rarely (both) held by managers, process performance currently cannot be conveniently measured by those stakeholders most interested in it. Instead, managers need to involve process analysts and domain experts to obtain the insights they desire, which can lead to considerable hindrance in terms of additional effort and delays, as well as possibly incorrect measurements caused by miscommunication [1].

In this work, we overcome this barrier by proposing an approach for measuring process performance based on event logs and textual descriptions. In this manner, our work allows managers to conveniently and quickly obtain useful insights by describing desired performance measures in a textual manner, such as "*the fraction of requests that are rejected*" or "*the maximum time between receiving and delivering orders that were approved*". With this goal, our work complements recent work on using natural language querying in process mining [2, 11], with an approach tailored to the task of process performance measurement.

Our approach builds on a language model fine-tuned to the task of extracting entities from PPI descriptions. To align these extracted entities to the contents of an event log, we propose various matching functions, as well as heuristics to infer missing information. Evaluation experiments using real-world event logs and PPI descriptions collected from industry and academic users highlight the potential of our approach, yet also reveal the challenging nature of the task.

In the remainder, Section 2 describes the challenges of automated transformation, whereas Section 3 provides essential definitions. Section 4 describes our automated approach to compute PPIs described in natural language. Section 5

presents a quantitative evaluation of our approach. Streams of related work are described in Section 6 and we conclude our paper in Section 7.

## 2 Problem Illustration

This section illustrates the main challenges associated with the transformation of textual PPI descriptions into measurable definitions. As a basis for this, we use the well-known *domestic declarations* event log from the 2020 BPI Challenge [5] and the PPI descriptions in Table 1. This process involves the submission, approval or rejection, and payment of travel declarations by employees.

**Table 1.** Exemplary PPI descriptions

| ID | Description |
|---|---|
| *ppi1* | *The average duration between submission and payment of a declaration* |
| *ppi2* | *The average time it takes for a declaration to be paid after its submission* |
| *ppi3* | *The amount of time until reimbursement* |
| *ppi4* | *The percentage of rejected requests* |
| *ppi5* | *The number of denied declarations as a fraction of the total submitted ones* |
| *ppi6* | *The number of declarations above 100 euros* |
| *ppi7* | *The total amount paid per year* |

**C1: High flexibility of natural language.** Textual PPI descriptions can describe the same measure in many different ways. For example, *ppi1* to *ppi3* all describe the time between *submission* and *payment* of declarations, using different structures (e.g., starting point first or last) and terminology (e.g., *duration* versus *time*). Similarly, whereas *ppi4* immediately indicates that this is a fractional measure, this information comes later in *ppi5*, which starts in the exact same manner ("*The number of [..]*") as is common for count measures, such as *ppi6*. These examples are only the tip of the iceberg, though, which means that a transformation approach must be able to deal with highly variable input.

**C2: Differences between description and data.** When describing a measure of interest, users do not necessarily account for the way that a process is recorded, which can result in large differences between the contents of a textual description and an event log. This often results in the use of synonyms (*denying* versus *rejecting*), though differences may also be process specific. For example, *ppi3* refers to the time until *reimbursement*, yet there is no activity in the event log that contains this term. Rather, the moment of *reimbursement* corresponds to the the *Payment handled* activity. Therefore, when matching information extracted from a PPI description to the contents of an event log, an approach must be able to find challenging correspondences.

**C3: Missing information.** Finally, PPI descriptions may leave certain information implicit that is required to define a measure. Common examples are: missing aggregation functions (does *ppi3* refer to the average, total, or individual time until payment?), missing starting points of time measures (what is the

starting point of *ppi3*?), and missing denominators of fractions (e.g., in *ppi4*). A transformation approach needs to be able to make the right choices in such situations, in order to still be able to compute a value for the desired measure.

## 3 Measurable PPI Definitions

This section presents the formal PPI definitions that our approach uses. These definitions are inspired by the PPINOT metamodel [4], which we specifically adapted to the way in which PPIs are commonly described in natural language, allowing for easier and more accurate transformation from textual description to a formal PPI. The values for PPIs defined in this manner can be automatically measured using a PPI computation tool (cf., [15]).

**Scope.** Our work covers a broad range of PPI definitions, allowing users to combine the following aspects. Each PPI definition should correspond to one of three types of base measures: *count*, *time*, and *data*. These can be complemented with additional operators such as *aggregation functions* (e.g., *min.*, *max.*, *average*), *group-by conditions* (e.g., *request per year or department*), *negation* (e.g., *requests not accepted*), and *filters* (such as *above 100 euros* or in the form of a fraction, such as the *fraction of rejected declarations*).

**Definitions.** We formalize measurable PPIs through the following definitions.

**Definition 1 (Universes).** *We define the following universes:*

- $\mathcal{U}_{att}$ *and* $\mathcal{U}_{val}$ *are the universes of* attribute names *and* values *in an event log, including an* activity *attribute to refer to activities and their names. For each* $a \in \mathcal{U}_{att}$*, we use* $dom(a) \subseteq \mathcal{U}_{val}$ *to refer to values that* $a$ *can take.*
- $\mathcal{U}_{agg}$ *is the universe of* aggregation functions*. In this paper we consider* $\mathcal{U}_{agg} = \{avg, max, min, sum, perc\}$.
- $\mathcal{U}_{op}$ *is the universe of* operations*. In this paper* $\mathcal{U}_{op} = \{==, \neq, >, <, \geq, \leq\}$.
- $\mathcal{U}_{case}$ *is the universe of* possible conditions *that refer to a case. In this paper,* $\mathcal{U}_{case} = \{begin, end\}$*, which refer to the beginning and end of a case.*
- $\mathcal{U}_{mval}$ *is the universe of possible values of base measures computed on an event log. This includes integers (for count measures), time intervals like* 7 days *for time measures, and* $\mathcal{U}_{val}$ *for data measures.*

We next define the different kinds of conditions used to specify measures:

**Definition 2 (Conditions).** *We define two sets of conditions:*

- Instant conditions $\mathcal{C}_I = \mathcal{C}_E \cup \mathcal{C}_C$ *comprise event and case conditions.* Event conditions $\mathcal{C}_E = \mathcal{U}_{att} \times \mathcal{U}_{op} \times \mathcal{U}_{val}$ *are tuples that relate an attribute name to a value using a comparison operator. Case conditions* $\mathcal{C}_C = \mathcal{U}_{case}$ *refer to either the beginning or end of a case.*
- Measure conditions $\mathcal{C}_M = \mathcal{U}_{op} \times \mathcal{U}_{mval}$ *are tuples that define a boolean expression based on a comparison against a possible measure value (*$\mathcal{U}_{mval}$*).*

An example of an event condition is $(activity, ==, submit\ declaration)$, which occurs when a *submit declaration* activity is completed, while to select cases

of *declarations above 100 euros*, we specify (>,€100) as a measure condition over a measure value that refers to a case attribute. Using instant and measure conditions, we can define the base measures supported by our approach.

**Definition 3 (Base measures).** Base measures $\mathcal{M}_B = \mathcal{M}_C \cup \mathcal{M}_T \cup \mathcal{M}_D$, *comprise three types:*

- Count measures $\mathcal{M}_C = \mathcal{C}_I \times (\mathcal{C}_M \cup \{\perp\})$ *are tuples that include an instant condition that specifies when to count, and an optional measure condition that is applied to the result of the count* ($\perp$ *as the absence of a condition).*
- Time measures $\mathcal{M}_T = \mathcal{C}_I \times \mathcal{C}_I \times (\mathcal{C}_M \cup \{\perp\})$ *are tuples that include two instant conditions specifying when the time measure starts and stops, respectively, and an optional measure condition* ($\perp$ *as the absence of a condition).*
- Data measures $\mathcal{M}_D = \mathcal{U}_{att} \times (\mathcal{C}_M \cup \{\perp\})$ *are tuples that include the attribute whose value we want to obtain, and an optional measure condition.*

For example, the count measure for *ppi4* is $((activity, ==, reject), (>, 0))$,[3] the time measure for *ppi3* is $(begin, (activity, ==, payment\ handled), \perp)$, and the data measure for *ppi7* is $(amount, \perp)$.

Finally, we define *aggregated measures*, which expand the expressiveness of base measures with aggregation, group-by, and filtering options:

**Definition 4 (Aggregated measures).** *The set of* aggregated measures $\mathcal{M}_A = \mathcal{M}_B \times \mathcal{U}_{agg} \times (\mathcal{U}_{att} \cup \{None\}) \times (\mathcal{C}_E \cup \perp)$ *is the set of tuples such that* $a = (b, agg, att, c)$ *means that the aggregation function agg is applied over the base measure b, grouping by attribute att, and filtering the cases that meet condition c. If* $att = None$*, this means that no grouping is applied; if* $c = \perp$*, this means that no condition is applied.*

For example, the full measure for *ppi1* is $(((activity, ==, submit), (activity, ==, payment\ handled), \perp), average, None, \perp)$, i.e., the average time between the two activities, and for *ppi7* we get $((amount, \perp), sum, 'year', \perp)$, to group the data measure (sum of the *amount* attribute) per year.

## 4 Approach

Figure 1 depicts an overview of our approach. The input is a textual description of a PPI and the output is the result of evaluating this PPI against a given event log. The approach consists of four main steps. Step 1 focuses on the extraction of relevant entities from the textual PPI description (tackling challenge C1). Step 2 matches the extracted entities against the contents of the event log in order to start establishing a measurable PPI definition (challenge C2). Then, for cases in which a user left out certain required information (challenge C3), Step 3 uses various heuristics to fill in the gaps and thereby complete the PPI definition. Finally, Step 4 uses the established definition in order to compute the desired PPI, thus directly measuring process performance for the event log.

[3] $(>, 0)$ is used to count the cases for which this activity happens at least once.

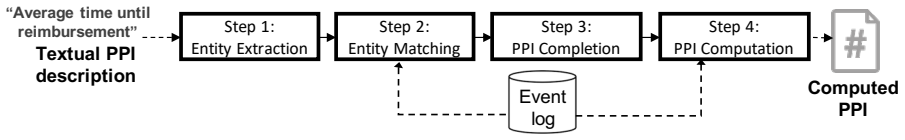


**Fig. 1.** Overview of our approach

### 4.1 Step 1: Entity Extraction

In this first step, our approach takes a textual PPI description $P$ as input and extracts the entities necessary to establish a PPI definition. We first discuss the kinds of entities we extract, before describing the extraction technique itself, and the data augmentation strategy we used to compensate for the small size of the available training data.

**Entity types.** Our approach aims to extract a number of entity classes, which we each denote with a specific tag. The tags for base measures are:

- Count measures: We use a count entity `CE` to capture what should be counted, e.g., *requests for reimbursement* or *accepted orders*.
- Time measures: We use start and endpoints (`TSE` and `TEE`) when present in descriptions, e.g., "*The amount of time until reimbursement*" contains an end point ("*reimbursement*"). However, descriptions of time measures may also use a single entity to refer to the range from start to end, for which we use a `TBE` tag, such as for "*The time to approve declarations*".
- Data measures: We use the `DMA` tag to refer to the description of the attribute to be measured, such as *the total amount* in *ppi7*.

On top of these classes for the measure types, we extract aggregation functions (`AGR`), such as *average* or *maximum*, group-by clauses (`GBC`), such as *per year* in *ppi7*, measure conditions, which are composed of an operator (`CCI`) and a measure value (`MEV`), such as *above 100 euros* in *ppi6*, and filters (`FDE`), such as *pre-approved* in *the total amount of requests that are pre-approved*.

**Extraction technique.** We apply a two-stage approach for entity extraction, using an annotated training dataset of PPI descriptions $\mathcal{D}_T$ (see below) and a pre-trained language model [20] as a basis for both stages. In this paper, we use DistilBERT [18] as the pre-trained language model, but the proposal is independent of the language model used.

In the first stage, we use *text classification* to categorize a PPI description $P$ according to its base type (count, time, or data). To this end, we fine-tune the pre-trained language model, with a linear layer on top of its pooled output, using the gold-standard measure types of the training collection $\mathcal{D}_T$. We use the resulting fine-tuned model to infer the measure type of unseen PPI descriptions.

In the second stage, we use *token classification* to identify the entities of interest in a description $P$. As a basis for this, we use the gold-standard tags of the descriptions in $\mathcal{D}_T$. Since entities can span multiple words, tags are assigned per *chunk*, e.g.: `The\O,average\AGR, time between\O submission\TSE and\O`

`payment of a declaration`\`TEE`.[4] Based on these gold-standard tags, we fine-tune the pre-trained language model for token classification, using a linear layer on top of its hidden-states output. We do this separately for count, time, and data measures, so that we obtain language models that are specifically fine-tuned to extract information from a given measure type, as identified by the aforementioned text classifier.

Given a textual PPI description $P$, we represent the output after token classification as a sequence $\Phi_P \backslash T_P = \langle \phi_1 \backslash t_1, \ldots, \phi_m \backslash t_m \rangle$. Each $\phi_i \backslash t_i$ represents a chunk of text ($\phi_i$) and its assigned tag ($t_i$). Each chunk, $\phi_i$ consists of one or more consecutive words from $P$ and each word is assigned to exactly one chunk.

**Data augmentation.** We only had a collection of 165 PPI descriptions available, including 129 from prior research [1]. However, the fine-tuning of language models requires a considerable amount of training data, especially when dealing with such diverse kinds of input and entities as in our work (challenge C1).

We address this challenge through data augmentation. Specifically, we greatly extend the initial set of PPI descriptions with automatically generated ones. To this end, we handcrafted textual patterns based on the descriptions in the initial set, making sure that a wide variety of different patterns is included. For instance, a typical text pattern for time measures is `[Agg] time from [cond1] to [cond2]`. Then, we used *Chatito*[5], a text generation tool, to generate distinct training phrases by combining all alternatives provided for each pattern. In this manner, we ended up with a total of 12,036 annotated descriptions in $\mathcal{D}_T$.

Using this augmented set to train the aforementioned text and token classification techniques, our entity extraction step can deal with highly flexible input.

### 4.2 Step 2: Entity Matching

We next set out to establish an actual measure $M_P$, according to the concepts defined in Section 3. To illustrate this step, we use *ppi1* as an example, which Step 1 identifies as a *time measure* with the tagged sequence $\Phi_P \backslash T_P$: $\langle$*average*\`AGR`, *submission*\`TSE`, *payment of a declaration*\`TEE`$\rangle$.

To establish $M_P$, we start with the structure of an aggregated measure $(b, agg, att, c)$, as defined in Definition 4, and expand its base measure $b$ with the structure of the identified measure type. Since *ppi1* is a time measure, $b$'s structure is $(s, e, (op, v))$ (cf. Definition 3), yielding $M_P = ((s, e, (op, v)), agg, att, c)$.

**Overall procedure.** Given a tagged sequence $\Phi_P \backslash T_P$, each tagged chunk $\phi \backslash t$ will be used to find a value for an element of $M_P$. The correspondence between a chunk $\phi \backslash t$ and an element $\epsilon$ follows from the tag $t$. For instance, *average*\`AGR` corresponds to $agg$, whereas *submission*\`TSE` and *payment of a declaration*\`TEE`, respectively, correspond to the time measure's start ($s$) and end ($e$).

For a chunk $\phi \backslash t$, we use a *matching function* `match`$(\phi \backslash t)$ to identify the right value for its corresponding element $\epsilon$ in $M_P$, from a target domain $D$. For instance, `match`$(\phi \backslash$`AGR`$)$ identifies the value for its corresponding element $agg$

[4] Tag `O` indicates that a chunk does not belong to any entity from the tag set.
[5] https://rodrigopivi.github.io/Chatito/

from its domain, which is $\mathcal{U}_{agg}$ (cf., Definition 4). As detailed next, we propose six instantiations of $\texttt{match}(\phi\backslash t)$, for different tags $t$. The general procedure is the same, though: `match` evaluates the similarity between the chunk $\phi$ and elements of the target domain $D$ using a similarity measure, returning the element with the highest score.

**Matching `AGR` tags.** Chunks with `AGR` tags correspond to aggregation functions, which means that the matching function's target domain is $\mathcal{U}_{agg} = \{avg, max, min, sum, perc\}$. Therefore, to match $\phi\backslash\texttt{AGR}$, we consider the similarity between $\phi$ and each of the possible values of $\mathcal{U}_{agg}$. Additionally, we also consider synonyms for each of them (e.g., *average*, *mean*, *total average*).

This is formalized as $\texttt{match}(\phi\backslash\texttt{AGR}) = argmax_{d \in \mathcal{U}_{agg}} \texttt{sim}_{comb}(\phi, d)$, where $\texttt{sim}_{comb}$ combines both the syntactic and semantic similarity of text chunk $\phi$ to the respective domain values $d$ of the tag and its synonyms as follows:

$$\texttt{sim}_{comb}(\phi, d) = w_1 \texttt{semSim}(\phi, d) + w_2 \texttt{synSim}(\phi, d)$$

The semantic similarity ($\texttt{semSim}(\phi, d)$) of $\phi$ with a domain value $d$ is the average of the semantic similarity of $d$ and its synonyms. This semantic similarity is the cosine distance between vectors obtained using an algorithm like word2vec[6]. The syntactic similarity ($\texttt{synSim}(\phi, d)$) is determined using the mean of the well-known Damerau-Levenshtein $dl(\phi, d)$ and the Jaro-Winkler $j(\phi, d)$ distances, where we consider the synonym of $\phi$ with the highest score.

**Matching `CCI` tags.** This tag corresponds to an operator (e.g., *above* or *greater than*). Therefore, its target domain is $\mathcal{U}_{op}$. The approach followed is exactly the same as for aggregation functions: $\texttt{match}(\phi\backslash\texttt{CCI}) = argmax_{d \in \mathcal{U}_{op}} \texttt{sim}_{comb}(\phi, d)$.

**Matching `DMA` and `GBC` tags.** These tags corresponds to the attribute name used in data measures (e.g., *amount* in "*the total amount paid per year*"), and *group by* clauses (e.g., "*orders per customer type*"), respectively. For these tags, the function `match` again uses $\texttt{sim}_{comb}$, this time to compare $\phi$ to $L$'s attribute names in $\mathcal{U}_{att}$, which is their target domain. However, for the `GBC` tag, we restrict the target domain to a subset of $\mathcal{U}_{att}$, so that it only includes attributes with a relatively low number of distinct values (we use 15 as a guideline). The rationale is that we expect users to be interested in groupings with a limited number of categories (e.g., per *customer_type*), as opposed to grouping by attributes that have unique values per case (e.g., per *order_ID*).

**Matching `CE` and `TBE` tags.** The target domain of these tags is the set of instant conditions $\mathcal{C}_I$ of the event log $L$, which are composed of three parts: attribute, operator, and value. Therefore, given a chunk $\phi_i\backslash\texttt{TBE}$ like *reimbursement* in *ppi3*, its matched value is (*activity*, ==, *payment handled*). We perform this matching at once, for just a single chunk $\phi$, since users generally do not specify conditions in an attribute-operator-value manner, but rather use a shorthand. For instance, "*reimbursement*" does not explicitly state that this condition refers to an *equals to* operator and that the relevant attribute is an event's *activity*.

A challenge is that the target domain is huge (spanning numerous combinations of attributes and their values), so we apply three actions to reduce it. First,

[6] We use the `en_core_web_lg` model provided by spacy (`https://spacy.io/`).

we only consider categorical attributes whose number of categories is lower than a threshold (we use 100). Second, we only consider the attributes whose value changes across the events of at least one case. The reason is that, if the attribute does not change, it is not useful to use it as a condition in count or time metrics. Finally, we restrict the operator of instant conditions to *equals to*, whereas we support *not equals to* in the PPI completion step (Section 4.3). This restriction of the operator does not apply to the operator identified by the `CCI` tag.

Let $\mathcal{S} \subseteq \mathcal{C}_I$ be the subset of conditions after applying these three actions. Then, to match a text chunk $\phi$ to an instance condition, we define $\texttt{match}(\phi \backslash t) = argmax_{d \in \mathcal{S}} \texttt{conSim}(\phi, d)$, where $t \in \{\texttt{CE}, \texttt{TBE}\}$. Here we quantify the condition similarity `conSim` between $\phi$ and a condition $d = (a, ==, v)$ as follows:

$$\texttt{conSim}(\phi, d) = (1 - w_{att})\texttt{valSim}(\phi, a, v) + w_{att} \sum_{v_i \in dom(a)} \frac{\texttt{valSim}(\phi, a, v_i)}{|dom(a)|}$$

The first part of the equation computes the value similarity (`valSim`) between the text chunk $\phi$ and the attribute-value pair of the condition. The second part of the equation considers the average similarity between the chunk and all values of the attribute domain ($dom(a)$). This allows us to prioritize those attributes whose domain is closer to $\phi$. $w_{att}$ represents the weight given to the latter. $\texttt{valSim}(\phi, a, v)$ itself is computed as follows:

$$\texttt{valSim}(\phi, a, v) = (1 - w_c)\texttt{indSim}(\phi, v) + w_c\texttt{indSim}(\phi, a + v)$$

Here, the first part of the equation computes the individual similarity (`indSim`) between the text chunk and the value of the attribute $v$, whereas the second part of the equation computes the same similarity but considering both the name of the attribute and its value ("$< a > < v >$"). This is done to account for cases where the text chunk either omits or includes the name of the attribute.

Finally, $\texttt{indSim}(\phi, v)$ can be computed using any established similarity measures. In this paper, we combine four similarity measures with a weighted sum: $\texttt{sim}_{comb}$ as defined above, $\texttt{sim}_{is}$, which uses a standard measure for bag-of-words-based similarity we applied in a previous work [1]. $\texttt{sim}_{emb}$, which uses the cosine distance between the sentence embeddings of the chunks [14]. $\texttt{sim}_{bert}$, which uses pre-trained natural language inference models as zero shot text classifiers [22].

**Matching `TSE` and `TEE` tags.** These tags refer to the conditions that determine the beginning (`TSE`) and end (`TEE`) of a time measure such as "*the duration between submission and payment of a declaration.*" When both of these tags are extracted from $P$, we can extend the approach used for `CE` and `TBE` with the following two heuristics to improve its performance. First, the *from* and *to* conditions are likely to be related to the same attribute (e.g., if the first condition refers to an activity, the second condition tends to refer to an activity as well). Second, the *from* condition should commonly occur before the *to* condition in the case. Therefore, given a text chunk $\phi$, we compute the similarity for each

possible pair of conditions $d_i = (a_i, ==, v_i)$ and $d_j = (a_j, ==, v_j)$ as follows:

$$\begin{aligned}\texttt{pairSim}(\phi, d_i, d_j) =&(1 - w_{mh})\frac{\texttt{conSim}(\phi, d_i) + \texttt{conSim}(\phi, d_j)}{2} + \\ &+w_{mh}\frac{\texttt{same}(a_i, a_j) + \texttt{cr}(d_i, d_j)}{2}\end{aligned} \tag{1}$$

Where $\texttt{same}(a_i, a_j)$ has a value of 1 if $a_i = a_j$ and 0 otherwise, and $\texttt{cr}(d_i, d_j)$ is the ratio of cases where the *from* condition occurs before the *to* condition from all cases where both conditions occur. To penalize only those conditions where the condition ratio is significantly low, we apply a logistic function normalized between 0.5 and 1 as follows: $2\left(\frac{1}{1+e^{-k\times cr}} - 0.5\right)$, where $k$ is a parameter that determines the steepness of the curve. We use $k = 10$ in our implementation.

**Matching `MEV` tag.** This tag refers to a measure value used in a measure condition, such as "*100 euros*" in *ppi6*. Its target domain is the set of possible values of base measures computed on an event log $L$ ($\mathcal{U}_{mval}$). The matching is performed using the context provided by the measure type identified in Step 1. For count measures, it just involves parsing an integer. For time measures, we use a syntactic parsing of time deltas. For data measures, it depends on the domain of the attribute used in it. For instance, in *The number of declarations with amount above 100 euros*, we would match *100* with the domain of *amount*, which is an integer. If the domain is categorical, we use $\texttt{sim}_{comb}$.

### 4.3 Step 3: PPI completion

After the previous step, a PPI definition $M_P$ has been built according to the information that is explicitly provided in the textual description $P$. However, as described in challenge C3, some details may have been left implicit, which means that certain mandatory slots in $M_P$ may still be empty. In this step, we fill these remaining slots based on several heuristics, which reflect common-sense interpretations of the missing pieces of information.

**Missing time points.** Descriptions such as "*The amount of time until reimbursement*" only describe a single point in time (the end, here), even though a time measure requires both a start and an end. Therefore, we complete time measures with an unspecified start point by setting it to the earliest timestamp of a case, and use the last timestamp for missing endpoints.

**Default conditions.** Users can provide descriptions of count measures like "*the percentage of rejected requests.*" Such percentage aggregations require measure conditions in order to be properly defined. Therefore, we add a measure condition $> 0$ to the result of the count, in our example: $((activity, ==, declaration\ rejected\ by\ employee), (>, 0))$, to capture that this activity should occur at least once for a case to be considered in the aggregation's numerator.

**Default aggregations.** Users may provide descriptions for time measures such as "*The amount of time until reimbursement*" (*ppi3*). Although these technically do not indicate that this is an aggregate measure, we recognize that a user is likely

not interested in the time of individual instances, which is why we automatically set the aggregation function to *average* here. Similarly, for a count measure such as "*the number of declarations*", we employ a *sum* aggregation by default.

**Applying negation.** Instant conditions can include negations, e.g., *the number of requests that are not paid*, which can be recognized using available *dependency parsers*.[7]. In these cases, we must ensure that the corresponding negated measure is properly defined. For time measures, this is straightforward, since we can just change the operator of the identified instant condition from *equals to* to *not equals to*. However, if we negate the condition of a count measure, such as $(activity, \neq, payment\ handled)$, we would get a situation in which all non-payment activities are counted. Therefore, we instead insert a measure condition, i.e., $(=, 0)$, which ensures that all cases for which the *payment handled* activity did not occur are counted.

### 4.4 Step 4: PPI computation

Step 3 yields a complete PPI definition. To actually compute a value for it, this definition can be translated into the input for a PPI computation tool, like ppinot4py [15] or the Celonis Process Query Language (PQL) [19]. In this paper, we use ppinot4py because there is a direct correspondence between the elements of the PPI definition and the PPINOT model so no transformation is needed.

## 5 Evaluation

To test our approach, we conducted an evaluation in which we compare PPI definitions obtained by our approach to a manually created gold standard.[8]

### 5.1 Evaluation Data

We collected two data sets of textual PPI descriptions for real-world event logs, whose main characteristics are summarized in Table 2. To allow for a high external validity of our evaluation, the data was obtained from different sources. The first one was gathered during different BPM courses with undergraduate and master students and includes 52 PPI descriptions related to the event log of a public traffic fine management process (TF) [12]. The second one was collected using an online questionnaire, through which industry and academic users from different countries provided 53 PPI definitions, all related to the domestic declarations process (DD) [5]. Participation was voluntary and anonymous.

In both datasets, a pre-processing step was needed and, as shown in Table 2, some PPI textual descriptions from the original data collection had to be excluded. These PPI descriptions could not be manually transformed into

[7] We again use the Spacy library for this.

[8] More information, our prototype, and links to the materials can be found at `https://github.com/isa-group/ppinat`.

**Table 2.** Information about the datasets in the test collection

| Dataset | Participants | PPIs | Reason to exclude | | | PPIs tested | | |
|---|---|---|---|---|---|---|---|---|
| | | | (1) | (2) | (3) | Time | Count | Data |
| Traffic fines (TF) | 18 | 52 | 7 | 7 | 18 | 12 | 8 | 0 |
| Declarations (DD) | 14 | 53 | 12 | 9 | 2 | 11 | 17 | 2 |

(1) Information not in the log. (2)Not supported. (3) Ambiguous information.

structured definitions to be computed against the information available in the corresponding event logs. Although most PPIs are time-related, the diversity of the PPI descriptions provided by the participants is noteworthy.

### 5.2 Experimental Setup

**Implementation.** To conduct the evaluation, we implemented the presented approach in Python, available in our repository.

**Hyperparameter search.** The entity-matching step uses various weights to operationalize the matchers. To find the appropriate settings, we perform an exhaustive grid search considering the following values: $w_{\texttt{sim}_x} \in \{0, 0.25, 0.5, 0.75, 1\}$, $w_{att} \in \{0, 0.1, 0.2\}$, $w_c \in \{0, 0.5, 1\}$, and $w_{mh} \in \{0, 0.25, 0.5\}$, where $w_{att}, w_c, w_{mh}$ are the weights defined in Section 4.2, and $w_{\texttt{sim}_x}$ captures the weights of the similarity measures used in `indSim`, testing a total of 945 combinations.

In this manner, we selected a configuration with the following weights: $w_{\texttt{sim}_{is}} = 0.25, w_{\texttt{sim}_{emb}} = 0.5, w_{\texttt{sim}_{bert}} = 0.25, w_{\texttt{sim}_{comb}} = 0, w_c = 0.5$, $w_{att} = 0.2$ and $w_{mh} = 0.25$. We report on the impact of these parameters below in the discussion of the results of Step 2.

**Evaluation measures.** To assess the quality of our approach we use the well-known *precision* and *recall* measures to compare generated PPI definitions to a manually created *gold standard*. The gold standard, available in the repository, was created by the three authors, who independently established measurable definitions for the gathered PPI descriptions based on their understanding of the respective event logs. The few differences were then resolved through a joint discussion. Here, *precision* reflects the fraction of slots that our approach filled correctly according to the gold standard, whereas *recall* represents the fraction of slots filled in the gold standard that were also correctly filled by our approach.

### 5.3 Results

In this section we report on the overall results obtained using our approach, followed by an assessment of its individual steps and configurations.

**Overall results.** Table 3 summarizes the results obtained in our evaluation. The *Approach* column reports on the results obtained by applying our full approach on the data, which shows that it obtains a precision and recall of above 0.70 for both datasets.

As expected, the best results are obtained for the matching of *aggregation* slots, which have a small, fixed target domain. *Condition* slots also get a high

**Table 3.** Evaluation results obtained for the two datasets and various configurations

| Dataset: | Domestic declarations (DD) | | | | | | | Traffic fines (TF) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Config.: | *Approach* | | *Perfect* | | *No comp.* | | | *Approach* | | *Perfect* | | *No comp.* | | |
| Slot type | n | prec. | rec. | prec. | rec. | prec. | rec. | n | prec. | rec. | prec. | rec. | prec. | rec. |
| Aggreg. | 29 | 0.93 | 0.93 | 1 | 1 | 0.88 | 0.76 | 18 | 0.89 | 0.89 | 0.95 | 0.95 | 0.75 | 0.33 |
| Cond. | 12 | 0.8 | 1 | 0.92 | 1 | 1 | 0.08 | 3 | 0.75 | 1 | 1 | 1 | 0 | 0 |
| From | 11 | 0.72 | 0.72 | 0.82 | 0.82 | 0.6 | 0.27 | 12 | 0.58 | 0.58 | 0.75 | 0.75 | 0.33 | 0.08 |
| To | 11 | 0.45 | 0.45 | 0.54 | 0.54 | 0.6 | 0.27 | 12 | 0.67 | 0.67 | 0.83 | 0.83 | 0.67 | 0.17 |
| When | 16 | 0.44 | 0.44 | 0.47 | 0.47 | 0.44 | 0.44 | 6 | 0.83 | 0.83 | 0.87 | 0.87 | 0.83 | 0.83 |
| Total | 82 | 0.70 | 0.72 | 0.78 | 0.80 | 0.67 | 0.44 | 52 | 0.72 | 0.75 | 0.86 | 0.86 | 0.64 | 0.27 |

precision and recall in both datasets, but in this case the majority of true positives comes from the PPI completion of Step 3 (see below). Regarding *from*, *to*, and *when* slots, their precision and recall changes significantly from one dataset to the other, which suggests that they are heavily domain-dependent. Note that we omitted slot types (group-by, data, and filters) with $n \leq 2$.

**Step 1: Impact of entity-extraction quality.** We evaluate if and how any mistakes made by the parser used in Step 1 affect the overall result of our approach. To do this, we also computed results obtained when using perfectly extracted entities (from the gold standard) as input for Step 2. As shown through the *Perfect* column in Table 3, we then obtain a precision of 0.78 and recall of 0.80 for DD, and precision and recall of 0.86 for the TF dataset, showing that our matching strategies are accurate. Compared to the results obtained with our full approach, we observe differences between 0.08 in DD and 0.14 in TF. This improvement is especially apparent for *from* and *to* slots, where more precise extraction leads to clear improvements for entity matching.

**Step 2: Matching configurations.** Next, we assess the impact of the various parameter settings and heuristics used in Step 2 to match extracted entities to the elements of an event log. The results obtained during hyperparameter search (available in the repository) show that the configuration of the matchers considerably affects the overall result quality. Out of the 945 matcher configurations, the best configured matchers outperform the worst matchers by 0.11 for DD and 0.14 for TF in terms of precision and recall. To further examine the effects of the weights, we compared the values of the top 25% and the worst 25% configurations. The best configurations typically combine at least two similarity metrics (out of the four $\texttt{sim}_x$ options) for matching extracted entities to instant conditions for the *from*, *to*, and *when* slots. Moreover, virtually all configurations in the top 25% use a similarity metric based on language models ($sim_{bart}$ or $sim_{emb}$), frequently combining both.

With respect to the matching heuristics, we find that top-25% configurations typically consider the order in which entities matched to *from* and *to* slots appear in traces (by using $w_{mh} > 0$), whereas the worst configurations tend to omit this

consideration (using $w_{mh} = 0$). Furthermore, when matching *from*, *to*, and *when* slots, the majority of the top-5% configurations consider attribute names (on top of attribute values), by setting $w_c = 0.5$. By contrast, there is no clear difference between configurations that assign positive weights to $w_{att}$, which also lets a matcher consider the entire domain of an attribute. Nevertheless, the positive impact of $w_{mh}$ and $w_c$ thus highlight the importance of considering the contents of an event log, in terms of event order and attribute names, during matching.

**Step 3: Impact of PPI completion.** We assess the relevance of the *PPI completion* step by comparing the results of our full approach to those obtained when omitting this step (*No comp.* in Table 3). The results reveal a significant drop in recall, i.e., of 0.28 in DD and 0.49 in TF. This shows that the proposed PPI completion heuristics help to identify many missing default values for *aggregation*, *condition*, *from*, and *to* slots. Precision also decreases (e.g., from 0.72 to 0.64 for TF), because without Step 3, negations are not properly interpreted.

**Runtime efficiency.** We tested our approach on an Intel i9 PC with 64GB of RAM, a 2TB SSD hard drive, and a consumer GPU GeForce RTX 3080 Ti. The average execution time for steps 1 to 3 for each PPI defined for DD and TF is 0.66 and 0.91 seconds, respectively. The initialization time, which is executed just once for each log and involves loading the log and computing the embeddings of the attribute names and values, is 10 seconds for DD and 42 seconds for TF.

### 5.4 Discussion

A post-hoc analysis of the results obtained reveals that the approach faces several challenges related to the entity-extraction and entity-matching steps.

**Entity-extraction challenges.** The usage of a state-of-the-art token classification technique allows our approach to deal with highly flexible input (challenge C1), and performs well with previously unseen terms. However, it is occasionally infeasible to distinguish different entity types based on just a description. For instance, in the TF dataset, the description "*Percentage of fines with an increment*" corresponds to a percentage aggregation over a count entity `CE` (counting the cases with an "*Add Penalty*" activity). By contrast, the description "*Percentage of fines appealed with vehicle class A*", involves a *filter entity* `FDE` on top of the count entity `CE`, even though the two descriptions are structurally identical. Here, one might envision a post-processing step that evaluates all different alternatives and picks the one that fits best with the event log or even fine-tuning the token classification for the specific domain at hand.

**Matching challenges.** The entity-matching step also faces the problem of lack of domain knowledge to resolve its task. For instance, in the DD dataset there are several activities with the text "approval" in them (e.g., *Declaration approved by Administration* and *Declaration final approved by supervisor*). If the PPI description does not provide indications about what kind of approval it refers to, e.g., "*Percentage of declarations are approved*", it is near impossible for the entity matching to tell to which approval activity it should be matched.

A related problem occurs when confounding words appear in the PPI description and the entity-matching step gives them more relevance than the actual key

words of the description. For instance, the DD dataset has two activities called *Request payment* and *Approve request*. When trying to match a PPI description like "*Average time to approve the request for reimbursement*", the matcher has to decide which part of the text is more relevant: "*approve the request*", which matches best with *Approved request* or the confounding "*request for reimbursement*", which matches best with *Request payment*. This problem may be addressed by recognizing when a PPI description refers to the overall goal of a process (e.g., *request for reimbursement*) and using that information to reduce the relevance of these confounding elements for matching.

## 6 Related Work

**Process performance measurement.** Most process mining tools support the computation of some types of PPIs. In most cases, however, they just support a predefined set of metrics, mainly related to time. There are some exceptions to this, e.g the *PQL* language to define customized PPIs in *Celonis* [19]. The main drawback in this case is that the computation results are not designed to be used outside the tool platform and integrated with other tools or workflows. Recently, *ppinot4py* was presented as a library that can be used to compute a wide variety of custom PPIs [15]. Yet, its main limitation is that users need to know low-level details of the log involved as well as technical aspects of the definition of PPIs. There is another thread of works that proposes user-friendly approaches to define custom PPIs, in the form of graphical representations [4,9] or templates [17]. A caveat is that the PPIs from these approaches cannot be directly computed over an event log.

**NLP interfaces for data base querying.** Highly related to our approach is the existing work on defining queries on tables or databases using natural language. There are two main threads in this regard. The first thread uses natural language text and a database schema as input to generate an SQL query that can be directly computed on the database. An example of this is the work presented in [8], but many more can be found in a recent survey [10]. The second thread does not generate an intermediate query model, but uses deep learning to learn the appropriate output, given a natural language query and a database table. Some examples are [7] and [6]. Although these works address a similar problem to the one presented in this paper, the nature of processes and the event logs that collect their information differs significantly from that of databases, which prevents us from using the same approaches off-the-shelf.

**NLP interfaces for process mining.** Finally, there is developing interest in using NLP to facilitate process mining tasks. For instance, Kobeissi et al. [11] present an intent-based natural language interface to allow users to perform queries on an event log. However, their work focuses on queries about event related data, with performance queries being out of scope. In addition, [11] needs to be adapted for each event log, unlike our proposal, which is log-independent and does not require human intervention to hand-craft the training set for each new event log. Barbieri et al. [2] present an architecture to support a conver-

sational, process mining oriented interface to existing process mining tools, but only focused on questions over process execution data. This preliminary work is extended in [3]. It introduces a taxonomy for natural language questions for process mining and also provides support to queries over process behavior and process mining analyses. However, their implementation and evaluation is performed with general questions applicable to any event log, leaving those associated with selected, domain-specific event logs for future work.

## 7 Conclusion

In this paper, we presented a first approach to calculate process performance indicators against a given event log based on textual descriptions. Our work builds on a fine-tuned language model to extract relevant entities, tailored techniques to match these entities to the contents of an event log, as well as completion heuristics to deal with incomplete descriptions provided by users. The evaluation performed yielded promising results, although there are also clear open challenges. It is worth noting that the heuristics used for completing PPIs have been useful in improving the performance of our approach. In addition, the publicly available dataset we established for this work is valuable in itself, as it allows other researchers and us to continue advancing in this direction.

In future work, we aim to improve both the scope and accuracy of our work. For this, we naturally aim to use the exponentially increasing potential of large language models (LLMs) such as GPT-4. A first direction, aiming for accuracy improvements, is to incorporate LLMs directly into our proposed approach by using their functionality to replace parts of our work that currently rely on other NLP technologies, such as the entity-extraction step and the computation of semantic similarity scores. Next to that, we aim to use the conversational capabilities of LLMs to facilitate interaction between a user and an approach such as ours, in order to guide users through the step-by-step definition of PPIs using textual input. In such an interactive setting, the agent can ask clarifying questions where appropriate and, furthermore, allow users to iteratively build up highly expressive performance measures, thus improving both the accuracy and scope of our work.

## Acknowledgments

We thank Maria Isabel Ramos and Javier Vilariño for their support in the implementation.

## References

1. van der Aa, H., Leopold, H., del Río-Ortega, A., Resinas, M., Reijers, H.A.: Transforming unstructured natural language descriptions into measurable process performance indicators using hidden markov models. Inf. Syst. **71**, 27–39 (2017)

2. Barbieri, L., Madeira, E.R.M., Stroeh, K., van der Aalst, W.M.: Towards a natural language conversational interface for process mining. In: ICPM Workshops. pp. 268–280. Springer, Cham (2021)
3. Barbieri, L., Madeira, E.R.M., Stroeh, K., van der Aalst, W.M.: A natural language querying interface for process mining. J Intell Inf Syst (2022)
4. del-Río-Ortega, A., Resinas, M., Durán, A., et al.: Visual PPINOT:A graphical notation for process performance indicators. Bus.Inf.Syst.Eng. **61**(2), 137–161 (2019)
5. van Dongen, B.: BPI challenge 2020 domestic declarations. https://doi.org/10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51
6. Eisenschlos, J.M., Gor, M., Müller, T., Cohen, W.W.: Mate: Multi-view attention for table transformer efficiency (2021)
7. Herzig, J., Nowak, P.K., Müller, T., Piccinno, F., Eisenschlos, J.: TaPas: Weakly supervised table parsing via pre-training. In: ACL. pp. 4320–4333 (2020)
8. Hui, B., Shi, X., Geng, R., Li, B., Li, Y., Sun, J., Zhu, X.: Improving text-to-SQL with schema dependency learning. arXiv preprint arXiv:2103.04399 (2021)
9. Janiesch, C., Matzner, M.: BAMN: a modeling method for business activity monitoring systems. J. Decis. Syst. **28**(3), 185–223 (2019)
10. Katsogiannis-Meimarakis, G., Koutrika, G.: A survey on deep learning approaches for text-to-SQL. The VLDB Journal (2023)
11. Kobeissi, M., Assy, N., Gaaloul, W., Defude, B., Haidar, B.: An intent-based natural language interface for querying process execution data. In: ICPM. pp. 152–159. IEEE (2021)
12. de Leoni, M., Mannhardt, F.: Road traffic fine management process. https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5 (2015)
13. Popova, V., Sharpanskykh, A.: Modeling organizational performance indicators. Inf. Syst. **35**(4), 505–527 (2010)
14. Reimers, N., Gurevych, I.: Sentence-bert: Sentence embeddings using siamese bert-networks. In: EMNLP. ACL (11 2019)
15. Resinas, M., del Río-Ortega, A., Ruiz-Cortés, A.: PPINOT computer and ppinot4py: Two libraries to compute process performance indicators. In: ICPM (Demo track) (2021)
16. del Río-Ortega, A., Resinas, M., Cabanillas, C., Ruiz-Cortés, A.: On the definition and design-time analysis of process performance indicators. Inf. Syst. **38**(4), 470–490 (2013)
17. del Río-Ortega, A., Resinas, M., et al.: Using templates and linguistic patterns to define process performance indicators. Ent. Inf. Sys. **10**(2), 159–192 (2016)
18. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108 (2019)
19. Vogelgesang, T., Ambrosy, J., Becher, D., Seilbeck, R., Geyer-Klingeberg, J., Klenk, M.: Celonis PQl: A query language for process mining. In: Process Querying Methods, pp. 377–408. Springer (2022)
20. Wang, H., Li, J., Wu, H., Hovy, E., Sun, Y.: Pre-Trained Language Models and Their Applications. Engineering (Sep 2022)
21. Wetzstein, B., Ma, Z., Leymann, F.: Towards measuring key performance indicators of semantic business processes. In: BIS. pp. 227–238. Springer (2008)
22. Yin, W., Hay, J., Roth, D.: Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. In: EMNLP. pp. 3914–3923 (2019)