Enabling Semantics-aware Process Mining through the Automatic Annotation of Event Logs

Adrian Rebmann, Han van der Aa

Data and Web Science Group, University of Mannheim, Mannheim, Germany

Abstract

Process mining is concerned with the analysis of organizational processes based on event data recorded during their execution. Foundational process mining techniques analyze such data in an abstract manner, without taking the meaning of these events or their payload into consideration. By contrast, other techniques may exploit specific kinds of information contained in event data, such as resources in organizational mining and business objects in object-centric analysis, to gain more specific insights into an organization's operations. However, the information required for such analyses is typically not readily available. Rather, the meaning of events is often captured in an ad hoc manner, commonly through unstructured textual attributes, such as an event's label, or in unclearly named attributes. In this work, we address this gap by proposing an approach for the automatic annotation of semantic components in event logs. To achieve this, we combine the analysis of textual attribute values, based on a state-of-the-art language model, with novel attribute classification and component categorization techniques. In this manner, our approach first identifies up to eight semantic components per event, revealing information on the actions, business objects, and resources recorded in an event log. Afterwards, our approach further categorizes the identified actions and actors, allowing for a more in-depth analysis of key process perspectives. We demonstrate our approach's efficacy through an evaluation using a broad range of event logs and highlight its usefulness through four application scenarios enabled by our approach.

Keywords: Process mining, Natural language processing, Semantic analysis

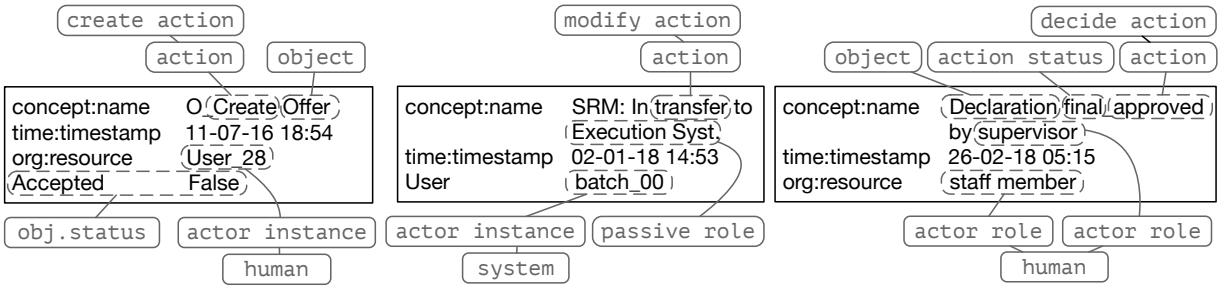
1. Introduction

Process mining refers to a family of techniques that analyze how organizational processes are truly executed [1]. As a basis for this, these techniques use data recorded during the execution of processes, stored in so-called *event logs*. An event log consists of a number of *traces*, each of which represents a sequence of events that was performed for a particular case, such as an individual order or request. Most

Email addresses: rebmann@informatik.uni-mannheim.de (Adrian Rebmann), han@informatik.uni-mannheim.de (Han van der Aa)

foundational process mining techniques treat these traces as sequences of abstract symbols, e.g., $\langle a, b, c, d \rangle$. In this manner, they recognize that events have different classes (e.g., a or b), but they do not consider what these actually mean in relation to the process (what is the role of step a in a process), or what the payload of an event, in terms of its data attributes, says about the performed process step. Beyond such foundational control-flow analysis of business processes, more advanced techniques do take the meaning of events or a part of their data attributes into account. Examples for this include social network analysis [2], which considers the actors that perform events, object-centric process analysis [3], which considers the business objects that are handled in a process, and semantic anomaly detection [4], which detects abnormal process behavior by considering the meaning of performed actions.

A key inhibitor of such advanced techniques is that the information required to conduct these analyses, such as the actions, business objects, and actors associated with events, is not readily available in most event logs. A prime cause for this is the limited standardization of event data, which is neither enforced nor complete with respect to the relevant pieces of information, which we shall refer to as *semantic components*. To illustrate these issues, consider Figure 1, which shows three events from real-life event logs, capturing information on semantic components in various manners.



(a) Event e_1 from BPI17 [5] (b) Event e_2 from BPI19 [6] (c) Event e_3 from BPI20 [7]

Figure 1: Exemplary events and their semantic components

All events have an event label (`concept:name`) and a timestamp (`time:timestamp`), specified using attributes from the XES standard [8], whereas event e_1 also uses the standard `org:resource` attribute to capture which actor performed the event. However, this XES standard is not always followed properly. For example, event e_2 uses `User`, rather than the standard `org:resource` attribute, to indicate the actor, whereas event e_3 erroneously uses this standard attribute to capture information on the actor's role (a *staff member*) rather than on the specific actor instance. Furthermore, the XES standard only covers a limited set of attributes, which means that information on semantic components such as *actions*, *business objects*, and their *status* are not covered by the standard at all. Particularly problematic here is that information on these and other components is commonly not explicitly captured through event attributes, but is rather part of unstructured, textual data attributes associated with events, usually as part of their labels. For example, the

"*Declaration final_ approved by supervisor*" label of e_3 captures the event's business object (*declaration*), the action (*submitted*) along with its status (*final*), and the role of the actor (*supervisor*). Since these components are all contained in the same label, the information cannot be used by process mining techniques.

Enabling this use requires the processing of each individual attribute value in order to identify the included semantic information. Clearly, this is an extremely tedious and time-consuming task when considered in light of the complexity of real-life logs, with hundreds of event classes, dozens of attributes, and thousands of instances. Therefore, this calls for automated support for the semantic annotation of event data, in order to make the information on relevant semantic components available to process mining techniques.

To accomplish this, we propose an approach that automatically identifies semantic information from events without imposing any assumptions on the content or structure of a log's attributes. In particular, our approach aims to identify information on eight *semantic component types*, covering various kinds of information related to business objects, actions, actors, and other resources. After this, our approach further categorizes the identified actions and actors into various categories, allowing for a more fine-granular analysis of the way in which a process is executed and what kind of resources are involved in its execution. To achieve its goal, our approach combines state-of-the-art natural language processing (NLP) techniques, tailored to the task of semantic role labeling, with novel techniques for semantic attribute classification and component categorization.

We assess the accuracy of our approach by applying it on a collection of 14 real-world event logs. Furthermore, we demonstrate its usefulness by showcasing four application scenarios that are only possible thanks to the semantic annotations that our approach provides. Specifically, we show how our approach can be used to (1) refine event labels in order to reduce the complexity of discovered process models, (2) enable object-centric process analysis, (3) abstract event data by grouping together event classes belonging to the same types of actions, and (4) analyze the automation degree of processes based on actor information.

This paper is an extended and revised version of our earlier work on the extraction of semantic process information from event logs [9]. The approach presented here extends its previous version in terms of its accuracy and scope. We improved its *accuracy* by expanding the attribute classification technique that we developed to identify attribute-level semantic information, whereas the *scope* has been broadened by the addition of an additional stage, which covers the aforementioned semantic categorization of identified actions and resources. The utility of both extensions is assessed in additional evaluation experiments, whereas we also added two new application scenarios to demonstrate the value of the expanded scope of our work.

The remainder of the paper is structured as follows. Section 2 defines the scope of our work in terms of the covered semantic information and the main aspects involved in the annotation task. Section 3 presents our improved and extended annotation approach. Section 4 reports on evaluation experiments that show that our approach achieves accurate results on real-life event logs, spanning various domains and varying considerably in terms of their informational structure. Afterwards, Section 5 highlights the usefulness of our

approach in a qualitative manner by using it in four application cases. Finally, Section 6 discusses streams of related work, whereas Section 7 concludes the paper.

2. Scope

This section describes the scope of our work in terms of the covered semantic information and the main aspects involved in the annotation task. We divide this discussion into two parts: Section 2.1 covers the scope with respect to the identification of semantic components in event data, whereas Section 2.2 discusses the subsequent further categorization of identified components into more specific types.

2.1. Semantic Component Identification

Given an event log, our work first sets out to annotate pieces of information associated with events that correspond to particular *semantic components*. This identification task covers the following semantic component types (Section 2.1.1) and aspects (Section 2.1.2).

2.1.1. Semantic Component Types

Our work covers various component types that support a detailed analysis of business process execution from a behavioral perspective, i.e., we target semantic components that are commonly observed in event logs and that are relevant for an order-based analysis of event data. Therefore, we consider information related to *business objects*, *actions*, and active and passive *resources* involved in a process' execution. For each of these categories, we define multiple semantic component types:

Business objects. In line with convention [10], we use the term *business object* to broadly refer to any artifact or entity that is being handled in a process, which covers documents such as an *offer* or a *declaration*, physical objects such as a *car* or a *computer*, but can also an relate to, e.g., a *customer* or an *applicant*. For the events in a log, our work annotates two component types:

- `object` as the type of a business object, such as an *offer* or a *declaration* associated with an event, and
- $\texttt{object}_{status}$ as the reported status of a business object, e.g., whether an identified order is indicated as being *open*, *canceled*, or *accepted*.

Actions. Similarly, we define two types of components to capture information on the actions that are applied to business objects:

- `action` as the performed action itself, e.g., *create*, *transfer*, or *approve*, and
- $\texttt{action}_{status}$ as further information on its lifecycle status, e.g., *started*, *paused*, *final*, or *completed*.

Resources. We capture information regarding the active resources of events, i.e., the entities that actually performed the recorded actions, in the following two component types:

- `actor`$_{role}$ as the type or role of active resource in the event, e.g., a "*supervisor*" or a "*system*", and
- `actor`$_{instance}$ for information indicating the specific actor instance, e.g., an employee identifier such as "*User_28*".

Aside from the actor, events may also store information on *passive* resources involved in an event, primarily in the form of *recipients*. For this, we again define two component types:

- `passive`$_{role}$ as the type or role of passive resource related to the event, e.g., the "*supervisor*" receiving a document or a *system* on which a file is stored or transferred through, and
- `passive`$_{instance}$ for information indicating the specific resource, e.g., an employee or system identifier such as "*batch_00*".

Coverage and extensibility. The considered semantic component types enable a broad range of fine-granular insights into the execution of a process. For example, the *business object* and *action* categories allow one to obtain detailed insights into the business objects moving through a process, their inter-relations, and their life-cycles. Furthermore, by also considering the resource-related components, one can, for instance, gain detailed insights into the resource behavior associated with a particular business object, e.g., how resources jointly collaborate on the processing of a specific document.

While the covered component types, thus, support a wide range of analyses and are purposefully selected based on their relevance in real-life event logs, our approach is by no means limited to these specific component types. Given that we employ state-of-the-art NLP and classification technology that generalizes well, the availability of appropriate event data allows our approach to be easily extended to cover additional semantic component types, both within and outside the informational categories considered here.

2.1.2. The Component Identification Task

To ensure that all relevant information is identified in an event log, our work considers two aspects of the *semantic component identification* task, concerned with two kinds of event attributes: *attribute-level classification* for attributes dedicated to a single semantic component type and *instance-level labeling* for textual attributes covering various component types:

Attribute-level classification. Attribute-level classification aims to determine the component types of attributes that provide the same kind of information throughout an event log, e.g., a `doctype` attribute indicating a business object (`object`) or an `org:resource` attribute capturing information on the specific resource performing an event (`actor`$_{instance}$).

Although the XES standard [8] specifies several dedicated event attributes, such as `org:resource` and `org:role`, these only cover a subset of the semantic component types relevant for our approach. Specifically, they omit component types related to business objects, actions, and passive resources. Information on these types may, thus, be captured in attributes with diverse names, e.g., in the real-life Hospital log [11], the status of business objects (`object`$_{status}$) is jointly indicated by several event attributes, such as `isCancelled`

and `isClosed`. Furthermore, even for component types covered by standard attributes, there is no guarantee that event logs adhere to the conventions, e.g., rather than using `org:group`, the BPI14 [12] log captures information on actors in an `Assignment_Group` attribute.

Instance-level labeling. Instance-level labeling, in turn, aims to identify semantic information for attributes with unstructured, textual values that encompass various semantic component types, differing per event instance. This task is most relevant for so-called event labels, often stored in a `concept:name` attribute.

Log	ID	Event label	Semantic components
WABO [13]	l_1	T08 Draft and send request for advice	`action` (×2), `object`
BPI15 [14]	l_2	send design decision to stakeholders	`action`, `object`, $\texttt{passive}_{role}$
BPI15 [14]	l_3	send letter in progress	`action`, `object`, $\texttt{action}_{status}$
RTFM [15]	l_4	Insert Date Appeal to Prefecture	`action`, `object`, $\texttt{passive}_{role}$
BPI19 [6]	l_5	Vendor creates invoice	$\texttt{actor}_{role}$, `action`, `object`
BPI19 [6]	l_6	SRM: In transfer to Execution Syst.	`action`, $\texttt{passive}_{role}$
BPI20 [7]	l_7	Declaration final_approved by supervisor	`object`, $\texttt{action}_{status}$, `action`, $\texttt{actor}_{role}$
BPI17 [5]	l_8	O_Create Offer	`action`, `object`

Table 1: Exemplary event labels from real-life event logs with their semantic components.

These labels contain highly valuable semantic information, yet also present considerable challenges to their proper handling, as illustrated through the real-life event labels in Table 1. The examples highlight the diversity of textual labels, in terms of their structure and the semantic component types that they cover. It is worth mentioning that such differences may even exist for labels within the same event log, e.g., labels l_5 and l_6 (the label of e_2 of the running example) differ considerably in their textual structure and the information they cover, yet they both stem from the same event log. Another characteristic to point out is the possibility of recurring component types within a label, such as seen for label l_1, which contains two `action` components: *draft* and *send*. Hence, an approach for instance-level labeling needs to be able to deal with textual attribute values that are highly variable in terms of the information they convey, as well as their structure.

2.2. Semantic Component Categorization

In this extended version of our initial semantic extraction work [9], we follow the identification of semantic components with a further component categorization step. In this step, we use *action categorization* to classify identified actions into pre-defined types of high-level actions, whereas we use *resource categorization* to distinguish between human and system actors involved in a process.

Action categorization. Due to the wide range of domains in which organizational processes occur, processes can consist of a plethora of different actions, resulting in a virtually unlimited universe of potential actions. Thanks to the state-of-the-art NLP technology on which our work builds, our component identification approach can still recognize `action` components in an accurate manner, despite this variety. Nevertheless, various kinds of process analysis, such as abstraction, filtering, and conformance checking, can benefit from having an understanding about which actions (or activities) in a process serve a similar purpose and what that purpose actually entails, e.g., *improving* and *updating* both refer to a modification of a certain business object. This calls for a reduction of the range of actions observed in a process to a set of known higher-level action types.

To operationalize this, we use *action categorization* to assign a meaningful action type to each `action` component identified for a process. As a basis for this categorization, we adopt the established action classification framework of the MIT Process Handbook [16], which classifies process-related actions in a hierarchical manner[1]. The top-most level of this hierarchy defines eight high-level actions, as follows:

- *Create*: An action is classified as *create* if its essence is focused on the creation of an output of some sort, e.g., *producing* or *documenting* something.
- *Modify*: An action is classified as *modify* if its essence is focused on changing some attribute of the input as the output, e.g., *improving* or *sending* something.
- *Preserve*: An action is classified as *preserve* if its essence is to keep the input unchanged as an output, just at a later point in time, e.g., *storing* or *packaging* something.
- *Destroy*: An action is classified as *destroy* if its essence is focused on the destruction of an input of some sort, e.g., *retiring* or *eliminating* something.
- *Combine*: An action is classified as *combine* if its essence is grouping or integrating multiples of an input into a single collected output, e.g., *grouping* or *matching* something.
- *Separate*: An action is classified as *separate* if its essence is ungrouping or splitting a single collected input into multiple outputs, e.g., *dividing* or *extracting* something.
- *Decide*: An action is classified as *decide* if its essence is a choice among multiple alternatives, e.g., *determining* or *approving* something.
- *Manage*: An action is classified as *manage* when the actual process to be used is yet unspecified, such as a means of coordinating a dependency or other process, e.g., *assigning* or *scheduling* something.

A categorization of actions into these top-level action types then enables a more detailed view on the process allowing analyses based on their meaning. For instance, it allows us to separate a process into different stages, such as creation, processing (modifying), and decision phases, as we later illustrate in a

[1]Note that our work is independent of this specific classification scheme. It can be replaced with any categorization for which instance data is available, such as the broader verb classification framework by Levin [17].

real-world application scenario (Section 5).

Resource categorization. Being aware whether process steps are performed by humans or systems allows for a detailed analysis of a process with respect to the resource perspective. For instance, it enables an assessment of the degree of automation or system support of a process, as later shown in an application scenario (Section 5). Furthermore, in the context of organizational mining, the interactions among employees are of particular interest, which requires the ability to distinguish human from non-human resources. To enable such analyses, we further categorize identified actors into system and human resources through *resource categorization*.

While this categorization thus involves only two classes, properly operationalizing it is particularly interesting. Specifically, a categorization approach has to take into account that the nature of a resource can be derived from various kinds of information, which may or may not be available in a given event log or for a particular resource. For example, while $\texttt{actor}_{role}$ components that indicate roles such as a *supervisor* or a *service*, already reveal the category of an actor from a semantic perspective, such clear descriptions are rarely available. Similarly, $\texttt{actor}_{instance}$ components can be expressive, e.g., *User_28* or *batch_00*, but may also be unspecific, e.g., *res_90*. Thus, both types of semantic components can provide useful information, but cannot be solely relied on. Nevertheless, even when no meaningful information is available in the actor-specific components themselves, insights about the resource category may still be derived by looking at the context in which an actor operates. This can, for example, be achieved by considering the kind of activities an actor performs—a resource associated with "*SRM: In transfer to Execution Syst.*" events is likely to be a system—whereas also the duration of activities (instant versus highly variable), can be useful to distinguish between actor types.

3. Semantic Annotation Approach

This section presents our approach for the semantic annotation of event data. Section 3.1 first introduces the approach at a high level, whereas Section 3.2–Section 3.4 describe its main stages in detail. Finally, Section 3.5 describes the output our approach generates in detail.

3.1. Approach Overview

The main input, main structure, and output of our semantic annotation approach are as follows:

Approach input. Our approach takes as input an event log L that consists of events recorded by an information system. Each event $e \in L$ carries information in its payload. This payload is defined by a set of (data) *attributes* $\mathcal{D} = \{D_1, \ldots, D_p\}$ with $\texttt{dom}(D_i)$ as the domain of attribute D_i, $1 \leq i \leq p$, and $\texttt{name}(D_i)$, its name. We write $e.D$ for the value of D for an event e.

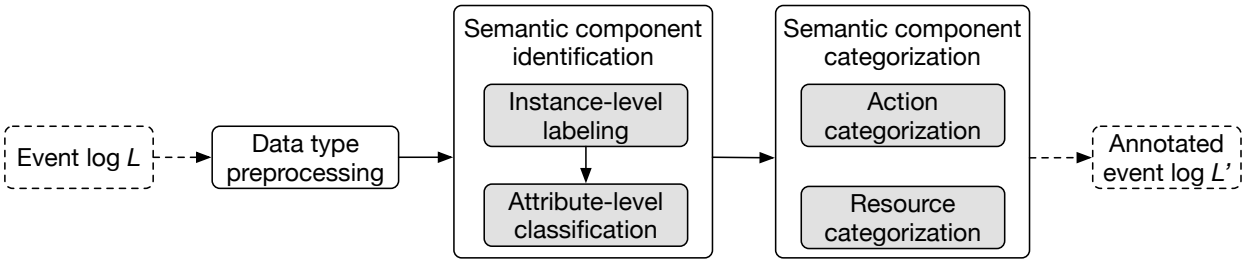


Figure 2: Overview of the approach.

Note that we do not impose any assumptions on the attributes contained in an event log L, meaning that we do not assume that attributes such as `concept:name` and `org:role` are included in $\mathcal{D}$.

Approach structure. The goal of our approach is to annotate the events in a log with additional information about the semantic components contained in an event's payload, as described in Section 2. To achieve this, our approach consists of three main parts, as illustrated in Figure 2.

Given a log L, the *data type preprocessing* step first analyzes the domains of attributes in order to differentiate among textual, miscellaneous, and irrelevant attributes. Next, the *semantic component identification* stage aims to annotate each event with up to eight different component types. This stage consists of two subsequent steps: *instance-level labeling*, which annotates the individual values of textual attributes, and *attribute-level classification*, which determines the appropriate component type for entire attributes. Afterwards, our approach proceeds with the *semantic component categorization* stage, which consists of two independent steps: in the *action categorization* step, identified `action` components are classified according to eight high-level types, whereas *resource categorization* differentiates between `human` and `system` resources.

Output. As output, our approach returns an augmented event log L', in the XES format, in which each event e is extended with additional semantic information, i.e., with its action(s) and action type(s), business object(s), and various kinds of resource-related information.

3.2. Data Type Preprocessing

In this step, our approach identifies sets of textual attributes $\mathcal{D}^T$, which will serve as input for the instance-level labeling step, and miscellaneous attributes $\mathcal{D}^M$, which will become the input to the attribute-level classification step. Attributes not included in either of these sets are deemed as irrelevant for our purposes and, thus, omitted from further consideration. This preprocessing step is performed as follows.

Data type classification. We first use standard techniques, such as provided by the *Pandas* library[2], to classify each attribute in $\mathcal{D}$ as either `timestamp`, `numeric`, `boolean`, or `string`, based on its domain.

[2] https://pandas.pydata.org

Identifying textual attributes. To identify the set of textual attributes $\mathcal{D}^T$, we differentiate between `string` attributes with true natural language values, e.g., "*Declaration final_ approved by supervisor*" or "*O_ Create Offer*", and other kinds of alphanumeric attributes, with values such as "*User_ 28*", "*A*", and "*R_ 45_ 2A*". Only the former kind of attributes will be assigned to $\mathcal{D}^T$ and, thus, analyzed on an instance-level in the remainder of the approach. We identify such true textual attributes as follows:

1. Given a `string` attribute, we first apply a tokenization function `tok`, which splits an attribute value into lowercase tokens (based on whitespace, camel-case, underscores, etc.) and omits any numeric ones. E.g., given s_1 = "*Declaration final_ approved by supervisor*", s_2 ="*User_ 28*", and s_3 = "*08_ AWB45_ 005*", we obtain: $\texttt{tok}(s_1)$ = [*declaration*, *final*, *approved*, *by*, *supervisor*], $\texttt{tok}(s_2)$ = [*user*] and $\texttt{tok}(s_3)$ = [*awb*].
2. We apply a *part-of-speech tagger*, provided by standard NLP tools (e.g., Spacy [18]), to assign a token from the Universal Part of Speech tag set[3] to each token. In this manner, we obtain [(declaration, *NOUN*), (final, *NOUN*), (approved, *VERB*), (by, *ADP*), (supervisor, *NOUN*)] for s_1, [(user, *NOUN*)] for s_2, and [(awb, *PROPN*)] for s_3.
3. Finally, we exclude any attribute that only has values with the same token in $\texttt{tok}(s)$ or that do not contain any `NOUN`, `VERB`, `ADV`, `ADP`, or `ADJ` tokens. In this way, we omit attributes with values such as s_2 = "*User_ 28*" and s_3 = "*08_ AWB45_ 005*", which are identifiers, rather than real textual attributes. The other attributes, which have diverse, textual values, e.g., s_1 ="Declaration final_approved by supervisor", are assigned to $\mathcal{D}^T$.

Selecting miscellaneous attributes. We also identify a set of non-textual attributes that are candidates for semantic labeling, referred to as the set of miscellaneous attributes, $\mathcal{D}^M \subseteq \mathcal{D} \setminus \mathcal{D}^T$. This set contains attributes that are not included in $\mathcal{D}^T$, yet have a data type that may still correspond to certain semantic component types, such as statuses or identifiers.

To establish $\mathcal{D}^M$, we first discard those attributes in $\mathcal{D} \setminus \mathcal{D}^T$ categorized as `timestamp` attributes, as well as `numeric` attributes that include *real* or *negative* values. We exclude these because they are not used to capture semantic information. By contrast, the remaining attributes have data types that may correspond to component types, such as `boolean` attributes that can be used to indicate specific states, e.g., `Accepted`, whereas non-negative integers are commonly used as identifiers, e.g., a `customer` attribute with values such as "*32015*" and "*49102*". These remaining attributes are then joined with the `string` attributes that were not selected for $\mathcal{D}^T$, e.g., attributes with values such as the aforementioned "*User_ 28*", and "*08_ AWB45_ 005*" examples, in order to form the set of miscellaneous attributes $\mathcal{D}^M$.

In this manner, given the log of example event e_1, for instance, `concept:name` and `org:resource` are assigned to $\mathcal{D}^T$, `Accepted` is assigned to $\mathcal{D}^M$, and `time:timestamp` is omitted from consideration.

[3] https://universaldependencies.org/docs/u/pos/

3.3. Semantic Component Identification

The *semantic component identification* stage consists of two subsequent steps. First, *instance-level labeling* processes the values of textual attributes to extract the parts that correspond to semantic component types, e.g., recognizing that a "*document received*" event label contains the business object "*document*" and the action "*received*". Afterwards, the *attribute-level classification* step identifies the appropriate component type for each of the remaining attributes, i.e., it aims to determine the semantic component type that corresponds to all values of a certain attribute by considering its value domain as well as its name. For example, it recognizes that all values of a `doctype` attribute correspond to the `object` component type. The details of these steps are as follows.

3.3.1. Instance-level Labeling of Textual Attributes

In this step, our approach annotates the values of textual attributes in order to extract the parts that correspond to certain semantic component types, e.g., recognizing that the "*O_ Create Offer*" label of event e_1 contains "*offer*" as the `object` and "*create*" as the `action`. As discussed in Section 2.1.2, this task comes with considerable challenges due to the high diversity of textual attribute values in terms of their linguistic structure and informational content. To be able to deal with these challenges, we therefore build on state-of-the-art developments in the area of natural language processing.

Tagging task. We approach the labeling of textual attribute values with semantic component types as a text tagging task. Therefore, we instantiate a function that assigns a type from the eight component types described in Section 2.1 to chunks (i.e., groups) of consecutive tokens from a tokenized textual attribute value. Formally, we use $\mathcal{T}$ to refer to the set of component types and we denote the outcome of the tokenization of an attribute value for a given event $e \in L$ and attribute $D \in \mathcal{D}^T$, as $\texttt{tok}(e.D) = \langle w_1, \ldots, w_n \rangle$, where each token represents a word w_i. Then, we define a function $\texttt{tag}(\langle w_1, \ldots, w_n \rangle) \rightarrow \langle c_1 \backslash t_1, \ldots, c_m \backslash t_m \rangle$, where c_i for $1 \leq i \leq m$ is a chunk consisting of one or more consecutive tokens from $\langle w_1, \ldots, w_n \rangle$, with $t_i \in \mathcal{T} \cup \{\texttt{other}\}$ as its associated semantic component type (or `other` for words that are not part of a semantic component). For instance, $\texttt{tag}(\langle \textit{create}, \textit{offer} \rangle)$ yields $\langle \textit{create} \backslash \texttt{action}, \textit{offer} \backslash \texttt{object} \rangle$, while $\texttt{tag}(\langle \textit{declaration}, \textit{final}, \textit{approved}, \textit{by}, \textit{supervisor} \rangle)$ yields $\langle \textit{declaration} \backslash \texttt{object}, \textit{final} \backslash \texttt{action}_{status}, \textit{approved} \backslash \texttt{action}, \textit{by} \backslash \texttt{other}, \textit{supervisor} \backslash \texttt{actor}_{role} \rangle$.

BERT. To instantiate the `tag` function, we employ BERT [19], a state-of-the-art transformer-based language model that is capable of dealing with highly diverse textual input and achieves accurate results on a wide range of NLP tasks. BERT has been pre-trained on huge text corpora in order to develop a general understanding of a language. This model can then be *fine-tuned* by training it on an additional, smaller training data collection to target a particular task. In this manner, the trained model combines its general language understanding with aspects that are specific to the task at hand. In our case, we thus fine-tune BERT in order to tag chunks of textual attribute values that correspond to semantic component types.

Fine-tuning. For the fine-tuning procedure, we manually labeled a collection of 13,231 unique textual values stemming from existing collections of process models [20], textual process descriptions [21], and event logs (see Section 4.1). As expected, the collected samples do not capture information on resource instances, and rather contain information on the type level (i.e., $\texttt{actor}_{role}$ and $\texttt{passive}_{role}$). For those semantic component types that are included in the samples, we observe a considerable imbalance in their commonality, as depicted in Table 2. In particular, while component types such as `object` (14,629 times), `action` (12,573), and even $\texttt{passive}_{role}$ (1,191) are relatively common, we only found few occurrences of $\texttt{actor}_{role}$ (135), $\texttt{object}_{status}$ (92), and $\texttt{action}_{status}$ (30) component types.

Table 2: Characteristics of the training data used to fine-tune our BERT-based language model

Source	**Count**	`object`	$\texttt{object}_{status}$	`action`	$\texttt{action}_{status}$	$\texttt{actor}_{role}$	$\texttt{passive}_{role}$	`other`
Process models	11,658	13,543	50	11,445	3	58	1,058	4,966
Textual desc.	498	503	11	498	0	8	114	206
Event logs	625	583	31	630	27	69	19	291
Augmentation	450	350	100	350	150	200	0	150
Total	13,231	14,979	192	12,923	180	335	1,191	5,613

To counter this imbalance, we created additional training samples with $\texttt{object}_{status}$, $\texttt{action}_{status}$, and $\texttt{actor}_{role}$ component types using established data augmentation strategies. In particular, we created samples by complementing randomly selected textual values with (1) known $\texttt{actor}_{role}$ descriptions, e.g., "*offer created*" is extended to "*offer created by supervisor*", and (2) common life-cycle transitions from [1, p.131] to create samples containing $\texttt{object}_{status}$ and $\texttt{action}_{status}$ component types, e.g., "*check invoice*" is extended to "*check invoice completed*". However, as shown in Table 2, we limited the number of extra samples to avoid overemphasizing the importance of these component types.

Given this training data, we operationalize the *tag* function using the *BERT base uncased pre-trained language model*[4] with 12 transformer layers, a hidden state size of 768 and 12 self-attention heads. As suggested by its developers [19], we trained 2 epochs using a batch size of 16 and a learning rate of 5e-5.

Reassigning noun-only attributes. After applying the *tag* function to the values of an attribute $D \in \mathcal{D}^T$, we check whether the tagging is likely to have been successful. In particular, we recognize that it is hard for an automated technique to distinguish among the `object`, $\texttt{actor}_{role}$, and $\texttt{passive}_{role}$ component types, when there is no contextual information, since their values all correspond to nouns. For instance, the `User` attribute of event e_2, which encompasses noun-based values like "*user*" and "*batch*", may be falsely tagged as `object` rather than $\texttt{actor}_{role}$. This happens because business objects are much more common

[4] https://github.com/google-research/bert

in the training data and the attribute values do not provide any further context that indicates the correct component type. To overcome this issue, we establish a set $\mathcal{D}_n^T \subseteq \mathcal{D}^T$ that contains all such *noun-only* attributes, i.e., attributes of which all values correspond solely to the `object` component type. This set is then forwarded to the attribute-level classification step of our approach, whereas the tagged values of the other attributes directly become part of our approach's output in the form of semantic annotations.

In this manner, we annotate the values of our example events' `concept:name` attributes. For e_1 we annotate: *create*: `action`, *offer*: `object`, for e_2: *transfer*: `action`, *execution sys.*: $\texttt{passive}_{role}$, and e_3: *declaration*: `object`, *final*: $\texttt{action}_{status}$, *approve*: `action`, *supervisor*: $\texttt{actor}_{role}$. The events' noun-only textual attributes, i.e., `org:resource` and `User`, are reassigned to be handled in the next step.

3.3.2. Attribute-level Classification

In this step, our approach determines the semantic component types of the miscellaneous attributes in $\mathcal{D}^M$, such as the boolean `Accepted` attribute of e_1, identified in the preprocessing stage, and the noun-only textual attributes in $\mathcal{D}_n^T$, such as `User` of e_2 and `org:resource` of e_3, stemming from the previous step. We target this task as a classification problem at the attribute level, i.e., we aim to identify a single semantic component type $t \in \mathcal{T} \cup \{\texttt{other}\}$ for each $D \in \mathcal{D}^M \cup \mathcal{D}_n^T$ and then assign type t to each occurrence of D in the event log. For attributes in $\mathcal{D}^M$, our approach operationalizes this classification task based on just an attribute's name, whereas it considers the name as well as the values of attributes in $\mathcal{D}_n^T$.

Note that we initially assign each attribute a component type $t \in \mathcal{T}'$, where $\mathcal{T}'$ excludes the *instance* component types, i.e. $\texttt{actor}_{instance}$ and $\texttt{passive}_{instance}$, from $\mathcal{T}$. Afterwards, our approach then distinguishes between type-level and instance-level resource attributes based on their domain.

Classifying miscellaneous attributes in $\mathcal{D}^M$. To determine the component type of miscellaneous attributes, we recognize that their values, typically alphanumeric identifiers, integers or boolean, are mostly uninformative and thus not helpful for the classification task. Therefore, we determine the component type of an attribute $D \in \mathcal{D}^M$ based on its name. To do this, we build a classifier that classifies a $\texttt{name}(D)$ based on a set of available attribute names, which we each manually assigned a class from the set $\mathcal{T}'$.

Attribute classifier. As training data for this classification task, we take the set of attribute names from the available real-world event logs used in our evaluation (see Section 4.1 for further details), complemented with attribute names from the *schema.org* vocabulary. This latter resource provides suggestions for standard attribute names that cover a broad range of object-related terms, e.g., *product*, as well as status and resource-related terms, e.g., *pending* or *agent*. Table 3 provides an overview of the training data obtained in this manner, which provides a good basis to train a classifier for our purpose.

Using this training set, we built a multi-class text classifier function $\texttt{classify}(D)$, which, given an attribute D, returns $t_D \in \mathcal{T}' \cup \{\texttt{other}\}$ as the semantic component type closest to $\texttt{name}(D)$, with $\texttt{conf}(r_D) \in$

Table 3: Characteristics of the data used to training our attribute classifier.

Source	`object`	$\texttt{object}_{status}$	$\texttt{action}_{status}$	$\texttt{actor}_{role}$	`other`
Real-life logs	6	6	1	8	38
Schema.org	68	22	4	91	70
Total	74	28	5	99	108

$[0, 1]$ as the respective confidence value. To operationalize the `classify` function, we encode the training data using the GloVe [22] vector representation for words. Subsequently, we train a *logistic regression* classifier on the obtained vectors, which can then be used to classify unseen attribute names. Since GloVe provides a state-of-the-art representation to detect semantic similarity between words, the classifier can recognize that, e.g., an `item` attribute is more similar to `object` attributes like "*product*", than to the names associated with other component types in the dataset.

Detecting status attributes. Although the `classify` function is able to recognize the majority of relevant attribute classes, we observe that it relatively often fails to recognize $\texttt{object}_{status}$ attributes, which may be falsely classified as either `object` or `other`. A primary reason for this is that examples of the $\texttt{object}_{status}$ class are underrepresented in the available training data, whereas the class also relates to a broad range of different kinds of statuses. However, we also observe that such $\texttt{object}_{status}$ attributes commonly follow a particular style. Specifically, status attributes often have a name that contains the past participle of a verb, e.g., "*selected*", "*is closed*", "*accepted*", accompanied by Boolean or categorical attribute values, e.g., indicating that the respective object is indeed closed.

Using this insight, we therefore re-assign the class of any attribute $D \in \mathcal{D}^M$ of which $\texttt{name}(D)$ ends with a past participle, thus overwriting the class t_D assigned by the classifier with $\texttt{object}_{status}$. The detection of such cases can be achieved by using standard NLP techniques, such as SpaCy in Python [18]. In this manner, our approach annotates the values of the boolean `Accepted` attribute of e_1 as $\texttt{object}_{status}$.

Classifying noun-only attributes in $\mathcal{D}_n^T$. Next, we turn to the classification of the noun-only attributes in $\mathcal{D}_n^T$, which were identified in the instance-level labeling step. Recall that these are textual attributes of which all values were entirely classified as being of the `object` component type, a situation that hints at a lack of context for their proper analysis (see, e.g., the `User` attribute of event e_2). To properly classify such attributes, we therefore first apply the same classifier as used for miscellaneous attributes. If $\texttt{classify}(D)$ provides a classification with a high confidence value, i.e., $\mathit{conf}(t_D) \geq \tau$ for a threshold τ, our approach uses t_D as the component type for an attribute $D \in \mathcal{D}_n^T$. In this way, we directly recognize cases where $\texttt{name}(D)$ is equal or highly similar to the attributes in our training data. However, if the classifier does not yield a confident result, we instead analyze the textual values in $\texttt{dom}(D)$.

Since noun-only attributes were previously re-assigned due to the lack of context available for their values, e.g., they just consist of words like "*user*" or "*vendor*", we overcome this issue by placing them in artificial contexts that cover various kinds of semantic components, allowing us to recognize the appropriate role of an attribute value. To provide these contexts, we use a selection of highly expressive textual attribute values from the training collection of the instance-level labeling step. Specifically, we use a set T of texts consisting of the 891 unique attribute values that contain at least three different kinds of semantic components.

To illustrate this, consider "*vendor*" as an attribute value, and t ="*confirm to customer that paperwork is ok*" as the context value from T, which contains information on action (*confirm*), a passive resource (*customer*), a business object (*paperwork*), and the object's status (*ok*). As shown in Figure 3, we create artificial texts for the attribute value by replacing a semantic component type from t with the word "*vendor*". We subsequently feed these artificial texts into the language model used for instance-level labeling (Section 3.3.1), which can then be used to quantify the confidence score that the attribute value corresponds to the semantic component type it replaced in an artificial text. For instance, in Figure 3, "*vendor*" is regarded as most likely corresponding to a passive resource for the context t, which we consider as a vote in favor of the $\texttt{passive}_{role}$ component type. Having computed these confidence scores for all attributes values in $\texttt{dom}(D)$ against all exemplary contexts in T, we assign $t_D \in \mathcal{T}' \cup \{\texttt{other}\}$ as the component type that received the most votes overall.

Context t:	confirm (`action`) to customer ($\texttt{passive}_{role}$) that paperwork (`object`) is ok ($\texttt{object}_{status}$)		
Replace `action`	*vendor* to customer that paperwork is ok	$\rightarrow$ *conf*(`action`)	= 0.26
Replace $\texttt{passive}_{role}$	confirm to *vendor* that paperwork is ok	$\rightarrow$ *conf*($\texttt{passive}_{role}$)	= **0.81**
Replace `object`	confirm to customer that *vendor* is ok	$\rightarrow$ *conf*(`object`)	= 0.68
Replace $\texttt{object}_{status}$	confirm to customer that paperwork is *vendor*	$\rightarrow$ *conf*($\texttt{object}_{status}$)	= 0.66

Figure 3: Insertion of value "*vendor*" into an existing context t, providing support for it being a passive resource.

Recognizing instance-level attributes. Since we only focused on the type-level components $\mathcal{T}'$ in the above, we lastly check for every attribute that was classified as being resource related, i.e., with $t_D \in \{\texttt{actor}_{role}, \texttt{passive}_{role}\}$, if it actually corresponds to an instance-level component type instead. Particularly, we change t_D to the corresponding instance-level component type if $\texttt{dom}(D)$ has values that contain a numeric part or only consist of named-entities (e.g., "*Pete*"). For instance, a `User` attribute (cf. event e_2) with values like *batch_ 00*, contains numeric parts and is, thus reassigned to $\texttt{actor}_{instance}$, while the attribute of event e_3, with $\texttt{dom}(\texttt{org:resource}) = \{$*staff member*, *system*$\}$, clearly does not describe individual resource instances and, therefore, will retain its $\texttt{actor}_{role}$ component type.[5]

[5]Note that this is an interesting case, given that `org:resource` attributes are normally used to indicate actor instances.

3.3.3. Component Identification Output

Having completed both the instance-level labeling and attribute-level classification steps, the component identification stage of our approach returns a collection of tuples (t, v) with $t \in \mathcal{T}$ a semantic component type and v a value, for each event $e \in L$. For values of a textual attribute $D_1 \in \mathcal{D}^T \setminus \mathcal{D}_n^T$), v corresponds to part of the attribute value $e.D_1$. For those attributes that were classified at the attribute level, i.e., $D_2 \in \mathcal{D}^M \cup \mathcal{D}_n^T$, a tuple receives the entire value. In case of boolean values, the attribute's name is used if the original value is *true*, if it is *false*, "*not*" is prepended to the attribute's name. For event e_1 of our running example, we obtain the tuples (`action`, *create*) and (`object`, *offer*) based on instance-level labeling and ($\texttt{action}_{status}$, *notAccepted*) and ($\texttt{actor}_{instance}$, *User_28*) based on attribute-level classification. For e_2, we obtain (`action`, *transfer*) and ($\texttt{passive}_{role}$, *execution syst.*) based on instance-level labeling and ($\texttt{actor}_{instance}$, *batch_00*) based on attribute-level classification. Finally, for e_3, we obtain (`object`, *declaration*), ($\texttt{action}_{status}$, *final*), (`action`, *approve*), and ($\texttt{actor}_{role}$, *supervisor*) based on instance-level labeling and ($\texttt{actor}_{role}$, *staff member*) based on attribute-level classification.

3.4. Semantic Component Categorization

In this section, we describe the two steps of the component categorization stage: action categorization (Section 3.4.1) and resource categorization (Section 3.4.2).

3.4.1. Action Categorization

In this step, our approach aims to classify identified `action` components, stemming from the previous stage, according to the eight high-level action categories described in Section 2.2. In this manner, we are able to recognize which events in a log relate to similar kinds of process steps, such as events that create, modify, or combine objects. For instance, we recognize that the *transfer* action of event e_2 modifies the handled item, whereas the *approve* action of event e_3 refers to a decision about a declaration.

We tackle this categorization task by establishing a set of *reference actions*, derived from the same MIT Process Handbook [16] that defines the eight high-level action categories. Then, given an action identified in the log, we use these reference actions to determine the most suitable high-level category.

Table 4 provides an overview of the collection of reference actions A_r established for this purpose, which corresponds to the actions found in the first four layers of the action hierarchy defined by the handbook. We note that lower parts of the hierarchy do not provide additional reference actions, but rather contain more specific versions. For example, given "*retire*" as as a reference action for *Destroy*, further layers of the hierarchy include "*retire physical object*" and "*retire digital object*" as specializations, whose inclusion would not help to categorize individual actions. It is important to remark here that some reference actions are part of multiple high-level categories, given that an action can have different impacts depending on its context. For example, the reference action "*document*" is part of both the *Create* and *Preserve* categories,

since documenting something can both indicate the creation of a new informational object, as well as the preservation of information, such as writing down a decision.

Category	**Exemplary reference actions**	**Count**
Create	*build*, *duplicate*, *design*, *produce*, *document*	15
Destroy	*retire*, *dispose*, *eliminate*, *obliterate*, *depreciate*	7
Modify	*improve*, *update*, *complete*, *move*, *send*	24
Preserve	*wait*, *retain*, *store*, *document*, *package*	10
Combine	*group*, *organize*, *match*, *aggregate*, *link*	10
Separate	*disaggregate*, *divide*, *segment*, *diversify*, *extract*	10
Decide	*select*, *determine*, *assign*, *assess*, *approve*	20
Manage	*assign*, *organize*, *allocate*, *schedule*, *budget*	11

Table 4: High-level action categories and exemplary reference actions.

To categorize an identified `action` component a, we first determine the most similar reference action $a_r \in A_r$. For this purpose, we again employ the *GloVe* vector representations for words [22]. Given these vectors, we compute the cosine similarity between all vector pairs (a, a'), with $a' \in A_r$, retrieving the reference action a_r with the highest similarity to a. We then categorize action a according to the high-level category that a_r is part of. For example, given $a =$ "*transfer*", we obtain $a_r =$ "*move*" as the reference action, so that a is accordingly recognized as belonging to the *Modify* category.

Note that in cases where there are multiple reference actions with the same, highest similarity score for a given action a (and those actions being part of different categories), or if the most similar reference action is part of multiple high-level categories, we break the tie by computing the similarity between a and the high-level actions themselves, assigning a to the category with the highest similarity score. For example, given the action "*write*", we obtain "*document*" as the reference action, which is part of both the *Create* and *Preserve* categories. Because "*write*" has a higher semantic similarity to the term "*create*" than to "*preserve*", we assign the action to the former category.

In this manner, we annotate the *create* action of the example event e_1 with the *Create* action type, the *transfer* action of e_2 with *Modify*, and the *approve* action of e_3 with *Decide*.

3.4.2. Resource Categorization

Finally, our approach turns to the categorization of the resources identified in an event log, determining whether they correspond to `human` or `system` actors. For instance, we categorize the $\texttt{actor}_{instance}$ component "*User_28*" of event e_1 as `human` and the $\texttt{actor}_{instance}$ component "*batch_00*" of event e_2 as `system`. Depending on the event log or specific resource, the information that may be available to perform this categorization can vary greatly, ranging from specific descriptions of actor roles, e.g., *supervisor* and *staff*

member in event e_3, to only having information on identifiers, e.g., *batch_ 00* or or *User_ 28* in the other two example events. Therefore, we propose several different strategies, exploiting different kinds of information.

Identifying individual actors. We perform this categorization task for each actor contained in a log, which means that we first determine the set of resources R that performed events, including all information available on them. To achieve this, we populate R with all distinct $\texttt{actor}_{instance}$ components, since these represent unique actors. We associate these with any $\texttt{actor}_{role}$ components found in an actor's events. For example, given an event with $\texttt{actor}_{instance}$ = "*0011*" and $\texttt{actor}_{role}$ = "*supervisor*", we add the resource tuple r_1 = ($\texttt{actor}_{role}$ = "*supervisor*", $\texttt{actor}_{instance}$ = "*0011*") to R. For events without $\texttt{actor}_{instance}$ but with $\texttt{actor}_{role}$ information, we consider unique combinations of $\texttt{actor}_{role}$ components as additional actors and add them to R. For instance, for event e_3, which has two $\texttt{actor}_{role}$ components but no instance information, we add r_2 = ($\texttt{actor}_{role}$ = "*supervisor*", $\texttt{actor}_{role}$ = "*staff member*") to R.

Categorization strategies. To categorize a resource, we apply up to four categorization strategies in a sequential manner: (1) WordNet-based categorization of $\texttt{actor}_{role}$ components, (2) named entity recognition of $\texttt{actor}_{instance}$ components, (3) BERT-based resource classification, and (4) execution time analysis. The underlying idea is that the earlier strategies are highly precise, but may not be applicable for every resource $r \in R$. Therefore, if a strategy yields a hit for r, we use that outcome to categorize it, while otherwise our approach moves to the next strategy in the list.

WordNet-based categorization of $\texttt{actor}_{role}$ *components.* This strategy aims to categorize a resource by comparing its $\texttt{actor}_{role}$ component, if available, to categories established in the lexical database *WordNet* [23]. This database relates words to each other via semantic relationships, such as hypernymy. A *hypernym* is a more general term for a given word, e.g., "*color*" is the hypernym of "*red*".

If the $\texttt{actor}_{role}$ value of a resource r is included in WordNet, we use the hypernymy relation to check if the value has a hypernym that corresponds to *person* or *organization*, in which case we categorize r as being `human`, or to *system*, *computer*, or *information*, in which case we categorize r as a `system`. For example, this strategy detects that a "*vendor*" is a *person* and, thus, a `human` actor, whereas a "*database*" corresponds to *information*, and is thus categorized as a `system`.

While this strategy is highly precise, its applicability is impeded by the relatively limited scope of WordNet, which only covers rather common English terms. As such, this strategy requires that a resource has a semantically meaningful $\texttt{actor}_{role}$ component and, furthermore, that this is not too domain-specific.

Named entity recognition of $\texttt{actor}_{instance}$ *components.* This strategy aims to detect `human` resources by determining if information contained in the $\texttt{actor}_{instance}$ component corresponds to a known name, e.g., *Pete* or *Wil*. For this purpose, we use a standard named entity recognizer (NER) [18] to check if the available instance information is found to be a *person*. In this manner, we are able to recognize a broad range of given and surnames of involved human actors, though this strategy can naturally only be applied to event

logs that have not been properly pseudonomymized.

BERT-based resource classification. If the two previous deterministic strategies do not yield a result, we next apply a probabilistic strategy that aims to classify a resource according to the textual information contained in its components (if any), as well as based on the activities that it performs. While the former speaks for itself, the idea of the latter is that the names of activities can indicate whether they are likely to be performed in an automated manner, i.e., by a system, or not. For instance, a resource performing a "*transmit data*" activity is more likely to be a system than a resource performing a "*conduct quality check*" activity, which would require the expertise of a human.

We operationalize this step by fine-tuning a BERT-based language model to the task of classifying textual fragments, consisting of actor descriptions and activity names, as either *system* or *human*. To train this model, we employ a dataset and gold standard established by Leopold et al. [21], which contains sets of activities that are classified as being performed in an automated or in a manual fashion. In this manner, the fine-tuned model will learn to distinguish texts that relate to automated activities performed by `system` actors and manual activities performed by `human` actors. Given a resource r, we consider each unique event label l of an event that r performs and apply the fine-tuned BERT model on the string $l+$ "*by*" $+r.\texttt{actor}_{role}$ (or just l if r has no $\texttt{actor}_{role}$). For instance, with $l =$ "*in transfer to execution sys*" and $res =$ "batch", we feed "*in transfer to execution sys by batch*" to the classifier. The classifier recognizes this activity as being performed by a system, resulting in a vote for the `system` category. Afterwards, we assign the category of r according to the most commonly predicted class across all unique labels that r has executed in the log.

Naturally, this strategy requires event labels with sufficient semantic information, i.e., activities that refer to an action and/or business object. Therefore, we only apply this strategy for labels that contain at least one of these components. Alternatively, in case $\texttt{actor}_{role}$ components are available for a resource, we can use these as standalone input to the classifier if no labels with sufficient semantic information are available. If neither informative labels nor actor descriptions are available, we turn to the last strategy.

Execution time analysis. Finally, for resources where none of the previous semantic strategies can be applied, we use a final categorization heuristic. Specifically, we consider the execution times of the events that are performed by a given resource r. If these times hint at instantaneous execution of process steps, i.e., the timestamp of an event and its predecessor is equal[6], we categorize r as a `system`, otherwise as a `human` actor.

For our running example, we categorize the actor of e_1, *User_28*, as `human` using BERT-based resource classification, which is based on the word *User* itself, in combination with the labels of the events that the actor performed. The actor of event e_2 is categorized as `system`, based on the same strategy, whereas the actor of e_3 is categorized as `human` based on WordNet, which relates both its $\texttt{actor}_{role}$ components, *supervisor* and *staff member*, via the hypernymy relation to the word *person*.

[6]While the timestamp is at least specific to the second.

3.5. Output

The component identification stage of our approach returns a collection of tuples (t, v) with $t \in \mathcal{T}$ a semantic component type and v a value, for each event $e \in L$, as described in Section 3.3.3. Afterwards, the component categorization stage adds a tuple (`action:type`, cat_a), with cat_a as a high-level action, for each identified `action` component, and a tuple (`actor:type`, cat_r), with cat_r as either `human` or `system`, if applicable. To enable the subsequent application of process mining techniques, the implementation of our approach returns an augmented XES event log that captures these tuples as dedicated, additional event attributes. We, thus, do not override any attributes from the original log.

Note that we support different ways to handle cases where an event has multiple tuples with the same semantic component type, such as the "*draft*" and "*send*" actions stemming from a "*draft and send request*" label or *staff member* and *supervisor*, both $\texttt{actor}_{role}$ components stemming from the same event e_3. Particularly, users can choose to collect the values into one attribute, i.e., `action` = [*draft, send*] and $\texttt{actor}_{role}$ = [*staff member, supervisor*], or into multiple, uniquely-labeled attributes, i.e., `action:0` = *draft*, `action:1` = *send* and `actor:role:0` = *supervisor*, `actor:role:1` = *staff member*. Information on the respective action types is then analogously captured in one or more event attributes. Lastly, if multiple $\texttt{object}_{status}$ (or $\texttt{action}_{status}$) attributes exist that each have Boolean values, e.g., `isCancelled` and `isClosed` for the Hospital log [11], these are consolidated into a single attribute, for which events are assigned a value based on their original Boolean attributes, e.g., $\{\perp, isCancelled, isClosed\}$.

For our running example, we obtain the annotated events as shown in Figure 4 as the final output of our approach, which make the semantic components available for process analysis techniques.

concept:name	O_Create Offer
time:timestamp	11-07-16 18:54
org:resource	User_28
Accepted	False
action:name	create
action:type	create
object:name	offer
object:status	notAccepted
actor:instance	User_28
actor:type	human

(a) Output for event e_1

concept:name	SRM: In transfer to Execution Syst.
time:timestamp	02-01-18 14:53
User	batch_00
action:name	transfer
action:type	modify
passive:role	Execution Syst.
actor:instance	batch_00
actor:type	system

(b) Output for event e_2

concept:name	Declaration final_approved by supervisor
time:timestamp	26-02-18 05:15
org:resource	staff member
action:name	approve
action:status	final
action:type	decide
object:name	declaration
actor:role:0	supervisor
actor:role:1	staff member
actor:type	human

(c) Output for event e_3

Figure 4: Final output of our approach for the running example's events

4. Evaluation

In this section we describe evaluation experiments we conducted to demonstrate the accuracy of our proposed event log annotation approach with respect to its ability to both identify and categorize semantic

components. Section 4.1 presents the collection of 14 real-life event logs we use as a basis for these experiments, Section 4.2 describes the employed implementation and experimental setup, whereas the results are presented and discussed in Section 4.3. To support reproducibility, the employed implementation, gold standard, and links to the event logs are all available through our project repository.[7]

4.1. Evaluation Data

To conduct our evaluation, we selected all real-life event logs publicly available in the common 4TU repository[8], excluding those capturing data on software interactions or sensor readings, given their lack of natural language content. For collections that included multiple event logs with highly similar attributes, i.e., BPI13, BPI14, BPI15 and BPI20, we only selected one log per collection, to maintain objectivity of the obtained results. Table 5 depicts the details on the resulting collection of 14 event logs. They cover processes of different domains, for instance financial services, public administration and healthcare. Moreover, they vary significantly in their number of event classes, textual attributes, and miscellaneous attributes.

Table 5: Event log characteristics, with $\mathcal{C}$ as the set of event classes, $\mathcal{D}$ of data attributes, and $\mathcal{D}^T$ of textual data attributes.

Log name	**Ref.**	$\|\mathcal{C}\|$	$\|\mathcal{D}\|$	$\|\mathcal{D}^{\mathbf{T}}\|$	**Log name**	**Ref.**	$\|\mathcal{C}\|$	$\|\mathcal{D}\|$	$\|\mathcal{D}^{\mathbf{T}}\|$
BPI12	[24]	24	4	2	BPI20	[7]	51	5	4
BPI13	[25]	4	11	4	CCC19	[26]	29	9	4
BPI14	[12]	39	7	2	Credit Req.	[27]	8	4	3
BPI15	[14]	289	13	3	Hospital	[11]	18	20	2
BPI17	[5]	26	15	4	RTFM	[15]	11	15	2
BPI18	[28]	41	12	5	Sepsis	[29]	16	31	1
BPI19	[6]	42	4	2	WABO	[13]	27	6	2

4.2. Setup

We conducted our evaluation experiments based on the following.

Implementation. We implemented our approach in the form of a Python prototype, which is publicly available through the aforementioned project repository and also includes a command that allows users to directly incorporate our approach in their Python projects through *pip* installation.

Our implementation uses the PM4Py [30] library to handle event logs, Pandas[9] for the data type preprocessing stage, the BERT base uncased pre-trained language model[10] as a foundation for the instance-level labeling step, and GloVe vector representations [22] to determine semantic similarity between words.

[7] https://gitlab.uni-mannheim.de/processanalytics/semantic-event-log-annotation
[8] https://data.4tu.nl/search?q=:keyword:%20%22real%20life%20event%20logs%22
[9] https://pandas.pydata.org
[10] https://github.com/google-research/bert

Gold standard. As a basis for our evaluation, we established a *gold standard* in which we manually annotated the contents of all 14 event logs used in the evaluation. For the component identification stage, we annotated all unique textual values, for instance-level labeling, and attributes, for attribute-level classification with their proper semantic component types. For the component categorization stage, we labeled identified resource components as `system` if the description of the data set clearly stated which resources are systems or if explicit information about the resource type was available from the event log itself. The `action` components that our approach identified in the 14 evaluation logs were labeled with their action type according to the MIT Process Handbook [16].

For reproducibility, the gold standard is published in the aforementioned project repository.

Cross-validation procedure. The semantic component identification stage of our approach uses a language model in the instance-level labeling step (Section 3.3.1) and a classifier in the attribute-level classification step (Section 3.3.2) that are both, among others, trained on data from the same real-life event logs used in the evaluation. Therefore, to avoid biasing the results, we perform our evaluation experiments using *leave-one-out* cross-validation, in which we repeatedly train our approach using data from 13 event logs and evaluate it on the 14th. This procedure is repeated such that each log in the collection is considered as the test log once. For the component categorization (Section 3.4), this procedure is not required, since the training data we use does not stem from the collection of evaluation logs.

Evaluation metrics. To assess the performance of our approach, we compare the annotations obtained using our approach against the established gold standard. Specifically, we report on the standard *precision*, *recall*, and the F_1*-score*. Note that for instance-level labeling, we evaluate correctness per chunk, e.g., if a chunk (*purchase order*, `object`) is included in the gold standard, both "*purchase*" and "*order*" need to be associated with the `object` component type in the result, otherwise, neither is considered correct.

Baselines. We aim to place the results obtained by our approach into context by comparing them to those obtained by relevant baselines. While there is no baseline against which we can compare our entire work, we compare the accuracy of our instance-leveling step against an existing activity label parser [20] and we compare the improved version of our attribute-level classification step against its old version [9]. The details of these comparison are described in the respective parts of the results discussion below.

4.3. Results

In this section, we consider the accuracy of our approach when it comes to semantic component identification (Section 4.3.1) and categorization (Section 4.3.2).

4.3.1. Component Identification Results

We assess the results of the component identification stage by first considering the individual instance-level labeling and attribute-level classification steps, followed by a discussion of the overall results.

Instance-level labeling results. Table 6 shows the results obtained when labeling the 625 unique textual attribute values included in the event logs, where the *Count* column reflects the respective number of times a component type occurred in the gold standard. The table shows that our instance-level labeling technique is able to identify semantic components in textual attributes with high accuracy, achieving an overall F_1-score of 0.91. The comparable precision and recall scores, e.g. 0.94 and 0.95 for `action` or 0.89 and 0.88 for `object`, each suggest that the approach can accurately identify components while avoiding false positives. This is particularly relevant, given that nearly half of the textual attribute values also contain information beyond the scope of the semantic component types considered here (as shown in Table 2, there are 291 textual parts marked as `other`). Due to this ability to recognize which parts of texts are actually relevant for the component identification task, our approach even performs well on complex values. For example, for the "*t13 adjust document x request unlicensed*" from the WABO log [31], our approach correctly recognizes the business objects (*document* and *request*), the action (*adjust*) and status (*unlicensed*), while omitting the superfluous content (*t13* and *x*) from consideration.

Table 6: Results of the instance-level labeling step for the 625 unique textual attribute values.

Component	**Count**	**Prec.**	**Rec.**	**F_1**
`object`	583	0.89	0.88	0.88
$\texttt{object}_{status}$	31	0.85	0.77	0.78
`action`	630	0.94	0.95	0.94
$\texttt{action}_{status}$	27	0.85	0.81	0.82
$\texttt{actor}_{role}$	69	0.93	0.84	0.88
$\texttt{passive}_{role}$	19	0.84	1.00	0.91
Overall	1,359	0.91	0.91	0.91

Challenges. We observe that the primary challenge for our approach relates to the differentiation between relatively similar semantic component types, namely between the two kinds of statuses, $\texttt{object}_{status}$ and $\texttt{action}_{status}$, as well as the two kinds of resources, $\texttt{actor}_{role}$ and $\texttt{passive}_{role}$. Making this distinction is particularly difficult in cases that lack sufficient contextual information or proper grammar. For example, an attribute value like "*denied*" can refer to either type of status, whereas it is even hard for a human to determine whether the "*create suspension competent authority*" label describes *competent authority* as a primary actor or a passive resource.

Baseline comparison. To put the performance of the instance-level labeling step into context, we compared it to a state-of-the-art technique for the parsing of process model activity labels, proposed by Leopold et al. [20]. For a fair comparison, we retrained our approach on the same training data as used to train the baseline (corresponding to the collection of process models in Table 2) and only assess the performance with

respect to the recognition of *business objects* and *actions*, since the baseline only targets these. Table 7 presents the results obtained in this manner for the event labels from all 14 considered event logs.

The table shows that our approach greatly outperforms the baseline, achieving an overall F_1-score of 0.75 versus the baseline's 0.47. Post-hoc analysis reveals that this improved performance primarily stems from event labels that are more complex (e.g., multiple actions, various semantic components or compound nouns spanning multiple words) or lack a proper grammatical structure. This is in line with expectations, given that the baseline approach has been developed to recognize several established labeling styles, whereas we observe that event data often does not follow such modeling guidelines in practice. Finally, it is worth observing that the performance of our approach in this scenario is considerably lower than when trained on the full data collection (e.g., an F_1 of 0.66 versus 0.88 for the `object` component type), which highlights the benefits of our data augmentation strategies as well as the benefits of also training on event labels besides process model activities.

Table 7: Comparison of our instance-level labeling technique against a state-of-the-art label parser. Both techniques are trained on process model activity labels and evaluated on the event labels in our data collection.

		Our approach			**Baseline** [20]		
Component	**Count**	**Prec.**	**Rec.**	**F_1**	**Prec.**	**Rec.**	**F_1**
`object`	562	0.65	0.68	0.66	0.40	0.40	0.40
`action`	618	0.86	0.81	0.83	0.59	0.48	0.53
Overall	1,180	0.76	0.75	0.75	0.50	0.44	0.47

Attribute-level classification results. After discarding 61 out of the total of 156 attributes in the preprocessing step and handling 24 attributes at the instance-level, a total of 71 attributes in $\mathcal{D}^M \cup \mathcal{D}_n^T$ reach the attribute-level classification step. 36 of these attributes relate to one of the semantic component types, whereas the remaining 35 are of the `other` category. As shown in Table 8, our approach achieves highly accurate results for this step, with an overall precision of F_1 score of 0.92 for the 36 attributes corresponding to semantic components and of 0.91 for the entire set. Notably, these results reveal that our approach is able to avoid false positives well, even though a substantial amount of event attributes are beyond the scope of our semantic component types, such as monetary amounts or timestamps. This achievement can largely be attributed to the domain analysis employed in our approach's first step.

We remark that the outstanding performance of our approach with respect to the $\texttt{action}_{status}$ and $\texttt{actor}_{instance}$ component types is in part due to the usage of standardized XES names for some of these attributes, enabling easy recognition. Yet this is not always the case: 5 out of 18 $\texttt{actor}_{instance}$ attributes use different names than the XES standard (`org:resource` or `org:group`), such as `User` or `Assignment_Group`. Our approach nevertheless maintains a high accuracy for these cases, correctly recognizing all such attributes.

Table 8: Results of the attribute-classification step for the non-textual attributes

		Our approach			Old approach [9]		
Component	**Count**	**Prec.**	**Rec.**	**F_1**	**Prec.**	**Rec.**	**F_1**
`object`	6	0.67	1.00	0.80	1.00	0.33	0.50
$\texttt{object}_{status}$	6	0.83	0.83	0.83	0.50	0.33	0.40
$\texttt{action}_{status}$	6	1.00	1.00	1.00	1.00	1.00	1.00
$\texttt{actor}_{instance}$	18	0.95	1.00	0.97	0.95	1.00	0.97
`other`	35	0.94	0.86	0.90	0.81	0.92	0.86
Overall (without `other`)	36	0.89	0.97	0.92	0.89	0.78	0.80
Overall (with `other`)	71	0.92	0.91	0.91	0.85	0.88	0.83

Overall, however, it is important to consider that these results were obtained for a relatively small set of 36 semantic attributes. Therefore, both the remarkable performance for most component types, as well as the comparably lower accuracy for `object` attributes should be considered with care.

Baseline comparison. In comparison to our original attribute-level classification technique [9], we improved this step through the incorporation of additional training data and by adding an additional heuristic technique to improve the detection of $\texttt{object}_{status}$ attributes (see Section 3.3.2). As shown in Table 8, these adaptations resulted in an improved classification accuracy, raising the F_1 score from 0.80 to 0.92 for the semantic attributes. The increase in F_1 for the $\texttt{object}_{status}$ attributes from 0.40 to 0.83 highlights the value of the heuristic technique, whereas also our approach's ability to detect `object` attributes improved, with an F_1 of 0.80 versus 0.50 before.

Overall component identification results. The overall performance of the component identification stage can be considered as the average over the instance-level labeling and attribute-level classification results, weighted against the number of entities that were annotated with this component, i.e., a unique textual attribute value (instance-level) or an entire attribute (attribute-level). These overall scores displayed in Table 9, which are naturally skewed heavily towards the performance of the instance-level labeling step, given that this step covered 1,359 out of the 1,395 entities.

We observe that the approach achieves highly accurate overall results, with a micro-average precision, recall, and F_1-score of 0.91. Still, when considering the results per semantic component type, we observe that there exist considerable differences. These differences are largely due to the lower accuracy achieved for the underrepresented component types in the data set, since it is clear that our approach is highly accurate on more common component types, such as the F_1 score of 0.94 for the recognition of actions.

Table 9: Overall results of the component identification stage

Component	Count	Prec.	Rec.	F_1
`object`	589	0.89	0.88	0.88
$\texttt{object}_{status}$	37	0.85	0.78	0.79
`action`	630	0.94	0.95	0.94
$\texttt{action}_{status}$	33	0.88	0.84	0.85
$\texttt{actor}_{role}$	69	0.93	0.84	0.88
$\texttt{actor}_{instance}$	18	0.95	1.00	0.97
$\texttt{passive}_{role}$	19	0.84	1.00	0.91
Overall	1,395	0.91	0.91	0.91

4.3.2. Component Categorization Results

This section provides the results of the component categorization experiments. First, the results of the action categorization are discussed, before focusing on the categorization of resources.

Action Categorization. Table 10 shows the results of the categorization of the 235 `action` components identified by our approach in the 14 evaluation logs per action type, consisting of 42 *create*, 4 *destroy*, 113 *modify*, 14 *preserve*, 6 *combine*, 5 *separate*, 49 *decide*, and 2 *manage* actions.

Overall, our approach rather accurately categorizes the identified actions into their respective types, achieving an F_1 score of 0.79. For the more common action types, our approach performs well, achieving an F_1 score of > 0.73 for *create*, *modify*, and *decide*. However, the results for the less common action types, i.e., *combine*, *destroy*, *manage*, *preserve*, and *separate* vary, with an F_1 ranging from 0.57 (*separate*) to 1.00 (*manage*).

Looking at specific cases, we find that our approach is able to categorize both rather common actions, e.g., such as "*generate*", "*accept*", and "*notify*", as well as actions performed in specialized processes, such as surgery [26]. For instance, the action "*anesthetize*" is correctly categorized as *modify*, whereas the action "widen" is correctly categorized as *separate*. Though, this does not hold in all cases. For instance, the action "*clean*" is categorized as *create* rather than *modify*, which can be attributed to the limited amount of available training data.

A main challenge in this categorization task is the ambiguity of some of the actions. For instance, the action "*suspension*" could be considered as either a *destroy*, *modify*, or a *decide* action. This makes it difficult, also for a human, to categorize such actions. Another challenge that naturally follows from this is that the framework does not classify actions into disjoint top-level categories. For instance, the action "*document*" is both categorized as *create* and *preserve*, which both makes sense given that an artifact can be considered both as preserved or created when it is documented. Similarly, the action "*allocate*" is both categorized

Table 10: Results of the action categorization step

Category	Count	Prec.	Rec.	F_1
Create	42	0.70	0.90	0.79
Destroy	4	0.75	0.75	0.75
Modify	113	0.85	0.84	0.85
Preserve	14	0.62	0.62	0.62
Combine	6	0.62	0.83	0.71
Separate	5	0.44	0.80	0.57
Decide	49	0.91	0.61	0.73
Manage	2	1.00	1.00	1.00
Overall	235	0.81	0.79	0.79

as *manage* and *decide*. While such framework-specific issues are problematic, even with this set-up, our approach can provide a helpful categorization of actions and, thus, process activities. We demonstrate this utility through an application scenario below (Section 5).

Resource Categorization. The 14 evaluation logs contain a total of 5,236 distinct resources. 5,204 of these are `human`, while only 32 are `system`, an imbalance that clearly reflects the fact that many different human actors can be involved in a process, while the number of different systems is typically limited.

Overall, we achieve an F_1 score of 0.999 for `human` (precision and recall also 0.999) and of 0.80 for `system` actors (with a precision of 0.86 and recall of 0.75). The performance of the individual strategies in terms of their number of *hits*, i.e., how often they were applicable, and their precision is depicted in Table 11.

Table 11: Performance of the resource categorization strategies, with the asterisks (*) indicating class-level hits.

Strategy	Category	Hits	Prec.
WordNet	`human`	6*	1.00
	`system`	2*	1.00
NER	`human`	585	1.00
	`system`	0	-
BERT	`human`	1,340	0.99
	`system`	30	0.85
Time	`human`	1,299	1.00
	`system`	0	-

As shown in the table, the WordNet-based strategy has perfect precision, but is only applied to a few

cases. However, it should be noted here that these eight hits correspond to entire resource classes (i.e., $\texttt{actor}_{role}$ components), rather than individual resources like the other strategies. For seven of these cases there are no instances contained in the log, such as the *Employee*, *Director*, and *Supervisor* roles in the BPI20 log. However, the *Vendor* role from the BPI19 log actually relates to 1,975 different resources, thus highlighting the overall relevance of this class-level strategy.

For the NER-based strategy, we again observe a perfect precision, though this strategy can only be applied to the BPI13 log. This event log uses first names, such as *Tomas*, *Carrie*, and *Niklas*, to refer to 585 specific resources.

The BERT-based strategy, which primarily focuses on activity label names, can be applied more broadly than the previous strategies. Yet, as shown by the precision of 0.99 for `human` resources and 0.85 for `system` ones, the accuracy of this strategy is still high. An in-depth analysis of these results reveals that a primary challenge here involves activities that can be executed by both human and system resources, such as seen for the BPI18 [28] log. In these situations, the analysis of activity labels does not always allow for the appropriate distinction between resource types, affecting the strategy's precision.

Finally, the results for the heuristic analysis of execution times are rather inconclusive, since this strategy was not applicable to `system` resources in the employed event log collection. However, the strategy also did not falsely categorize a `human` resource as a system, thus nevertheless achieving a precision of 1.00.

5. Application Cases

To highlight the benefits of our approach, we next look at four application cases involving real-life event logs. Through these application cases, we show how the semantic information identified by our approach can support (1) event class refinement, (2) object-centric process analysis, (3) semantics-aware event abstraction, and (4) the analysis of a process' automation degree.

5.1. Event Class Refinement

In this first application case, we show how semantic components identified in the instance-level labeling step of our approach can be used to establish more appropriate event classes for the *Permit log* from the BPI Challenge 2020 [7]. This log consists of 7,065 cases and 86,581 events, divided over 51 event classes (according to the event label, i.e., the `concept:name` attribute). This relatively large number of event classes is problematic when aiming to gain insights about the recorded business process. Particularly, any process model derived on its basis will automatically exceed the recommended maximum of 50 nodes in a process model [32] and quickly reach a spaghetti-like structure.

However, an assessment of the event labels and the semantic components identified in them, reveals that the labels in the log are polluted by superfluous information, resulting in an unnecessarily high number of

different event classes. Specifically, the majority of event labels mixes up information about the conducted activity, which should indeed be contained in a label, with information on the actor that performs the activity, which should rather be captured in a dedicated $\texttt{actor}_{role}$ attribute. Typical examples of this situation are labels such as "*declaration approved by budget owner*" and "*declaration approved by administration*".

Recognizing this situation, we can use the semantic components identified by our approach to establish refined event labels, which consist of only the information from the `action` and `object` roles, e.g., "*declaration approved*", while deferring the actor information to a $\texttt{actor}_{role}$ attribute. This operation yields an event log in which the number of event labels has been greatly reduced, from 51 to just 21. In this way, we have consolidated different pieces of semantic process information in dedicated places, i.e., activity information in the label and actor information in a separate attribute, whereas, when used as the new event class, the refined event labels allow for the discovery of smaller and hence more understandable process models. Finally, it is important to point out that this transformation does not lead to any loss of information, given that the old labels are not overwritten, whereas it is, as always, also possible to define event classes as combinations of the (refined) event labels plus the $\texttt{actor}_{role}$ attribute.

5.2. Object-centric Process Analysis

In this application case, we demonstrate how the semantic information identifed by our approach can be used to obtain an object-centric view on a process, which helps to provide clearer insights into processes that deal with various kinds of business objects. For this case, we again consider the Permit log [7].

After applying our approach, we observe that the log contains six different business objects (indicated as `object`): *permit*, *trip*, *request for payment*, *payment*, *reminder*, and *declaration*. Such information was not initially available in the log, given that these business objects were identified in the event labels themselves (see also the previous application case). Yet, after having identified them, we can investigate the execution of the process in both an inter-object and intra-object manner, which provides novel insights that the original log could not reveal.

To illustrate this potential, consider the directly-follows graph shown in Figure 5, which we obtained by selecting all events related to declarations, i.e., with `object` = '*declaration*', and using the identified actions to establish the event class. The figure clearly reveals how these objects are handled in the process. Mostly, declarations are *submitted*, *approved*, and then *final approved*. Interestingly, though, we also see 112 cases in which a declaration was definitely approved, yet rejected afterwards. Furthermore, we see 140 cases, in which a declaration was resubmitted after it was already definitely approved.

It is important to stress that such insights would not be possible from the original log, given that, in reality, the events related to declarations may be interspersed with events related to other business objects. Furthermore, by employing different filters and event classes, this object-centric view can be used as a basis for a wide range of other insights, e.g., to reveal how documents are handed over between different

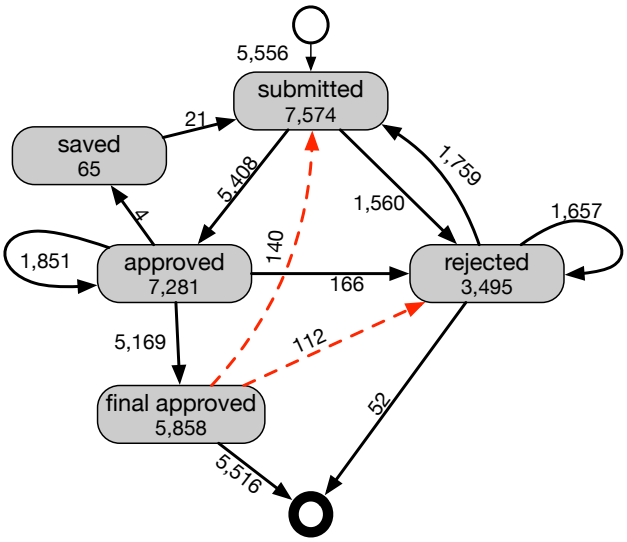


Figure 5: Example for object-centric analysis. The directly-follows graph shows the actions applied to the object *declaration* in the log (includes 100% activities, 50% paths).

employees or to reveal the inter-relations between business objects. Besides using the annotated object-centic information directly, it can also serve as a starting point to transform existing classical event logs into object-centric ones [33], which can in turn be used for object-centric process mining [34].

5.3. Semantics-aware Event Abstraction

In this third case, we demonstrate how the identified components and categorized actions can be used for meaningful event abstraction. Event abstraction is an established way of reducing complexity in event logs by grouping together related event classes into higher-level activities [35]. In recent work [36], we proposed the *GECCO* approach for this task, which is an abstraction approach that allows users to impose requirements on the characteristics of resulting higher-level activities and the corresponding log. The semantic components identified in the work at hand provide a highly suitable complement to GECCO, given that they allow us to establish requirements involving specific semantic information.

To demonstrate this potential, we apply our approach on the *Loan Application* log from the BPI Challenge 2017 [5]. Although this event log only contains 24 event labels, the complexity of this process when visualized in the form of, e.g., a directly-follows graph is considerable, resulting in a graph with 160 different edges. Aiming for meaningful abstraction, we observe that the process primarily relates to actions performed for two main types of business objects: *applications* and *offers*. Moreover, while various actions are applied to these objects, our categorization step recognizes that these primarily relate to *create*, *modify*, and *decide* actions. We use these insights to establish two abstraction requirements that we can impose when using GECCO: (1) grouped event classes should refer to the same business object (`object`) and (2) they should refer to the same type of action. We then name the obtained high-level activities as a combination of their

action type and business object, resulting in the abstracted directly-follows graph of Figure 6. The graph contains the 80% most frequent edges, while omitting event classes that do not refer to either of the main types of business objects.

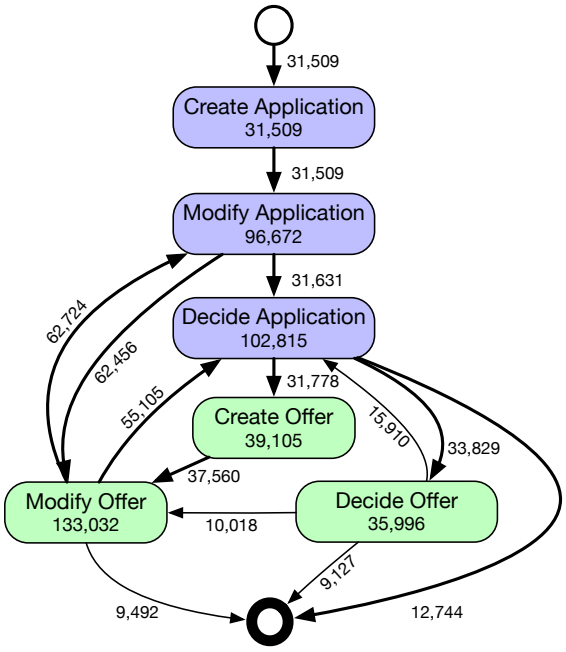


Figure 6: The directly-follows graph abstracted to show the actions types applied to the main business objects in the log, if any (includes 100% activities, 80% paths).

Having obtained this abstracted view on the process reveals clear dependencies between action types within and across the two business objects. For instance, we find that applications are first created and processed (modified). Moreover, a decision about an application, i.e., its acceptance, is necessary before an offer can be created. Further, there can be multiple iterations between offer and application. After modifying the offer, the application can be modified before the offer is modified again. Finally, the majority of cases end with a decision, either about an offer or about the application. Another common end of the process is a final modification of the offer. By looking into the original activities, this corresponds to sending a notification about an offer.

5.4. Analysis of Automation Degree

Finally, we use the resource categorizations provided by the final step of our approach to assess the automation degree and impact of system resources for the *Purchase Order* event log, part of the BPI Challenge 2019 [6]. The event log contains 251,734 cases, 1,595,923 event, and 628 unique resources.

By applying the resource categorization step of our approach, we discover that the events in the log originate from 608 `human` resources and 20 `system` resources, primarily captured in the log's `User` attribute.

We can leverage this categorization, stored in a `resource:type` event attribute, to analyze the automation degree of the Purchase Order process. At the event level, we find that about 79% of the events in the log are performed by employees, whereas the other 21% are performed in an automated manner, i.e., by a `system`. When considering full traces, we recognize that just 2,338 cases (0.9%) are entirely performed by systems, showing that only a fraction of cases can be handled in an automated manner. By contrast, we find that 162,505 (64.9%) do not involve any automatically executed steps. Both insights, thus, hint at clear opportunities for further automation, e.g., through the application of robotic process automation (RPA) technology.

6. Related Work

Our work relates to streams of research focused on the analysis of event and activity labels, semantic annotations of process information, and semantic role labeling in NLP.

Event and activity label analysis. Various approaches strive to either disambiguate or consolidate labels in event logs. Lu et al. [37] propose an approach to detect duplicate event labels, i.e., labels that are associated with events that occur in different contexts. By refining such duplicates, the quality of subsequently applied process discovery algorithms can be improved. By contrast, Sadeghianasl et al. [38, 39] aim to detect the opposite case, i.e., situations in which different labels are used to refer to behaviorally equivalent events. They achieve this through context-aware metrics [38] and crowdsource-based gamification [39]. Other approaches strive for the semantic analysis of labels, such as work by Deokar and Tao [40], which group together event classes with semantically similar labels, as well as the label parsing approach by Leopold et al. [20] against which we compared our work in the evaluation. Wynn et al. aim to improve event log quality by relabeling activities based on proposed quality metrics. To this end, they suggest standard NLP techniques, such as POS Tagging and Lemmatization, based on the computed quality of a log [41].

Semantic annotation of process information. Various works are complementary to ours in that they also strive to annotate event data or process models with different kinds of semantic information. For instance, work by Tsoury et al. [42] strives to augment logs with additional information derived from database records and transaction logs. Works by Leopold et al. focus on the categorization of process activities, achieved by mapping process model components to an existing process categorization [43] and by categorizing activities according to their degree of automation [21].

Role labeling in NLP. Beyond the scope of process analysis, our work also relates to semantic annotation applied in various other contexts. Most prominently, semantic role labeling is a widely recognized task in NLP [44, 45], which labels spans of words in sentences that correspond to semantic roles. The task's goal is to answer questions like *Who is doing what, where and to whom?* While early work in this area mostly applied feature engineering methods [46], recently deep learning-based techniques have been successfully

applied, e.g., [47, 48]. In the context of web mining, semantic annotation focuses on assigning semantic concepts to columns of web tables [49], while in the medical domain it is e.g. used to extract the symptoms and their status from clinical conversations [50].

7. Conclusion

In this paper, we proposed an approach for the automatic semantic annotation of event data. Namely, our approach identifies up to eight semantic component types per event, covering business objects, actions, actors, and other resources, without imposing any assumptions on the structure of an event log's attributes. It then further categorizes the identified action and resource components into pre-defined categories, enabling new analysis opportunities that consider the meaning of events.

We demonstrated our approach's efficacy through evaluation experiments using a wide range of real-life event logs. The results show that our approach accurately identifies the targeted semantic components from textual attributes, whereas our attribute classification techniques were also shown to yield good results when dealing with the information contained in non-textual attributes. In both cases, we showed that our techniques outperform existing state-of-the-art work. Furthermore, our approach performs well in assigning identified semantic components to predefined categories. Finally, we highlighted the potential of our work by illustrating some of its benefits in four application cases based on real-life data. Particularly, we showed how our approach can be used to refine and consolidate event classes in the presence of polluted labels, as well as to obtain object-centric insights about a process. Moreover, we showed that by categorizing action components, we can abstract an event log to analyze higher-level dependencies in the process, while a detailed analysis of the automation degree of a process is enabled by categorizing resource components.

In future work, our approach itself could be extended by incorporating additional semantic component types or component categories, given relevant use cases and data. Furthermore, the accuracy of the approach may be improved by incorporating additional semantic technologies in its annotation techniques, such as using large-scale knowledge graphs for better categorization. Nevertheless, we see the biggest future potential in terms of the semantics-aware process mining techniques and use cases that our work enables, of which the demonstrated use for meaningful event abstraction and our proposed semantic anomaly detection approach [4] are just a starting point.

Reproducibility: The implementation, dataset, and gold standards employed in our work are all available through the repository linked in Section 4.

References

[1] W. M. P. van der Aalst, Process Mining: Data Science in Action, Springer, 2016.

[2] W. M. P. van der Aalst, H. A. Reijers, M. Song, Discovering social networks from event logs, Computer Supported Cooperative Work (CSCW) 14 (6) (2005) 549–593.

[3] W. M. P. van der Aalst, Object-Centric Process Mining: Dealing with Divergence and Convergence in Event Data, in: SEFM, Springer, 2019, pp. 3–25.

[4] H. van der Aa, A. Rebmann, H. Leopold, Natural language-based detection of semantic execution anomalies in event logs, Information Systems 102 (2021) 101824.

[5] B. van Dongen, BPI Challenge 2017 (2017). doi:10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b.

[6] B. F. van Dongen, BPI Challenge 2019 (2019). doi:10.4121/uuid:d06aff4b-79f0-45e6-8ec8-e19730c248f1.

[7] B. F. van Dongen, BPI challenge 2020 (2020). doi:10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51.

[8] G. Acampora, A. Vitiello, B. Di Stefano, W. M. P. van der Aalst, C. Günther, E. Verbeek, IEEE 1849tm: The XES standard, IEEE Computational Intelligence Magazine (2017) 4–8.

[9] A. Rebmann, H. van der Aa, Extracting semantic process information from the natural language in event logs, in: International Conference on Advanced Information Systems Engineering, Springer, 2021, pp. 57–74.

[10] J. Mendling, H. A. Reijers, J. Recker, Activity labeling in process modeling: Empirical insights and recommendations, Inf. Syst. 35 (4) (2010) 467–482.

[11] F. Mannhardt, Hospital billing - event log (2017). doi:10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741.

[12] B. F. van Dongen, BPI challenge 2014 (2014). doi:10.4121/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35.

[13] J. Buijs, Receipt phase of an environmental permit application process ('WABO') (2014). doi:10.4121/uuid:a07386a5-7be3-4367-9535-70bc9e77dbe6.

[14] B. F. van Dongen, BPI challenge 2015 (2015). doi:10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1.

[15] M. M. De Leoni, F. F. Mannhardt, Road traffic fine management process (2015). doi:10.4121/UUID:270FD440-1057-4FB9-89A9-B699B47990F5.

[16] T. W. Malone, K. Crowston, G. A. Herman, Organizing business knowledge: The MIT process handbook, MIT press, 2003.

[17] B. Levin, English verb classes and alternations: A preliminary investigation, University of Chicago press, 1993.

[18] M. Honnibal, I. Montani, spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing, To appear 7 (1) (2017).

[19] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: NAACL, ACL, 2019, pp. 4171–4186.

[20] H. Leopold, H. van der Aa, J. Offenberg, H. A. Reijers, Using Hidden Markov Models for the accurate linguistic analysis of process model activity labels, Inf. Syst. 83 (2019) 30–39.

[21] H. Leopold, H. van der Aa, H. A. Reijers, Identifying candidate tasks for robotic process automation in textual process descriptions, in: BPMDS, Springer, 2018, pp. 67–81.

[22] J. Pennington, R. Socher, C. D. Manning, GloVe: Global vectors for word representation, in: EMNLP, 2014, pp. 1532–1543.

[23] G. A. Miller, WordNet: a lexical database for english, Communications of the ACM 38 (11) (1995) 39–41.

[24] B. F. van Dongen, BPI Challenge 2012 (2012). doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.

[25] W. Steeman, BPI challenge 2013 (2014). doi:10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07.

[26] J. Munoz-Gama, R. R. de la Fuente, M. M. Sepúlveda, R. R. Fuentes, Conformance checking challenge 2019 (ccc19) (Feb 2019). doi:10.4121/uuid:c923af09-ce93-44c3-ace0-c5508cf103ad.

[27] A. Djedović, Credit Requirement Event Logs (2017). doi:10.4121/uuid:453e8ad1-4df0-4511-a916-93f46a37a1b5.

[28] B. F. van Dongen, F. Borchert, BPI Challenge 2018 (2018). doi:10.4121/uuid:3301445f-95e8-4ff0-98a4-901f1f204972.

[29] F. Mannhardt, Sepsis Cases - Event Log (2016). doi:10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460.

[30] A. Berti, S. J. van Zelst, W. van der Aalst, Process mining for python (PM4Py): Bridging the gap between process-and

data science, in: ICPM Demo Track 2019, 2019, pp. 13–16.

[31] J. Buijs, Environmental permit application process ('wabo'), coselog project (2014). doi:10.4121/uuid:26aba40d-8b2d-435b-b5af-6d4bfbd7a270.

[32] J. Mendling, H. A. Reijers, W. M. P. van der Aalst, Seven process modeling guidelines (7PMG), Information and Software Technology 52 (2) (2010) 127–136.

[33] A. Rebmann, J.-R. Rehse, H. van der Aa, Uncovering object-centric data in classical event logs for the automated transformation from XES to OCEL, in: Business Process Management - 20th International Conference, BPM 2022, Muenster, Germany, September 11-16, 2021, Proceedings, 2022 to appear.

[34] W. van der Aalst, Object-centric process mining: Dealing with divergence and convergence in event data, in: Software Engineering and Formal Methods, Springer, 2019, pp. 3–25.

[35] S. J. van Zelst, F. Mannhardt, M. de Leoni, A. Koschmider, Event abstraction in process mining: literature review and taxonomy, Granular Computing 6 (3) (2021) 719–736.

[36] A. Rebmann, M. Weidlich, H. van der Aa, GECCO: constraint-driven abstraction of low-level event logs, in: 38th IEEE International Conference on Data Engineering, IEEE, 2022, to appear.

[37] X. Lu, D. Fahland, F. J. van den Biggelaar, W. M. van der Aalst, Handling duplicated tasks in process discovery by refining event labels, in: BPM, 2016, pp. 329–347.

[38] S. Sadeghianasl, A. ter Hofstede, M. Wynn, S. Suriadi, A contextual approach to detecting synonymous and polluted activity labels in process event logs, in: OTM, Springer, 2019, pp. 76–94.

[39] S. Sadeghianasl, A. ter Hofstede, S. Suriadi, S. Turkay, Collaborative and Interactive Detection and Repair of Activity Labels in Process Event Logs, in: ICPM, 2020, pp. 41–48.

[40] A. V. Deokar, J. Tao, Semantics-based event log aggregation for process mining and analytics, Information Systems Frontiers 17 (6) (2015) 1209–1226.

[41] B. Ramos-Gutiérrez, Á. J. Varela-Vaca, F. J. Ortega, M. T. Gómez-López, M. T. Wynn, A NLP-oriented methodology to enhance event log quality, in: Enterprise, Business-Process and Information Systems Modeling, Springer International Publishing, Cham, 2021, pp. 19–35.

[42] A. Tsoury, P. Soffer, I. Reinhartz-Berger, A conceptual framework for supporting deep exploration of business process behavior, in: ER, Springer, 2018, pp. 58–71.

[43] H. Leopold, F. Pittke, J. Mendling, Automatic service derivation from business process model repositories via semantic technology, Journal of Systems and Software 108 (2015) 134–147.

[44] X. Carreras, L. Màrquez, Introduction to the CoNLL-2005 shared task: Semantic role labeling, in: CoNLL 2005, 2005, pp. 152–164.

[45] D. Gildea, D. Jurafsky, Automatic labeling of semantic roles, Computational linguistics 28 (3) (2002) 245–288.

[46] S. Pradhan, W. Ward, K. Hacioglu, J. H. Martin, D. Jurafsky, Semantic role labeling using different syntactic views, in: ACL, 2005, pp. 581–588.

[47] L. He, K. Lee, M. Lewis, L. Zettlemoyer, Deep semantic role labeling: What works and what's next, in: ACL, 2017, pp. 473–483.

[48] Z. Zhang, Y. Wu, H. Zhao, Z. Li, S. Zhang, X. Zhou, X. Zhou, Semantics-aware BERT for language understanding, in: AAAI, Vol. 34, No. 05, 2020, pp. 9628–9635.

[49] Z. Zhang, Effective and efficient semantic table interpretation using tableminer+, Semantic Web 8 (6) (2017) 921–957.

[50] N. Du, K. Chen, A. Kannan, L. Tran, Y. Chen, I. Shafran, Extracting symptoms and their status from clinical conversations, in: ACL, 2019, pp. 915–925.