Multi-perspective Identification of Event Groups for Event Abstraction

Adrian Rebmann[1], Peter Pfeiffer[2,3], Peter Fettke[2,3], and Han van der Aa[1]

[1] Data and Web Science Group, University of Mannheim, Mannheim, Germany
{rebmann|han.van.der.aa}@uni-mannheim.de
[2] German Research Center for Artificial Intelligence, Saarbrücken, Germany
peter.{pfeiffer|fettke}@dfki.de
[3] Saarland University, Saarbrücken, Germany

Abstract. In process mining settings, events are often recorded on a low level and cannot be used for meaningful analysis directly. Moreover, the resulting variability in the recorded event sequences leads to complex process models that provide limited insights. To overcome these issues, event abstraction techniques pre-process the event sequences by grouping the recorded low-level events into higher-level activities. However, existing abstraction techniques require elaborate input about high-level activities upfront to achieve acceptable abstraction results. This input is often not available or needs to be constructed, which requires considerable manual effort and domain knowledge. We overcome this by proposing an approach that suggests groups of low-level events for event abstraction. It does not require the user to provide elaborate input upfront, but still allows them to inspect and select groups of events that are related based on their common multi-perspective contexts. To achieve this, our approach learns representations of events that capture their context and automatically identifies and suggests interesting groups of related events. The user can inspect group descriptions and select meaningful groups to abstract the low-level event log.

Keywords: Process mining · Event abstraction · Multi-perspective analysis.

1 Introduction

Process mining comprises methods to analyze event data that is recorded during the execution of organizational processes. Specifically, by automatically discovering process models from event logs, process discovery yields insights into how a process is truly executed [1]. Events recorded by information systems are often too fine-granular for meaningful analysis, though, and the resulting variability in the recorded event sequences leads to overly complex models. To overcome this issue, *event abstraction* techniques aim to lift the low-level events recorded in a log to a more abstract representation, by grouping them into high-level activities [17].

Existing techniques for event abstraction (cf., [4, 17]) are either *unsupervised* or *supervised*. Unsupervised techniques do not require any input about targeted high-level activities. Instead, they rely on control-flow similarities between low-level event types. Yet, they do not consider any other dependencies between events, such as the amount of time between their execution. Since the user of such techniques has no control over the abstraction result, there is no guarantee that they yield meaningful high-level activities, making it hard to ensure that an abstraction is appropriate for a specific analysis goal. For instance, if the goal is to understand interactions between employees in a process, grouping events based on control-flow similarity might lead to high-level activities that encompass different employees. This makes it difficult—if not impossible—to analyze interactions in the process. Supervised event abstraction techniques aim to overcome such issues by requiring input about high-level activities upfront, e.g., high-level process models [2] or predefined event patterns [10]. In this manner, such techniques give the user control over high-level activities. However, in practice the required information is often not available beforehand. For instance, when applying event abstraction as a preprocessing step to process discovery, high-level process models are typically not available [17]. Even if knowledge on the desired high-level activities is available, it may require a lot of manual effort to translate it into the necessary input, e.g., by defining how these high-level activities manifest themselves in low-level event patterns [10].

These two extremes, between not giving the user any control over high-level activities and requiring too much input, call for a common middle ground, i.e., a convenient means to support users in their abstraction tasks. In particular, users should be enabled to control the characteristics of high-level activities, while reducing the upfront knowledge they need about the data. This is particularly challenging in situations where the events' labels do not reveal the purpose of the high-level activities they relate to. An *Update record* event, for instance, could relate to any activity that modifies a business object. In such situations it is inevitable to look at the context of events and identify high-level activities in a more indirect manner.

To enable this, we propose an approach that allows the user to inspect groups of events based on their common context, thus, guiding them towards identifying meaningful high-level activities that can be used for abstraction without requiring upfront input about these activities. Our approach learns representations that capture complex contextual dependencies between low-level events, e.g., that events are executed within a short period of time and are performed by the same resource. Based on these representations, it automatically identifies and suggests groups of events. The user can select meaningful groups that can in turn be used to abstract the low-level log.

We motivate the need for the multi-perspective identification of event groups for abstraction in Section 2, before introducing preliminaries in Section 3. We present our approach in Section 4. Then, Section 5 describes a proof of concept demonstrating the potential of our approach. Section 6 summarizes related work; Finally, Section 7 discusses limitations of our work, gives an outlook on next steps, and concludes.

2 Problem Illustration

Our work deals with situations in which there are complex *n:m* relations between low-level event classes and high-level activities, which means that events with the same label can relate to different activities, which themselves can relate to any number of events. Such low-level recording is a common issue, e.g., when dealing with UI logs, logs from messaging and document management systems, and logs of sensor data. In

Table 1: A single case of a request handling process recorded on a low level.

CaseID	EventID	Class	Timestamp	Role	Column
C1	e1	Receive email	05-23 07:45	Assistant	
C1	e2	Create record	05-23 09:07	Assistant	
C1	e3	Open document	05-23 10:40	Assistant	
C1	e4	Close document	05-23 10:51	Assistant	
C1	e5	Update record	05-23 10:52	Assistant	isComplete
C1	e6	Open document	05-25 15:03	Manager	
C1	e7	Update record	05-25 15:20	Manager	isAccepted
C1	e8	Close document	05-25 15:23	Manager	
C1	e9	Send email	05-26 10:03	Assistant	

such settings, individual events are often not informative and cause a high degree of variability in event logs resulting in the discovery of spaghetti models [17].

For illustration purposes, consider a request-handling process, which is supported by an information system logging events on a low level, i.e., on the level of database and document operations, such as *Open document*, *Update record*, and *Send email*. A single case of the low-level event log of this process is depicted in Table 1. On the activity level, blue events (*e1*–*e2*) record that a new request has been received, purple events (*e3*–*e5*) refer to checking required documents for completeness, whereas brown events (*e6*–*e8*) refer to a decision about a request. Finally, the gray event (*e9*) represents the notification about the outcome of the request.

Looking at the sequence of events in case *C1*, however, does not reveal these activities, because their purpose is not explicitly indicated in the available data. For instance, from an *Open document* event like *e3*, it is unclear if it refers to a check for completeness or a decision. Therefore, we have to discover meaningful activities in a more indirect manner, i.e., by looking for events that occur in a commonly recurring context. This may include the temporal context, e.g., that events occur within a short period of time, the organizational context, e.g., that events are executed by the same resource, and the data context associated with individual events. For instance, the purple events (*e3*–*e5*) happen within a short period of time (12 minutes), are executed by an assistant, while *e5* changes the value of the *isComplete* column. In contrast, the brown events (*e6*–*e8*) happen within 20 minutes, are executed by a manager, while *e7* changes the value of the *isAccepted* column. The events within these two groups share a common context from both the time and resource perspectives, whereas the different columns they update indicate a clear difference between the groups in *C1*, i.e., they hint at the purpose of an underlying business activity.

Therefore, our goal is to group events that have similar contexts, in order to make the purpose of activities that the low-level events represent more apparent. However, commonly recurring contexts of events, like the ones illustrated above, often cannot be identified from individual cases, because these represent single process instances in which contexts may not recur. Therefore, we have to consider the entire event log for this task, i.e., all events, across cases. The identification of these recurring contexts is highly complex from the low-level event log, though, which may have dozens of event classes and attributes and thousands of cases. Hence, this requires an automated identification of groups of events, yet, we also want to make sure that identified groups are actually meaningful for a user and their specific analysis purpose.

We tackle this through two main parts:

Multi-perspective event groups. We identify groups of events based on their multi-perspective context. In particular, we assign similar events that share commonly recurring contexts across process perspectives to the same group and distinct events that do not share such contexts to different groups. An example is shown in Fig. 1, where, e.g., an *Open document* is grouped with an *Update record* event, as both are executed by an assistant and happen within 20 minutes. In contrast, the *Update record* event executed by a manager changing the value of the *isAccepted* column is assigned to a different group. While these events belong to the same case *C1*, it is important to stress that we aim for groups of events that span individual cases. If, for instance, a hypothetical *Update record* event of a case *C2* is also executed by an assistant and changes the *isComplete* column, we aim to assign it to the same group as *e3* and *e5*, because they share contexts across cases.

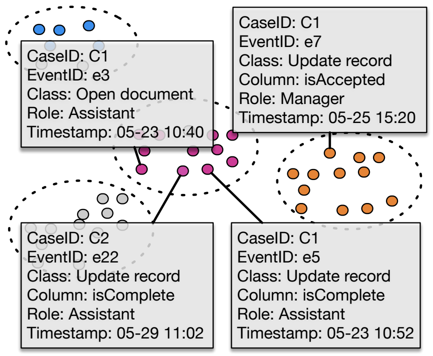


Fig. 1: Multi-perspective event groups.

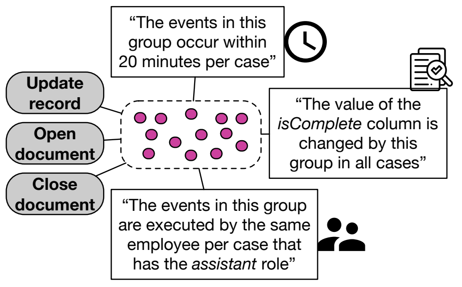

"The events in this group occur within 20 minutes per case"
Update record
Open document
Close document
"The value of the isComplete column is changed by this group in all cases"
"The events in this group are executed by the same employee per case that has the assistant role"

Fig. 2: Suggestion of a group of events.

Effective group suggestions. To ensure that identified groups are indeed meaningful, we support the user with understandable suggestions, allowing them to assess and select groups of related events based on their context. For instance, in our running example we identified that a group of events is executed within a short period of time by the same role, which changes the status of the request as shown in Fig. 2. Given that the events in this group occur in a similar context and there is a clear property that differentiates this from other groups, i.e., the change of the *isComplete* value, we aim to suggest it to the user. They might associate this group with a check for completeness in the request-handling process, select it for abstraction, and later assign it a suitable label.

3 Preliminaries

Events. We consider events recorded during the execution of a process and write $\mathcal{E}$ for the universe of all events. Events have unique identifiers and carry a payload containing their `Class` and optional contextual information, such as a timestamp, resource information, or relevant data values. We capture this payload by a set of attributes $\mathcal{D} = \{D_1, \ldots, D_p\}$, with $dom(D_i)$ as the domain of attribute D_i, $1 \leq i \leq p$. We write $e.D$ for the value of attribute D of an event e. For instance, for event $e1$ in Table 1, we write $e1.\texttt{Class} = \textit{Receive email}$ and $e1.\texttt{Role} = \textit{Assistant}$.

Event log. An event log is a set of *traces* L, with each trace a sequence of events $\sigma \in \mathcal{E}^*$, representing a single execution of a process, i.e., a case. An event belongs to exactly one trace. We write E_L for the set of all events of the traces in L.
Event groups. An event group is a set of events $g \subseteq E_L$. A grouping of events $G = \{g_1, ..., g_k\}$ is a set of event groups, such that G's members are disjoint and cover all events in E_L, i.e., $\bigcup_i^k g_i = E_L \wedge \bigcap_i^k g_i = \emptyset$.

4 Approach

As visualized in Fig. 3, our approach takes as input an event log and consists of four steps to create event group suggestions for event abstraction. Step 1 learns contextual dependencies between events and establishes multi-perspective representations. Step 2 groups the events based on these representations, which yields event groups as visualized in Fig. 1. Step 3 then computes key properties per group, which Step 4 uses to create suggestions by selecting groups with interesting properties and generating descriptions of the common contexts in which a groups' events occur. The output is a set of group suggestions and textual descriptions per group, such as shown in Fig. 2. The user can inspect these suggestions and select meaningful groups that serve their analysis purpose. The selected groups can then be used to abstract the low-level event log.

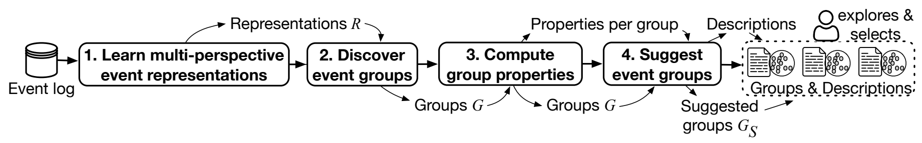
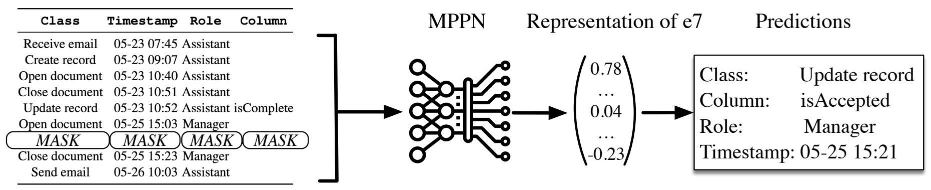


Fig. 3: The main steps of our approach.

4.1 Step 1: Learn multi-perspective event representations

In the first step, we establish event representations that capture the multi-perspective context of low-level events, i.e., we aim to derive a representation r for each low-level event e, which contains contextual information available in e's attributes as well as its context in terms of surrounding events in its trace. As illustrated in Section 2, it is essential to consider this multi-perspective context of events to obtain meaningful event groups. The challenge here lies in generating representations that contain the relevant context required to create such groups. To this end, we leverage the ability of the Multi-Perspective Process Network (MPPN) [11]. The approach processes traces with various perspectives of different types, i.e., categorical, numerical, and temporal event attributes, as well as the trace-based context.

For a trace σ, MPPN transforms the sequence of each available attribute's values into distinct 2D "images". Each image is processed by a pre-trained convolutional neural network (CNN) and results in one feature vector per attribute. Then, in order to obtain multi-perspective representations, the individual per-attribute vectors are pooled and processed by a fully-connected neural network resulting in one representation of adjustable size per trace, which contains features of all perspectives. Through the transformation of sequences of attribute values into images and the use of CNNs, the approach



Class	Timestamp	Role	Column
Receive email	05-23 07:45	Assistant	
Create record	05-23 09:07	Assistant	
Open document	05-23 10:40	Assistant	
Close document	05-23 10:51	Assistant	
Update record	05-23 10:52	Assistant	isComplete
Open document	05-25 15:03	Manager	
MASK	*MASK*	*MASK*	*MASK*
Close document	05-25 15:23	Manager	
Send email	05-26 10:03	Assistant	

Fig. 4: Masked event prediction to learn multi-perspective representations.

can focus on detecting similar patterns across traces in L, rather than focusing on the specific order in which events occur in individual traces. This flexibility in terms of how traces are processed makes MPPN a good choice for the task at hand, since, especially in event abstraction settings, we need to account for the considerable degree of variability present in low-level event sequences. Moreover, the learned representations include all process perspectives and thus, can be used for multi-perspective clustering tasks.

Originally, MPPN was developed to learn representations per trace $\sigma \in L$. Therefore, we have to adapt its learning strategy during training to be able to obtain one representation r per event $e \in E_L$, which captures e's multi-perspective context. To this end, we randomly mask all attribute values of events and train MPPN to predict these masked values given all other information in σ. For instance, as shown in Fig. 4, we replace the values of *e7*.`Class`, *e7*.`Timestamp`, *e7*.`Role`, and *e7*.`Column` with *MASK*. The task of MPPN is to predict all masked attribute values of *e7* using the information from the trace's other events. If MPPN is able to accurately predict the attribute values of the masked events, this indicates that the learned representations capture their events' contexts well. Since MPPN has access to all events and their attributes before and after *e7*, rich contextual information can be incorporated into r.

After being trained in this manner on the whole event log, we obtain a set R of representations: for each event e, we mask all attribute values of e, process σ with MPPN, and add the generated representation r to R.

4.2 Step 2: Discover event groups

Step 2 discovers groups of events with commonly recurring multi-perspective contexts, which may represent high-level activities. To establish a set G of event groups, we cluster events with similar learned representations since they are likely to share a similar context, for instance, because they are executed by the same resources and occur within a short period of time. For performance reasons, we reduce the complexity of the representations using *Principal Component Analysis* (PCA). Then, we apply the well-known *k-means* algorithm to obtain clusters. Instead of setting a specific number of clusters *k*, we use the *elbow method* [15] to select an appropriate k from a range of values (from 2 up to twice the number of event classes).

This clustering yields a grouping G as illustrated in Fig. 1. By assigning labels to each group $g \in G$, we could build a mapping between low-level events and high-level activities at this point already, which can be used to abstract a low-level event log. However, the remaining steps further process the groups to suggest only interesting ones to the user to make sure that they can assess how meaningful groups are for abstraction.

Table 2: Exemplary group properties used by our approach.

Level	**Perspective**	**Property**	**Example description based on template sentences**
Group	Control-flow	# of event classes	*This group has "Open document" and "Close document" events.*
	Resource	# of resources	*All events of this group are executed by 20 different resources.*
	Resource	# of roles	*All events of this group are executed by the "Assistant" role.*
	Time	Day of occurrence	*90% of events in this group happened on a Wednesday.*
	Time	Time of occurrence	*All events in this group happened before noon.*
	Data (cat.)	Distinct values	*All events have the value "Loan takeover" for the* `Goal` *attribute.*
	Data (num.)	Value range	*The* `Cost` *attribute ranges between 1,000 and 1,500 in this group.*
Case	Control-flow	# of event classes	*For this group, there are on average 3 events per case.*
	Control-flow	Range of positions	*The events of this group occur in a range of 2 to 3 events.*
	Resource	# of resources	*All events in this group are executed by the same resource per case.*
	Resource	# of roles	*All events in this group are executed by the "Manager" role.*
	Time	Duration	*This group of events takes 45 minutes on average per case.*
	Data (cat.)	Distinct values	*The value of the* `isAccepted` *attribute changes once on average.*
	Data (num.)	Value range	*Cost* attribute has a range of 50 on average for this group per case.

4.3 Step 3: Compute group properties

Next, based on the available event attributes, we compute a set of properties for each group $g \in G$, which jointly describe the multi-perspective common context of the events in g. These are later used to (1) assess how interesting a group is and (2) create textual descriptions of the group as exemplified in Fig. 2. An overview of considered properties is provided in Table 2. These do not necessarily consider all aspects of a particular input event log, yet, our approach can be easily extended with additional ones. As the table shows, properties either refer to all events in g or to the events in g per case. Moreover, each property refers to one attribute and, as such, to one main process perspective, i.e., the control-flow, resource, time, or data perspective. For instance, a group-based, resource-related property would be *the number of distinct roles that execute events within a group*, whereas a case-based one would be *the average number of distinct roles in a group per case*.

We compute group-level properties by aggregating the attribute values of events in a group, i.e., we collect distinct categorical and sum, average, and compute the range of numerical attribute values. For case-level properties, we first aggregate per trace and then take the average, minimum, and maximum. For instance, for a case-level property that explains the maximum number of distinct resources per trace, we count distinct resources that executed a group's events for each trace and take the maximum.

Handling noise. There may be events with attribute values that occur infrequently in the established groups, which may pollute otherwise clear, representative group properties. To deal with such noise, we introduce a noise-filtering threshold τ, which can take values between 0 and 1 with a default value of 0.2 (the commonly used noise filtering threshold to separate frequent from infrequent behavior). We remove an event from a group g if the value's relative frequency in g is less than τ times the values' relative frequency in the log and recompute the property.

4.4 Step 4: Suggest event groups

In the final step, we select those groups that have properties that are actually interesting, i.e., we establish a set $G_s \subseteq G$ of groups to be suggested to the user. For these, we then create textual descriptions providing the most interesting properties per group, such as visualized in Fig. 2 of our running example.

Selecting groups to suggest. Using the properties that have been derived for a group g, we make a selection of groups to present to the user based on the interestingness of their properties. We argue that there are primarily two aspects that determine if a property is interesting for multi-perspective event abstraction: *distinctness* and *uniqueness*.

Distinctness. The distinctness of a property assumes that the more a property of a group differs from that of others, the more interesting it is. For instance, if a group of events is the only one that contains the *Manager* role, this makes it interesting. We compute the *earth mover's distance* [13] using the property's value sets for categorical properties and the property's averages per case for numerical ones for each group versus all other groups. The sum of the distances is the distinctness score of a property. The larger this score, the more distinct this group is from others for the respective property.

Uniqueness. The uniqueness of a property reflects how similar events in a group are with respect to a specific property. For instance, a group that contains events that all refer to the *Assistant* role makes this group more interesting than a group, whose events refer to five different roles. The uniqueness of a categorical property is the number of distinct values that occur for it in this group, whereas for numerical ones, we calculate the variance of the values within a group. On the case level, the uniqueness can be quantified using the mean number of distinct values per case for categorical properties, respectively the mean value range (difference between minimum and maximum) for numerical ones. The smaller this score, the more unique a group is for the property.

Inclusion criterion. We rank the groups per property and include a group g in G_s if it ranks highest for at least one property for either uniqueness or distinctness.

Generating textual group descriptions. Next, we provide understandable explanations for the groups in G_s. To this end, we create natural language descriptions of the properties of a group g, such as exemplified in Fig. 2. For each property, we fill slots of pre-defined template sentences. Examples of already filled template sentences are provided in the rightmost column of Table 2.

4.5 Output

Our approach outputs the set G_s of event group suggestions for event abstraction along with their corresponding textual descriptions. A user can inspect the generated descriptions and select meaningful groups. In this manner, we introduce a means to ensure that the groups that are ultimately used for abstraction are actually useful for the user with respect to their downstream analysis goal. While these textual descriptions are a means to explain the generated suggestions in an intuitive manner, the set of suggested groups in G_s are the important output for the actual event abstraction. They can be used to build a mapping from low-level events to higher-level activities, once each selected group is assigned a label. The concrete abstraction of the low-level event log can then be instantiated in various manners. For instance, we can replace each low-level event's class with the label associated with its group, i.e., high-level activity, and only retain the last event with the same label per trace. An important aspect is to consider multiple instances of the same high-level activity within a trace [8], which we will address when further developing our approach.

5 Proof of Concept

We implemented our approach as a Python prototype and simulated an event log that mirrors the scenario outlined in the problem illustration (Section 2)[4]. We aim to show that our approach can find groups of low-level events that correspond to meaningful high-level activities and that these can be used for event abstraction.

Data. There are no public logs available that record data as considered in our work and for which a ground truth is known. Therefore, we modeled a high-level and corresponding low-level Petri net. We simulated the low-level net introducing multi-perspective contextual dependencies and n:m relationships between the event classes and high-level activities. For instance, the execution of the *Decide on acceptance* activity (cf. Fig. 5) yields *Open document*, *Close document*, and *Update record* events, is performed by one manager per case, and takes at most 20 minutes.

Settings. We trained MPPN on the event log (cf. Section 4.1) generating vectors r of size 128. It reached almost 100% accuracy on all attributes except `Resource` with 73%. For PCA, we chose an explained variance of 0.99 to minimize information loss.

Results. Table 3 shows the groups suggested by our approach, including the multi-perspective context found in their event attributes. How these groups relate to the original high-level activities is indicated in Fig. 5.

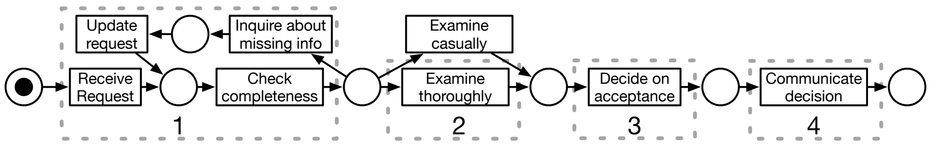


Fig. 5: High-level process model with groups suggested by our approach.

Table 3: Group suggestions found by our approach.

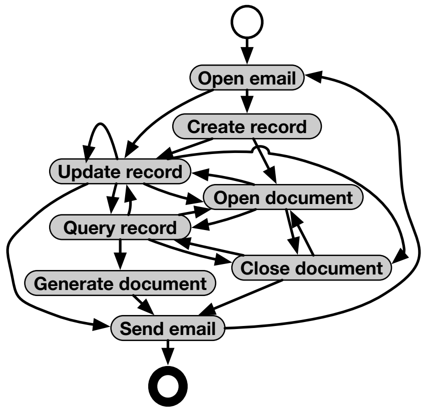
Group	1	2	3	4
Event classes	Open email, Open document, Create record, Update record, Close document, Send email	Open document, Query record, Update record, Close document	Open document, Update record, Close document	Generate document, Query record, Send email
Context	Roles: Assistant Resource: avg. 2.5 per case Duration: 3h30m per case Status: complete, incomplete	Roles: Expert Resource: 1 per case Duration: 20m per case Status: complete	Roles: Manager Resource: 1 per case Duration: 15m per case Status: accept, reject	Roles: Assistant Resource: 1 per case Duration: 8m per case Status: accept, reject

We found that our approach identified three groups of events that exactly resemble high-level activities. Group 2 corresponds to the *Examine thoroughly* activity, Group 3 to *Decide on acceptance*, and Group 4 to *Communicate decision*. Notably, *Decide on acceptance* consist of the same set of low-level event classes as *Examine thoroughly*. However, Group 1 represents the whole initial phase of the process, which actually consists of four high-level activities, i.e., our approach could not discriminate the intended

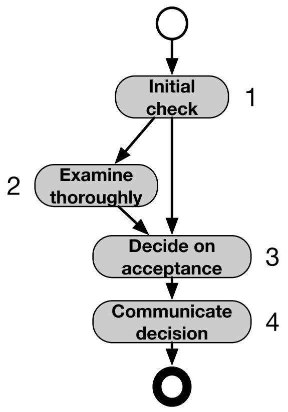
[4] The source code, high-level as well as low-level process models, simulation, and a detailed scenario description are all available at https://github.com/a-rebmann/exploratory-abstraction.

high-level activities. This could be due to ambiguous contextual information, e.g., because the events all happen at the beginning of their case and are executed by the same role. However, depending on the specific analysis purpose, this event group may still be meaningful. If, for instance, a user is interested in how requests are examined and how decisions are made, they do want to abstract from the details of this initial phase.

To highlight the usefulness of the suggested groups for abstraction, we applied them to the low-level event log, omitting events from groups not included in G_s. In particular, we map the low-level events of each group $g \in G_s$, to high-level activities. The DFG of the low-level event log and the DFG obtained after abstracting the log are visualized in Fig. 6. In the low-level DFG, the nodes refer to the distinct event classes in the log. Because one event class can be part of multiple high-level activities and one high-level activity can consist of multiple low-level event classes, limited insights can be obtained about the underlying process. From Fig. 6a it is, therefore, impossible to derive the actual relations to activities in Fig. 5. For instance, since *Send email* events relate to both inquiring about missing information (at the start of the process) and communicating a decision (at the end), there is a loop in the low-level DFG from the last to the first node, which obscures the distinct activities. However, our approach was able to group events in a way, such that a meaningful structure becomes visible (Fig. 6b), e.g., by assigning *Send email* events with different contexts (start vs. end of the process) to different groups. The initial process phase has been abstracted into a single activity, i.e., *Initial check* (the values of the *Status* attribute, i.e., *complete* and *incomplete*, hint at a checking activity). Moreover, clear behavioral patterns that were "hidden" in the low-level DFG are revealed for the later phase of the process: there is a choice between doing a thorough examination or not and there is a sequence between first examining the request, deciding on it, and finally communicating the decision. Note that we manually assigned meaningful labels to the new activities, since this is not yet supported by the approach. However, the descriptions of the multi-perspective event contexts our approach creates already provide the user with hints on how to label the groups.



(a) Low-level DFG (before abstraction).



(b) DFG after abstraction; omitting events from groups not included in G_s.

Fig. 6: Abstraction impact achieved with the suggested groups.

These results indicate the potential of the approach to identify meaningful groups of events for event abstraction without any knowledge of true high-level activities. Also, the necessity to involve the user becomes clear, who can inspect group descriptions and make the final decision if a group is meaningful and which activity it resembles.

6 Related Work

A broad range of event abstraction techniques has been proposed in the context of process mining [4, 17]. To conduct meaningful abstraction, techniques require explicit input about high-level activities, which has to be provided by the user beforehand. For instance, some techniques assume a data attribute to indicate higher abstraction levels [7, 9], whereas others assume high-level process models as input [2]. While a recent technique explains the relations between low-level events and activities, the high-level activities and a mapping to low-level event classes are still required [6]. Other techniques do not require users to explicitly provide information about higher-level activities themselves, but criteria about when events are considered to be part of the same high-level activity, e.g., using temporal information [3] or requirements about the specific characteristics high-level activities should have [12]. In contrast to these techniques, our approach does not require the user to provide input about high-level activities upfront, but supports them in finding suitable groups of events based on their properties, which can then be used to abstract the event log in a meaningful manner.

Beyond the context of event abstraction, a recent study [18] examined exploratory analysis practices in process mining finding that few techniques support the user in the exploration of event data. A notable example is the work by Seeliger et al. [14] who introduced a system for trace clustering, which recommends clusters to analyze based on process performance indicators and thus, suggests groups of cases rather than events. Tsoury et al. [16] strive to augment logs with information derived from database records and transaction logs to allow for deeper insights when exploring event data. While these works provide the user with valuable support when analyzing complex event logs, they do not consider lifting low-level event data to a more meaningful level of abstraction.

7 Conclusion

This paper proposed an approach to identify and suggest groups of low-level events based on their multi-perspective recurring contexts that it learns using only information available in the event log. Users can inspect and select suggested groups, which supports the meaningful abstraction of event logs without the need to provide elaborate input about high-level activities upfront. In an initial proof of concept, we showed that the approach can indeed identify groups that correspond to high-level activities.

The research presented in this workshop paper is work in progress. We aim to expand the current work in several directions. First, we aim to extend the scope of our approach by adding a phase in which users can explore groups and interactively refine meaningful but too coarse-grained ones (such as Group 1 in Section 5), e.g., by triggering a clustering of a single group. Also, we aim to provide the user with various options for abstracting events by clustering the same representations but with different settings.

Furthermore, if a group is discarded by the user because it does not make sense to them, e.g., because events with *complete* as the value for a `Status` attribute were assigned to the same group as events with *incomplete*, we want to incorporate their decision. In such cases, a re-clustering could be applied that takes this feedback into account and suggest groups that adhere to it. Second, motivated by the shift towards conducting data-driven process analysis in an object-centric and view-based manner [5], we aim to overcome the assumption that low-level events belong to exactly one case. Finally, to assess the usefulness of our (extended) approach, we aim to go beyond an evaluation using synthetic logs, by applying it in real-word settings and involving participants in a user study to assess the value of the suggestions our approach provides.

References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer (2016)
2. Baier, T., Mendling, J., Weske, M.: Bridging abstraction layers in process mining. Information Systems **46**, 123–139 (2014)
3. De Leoni, M., Dündar, S.: Event-log abstraction using batch session identification and clustering. Proceedings of the ACM Symposium on Applied Computing pp. 36–44 (2020)
4. Diba, K., Batoulis, K., Weidlich, M., Weske, M.: Extraction, correlation, and abstraction of event data for process mining. Wiley Interdisciplinary Reviews **10**(3), 1–31 (2020)
5. Fahland, D.: Process mining over multiple behavioral dimensions with event knowledge graphs. In: Process Mining Handbook, pp. 274–319. Springer (2022)
6. Fazzinga, B., Flesca, S., Furfaro, F., Pontieri, L.: Process mining meets argumentation: Explainable interpretations of low-level event logs via abstract argumentation. Information Systems **107**, 101987 (01 2022)
7. Leemans, S.J., Goel, K., van Zelst, S.J.: Using multi-level information in hierarchical process mining: Balancing behavioural quality and model complexity. In: ICPM. pp. 137–144 (2020)
8. Li, C.Y., van Zelst, S.J., van der Aalst, W.M.: An activity instance based hierarchical framework for event abstraction. In: ICPM. pp. 160–167. IEEE (2021)
9. Lu, X., Gal, A., Reijers, H.A.: Discovering hierarchical processes using flexible activity trees for event abstraction. In: ICPM. pp. 145–152. IEEE (2020)
10. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M., Toussaint, P.J.: Guided process discovery – a pattern-based approach. Information Systems **76**, 1–18 (2018)
11. Pfeiffer, P., Lahann, J., Fettke, P.: Multivariate business process representation learning utilizing gramian angular fields and convolutional neural networks. In: BPM. pp. 327–344. Springer (2021)
12. Rebmann, A., Weidlich, M., van der Aa, H.: GECCO: Constraint-driven abstraction of low-level event logs. In: ICDE 2022. pp. 150–163. IEEE (2022)
13. Rubner, Y., Tomasi, C., Guibas, L.J.: A metric for distributions with applications to image databases. In: Sixth International Conference on Computer Vision. pp. 59–66. IEEE (1998)
14. Seeliger, A., Sánchez Guinea, A., Nolle, T., Mühlhäuser, M.: Processexplorer: intelligent process mining guidance. In: BPM. pp. 216–231. Springer (2019)
15. Tibshirani, R., Walther, G., Hastie, T.: Estimating the number of clusters in a data set via the gap statistic. Journal of the Royal Statistical Society **63**(2), 411–423 (2001)
16. Tsoury, A., Soffer, P., Reinhartz-Berger, I.: A conceptual framework for supporting deep exploration of business process behavior. In: ER. pp. 58–71. Springer (2018)
17. van Zelst, S.J., Mannhardt, F., de Leoni, M., Koschmider, A.: Event abstraction in process mining: literature review and taxonomy. Granular Computing **2** (2020)
18. Zerbato, F., Soffer, P., Weber, B.: Initial insights into exploratory process mining practices. In: BPM. pp. 145–161. Springer (2021)