

Event Log Abstraction in Client-Server Applications

M. Amin Yazdi¹^a, Pejman Farhadi Ghalatia²^b and Benedikt Heinrichs¹^c

¹IT Center, RWTH Aachen University, Aachen, Germany

²Aachen Institute for Advanced Study in Computational Engineering Science, RWTH Aachen University, Aachen, Germany

Keywords: Business Process Intelligence, Event Log Abstraction, Process Discovery, Data Pre-processing.

Abstract: Process mining provides various techniques in response to the increasing demand for understanding the execution of the underlying processes of software systems. The discovery and conformance checking techniques allow for the analysis of event data and verify compliances. However, in real-life scenarios, the event data recorded by software systems often contain numerous activities resulting in unstructured process models that are not usable by domain experts. Hence, event log abstraction is an essential preprocessing step to deliver a desired abstracted model that is human-readable and enables process analysis. This paper provides an overview of the literature and proposes a novel approach for transforming fine-granular event logs generated from client-server applications to a higher level of abstraction suitable for domain experts for further analysis. Moreover, we demonstrate the validity of the suggested method with the help of two case studies.

1 INTRODUCTION

Newly born software solutions emerge every day to fulfill the demand for digitalization. The continuous development of new applications is leading us toward the establishment of software systems with complex internal processes. For instance, universities provide distributed software solutions to support research data management and enable reusability, findability, and accessibility of research data (Yazdi et al., 2016; Politze et al., 2020). It results in a heterogeneous and distributed IT infrastructure with a complex and unstructured landscape of user processes. To tackle this issue, data modeling and exploring researchers' processes would help domain experts understand and monitor user behaviors and discover the underlying operational processes. Such a process-oriented modeling allows for the discovery of the challenges researchers face during their research's life-cycle (Valdez et al., 2015).

To analyze the dynamic behavior of users and monitor the status of operational processes, we use process mining techniques which enables us to discover process models (Yazdi, 2019). Two main process mining tasks are process *discovery* and *conformance* checking (van der Aalst, 2016). Process *dis-*

covery aims to discover a process model that represents the real descriptive behavior, based on real-life events. There are various process discovery algorithms including but not limited to Inductive Miner (IM), Alpha miner, Fuzzy miner, or Heuristic miner. *Conformance* checking compares a process model with an event log to find commonalities and differences. It enables us to measure the fitness of the discovered model (Van der Aalst et al., 2012). Various software products such as ProM (Van Dongen et al., 2005), Disco (Günther and Rozinat, 2012), Rapidminer (Mans et al., 2014) are available to assist us with process analysis tasks.

Often, software systems work in a complex infrastructure where a user request may require many software components and microservices to help for executing a task. This complex infrastructure often results in a $n:m$ relationship between server-side events and client-side events that are too fine-grained. Applying process discovery to these low-level event logs often results in obtaining an unstructured process model called "Spaghetti", that is unsuitable for process analysis (van der Aalst, 2016). Hence, various event log abstraction techniques have been developed to transform fine granular (low-level) event logs to a higher level of abstraction (high-level). However, the main challenge is to find a desired abstraction level that is interpretable by domain experts and still describes the reality of the business activities.

^a <https://orcid.org/0000-0002-0628-4644>

^b <https://orcid.org/0000-0002-7627-533X>

^c <https://orcid.org/0000-0003-3309-5985>

Table 1: The work of literature and classification based on abstraction techniques.

Author	Grouping		Input Data		Event Perspective	Mapping Relationship	Internal Abstraction		Validity	Quality Indicator	Target Domain
	Su.	Un.	Co.	Di.			Pr.	De.			
(Begicheva and Lomazova, 2017)	✓		✓		Sequential	n:1	✓		Formal	Fitness	Acyclic Cases
(Mannhardt et al., 2016)	✓		✓		Sequential	n:1		✓	Real-Life	Fitness & Matching Error	Information Systems
(Tax et al., 2016)	✓		✓		Sequential	n:m	✓		Synthetic	Levenshtein Distance	Parallel Activities
(Liu et al., 2018)	✓		✓		Sequential	n:1		✓	Synthetic	Fitness & Precision & Generalization	Information Systems
(de Leoni and Dündar, 2020)		✓		✓	Non-Sequential	n:1		✓	Real-Life	Fitness	Information Systems
(Mannhardt and Tax, 2017)		✓	✓		Sequential	n:1		✓	Real-Life	Fitness	Automatic Pattern Discovery
Our Approach	✓		✓	✓	Non-Sequential	n:m		✓	Real-Life	Fitness, PCC	Client-Server Applications

In order to deal with the challenge of abstracting low-level events, many methods are proposed (and discussed in Section 2). Some of the existing methods rely on unsupervised learning techniques to cluster groups of events into a higher-level of events. On one hand, current unsupervised learning methods are not able to acquire a suitable degree of abstraction and on the other hand, it leads us to an unclear relabeling strategy for the cluster of abstracted events. These learning methods either require manual labeling by domain experts or automatic concatenation of labels that create unreadable and long activity names. Furthermore, there are several supervised learning methods to overcome the mentioned challenges. However, these methods either utilize training sets to guide the abstraction process or rely on domain experts' inputs for the task of relabeling the activity names. Hence relabeling abstracted activity names is a non-trivial task. The challenges mentioned earlier describe the necessity for an approach to abstract low-level event logs to a suitable higher-level of granularity.

In this paper, we describe an iterative supervised abstraction technique for client-server applications. Whereas client-side event logs are used as the training set to describe the server-side event logs, and the Pearson Coefficient Correlation (PCC) is employed to measure event similarity for the task of abstracting activity names. The method described in this study is built on top of the suggested approach in (Yazdi and Politze, 2020) for acquiring the server-side event logs. Accordingly, we are relying on the OAuth2 workflow for recording the server-side event logs. The client-side event log plays the role of descriptive user activities that are understandable by domain experts and enable us to map low-level server-side event logs to high-level activities.

Overall, in this paper, we try to answer the following research questions in client-server applications:

1. How can we abstract event logs so that they are reliable and descriptive for the domain experts?
2. How to balance the abstraction granularity with respect to process models' fitness?

As suggested in the literature (Buijs et al., 2012), we use the model fitness as an indicator of the suitability of the discovered model. Furthermore, with the help of two case studies, we evaluate the validity and

scalability of our approach.

The rest of this paper is as follows. Section 2 introduces the related work and studies related to event log abstraction techniques. In Section 3, we provide the preliminaries and formal models used throughout the paper. Section 4 describes the approach used to enable the event log abstraction for client-server applications. We examine our approach with the help of two case studies in Section 5. In Section 6, we address the challenges and possible future work. Finally, in Section 7, we discuss the benefits and the contributions of our work.

2 RELATED WORK

There has been a recent research trend on event log abstraction methods and their challenges (van Zelst et al., 2020). This section categorizes different criteria that we find essential for every abstraction method and describes the work of literature accordingly. Additionally, we position our work concerning each criterion. Table 1 provides an overall birds-eye-view of related works and how the suggested approach differs from other existing solutions.

2.1 Log Abstraction Strategies

In this part, we elaborate on several event log abstraction criteria, their benefits and drawbacks. Furthermore, we also position our work accordingly.

2.1.1 Grouping Strategy

The grouping strategy refers to the learning techniques used for converting low-level event logs to a higher-level of granularity. There are two main categories of abstraction operations: *aggregation* and *elimination* (Smirnov et al., 2012). The event *elimination* operation leads to a naive approach by omitting insignificant events. However, this way results in models that are underfitting and misses information about eliminated events. The *aggregation* operation generates a significant group of events in which a combination of relatively insignificant events exists. There are two sub-domains of the *aggregation* oper-

ation, namely, *unsupervised* and *supervised* learning methods.

Unsupervised abstraction methods rely on completely automated techniques to group events and often result in less accurate high-level models. The labels in these methods are regenerated either by concatenation of activities or by using assumptions without the involvement of domain knowledge. Authors in (Mannhardt and Tax, 2017) use Local Process Model (LPM) discovery to identify frequent activity patterns in the process model automatically, and authors of (de Leoni and Dünder, 2020) use clustering methods to group activities based on the frequency of occurrences and use clusters' centroids for relabeling corresponding groups. Unsupervised learning methods are beneficial if no domain knowledge is available or acquiring additional information is not feasible.

On the contrary, *supervised* learning methods are the most common and practiced technique used for simplifying process models. These methods rely on some form of external or additional information to group and relabel event logs. Authors in (Begicheva and Lomazova, 2017; Mannhardt et al., 2016; Tax et al., 2016; Liu et al., 2018) use *supervised* learning techniques either by capturing additional information or creating a training data set to annotate fine-granular event logs. As the *supervised* learning technique is considered to be a more reliable abstraction technique for real-life scenarios and allows for additional domain experts' interventions, our approach also focuses on a *supervised* learning method.

2.1.2 Input Data

Input Data notes the nature of data in the form of discrete or continuous events. The discrete events are a set of infinite numbers of values (ex. categorical or real values), and continuous events refer to as a set of data that can take any values (ex. activity names). Event logs can be in the form of discrete sequences of events or continuous as a result of the execution of activities. Most techniques covered in the literature review are using some form of continuous activity names for the analysis. However, authors in (de Leoni and Dünder, 2020) vectorize the sequences of continuous event data and transformed the event log into the discrete data format. The discrete data, facilitates the execution of data mining algorithms, such as analyzing the frequency of occurrences of activities within an event log. Likewise, our approach adopts both continuous event data for discovering process models and discrete event data for the task of measuring the similarity of two activities.

2.1.3 Event Perspective

Event Perspective points to the effect of the order of appearing event logs on the abstraction technique. Most of the literature's procedures exploit only the order of the sequence of events as they appear in the low-level events. Authors in (de Leoni and Dünder, 2020) use the non-sequential order of events with unsupervised abstraction methods. In the system under study, the order of execution of events are not sequential due to the nature of the distributed systems, hence, our approach has to incorporate non-sequential events.

2.1.4 Mapping Relationship

The mapping relationship between instances of low-level to higher-level activities. There are $1:1$, $n:1$, $n:m$ mapping relationships. In the case of $1:1$ mapping, there is no abstraction that can be performed as the level of granularity does not change, i.e., for every event, there is only one corresponding activity. Most of the literature focuses on $n:1$ mapping or also known as shared functionality, i.e., for every low-level event log, there might be multiple higher-level activities. In software systems, user activities are often not directly corresponding to a recognizable event on the server-side, hence resulting in a $n:m$ mapping relationship. We position our work as $n:m$ mapping relationship as there are both $1:1$ as well as $n:1$ mappings between event logs of each level.

2.1.5 Internal Abstraction Modeling

Internal Abstraction Modeling highlights the use of probabilistic or deterministic factors to identify the sequence of event distributions. Similar to our suggested approach, most of the literature focuses on deterministic strategies. However, authors in (Begicheva and Lomazova, 2017) also employ the theory of regions to classify corresponding high-level activities, and authors in (Tax et al., 2016) adopt the Conditional Random Fields (CRF) as a probabilistic factor for the task of event sequence relabeling.

2.1.6 Validity

Validity describes the evaluation procedure used to demonstrate the soundness of a suggested technique. Unlike the authors of (Begicheva and Lomazova, 2017) that only use formal mathematical representation, all other works examine their approach over either real-life or synthetic event logs. However, only the authors of (Mannhardt et al., 2016) evaluate their technique by implementing their approach in a real

infrastructure. It is particularly important because despite the existence of real-life event logs, it does not include real challenges that exist in the field, such as noise, outliers, or missing data in event logs. We validate our method with real-life event logs by directly collecting and utilizing event logs from an existing software system to demonstrate the effectiveness of our approach.

2.1.7 Quality Indicator

These indicators support the quality dimensions of the discovered abstracted process model. Usually discovered models are evaluated based on metrics such as fitness, precision and generalization, which gives additional insights into the changes by the event log abstraction process (van der Aalst, 2016). According to (Buijs et al., 2012), fitness quantifies the extent to which the discovered model can accurately reproduce the cases recorded in the log. Precision quantifies the fraction of the behavior allowed by the model, which is not seen in the event log. Generalization assesses the extent to which the resulting model will be able to reproduce the future behavior of the process.

With respect to our literature study, only the authors of (Liu et al., 2018) have successfully reported the quality of their approach with the help of fitness, precision, and generalization. The rest of scholars in the field have either reported the model fitness as the only quality factor or have backed their work with supplementary measures such as Matching Error as a threshold for excluding unreliable matches between fine-granular event logs and high-level activities (Mannhardt et al., 2016). Nevertheless, authors in (Tax et al., 2016) solely leverage from the Levenshtein similarity distance as a degree to express the closeness of two traces from different granularity levels. In our study, besides the model fitness, we additionally use the Pearson Coefficient Correlation to compare and identify similar high-level activities.

2.1.8 Target Domain

The application domain that a technique is primarily focused on. Unfortunately, most of the literature is not generic enough and aims for a particular application domain. For instance, the suggested solution in (Begicheva and Lomazova, 2017) only targets the non-recursive activities (acyclic cases). The technique in (Tax et al., 2016) intends for the abstraction of only parallel activities with any executive order. The automatic pattern discovery and clustering of activities is the focus of the authors in (Mannhardt and Tax, 2017). Other literature under study appear to focus on the analysis of information systems such as a

digital whiteboard software (Mannhardt et al., 2016), user behavior analysis on software systems (Liu et al., 2018), and analysis of the frequency of a website visits (de Leoni and Dündar, 2020). On the contrary, we propose a universal approach that can be employed over client-server applications.

2.2 Summary

Despite the focus on acyclic cases in (Begicheva and Lomazova, 2017), authors only use a formal representation to demonstrate the handling of duplication in the event logs but have ignored the stuttering subsequences of activities. Moreover, the approaches suggested by (Mannhardt et al., 2016) and (Liu et al., 2018) rely on manual labeling of low-level events to high-level activities through interviews, observations, and excluding traces with missing or unknown events. The method proposed in (Tax et al., 2016) is capable of dealing with parallel activities; However, it relies on excessive domain knowledge. Methods such as (de Leoni and Dündar, 2020) and (Mannhardt and Tax, 2017) take advantage of using unsupervised learning techniques to relabel the abstract traces automatically. Cluster centroids or label concatenations generate the new labels for the higher-level activities, but the technology acceptance for the domain experts is uninvestigated. Although the authors in (Mannhardt and Tax, 2017) use Local Process Models (LPM) and automatic pattern discovery techniques to automatically detect start and end of activities, it is limited to a fixed number of candidates LPMs based on the event frequency ranking.

Overall, almost all the authors have used the Inductive Miner (IM) algorithm to discover process models. The IM algorithm can deal with infrequent behaviors while ensuring the soundness of the discovered process model (Leemans et al., 2014). Additionally, different variations of Petri Nets notations have been used to illustrate the discovered model. By referring to Table 1, it is clear that none of the approaches is suitable to be utilized in the existing infrastructure and either relies on internal probabilistic abstraction techniques or requires extensive domain knowledge to help with the task of relabeling the abstracting activities.

3 PRELIMINARIES

In this section, we provide the basic concepts and formal models used throughout the paper.

Given a set A , $\sigma = \langle a_1, a_2, \dots, a_n \rangle \in A^*$ is the set of all finite sequence over the A . $\sigma(i) = a_i$ denotes

the i^{th} element of the sequence. $\beta(A)$ is the set of all multi-sets over set A . $P_{NE}(A)$ is the set of all non-empty subsets over the set A . $|\sigma| = n$ is the length of σ . For $\sigma_1, \sigma_2 \in A^*$, $\sigma_1 \subseteq \sigma_2$ if σ_1 is a sub-sequence of σ_2 . e.g., $\langle g, b, o \rangle \subseteq \langle z, g, b, b, g, b, o, q \rangle$. For $\sigma \in A^*$, $\{a \in \sigma\}$ is a set of elements in σ . For $\sigma \in A^*$, $[a \in \sigma]$ is a multi-set of elements in σ , e.g., $[a \in \langle g, b, z, g, b \rangle] = [g^2, b^2, z]$.

(Event, Event Log) An event is a n-tuple $e = (t, c, a_c, a_s)$ and $e \in L$. Where $t \in T$ is the event timestamp, $c \in C$ is the case id, $a_c \in A_c$ is the client-side activity, $a_s \in A_s$ is the server-side activity, such that for each client-side activity, there is a set of corresponding server-side activities in respond to a user request. We denote $\xi = T \times C \times A_c \times A_s$ as the universe of all events for the original event log, s.t. $L \subseteq \xi$. In the event log, each event can appear only once. Hence, events are uniquely identifiable by their attributes. Additionally, we denote $\xi_f = T_{start} \times T_{complete} \times C \times A_c$ as the universe of abstracted events. The abstracted event log is represented as $L_f \subseteq \xi_f$, and each event $e_f \in L_f$ is a n-tuple $e_f = (t_{start}, t_{complete}, c_c, a'_c)$ where $\{t_{start}, t_{complete}\} \in T$ are the start and completion timestamp of an event respectively, representing a full software execution life-cycle, $c_c \in C$ is the case id, and $a'_c \in A_c$ is the abstracted activity name.

(Trace) Let x be a set of activities. Then $x \in e$, $\sigma_x = \langle e_{x_1}, e_{x_2}, \dots, e_{x_n} \rangle$ is defined as a trace in the event log, s.t., for each $e_{x_i}, e_{x_j} \in \sigma_x$, $1 \leq i < j \leq n: \pi_C(e_{x_i}) = \pi_C(e_{x_j})$ and $\pi_T(e_{x_i}) \leq \pi_T(e_{x_j})$. Trace $\sigma \in L$ is a sequence of activities. Note that there are no special start and end activities.

(Labeled Petri net) A Petri net is a n-tuple $N = (P, T, F)$ where P is the set of places, T the set of transitions, $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ represents the flow relation. A labeled Petri net $N = (P, T, F, l, A)$ extends the Petri nets with a labeling function $l \in T \rightarrow A$ which maps a transition to an activity from A . As an example, if $l(t) = a$, then a is an observed activity when transition t is executed.

4 APPROACH

Figure 1 represents the overview of our approach to abstract event logs in client-server applications, accompanied with example event logs' transformation throughout the procedure. Note that in the sample logs of figure 1, the client activity names are represented as $[activityName]@[webPage]$ to indicate the web page that a client activity is occurring. In the following section, we elaborate on every step for the recursive log abstraction.

Algorithm 1: Recursive Event Log Abstraction.

Input: Fine-granular event log L
Output: Abstracted event log L_f

- 1 set PCC threshold (Θ) to 1;
- 2 discover Petri net model for L using the Inductive Miner;
- 3 clone L to L' as a temporary event log;
- 4 **while** $fitness \geq 0.8$ **do**
- 5 **foreach** L' **do**
- 6 convert L' to a vectorized weighted matrix $M_{A_c \times A_s}$;
- 7 **foreach** e **do**
- 8 calculate the vector similarity(Δ) between every a_c ;
- 9 **if** $\Delta \geq \Theta$ **then**
- 10 relabel the activity a_c by the concatenation of the two similar activity names a'_c ;
- 11 replace the corresponding event in L' with $e' = (t, c, a'_c, a_s)$;
- 12 **end**
- 13 **end**
- 14 **end**
- 15 Lower PCC threshold Θ by Υ
- 16 **end**
- 17 **foreach** c in L' **do**
- 18 **while** $a'_{c_i} = a'_{c_j}$ **do**
- 19 capture the first (t_{start}) and last ($t_{complete}$) occurring event for the activity a'_c ;
- 20 create event $e_f = (t_{start}, t_{complete}, c, a)$, and append to L_f ;
- 21 **end**
- 22 **end**
- 23 **return** L_f

Additionally, algorithm 1 demonstrates a recursive log abstraction process using PCC (Benesty et al., 2009). The PCC enables us to calculate the similarity of different client-side activities by assessing the corresponding server-side activity occurrences.

Our approach applies the IM algorithm on top of the event log for the process model discovery task (step 1 of fig. 1). We clone the original event log L to a temporary event log L' to be further abstracted and reused for the recursive process (Step 2 of fig. 1). Then the abstracted event log is replayed over the discovered model generated from the original log to evaluate model fitness (step 3 of fig. 1). During this process, the fitness of the discovered model is calculated by: $fitness = \frac{True\ Positive}{True\ Positive + False\ Negative}$. The *True*

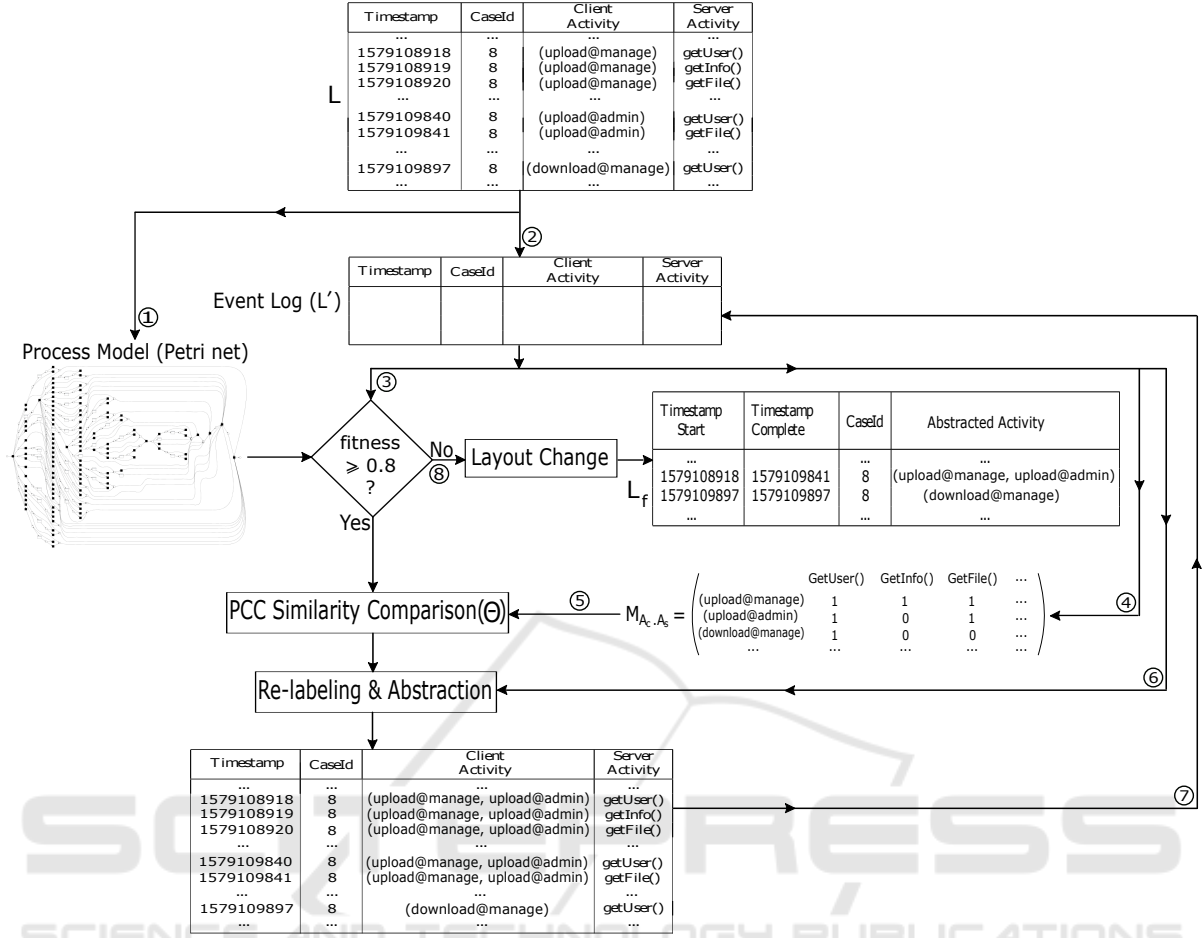


Figure 1: The general overview of the approach and sample event logs transformation throughout the abstraction process.

Positive is the number of traces that align with the discovered model, and the *False Negative* is the number of traces that do not align with the discovered model (Jouck et al., 2018). If the discovered model can completely replay the traces from L' , the model's fitness is 1. Hence, as the log abstraction increases, the fitness decreases.

In line 6 of the Algorithm 1, the event log $e = (t, c, a_c, a_s) \in L$ is transforming to a vectorized descriptive matrix $M_{A_c.A_s}$, i.e., the first column of the matrix is the group of client-side activities (a_c), and every other column represents a server-side activity (a_s). Each row carries the number of executions of server-side activities to a corresponding client-side activity. Note that we increment the number of a_s occurrences per a_c (step 4 of fig. 1). The weighted matrix M enables vector similarity analyses with higher accuracy.

In the block between lines 7 to 13 of Algorithm 1, we evaluate the similarity between two client-side activities (step 5 of fig. 1) and relabel logs by concatenating the client-side activity names (with a comma)

that falls into the PCC threshold (step 6 of fig. 1). The PCC evaluates the strength of a linear association between two vector variables and therefore quantifies two events' similarity. As the PCC description is out of this paper's scope, we refer the readers to (Benesty et al., 2009) for the formula and extensive explanation. The result of abstraction and relabeling is then cycled back into the method (step 7 of fig. 1).

In our proposed approach, we initialize the abstraction process with PCC threshold $\Theta = 1$ and gradually reduce the threshold Θ by γ . As shown in Figure 1, the abstraction process is continued until a fitness quality of 0.8 is reached. We propose the fitness of 0.8 as a criterion to stop the iterative abstraction process in our algorithm as a fitness value below 0.8 cannot replay event logs by that model completely (van der Aalst, 2016; Buijs et al., 2012) and thus lose its crucial quality dimension of a process model.

The block between lines 17 to 22 transforms the event log format and thus, decreases the number of events with each case's same activity (step 8 of fig. 1). At this stage, every event would be accompa-

Table 2: Results of each abstraction iteration over model fitness, number of activities and total number of events.

(a) Metadata Manager tool- case study 1.									
PCC Similarity Threshold	1	97-1	94-1	91-1	82-1	79-1	52-1	40-1	22-1
Fitness	1	0.91	0.89	0.88	0.88	0.84	0.79	0.77	0.73
# Client Activities (a_c)	21	16	15	14	11	10	8	7	6
# Events	13794	10302	9824	9268	7261	6414	5378	4735	4019

(b) Data Archiving tool - case study 2.									
PCC Similarity Threshold	1	97-1	91-1	88-1	85-1	82-1	55-1	28-1	
Fitness	1	0.98	0.88	0.83	0.81	0.76	0.70	0.59	
# Client Activities (a_c)	19	18	17	15	14	12	10	4	
# Events	10487	9973	9481	8297	7920	6553	5481	2203	

nied by the events' start and completion time, i.e., the time taken for the server-side components to process and respond to a client-side request. In the case of a single appearance of an event, the start and completion timestamp is the same to signify an instant event occurrence. The resulting abstracted event log is $L_f \subseteq \xi_f$ where $e_f \in L_f$ and $e_f = (t_{start}, t_{complete}, c, a'_c)$.

5 EXPERIMENTAL EVALUATIONS

To manifest a suggested method's quality and reliability, one needs to examine a system with at least two experiments (Dean et al., 1999). Consequently, we demonstrate the validity, robustness, and generalization of our methodology to obtain a readable and utilizable abstracted event log by conducting two rounds of empirical evaluations in real-life scenarios. The event logs obtained are from information systems provided by the RWTH-Aachen University for metadata management and archiving of research data. We follow (Rafiei and van der Aalst, 2019)'s instructions for securing event logs that comply with EU GDPR privacy regulations. Additionally, as mentioned earlier, there is an n:m relationship between client and server activities for the system under study. Respectively, every user activity triggers a set of server-side activities to process and execute the user's demand for a client-side operation. For the purpose of the case studies, we have collected and analyzed client-server event logs from two web applications, namely Metadata Manager and Data Archiving tool.

5.1 Case Study 1: Metadata Manager Tool

As the name represents, the Metadata Manager is a web application tool for researchers at the university to provide customizable metadata schemas and enable the classification and exploration of research data by their metadata. The most typical user interactions are providing metadata schemas, searching metadata, and adding/editing metadata. Overall, we obtained event logs for a duration of six months with 573 case ids and 13794 events. Metadata Manager generates 21 unique client-side activities and 32 server-side activities responsible for processing user requests.

Results: By applying the proposed approach, we obtained an abstracted process model portraying the presumed user interaction life-cycle. Figure 2a illustrates the discovered complex Petri net before the abstraction of event logs. Figure 2c shows the discovered abstracted model using our suggested approach. Table 2a demonstrates the result of the iterations of process abstraction until no further abstraction is possible. For the proof of concept, we have executed the process beyond our suggested fitness threshold. While considering our recommended fitness criteria as the breaking point for the abstraction process, we obtain an event log with 10 activities, 6414 events, and 573 cases with an overall fitness accuracy of 0.84. Furthermore, by exploring the results in table 2a, we observe that a suitable PCC threshold for classifying similar events and combining events without compromising the fitness on the Metadata Manager tool is 79-1. Note that no further abstraction is possible below the fitness of 0.73 as no more two events are similar below the PCC threshold of 22-1. Thus, we achieved a 52.39% decrease in the number of activities and



Figure 2: The discovered Petri net models before and after applying the event log abstraction technique.

53.51% abstraction in the total number of events for the system under study.

5.2 Case Study 2: Data Archiving Tool

The Data Archiving tool is a web application for researchers of the RWTH-Aachen University to archive research data and restore files as demanded. The users of this service can define a storage expiry date to adhere to the requirements of funding organizations for research projects to ensure that research data are available despite the expiry of a project. The most typical user activities are uploading/downloading data, searching for a research record, and restoring research data. In this case study, we collected event logs for a duration of six months with 413 case ids and 10487 events. The Data Archiving tool generates 19 unique client-side activities and 28 server-side activities responsible for processing user's client-side requests.

Results: Figure 2b and 2d respectively illustrate the discovered process model before and after applying the abstraction technique respectively. By checking the abstraction results at the suggested fitness criteria, we reach an event log with 7920 events, 14 client-side activities, with a fitness accuracy of 0.81. Table 2b reports on the results with every abstraction iteration until no two further activities are comparable. Moreover, we can find a suitable PCC threshold of 85-1 for achieving a desired level of abstraction. We also observe that no further abstraction is possible below the fitness of 0.59 as no more two events are similar below the PCC threshold of 28-1. Therefore, we managed a 26.32% decrease in the number of activities and a 24.47% reduction in the total number of events for the system under study.

6 DISCUSSION AND FUTURE WORK

We used the Data Follow Graph model to illustrate the abstracted models due to ease of understanding for the domain experts. We received various qualitative feedback, such as: “[...] I can now understand the user interactions and assess the user behaviors without being overwhelmed by many activities that do represent the actual processes, [...] I am wondering why we have such a bad performance on restoring files, I need to investigate this.”, referring to a violation of a KPI for restoring files that was previously unknown. Thus, the proposed method could successfully identify non-functional requirements and draw developers’ attention to the right software components for further technical investigations.

Despite our effort to propose a generalizable event log abstraction technique, the suggested method is only applicable to client-server applications. We have to acknowledge that one of the most time-consuming and manual efforts in this technique is to evaluate the input log for noise, outliers, and anomalies in the event log. Unfortunately, despite our attempt, our case studies’ report lacks the exploration of the precision and the generalization as other quality indicators due to extensive computing power required for calculating these factors via ProM implementation. Thus these calculations never terminated. Moreover, we use the concatenation of activity names for relabeling task; yet, this may cause readability issues for some domain experts if the number of original activities increases drastically. Lastly, we are also relying on event logs being generated by the OAuth2 workflow (authorization service). Consequently, there may be server-side activities that are not recorded. So, further study is required to validate our approach while including all executing software components.

7 CONCLUSIONS

This paper describes a novel approach for event log abstraction in client-server applications. In Section 2, we discussed the state of the art and elaborated on the benefits and shortcomings of each work, and explained how our suggested method could overcome existing limitations for the system under study. Our suggested approach enabled us to gradually abstract the fine-grained event logs to higher levels without losing essential information, enabling the domain experts to use the appropriate discovered model for further analysis. Besides answering our research questions, we evaluated the proposed approach with the

help of two real-life experiments. Overall, this work’s contributions are adaptability of the approach to any client-server application and enabling automatic relabeling of abstracted activity names at a desired level of granularity. Additionally, it empowers discovering the exact software execution life-cycle, facilitating the discovery of user behavior on client-server applications with high accuracy.

REFERENCES

- Beginicheva, A. K. and Lomazova, I. A. (2017). Discovering high-level process models from event logs. *Modeling and Analysis of Information Systems*, 24(2):125–140.
- Benesty, J., Chen, J., Huang, Y., and Cohen, I. (2009). Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer.
- Buijs, J. C., Van Dongen, B. F., and van Der Aalst, W. M. (2012). On the role of fitness, precision, generalization and simplicity in process discovery. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, pages 305–322. Springer.
- de Leoni, M. and Dünder, S. (2020). Event-log abstraction using batch session identification and clustering. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 36–44.
- Dean, A., Voss, D., Draguljić, D., et al. (1999). *Design and analysis of experiments*, volume 1. Springer.
- Günther, C. W. and Rozinat, A. (2012). Disco: Discover your processes. *BPM (Demos)*, 940:40–44.
- Jouck, T., Bolt, A., Depaire, B., de Leoni, M., and van der Aalst, W. M. (2018). An integrated framework for process discovery algorithm evaluation. *arXiv preprint arXiv:1806.07222*.
- Leemans, S. J., Fahland, D., and Van Der Aalst, W. M. (2014). Process and deviation exploration with inductive visual miner. *BPM (Demos)*, 1295(46):8.
- Liu, C., Wang, S., Gao, S., Zhang, F., and Cheng, J. (2018). User behavior discovery from low-level software execution log. *IEEE Transactions on Electrical and Electronic Engineering*, 13(11):1624–1632.
- Mannhardt, F., De Leoni, M., Reijers, H. A., Van Der Aalst, W. M., and Toussaint, P. J. (2016). From low-level events to activities-a pattern-based approach. In *International conference on business process management*, pages 125–141. Springer.
- Mannhardt, F. and Tax, N. (2017). Unsupervised event abstraction using pattern abstraction and local process models. *arXiv preprint arXiv:1704.03520*.
- Mans, R., van der Aalst, W. M., and Verbeek, H. (2014). Supporting process mining workflows with rapid-prom. *BPM (Demos)*, 56.
- Politze, M., Claus, F., Brenger, B., Yazdi, M. A., Heinrichs, B., and Schwarz, A. (2020). How to manage it resources in research projects? towards a collaborative

- scientific integration environment. *European Journal of Higher Education IT*, 2.
- Rafiei, M. and van der Aalst, W. M. (2019). Mining roles from event logs while preserving privacy. In *International Conference on Business Process Management*, pages 676–689. Springer.
- Smirnov, S., Reijers, H. A., and Weske, M. (2012). From fine-grained to abstract process models: A semantic approach. *Information Systems*, 37(8):784–797.
- Tax, N., Sidorova, N., Haakma, R., and van der Aalst, W. M. (2016). Event abstraction for process mining using supervised learning techniques. In *Proceedings of SAI Intelligent Systems Conference*, pages 251–269. Springer.
- Valdez, A. C., Özdemir, D., Yazdi, M. A., Schaar, A. K., and Zieffle, M. (2015). Orchestrating collaboration-using visual collaboration suggestion for steering of research clusters. *Procedia Manufacturing*, 3:363–370.
- van der Aalst, W. (2016). *Process mining: data science in action*. Springer.
- Van der Aalst, W., Adriansyah, A., and van Dongen, B. (2012). Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192.
- Van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H., Weijters, A., and van Der Aalst, W. M. (2005). The prom framework: A new era in process mining tool support. In *International conference on application and theory of petri nets*, pages 444–454. Springer.
- van Zelst, S. J., Mannhardt, F., de Leoni, M., and Koschmider, A. (2020). Event abstraction in process mining: literature review and taxonomy. *Granular Computing*, pages 1–18.
- Yazdi, M. A. (2019). Enabling operational support in the research data life cycle. In *Proceedings of the first International Conference on Process Mining*, pages 1–10. CEUR.
- Yazdi, M. A. and Politze, M. (2020). Reverse engineering: The university distributed services. In *Proceedings of the Future Technologies Conference (FTC) 2020, Volume 2*, pages 223–238. Springer.
- Yazdi, M. A., Valdez, A. C., Lichtschlag, L., Zieffle, M., and Borchers, J. (2016). Visualizing opportunities of collaboration in large research organizations. In *International Conference on HCI in Business, Government, and Organizations*, pages 350–361. Springer.