Merging event logs: Combining granularity levels for process flow analysis

Lihi Raichelson[1], Pnina Soffer[1], and Eric Verbeek[2]

[1]Department of Information Systems, University of Haifa, Haifa 31905, Israel
LihiRaOs@gmail.com, Spnina@is.haifa.ac.il

[2]Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
h.m.w.verbeek@tue.nl

Abstract. Process mining techniques enable the discovery and analysis of business processes and the identification of opportunities for improvement. Processes often comprise separately managed procedures documented in separate log files, which are impossible to mine in an integrative manner as the complete end-to-end process flow is obscure. These procedures can have simple (one-to-one) or complex (many-to-one or many-to-many) relationships among them. When complex relationships exist, typically different granularity levels are involved. In this paper, we present a merging algorithm that results in a comprehensive merged log that can handle all kinds of relationships between the procedures. Addressing differences in the granularity levels, it offers two views of the end-to-end process: a case view and an instance view. This enables the identification of process flow problems that could not be detected by previous techniques.

The unified log can be used by process mining techniques to identify flow problems, particularly at the point of integration between the processes under consideration. The procedure proposed in this paper has been implemented and evaluated using both synthetic and real-life logs.

Keywords: Process Mining, Multiple Instances, Merging Logs, End-To-End Process Flow, Abstraction Level.

1. Introduction

Process mining has emerged in the past few years as a means for improving overall business performance through discovery and analysis of the actual business process [2]. Process mining allows business processes to be visually mapped from inception through execution to termination, thereby facilitating the identification of the process problems and their causes [1]. This is done based on an event log, depicting all the events recorded during process execution;

such logs are largely available in information systems that support business processes. The work described in this paper has emerged from the need to identify flow and performance problems that can only be detected when a full, end-to-end business process is viewed through its different abstraction levels (i.e., main and sub- processes). For process mining analysis, a log that captures the full process is needed. However, often in organizational settings different parts of processes are managed by different information systems, each generating its own log. Our primary concern, then, is the need to connect logs recorded in different systems that are related to a single, full, end-to-end business process in a manner that would reflect different abstraction (or granularity) levels. As our work progressed it soon became evident that three additional issues required our attention: (a) the complexity of relationships between the different process parts; (b) the possible lack of equivalence in identifying log items that pertain to a single occurrence of the full, end-to-end business process; and (c) the burden that may be caused by irrelevant data in log entries. This paper thus delineates a solution for merging log files of processes distributed over different systems in an automated manner. The proposed solution reflects different granularity levels and is applicable to all possible types of inter-log relationships, matching log items that are not necessarily equivalent in their identification method, and which may be confusing due to additional data that is irrelevant for the merging (typically as free text). Challenges presented by the existence of disparate logs must be overcome and a single, unified log produced from which it is possible to trace the entire end-to-end business process at its different abstraction levels before performance analysis be conducted and flow anomalies be detected. Achieving these processes was the secondary goal of our work.

In the next section, we describe the different granularity levels which we delineate in detail in an example. In the third section, we explain the premises for our merging approach, addressing both our major and secondary concerns while in the fourth section we evaluate the effectiveness of this approach. The fifth section is devoted to related works found in the literature while the last two sections are devoted to a brief discussion and conclusion.

2. Combining different granularity levels

An obstacle to the effectiveness of process mining is that operations are carried out through different systems and recorded in disparate log files so that they cannot be mined in an integrative manner. To date, existing approaches [15] propose automated techniques that facilitate the reconstruction of the full, end-to-end process based on a single (merged) log file representing events recorded in both the instigating (main) process and the responding (sub) processes. However, these techniques are limited in scope to only one granularity level view whereas at least two granularity levels are relevant in such complex processes: a high level

representing the view of the main process and a low level representing the view of the subprocesses. Certain problems in the business process can only be detected in a combined view of the two granularity levels. In this section, we explore which problems require the combined view of the different granularity levels. We focus on two issues that are commonly addressed when investigating business processes: business performance and flow analysis. In order to discuss these, we first explain granularity levels, show an example to which we refer throughout this work, and discuss relationships between the process and the recorded events.

2.1. Understanding granularity levels

Processes can be viewed at a variety of granularity levels, specifically essential in processes that handle composite objects, so they can be traced at the composite level or at the level of its components. This need is amplified where several parties are involved in different parts of a single business process. For example, several departments order equipment from the company's warehouse, and there is some overlap between ordered equipment among the different departments. At the warehouse, items are picked collectively and then distributed individually to respective departments. This process can be viewed from two different perspectives: one concerning the order (composite object), placed by each department with all of its individual items, and the other relating to the flow of each single item (component) ordered by a department. We refer to the first perspective as the high level *case view* and to the second as the low level *instance view*.

It is possible to model the business process based on process mining from three perspectives: that of the case view (high level, emphasizing the main process), that of the instance view (low level, emphasizing the subprocess), and that of a combined view showing relationships between the two. When considering a full, end-to-end process involving multiple occurrences of a subprocess, a coarse-granularity view of the end-to-end flow at the case level provides limited analysis capabilities of problems that occur at the instance level and vice versa. In addition, some problems can only be detected by considering both views simultaneously, as we show below. To the best of our knowledge, to date, there are no studies that consider both high and low granularity levels of the end-to-end process flow together.

2.2. Motivating Example

To showcase the full extent of our approach, throughout this paper we use a running example based on a plausible scenario. The running example concerns a purchase ordering process, where different departments directly order office supplies through an ordering system. The

warehouse receives the deliveries from the suppliers with aggregated amounts of goods ordered by all departments, registers them, and then distributes them to the original ordering departments. We consider the ordering process as the main process and the delivery handling as a subprocess. Table 1 shows a simplified log of the main process while Table 2 shows a simplified log of the warehouse subprocess. The double line in each table serves to separate entries pertaining to a single "case" (referring to orders in Table 1 and deliveries in Table 2).

Table 1. Simplified log of the main process of ordering goods*

Order ID	Timestamp	User	Activity	Item No.	Department
3001	02/02/14 10:12	Ilana	Open order	1234 1235	Dept. 89
3001	02/02/14 10:13	Tsvi	Approve order	1234 1235	Dept. 89
3001	02/02/14 13:16	Ilana	Check status	1234 1235	Dept. 89
3001	03/02/14 16:18	Ilana	Receive item	1234	Dept. 89
3001	04/02/14 16:30	Ilana	Receive item	1235	Dept. 89
3001	04/02/14 16:34	Ilana	Receive item	1237	Dept. 89
3001	04/02/14 16:36	Ilana	Close order	1234 1235 1237	Dept. 89
3002	02/02/14 10:30	Sigal	Open order	1234 1236	Dept. 79
3002	02/02/14 10:31	Rachel	Approve order	1234 1236	Dept. 79
3002	02/02/14 15:31	Sigal	Check status	1234 1236	Dept. 79
3002	04/02/14 16:19	Sigal	Receive item	1234	Dept. 79
3002	04/02/14 16:35	Sigal	Receive item	1236	Dept. 79
3002	04/02/14 16:36	Sigal	Close order	1234 1236	Dept. 79

* Double line separates Order IDs

Table 2. Simplified log of the subprocess of delivery handling*

Delivery ID	Timestamp	User	Activity	Item No.	Department
5001	03/02/14 11:45	Mosh	Receive order	1234 1237	
5001	03/02/14 12:15	Mosh	Send to Dept.	1234 1237	Dept. 89
5001	03/02/14 12:17	Mosh	Send to Dept.	1234 1237	Dept. 79
5002	04/02/14 15:46	Mosh	Receive order	1235 1236	
5002	04/02/14 15:48	Mosh	Send to Dept.	1235	Dept. 89
5002	04/02/14 15:48	Mosh	Send to Dept.	1236	Dept. 79

* Double line separates Delivery IDs

As illustrated in Figure 1, both processes employ multiple instance procedures, where the lower-level instances in both processes refer to ordered items. However, the grouping of items to cases is different for the main and the subprocess. In the main process, the grouping is by

order (serving as case-ID in the log), and in the subprocess the grouping is by delivery (case-ID), where items ordered by different departments are supplied together. Furthermore, there is no explicit connection between the Case IDs of the two logs. As a result, multiple cases of the main process may correspond to multiple cases of the subprocess. To enable mining of the end-to-end process, the two logs must be merged in a manner that will overcome this difference in case-IDs, thus a choice must be made between two possible entities that would serve as the unique identifier for events in the unified log.

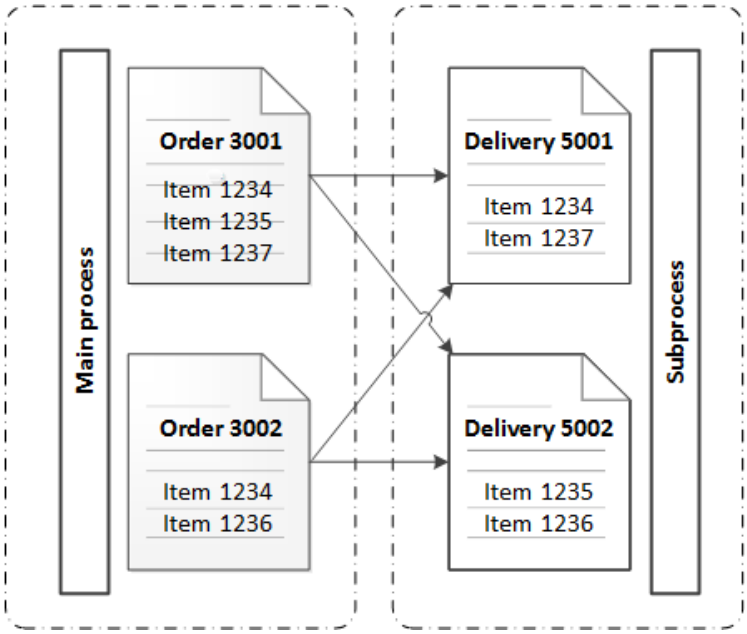
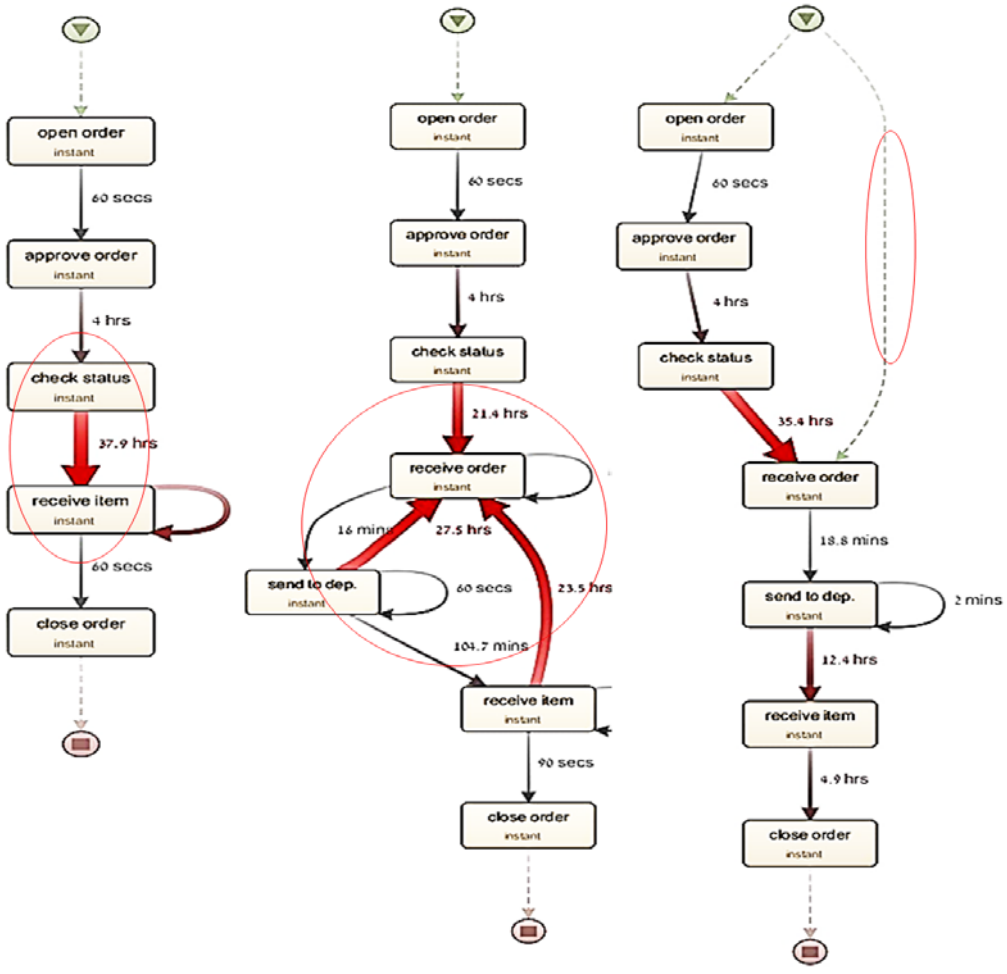


Figure 1. Illustration of many-to-many relationships between a main process and a subprocess of the running example

Consider two possible views of the process: one concerning the order (including all its items) and the other relating to the flow of each single item ordered by a department. We refer to the first one as the case view and to the second as the instance view. Should we support the case perspective when merging the logs, we would be finding for each case in the log of the main process (main log) its related cases in the log of the subprocess (*sub log*). In contrast, for a merged log supporting the instance perspective, each case of the sub log would be related to corresponding cases of the main log. It would be natural to assume that there is no substantial difference between the views. Yet, following each view solely might result in loss of information. In our example, item “1237” is not included in any of the orders. Following the end-to-end process with the case view would not reveal this problem. Hence, we claim that both views are imperative when analyzing the full, end-to-end process flow.

As an illustration, we manually generated an end-to-end case view and instance view of our running example, to enable the identification of different process flow problems. To

demonstrate this need, Figure 2 presents three mined models of the running example, using Fluxicon Disco (https://fluxicon.com/disco/) based on the manually merged logs.



2(a) **2(b)** **2(c)**

Figure 2. Illustration of mined models of (a) the main process (b) the end-to-end case view (c) the end-to-end instance view. Flow times are indicated by the thickness and color of the arrows, which are also annotated by the respective average time.

Depicting the *main process*, Figure 2 (a) reveals one flow (from “Check status” to “Receive item”), in which the average completion time is much longer than other flow phases in the process. However, it is impossible to identify where the delay occurs since no information about the subprocess is incorporated in the model.

More information is available in Figure 2 (b), where the hindrance is found to comprise two different delays: one between the request and the delivery process and the other within the delivery process. The second delay reflects the time that elapsed between different deliveries related to the same order; after an item was delivered, the process halted until the next item was received. These delays can be seen only at the case level of the merged log files.

Last, the instance view in Figure 2 (c), which is based on a merged view of both case and instance perspectives, reveals two interesting facts. First, it shows that the main delay for an ordered item occurs between the ordering and the delivery processes. This delay is inherent and expected for purchasing processes, and no other unexpected delays are observed. Second, it reveals that one of the logged instances started abnormally (an item was delivered and not ordered). This could not be detected with the case view.

In summary, both the case view and instance view in the merged log file, as demonstrated in Figure 2, are useful for revealing information about the end-to-end flow, not visible with the separate logs. Both views are needed since they emphasize different aspects, and each can reveal aspects that are not addressed by the other.

2.3. Establishing relationships between the processes

Our merging approach aims to merge two or more disparate event logs recording events of the main process and its subprocess(es). While our approach is applicable to numerous subprocesses, for the sake of simplicity, we focus on a single subprocess recorded in a single log file. In general, we consider a process as a partially ordered set of activities intended to achieve a goal [31], where each single occurrence of the process is termed a *Case*. Employing general definitions of an event log, a trace, and an event, commonly found in process mining literature [1] we define:

Definition 1 (event log, trace): An *event log***,** where each entry represents an event ***e***, is a multiset of *traces*; a trace is a sequence of events related to the same *Case ID*;

An event may consist of various *attributes*. For our suggested approach, it is required that each event will consist of the following attributes:

- #case id(***e***) - refers to the unique identifier of an entity of which the event relate, typically the entity whose life cycle is managed by the process
- #activity(***e***) - refers to the name of the activity the event relates to
- #timestamp(***e***) - refers to the timestamp when the event occurred
- #attribute values(***e***) – additional recorded attributes (e.g., resource) within an event

Addressing main and subprocesses, without a loss of generality we assume the former triggers the latter.

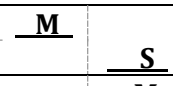
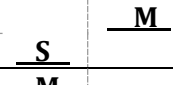
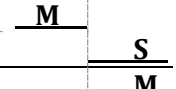
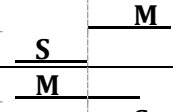
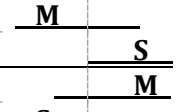
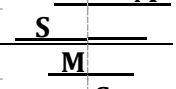
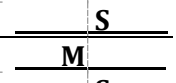
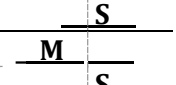
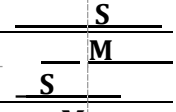
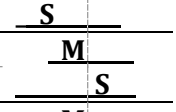
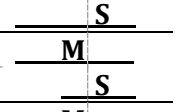
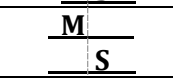
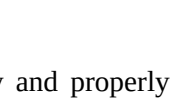
Definition 2 (Main process, subprocess): A triggering process which invokes other processes is the *main process* ***M***, while an invoked process is the *subprocess* ***S***.

We consider two types of relationships between the main process and its subprocess: (a) structural relationships, and (b) temporal relationships.

Structural relationships. Both the main process and its subprocess comprise several acts recorded in the logs. Four types of relationships are possible between traces in the logs: one-to-one, one-to-many, many-to-one, and many-to-many where the first quantifier represents the main process and the second – the subprocess. Simple relationships between process parts (one-to-one or one-to-many) are fairly easy to handle, as the connection between the main and the subprocess is typically explicit in the log or easily established. Real-life processes, however, often reveal complex inter-process relations (e.g., many-to-one or many-to-many), for example, repeatedly performed procedures of multiple instances of diverse entities, obscuring the connection between main and subprocess traces. When complex relationships exist, typically different granularity levels are of relevance. In order to join the cases of different logs correctly, some reference or a common property is needed. If the ID of the cases in the main process exists in the sub log as a foreign key or a descriptive attribute, joining the logs is straightforward. Similarly, finding a match between log cases when the logs have a one-to-one relationship and both have the same case ID is immediate. For logs whose case IDs are not identical, we match cases based on similarity of attribute values and appropriate temporal relations. This approach differs from approaches that discover foreign keys in disparate datasets (e.g., [53]) or those that discover attributes of similar values (e.g., [19]) as it considers dissimilarities in free text attributes and make no a priori assumptions regarding links among the two logs.

Temporal relationships. Based on Allen's interval algebra [7], thirteen types of temporal relationships may exist between any two protracted events in terms of their interface and overlap. These are described in Table 3, where M denotes the main process and S denotes the subprocess. Following Definition 2, we consider a fully and properly executed end-to-end process as one in which the flow between the main process and the subprocess follows this order: (a) the main process starts first, instigating the subprocess; (b) the subprocess overlaps main process, starting any time between the beginning and the end of the main process; and (c) the main process ends any time after the subprocess had started. The subprocess can end any time after it had started, independently of the ending point of the main process. The rationale is that the main process should be instigating the subprocess (hence it starts first) and the subprocess should provide feedback to the main process, hence it must have at least some overlap with the main process. Thus, only temporal scenarios 3, 5, 8, and 12 in Table 3 meet our conditions for a fully and properly executed end-to-end process whereas in others, (e.g., temporal scenario No.1) no feedback is returned from the subproces to the main process or the order of processes as defined above is not kept (e.g., temporal scenario No. 2).

Table 3. Allen's temporal relations applied to case matching*

Temporal scenario No.	Scenario from the perspective of the main process and the subprocess	Time relation[1]	Illustration
1	M before S	M < S	
	S after M		
2	S before M	S < M	
	M after S		
3**	M meets S	M m S	
	S is met by M		
4	S meets M	S m M	
	M is met by S		
5**	M overlaps S	M o S	
	S is overlapped by M		
6	S overlaps M	S o M	
	M is overlapped by S		
7	M during S	M d S	
	S contains M		
8**	S during M	S d M	
	M contains S		
9	M starts S	M s S	
	S is started by M		
10	S starts M	S s M	
	M is started by S		
11	M finishes S	M f S	
	S is finished by M		
12**	S finishes M	S f M	
	M is finished by S		
13	M equals S	M = S	
	S equals M		

* M denotes the main process and S denotes the subprocess

** Only temporal scenarios 3, 5, 8, and 12 meet our definition of a fully and properly executed end-to-end process

3. Log-merging and generation of case and instance views

As mentioned earlier, the criterion defining the scope of our solution is that the problems we seek to detect lie in the interface of two granularity levels. Our solution offers a way to merge disparate log files of the main process and its subprocess(es) that maintains all data of each log file while identifying the process to which each data entry pertains, considering the relationships discussed above. We suggest an automated technique to materialize a merged log supporting two granularity levels. First, a merged log with a focus on the case view, and second, a merged log that reveals the end-to-end instance view. The first step in achieving this goal is a preparatory step of "flattening" the sub log, required in order to support the view of multiple granularity levels. After this preparatory step, we merge the log files based on specific matching

considerations, generating a log that supports a case view. We can then generate the instance view of the merged log file that will allow us to analyze the full end-to-end process and identify different performance or flow problems. When presenting the approach, we focus on the most complex structural inter-log relationship, that of many-to-many. At the end of this section, we discuss the steps necessary when addressing logs where less complex relationships are concerned.

3.1. Flattening the sub log

To meet the complexity presented by many-to-many relations between event logs, before the logs are merged the sub log needs to be flattened, that is, we wish to dissolve clustered instance relationships in the sub log. We do this by generating a unique trace for each instance with its own attributes. The resulting traces will contribute to accomplishing the purpose of a comprehensive end-to-end process flow. The flattened subprocess will enable a more accurate matching between the main process and the corresponding subprocess, and reduce the chances of faulty matches. For example, where items to be supplied are distributed into several deliveries, and each delivery corresponds to many orders, the resulting match between an order and a delivery might entail a false attachment of items to an order. Matching is easier when we break the deliveries to single items.

Hence, flattening serves as a preparatory phase preceding the merging procedure, to remove ambiguity of relations and identify unrelated and possibly faulty instances (e.g., item 1237 in delivery 5002, not included in the order), as presented in Figure 3(a). The result, illustrated in Figure 3(b), is a reconstruction of the sub log, where each case of the sublog relates to a single instance, simplifying the merging between the main process and the corresponding subprocesses. Note, the relationship between the log can still remain a many-to-many one (Figure 3).

We dissolve clustered instance relationships based on an attribute that can uniquely identify these instances. Thus, preliminary domain knowledge that can typically be obtained from domain experts is required for the indication of the relevant attribute. To simplify, we consider a structured attribute rather than an unstructured free text one, however, additional preprocessing is possible if such structured attribute is not available.

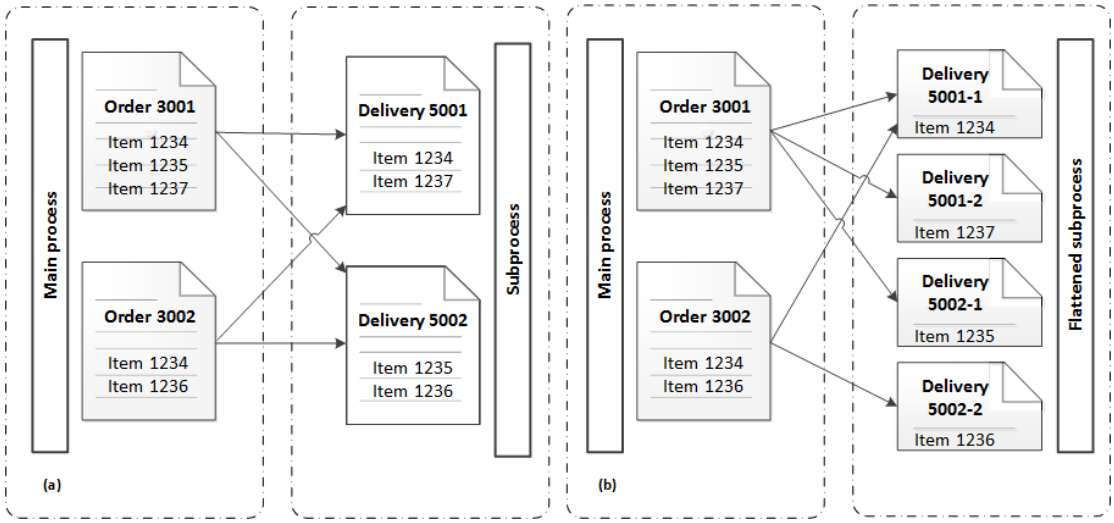


Figure 3. Dissolving clustered instance relationships within subprocess where (a) a case consists of clustered instances (b) a case consist of an individual instance.

Deconstruction Algorithm. With a known attribute that represents the instances addressed by the subprocess, this algorithm separates the grouped instances in the subprocess into individual traces. This is done in two steps. First, the algorithm extracts the list of instances for every case of the subprocess, that is, for every trace in the sub log. As every event in the log has an "instance" attribute, this corresponds to gathering all unique instance values for every event in that trace. Second, it iterates over all traces in the sub log and all instances extracted for that trace. For every combination of trace and instance it generates a new trace containing all events from that trace that are relevant to that instance. Every new trace is assigned a new case ID using a counter. The result is an automatically generated flattened sub log, where a single trace (marked by a single case ID) relates to each single instance (see Listing 1 in the Appendix). Note that the assumed availability of a structured attribute that marks the instances of the subprocess does not imply a structured connection between the logs. This connection can still be concealed in free text attributes whether the sub log is flattened or not.

Considering our running example, Table 4 presents the new flattened sub log (the equivalent of Table 2) where every instance now has a single trace. The original case IDs are kept as an attribute of the new events. The flattening process produces duplication in certain events (e.g., receive order by Mosh at 03/02/14 11:45); these are distributed over to separate instances later on in the procedure.

Table 4. Simplified log of the flattened new subprocess*

New Trace ID	Original Case ID	Timestamp	User	Activity	Item No.	Department
5001-1	5001	03/02/14 11:45	Mosh	Receive order	1234	
5001-1	5001	03/02/14 12:15	Mosh	Send to dept.	1234	Dept. 89
5001-1	5001	03/02/14 12:17	Mosh	Send to dept.	1234	Dept. 79
5001-2	5001	03/02/14 11:45	Mosh	Receive order	1237	
5001-2	5001	03/02/14 12:15	Mosh	Send to dept.	1237	Dept. 89
5001-2	5001	03/02/14 12:17	Mosh	Send to dept.	1237	Dept. 79
5002-1	5002	04/02/14 15:46	Mosh	Receive order	1235	
5002-1	5002	04/02/14 15:48	Mosh	Send to dept.	1235	Dept. 89
5002-2	5002	04/02/14 15:46	Mosh	Receive order	1236	
5002-2	5002	04/02/14 15:48	Mosh	Send to dept.	1236	Dept. 79

* Double lines separate New Trace IDs

3.2. Merging the logs: Main log and flattened sub log

We first describe our matching approach and then our merging procedure.

3.2.1. Matching main- and sub- process events

Typical log-merging methods are developed on the basis of limiting assumptions that hinder their applicability to real-life situations. It is thus imperative to find a new means by which to merge the main log with its sub log. The different considerations we discussed earlier are used for the purpose of matching process events in the two logs. In particular, the matching of cases is based on temporal relations and text similarity (of both structured and unstructured attributes).

Temporal relations. For establishing appropriate temporal relations, let us consider m_i and s_i, cases of the main and the subprocess respectively, both belonging to one execution of the end-to-end flow. We denote the time of the first event in m_i as $t^{start}[m_i]$ and the time of its last event as $t^{end}[m_i]$ (similar notations are used for s). Based on the discussion in Section 2.3, we require: $t^{start}[m_i] < t^{start}[s_i]$ and $t^{start}[s_i] \leq t^{end}[m_i]$ as a condition for m_i and s_i to be considered one trace of the end-to-end process.

Text similarity. Without making any assumption about the form of the connection between the logs (i.e., explicitly given as a structured attribute value or implicitly as part of a free text), we expect such connection to exist in the attribute values of events in the log. Where m_i and s_i are cases of the main and the subprocess respectively, both belonging to one execution of the

end-to-end flow, we expect the number of common words in their attribute values to be higher than the number of common words in the attribute values of m_i and s_j, cases that belong to different end-to-end executions. Following this premise, we calculate similarity scores for pairs of cases from the main and the sub log, based on the text-mining technique *Term Frequency-Inverse Document Frequency* (tf-idf) [44]. tf-idf relates to the frequency of a word in a given text (case attributes) as compared to its appearance in other texts (other cases). This filters out words that are common to all cases but do not serve as an efficient indicator of similarity between two specific cases (e.g., stop words such as "the"). We then extract remaining words to a *bag of words* for each case and calculate a similarity score of the two cases as the count of unique words that are common to them. For establishing a match between cases in the main and the sub log, we consider the highest scoring pairs, and disregard others. This makes the selection natural, considering the scores relatively, without a need for artificially setting thresholds for similarity scores.

To illustrate this using our running example, the bag of unique words of the first case in the main log would be {3001, Ilana, Tsvi, open, **order**, approve, check, status, **receive**, item, close, **1234**, 1235, **1237**, **dept**., **89**} and the bag of unique words of the first case in the sub log would be {5001, Mosh, **receive**, **order**, send, **1234**, **1237**, **dept**., 79, **89**}. The highlighted words are counted for the similarity score, which hence amounts to 6.

3.2.2. Merging procedure

Similar to our flattening approach, our merging approach, illustrated in Figure 4, generates a new case ID for cases in the new unified log so that each new case ID corresponds to a case of the main process. Corresponding cases of the sub log are augmented, duplicated when this is needed (e.g., case 5001-01 of the subprocess) in order to reflect the relations with the cases of the main process.

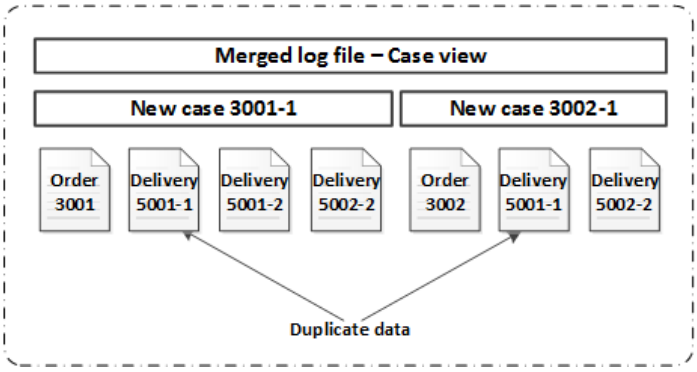


Figure 4. An illustration of relationships among cases and instances in the merged log file

The *Log-merging Algorithm*, then, takes the main log and the flattened sub log and generates a unified log. The algorithm, as portrayed in Listing 2 in the Appendix, generates a new case-id for each case of the main process, and calculates text similarity scores for the attributes of every case combination that meets the temporal relations requirements. For every case of the main process (and corresponding New Case ID), it selects the subprocess cases whose score is maximal and above zero and merges the logs accordingly.

Table 5 presents the merged log of one new, merged, case as extracted from our running example. While being related to a New Trace ID, the events maintain the record of the reference log where they originated and the respective original case ID.

Table 5. Simplified merged log for case 3001 in our running example and related flattened sub log cases 5001-1, 5001-2 and 5002-1

New Trace ID	**Case ID**	**Timestamp**	**User**	**Activity**	**Item No.**	**Dept.**
3001-1	3001	2/02/14 10:12	Ilana	Open order	1234,1235	89
3001-1	3001	2/2/2014 10:13	Tsvi	Approve order	1234,1235	89
3001-1	3001	2/2/2014 13:16	Ilana	Check status	1234,1235	89
3001-1	5001-2	3/2/2014 11:45	Mosh	Receive order	1237	
3001-1	5001-1	3/2/2014 11:45	Mosh	Receive order	1234	
3001-1	5001-2	3/2/2014 12:15	Mosh	Send to dept.	1237	89
3001-1	5001-1	3/2/2014 12:15	Mosh	Send to dept.	1234	89
3001-1	5001-2	3/2/2014 12:17	Mosh	Send to dept.	1237	79
3001-1	5001-1	3/2/2014 12:17	Mosh	Send to dept.	1234	79
3001-1	3001	3/2/2014 16:18	Ilana	Receive item	1234	89
3001-1	5002-1	4/2/2014 15:46	Mosh	Receive order	1235	
3001-1	5002-1	4/2/2014 15:48	Mosh	Send to dept.	1235	89
3001-1	3001	4/2/2014 16:30	Ilana	Receive item	1235	89
3001-1	3001	4/2/2014 16:34	Ilana	Receive item	1237	89
3001-1	3001	4/2/2014 16:36	Ilana	Close order	1234,1235,1237	89

3.3. Generating the Instance View

This section introduces a method of creation of the end-to-end instance view, created for each of the instances within a case. The proposed algorithm is applied to the unified log in order to unfold clustered instances into self-contained cases. Figure 5 illustrates the instance view, based on the merged log file, with a new case ID for each instances.

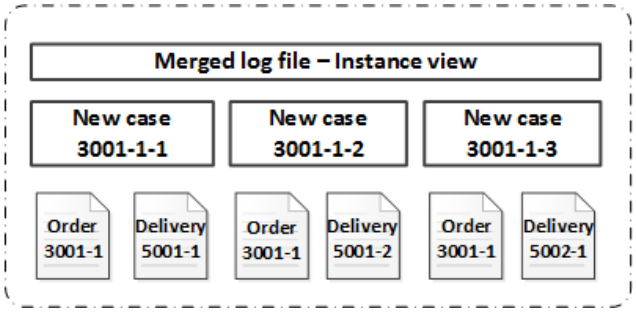


Figure 5. An illustration of the instance view in a merged log

Conversion Algorithm. The unfolding of clustered instances into self-contained cases is logically similar to the flattening process described earlier. Here, the addressed log is the unified log obtained by the merging procedure. A necessary basic assumption is that a known attribute in the log holds information that can be used for identifying the instances whose life-cycle is to be captured in the process model (as the "instance view"). We assume this information can be available based on domain knowledge, and that it appears as a structured value (and not embedded in free text). Again we note that this structured attribute is not necessarily the information that makes the connection between the two logs. In our running example, the items serve as the instances whereas the connection made for merging the logs includes a combination of the item and the department. The algorithm is portrayed in Listing 3 in the Appendix. Table 6 shows an end-to-end instance view for item 1234 of our running example. For each of the instances within case 3001-1 that was produced by our flattening procedure, an individual instance view is generated, creating to a new unique case ID (3001-1-1).

Table 6. Simplified log of an end-to-end instance view of a single instance (item 1234), based upon the merged log.

Instance Case ID	Merged Trace ID	Order	Timestamp	User	Activity	Item No.	Dept.
3001-1-1	3001-1	3001	2/02/14 10:12	Ilana	Open order	1234	89
3001-1-1	3001-1	3001	2/2/2014 10:13	Tsvi	Approve order	1234	89
3001-1-1	3001-1	3001	2/2/2014 13:16	Ilana	Check status	1234	89
3001-1-1	3001-1	5001-1	3/2/2014 11:45	Mosh	Receive item	1234	
3001-1-1	3001-1	5001-1	3/2/2014 12:15	Mosh	Send to dept.	1234	89
3001-1-1	3001-1	5001-1	3/2/2014 12:17	Mosh	Send to dept.	1234	79
3001-1-1	3001-1	3001	3/2/2014 16:18	Ilana	Receive item	1234	89
3001-1-1	3001-1	3001	4/2/2014 16:36	Ilana	Close order	1234	89

3.4. Performance and Flow Analyses

Performance and flow analyses of business processes are commonly performed. Here we show scenarios for which the double granularity view is essential for these analyses.

3.4.1. Performance analysis

Performance analysis concerns business metrics such as flow times or costs. For simplicity, we continue our example focusing only on flow times, both total and average, of the full process or its parts, aiming to identify bottlenecks in the overall process. That is, we seek to identify those segments of the process where delays are incurred as illustrated in **Figure 2**. In fact, delays occurring between cases (see **Figure 2** above) cannot be detected by analyzing just one view. They can, however, be detected through the combined analyses of both views since each view emphasizes different aspects of the process. A combination of both analyses presents a clear and full view of the relevant information from which it is possible to accurately pinpoint the exact location of the bottleneck. Table 7 summarizes the utility of each view given the location (process-wise) of the

Table 7. The utility of each view given the location (process-wise) of bottlenecks[1]

	View		
Bottleneck location	**Instance**	**Case**	**Explanation**
Main process	✓	✓	Details visible in both views
Sub-process	✓	Partially	Aggregated delays shown at case view, details at instance view
Between sub-processes	×	✓	Instance view captures each sub-process separately; delays between sub-processes seen only at case view

[1] ✓ indicates that a bottleneck can be identified by this view for this process; × indicates that a bottleneck cannot be identified; partially – when bottleneck identification points only to part of the problem.

3.4.2. Flow analysis

Flow analysis is used to identify irregular flow patterns that potentially attest to problems in the integration of the main and subprocesses. Such problems are typically manifested as an incomplete end-to-end flow in a process. Examples include cases where a main process starts but is not continued by a subprocess, or when a subprocess is instigated without being triggered by the main process. To systematically detect such cases, we utilize the merged logs at the

instance view, and consider the three expected phases of an end-to-end process, namely, main process begins, subprocess executed, and main process ends. Any divergence from this order is considered a flow anomaly. Table 8 lists all possible flow combinations of main and subprocess according to their expected phases. Out of eight possible flow combinations listed in Table 8, five (flow scenarios 2 to 7) are considered anomalies due to deviance in one or more of the flow phases described above.

Table 8. Possible end-to-end process flow combinations between main and subprocesses*****

Flow scenario No.	**Main process begins**	**Main process ends**	**Subprocess executed**	**Implication**
1	✓	✓	✓	Correct flow
2			×	Flow anomaly: missing sub-process
3		×	✓	Flow anomaly: abnormal ending
4			×	Flow anomaly: missing sub-process and abnormal ending
5	×	✓	✓	Flow anomaly: abnormal beginning
6			×	Flow anomaly: missing main process
7		×	✓	Flow anomaly: abnormal beginning and missing main process
8			×	No process executed

* ✓ indicates that this phase of the process is recorded in the logs; × indicates that it is not recorded.

Note that some of the combinations listed in Table 8 reflect business situations that are typical of poor integration scenarios. Scenario 2, for example, can be observed when an order is not delivered and is eventually closed. Similarly, Scenario 6 can stand for the delivery of an item that was not ordered. Furthermore, combinations of such anomaly patterns (if they are observed within a short timeframe), can indicate more complex situations. For example, consider a case where Scenario 4 takes place and is immediately followed by a case exhibiting Scenario 5. Such a scenario may indicate that these cases were indeed one occurrence of the end-to-end process, but some changes in the data (e.g., due to an error) made matching the cases together impossible. Similar conclusions can be drawn from the combination of Scenarios 3 and 7. The difference between the two combinations is in the location of the anomaly: while one combination (Scenarios 4 and 5) indicates an anomaly before or when triggering the subprocess, the second (Scenarios 3 and 7) indicates an anomaly after the subprocess has started.

3.5. Merging logs for other types of structural relationships

So far, we have discussed the most complex structural relationship between event logs, that of many-to-many. For this relationship, we proposed a procedure of three steps (a) flattening the subprocess, (b) merging the main and sub- processes, and (c) generating an instance view. Not all steps, however, are required for the simpler relationships of one-to-one, one-to-many, and many-to-one.

The first step, *flattening of the subprocess log*, is applicable when subprocess cases comprise multiple instances that may be triggered by different cases of the main process. Splitting these instances into self-contained cases of the subprocess, as discussed earlier, in order to avoid faulty matches is only relevant for many-to-many relationships between the logs, where such situations might emerge. The second step, *merging logs of the main process and the subprocess*, is applicable whenever the end-to-end process comprises multiple event logs, generated by different systems that operate different parts of the process. This step is thus relevant for all relationship types discussed here. The third step, *generating an instance view*, is applicable whenever multiple cases of the subprocess are triggered by at least one main process, namely one-to-many and many-to-many relationships. In these situations, following the process at the case level (of the main process) results in the loss of information, rendering it impossible to trace the flow at the subprocess level. The algorithm used in these situations aims to facilitate the discovery of the end-to-end flow for each instance (or subprocess cases). The applicability of each step to the different types of relationships is detailed in Table 9.

Table 9. Procedure steps for relationship types

	Relationship type[1]			
Step	**OTO**	**OTM**	**MTO**	**MTM**
Flattening subprocess				✓
Merging event logs	✓	✓	✓	✓
Generating an instance view		✓		✓

[1] OTO = one-to-one; OTM = one-to-many; MTO = many-to-one; MTM = many-to-many

4. Evaluation

We evaluated our procedure both in experimental conditions using synthetic logs and using real-life logs.[1] For each, we tested both the case and the instance perspectives. The experimental evaluation was used to determine precision (proportion of correctly identified matched cases among all indicated matches) and recall (proportion of correctly matched cases among all positive matches) of our procedure. Evaluating the approach using synthetic logs provides a full control over the experiment, with three important advantages. First, the correct match between cases is known in advance, facilitating an accurate measurement of precision and recall of results. Second, the logs can be generated to ensure a full coverage of all four relationship types and diverse samples of temporal relations between the cases and instances in the end-to-end process logs. Third, it is possible to control the amount of text-related noise (additional irrelevant text) in the attribute values. Using real-life logs, on the other hand, sheds light on the applicability and scalability of the approach.

We implemented our algorithms as two plug-ins in ProM [5], a well-known academic process mining tool that can be downloaded from http://www.promtools.org. The algorithms from Listing 1 and Listing 3 are implemented as the "Split traces" plug-in, while the algorithm from Listing 2 is implemented as the "Merge logs" plug-in. Both plug-ins are available in the "Log" package of ProM 6.6 [5].

4.1. Experimental evaluation using synthetic logs

4.1.1. Method

The experimental evaluation comprised four steps (1) generating noise-free logs, (2) merging the logs, generating both case and instance views (where applicable), (3) adding noise to the logs, and (4) merging the logs, generating both views. Precision and recall were measured for the results of the second and fourth steps.

Generating noise-free logs. Based on our running example, we generated four main process logs, one for each relationship type. Each log comprised 130 or 260 cases according to the relationship's target, "one" or "many" (Table 10), respectively. Each event had the following data attributes: Case ID, time stamp, user, department number, item number, and activity. We then generated corresponding subprocess logs that followed the four possible temporal relations

[1] Logs are available at http://dx.doi.org/10.17632/2hb3p36vf3.1. Note that sensitive information was altered in these files, so any reproduction of the process might slightly vary from our results.

between the main and subprocesses (main process meets or overlaps subprocess, subprocess during or ending with main process; see Table 3 above).

Table 10. Synthetic logs generated for algorithm evaluation

	Number of cases		
Type of relationship	**Main Log**	**Sub Log**	**Unified log (expected number)**
One-to-One	130	130	130
One-to-Many	130	260	260
Many-to-One	260	130	260
Many-to-Many	260	260	520

Generating noise-laden logs. With our end goal in mind, we created noise-laden logs mimicking the structure and context of real-life logs. Building on the noise-free logs, we introduced a free text attribute of 200 words to the main log and three free text attributes with up to five words each to each of the sub logs. We then randomly selected text for these attributes. This selection was repeated twice. First, the text was inserted in all logs by sampling free text attributes of a real-life log from a domain different from the one in the experiment. Second, we randomly replaced words in the text by words related to the process domain (348 words replaced by "delivery" and 466 words replaced by "item" in the main log). We performed the merging for both variations of noisy logs, in order to test whether the alphabet of the noise affect the merge.

4.1.2. Findings

Noise-free logs. Applying the procedure to each of the four log combinations resulted in perfectly unified logs with 100% recall and precision for both end-to-end case view and instance view.

Noise-laden logs. The obtained results were identical for the two variations of noise that were generated (with and without terms relevant for the process domain). These results are presented for the two perspectives: that of the case and that of the instance.

Case view results. Table 11 shows the results of the case view obtained by applying the merging procedure to logs that contain free text. Overall, with high recall (at least 94%) and high precision (at least 89%) it can be said that the merging performs well. The percentage of recall (94-97%) was predominantly higher than that of precision (89-93%), with the exception of the many-to-one relationship (96% for both). Exhibiting similar F-measure values (92-96%),

the merging can be considered as not sensitive to the type of relationship between the logs. Therefore, no relationship is identifiable as an "easier match."

Table 11. Recall and precision of the case view per relationship type[1]

Log relationship[2]	Total cases in main log	True positives (tp; correctly identified)	False positives (fp; incorrectly identified)	False negatives (fn; incorrectly rejected)	Recall	Precision	F-measure
OTO	130	126	16	4	97%	89%	93%
OTM	130	254	28	16	94%	90%	92%
MTO	260	239	11	10	96%	96%	96%
MTM	260	501	35	15	97%	93%	95%

[1] This table represents the two analysis variations where random words were both related and unrelated to the process as the difference between analyses was measured in fractions of the percent.

[2] OTO = one-to-one; OTM = one-to-many; MTO = many-to-one; MTM = many-to-many

Instance view results. Following the explanation in Section 3.5, the instance view was only generated for one-to-many and many-to-many inter-log relationships. The automated conversion of the case view into instance view involves the flattening of the subprocess for the one-to-many and many-to-many relationship types. The reduced number of *false positive* cases for these relationships (precision of 95% and 97% respectively; Table 12) can thus be attributed to this flattening. Merging results for the one-to-one and many-to-one relationships remained unchanged. Overall, the results of the merging procedure are better when involving log flattening for the subprocess.

Table 12. Recall and precision of the instance view following log flattening.

Log relationship[1]	Total cases in main log	True positives (tp; correctly identified)	False positives (fp; incorrectly identified)	False negatives (fn; incorrectly rejected)	Recall	Precision	F-measure
OTM	130	257	14	8	97%	95%	96%
MTM	260	511	17	8	98%	97%	98%

[1] OTO = one-to-one; OTM = one-to-many; MTO = many-to-one; MTM = many-to-many

4.2. Evaluation using real-life logs

While the experimental evaluation using synthetic logs focused on the merge accuracy, a second evaluation aimed to examine the scalability of our merging procedure and its practical implications. This was achieved using two real life logs, one to which both merging and analysis were applied and another in which only merging was performed for the sake of scalability evaluation. The first log, in which the full analysis was performed, did not suffer from any substantial delays and bottlenecks. Hence, our focus here was on scalability and flow analysis. The second log originates in the 2014 BPI challenge (http://www.win.tue.nl/bpi/doku.php?id=2014:challenge), a publically available log, commonly used for evaluation studies. In our context, it was used for the scalability evaluation due to its sheer volume.

4.2.1. Scalability

Our first goal was to test the scalability of our algorithms and the required processing time when realistically handling large logs. We tested it using two real-life logs. The first involves an access permission process (involving one-to-many relationships), where the main process issues permission requests and the subprocess handles the approval procedure, notifying the main process when they are granted. The second originates in the 2014 BPI challenge, involving a service desk support process, where the main process tracks technical support incidents and the subprocess handles requests for change. Table 13 summarizes log properties relevant for our analysis as well as the resulting processing time for both of these test cases. Precision and recall were not sought because the real matches would not be known a priori.

Table 13. Properties of experimental logs

Property	Real-life log 1	BPI Challenge
Business process	Access permissions	Service desk support
Main process	Permission requests	Technical support incidents
Subprocess	Approval procedure	Request for change
Time span	2012 (1 year)	2011-2014
Main process events	26,525	466,738
Subprocess events	44,792	30,276
Main process cases	4,472	46,616
Subprocess cases	13,762	10,193
Main process free text words per trace	3-57	3-14
Subprocess free text words per trace	1-23	9-17
Processing time (result)	92,400 milliseconds	294,000 milliseconds

4.2.2. Identification of anomalies

We followed the analysis described in section 3.4.2 above to seek anomalies that could be indicative of consistency or integration problems in the log that served for the full analysis (real-life log 1 described in Table 13 above). The instance view of the one-to-many real-life log was used for identifying the types of flow anomalies discussed earlier (see Table 8). Anomalies were detected by distinguishing between 1,104 full process traces and 102 partial process traces (Table 14). In addition, we validated our interpretation of these cases as flow anomalies by showing them to a domain expert, who supported our conclusions. Figure 6 delineates the types of flow anomalies as they emerged from the real-life log as single patterns (Table 8) or combinations of patterns, whose distribution in the log is given in Table 14.

***Uninvoked subprocess*.**

Main process closed (Figure 6a). Since no subprocess was invoked, only main process activities are shown. Inquiring the specific cases with the domain expert, we found that in all cases the application for permission had reached the wrong department, then an email was sent to the applicant, and the application was closed. In these cases, there are no corresponding subprocesses.

Main process remained open (Figure 6b). Since no subprocess was invoked, only main process activities are shown. Note that in all these cases the application for access permissions was denied and therefore did not proceed to the corresponding subprocess. However, the main-process remained open.

Modification of data within the main process (Figure 6c). Pairs of related process flows, where one has a normal starting point but an unexpected ending point at the main process, and the other starts right after the point the first one ended. It appears that originally, both partial traces corresponded to the same triggered case of the main process. As confirmed by the domain expert, it could be interpreted as an update of a key data value, due to which the traceability was not kept.

Modification of data within the subprocess (Figure 6d). Pairs of related process flows, where one has an unexpected ending point at the subprocess, and the other starts right after or by repeating the point where the first one ended. It appears that originally, both partial traces corresponded to the same triggered case of the main process. It could be interpreted as an update of a key data value at the subprocess (done while repeating the Approved and Completed activity), due to which the traceability was not kept.

Table 14. Distribution of detected anomalies in real-life log

Flow anomalies	Number of traces
Uninvoked subprocess	49 (15 - closed, 34 – remained open)

Modification of data within the main process	6
Modification of data within the subprocess	47

In summary, it can be concluded that flow anomalies and integration problems between the main and the subprocess can be detected and distinguished by our approach. The experimental evaluation we presented shows that our algorithm yields a precision rate of 89-97% in properly assigning cases to instances while our real-life evaluation shows the algorithm successfully identifies flow anomalies and performs well on a realistic dataset.

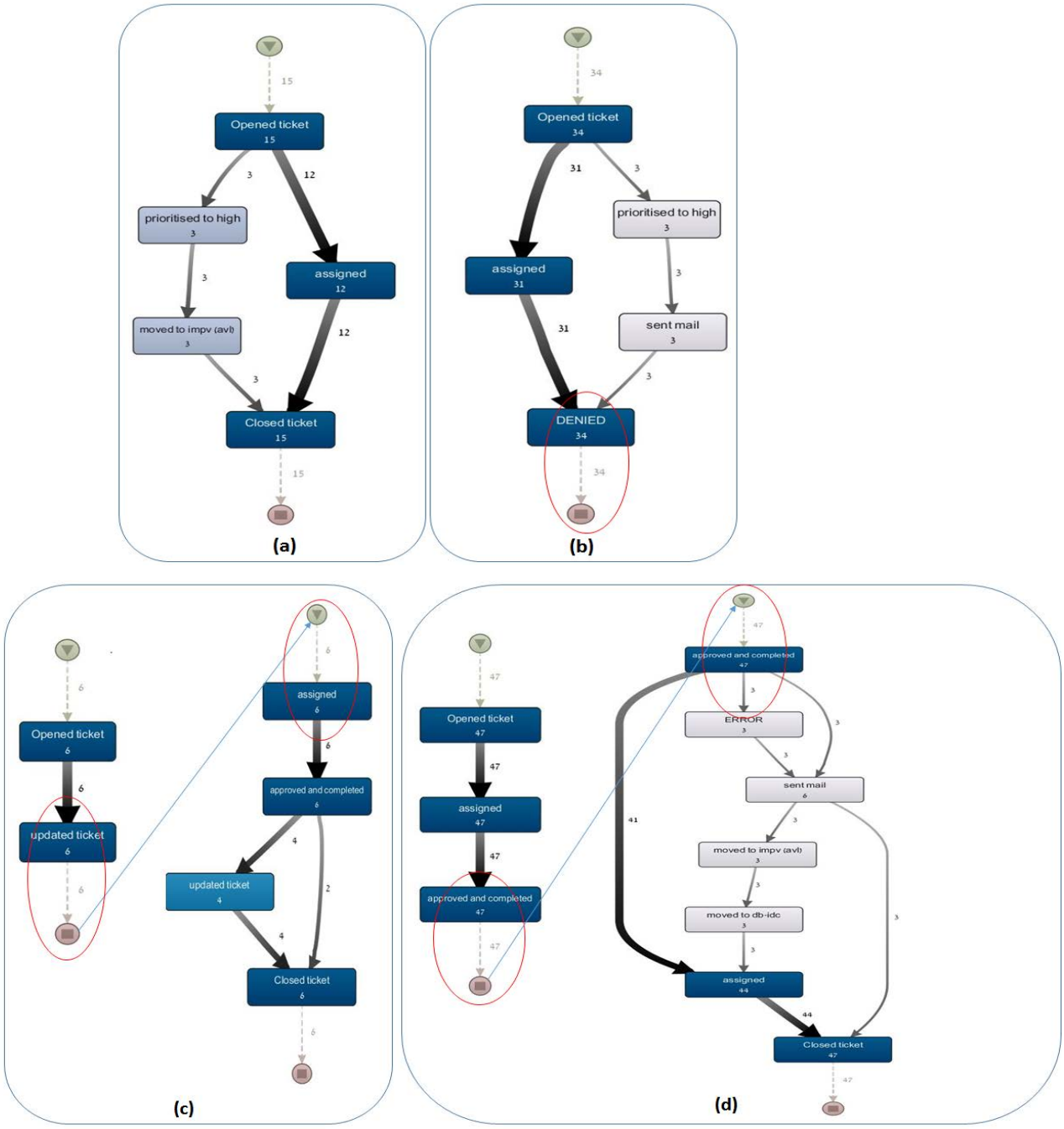


Figure 6. Mining results using filtering abilities to enable the refining of specific flow problems as follows: (a) uninvoked subprocess – main process closed (b) uninvoked subprocess – main process remained open (c) modification of data within the main-process (d) modification of data within the subprocess

5. Related work

The purpose of this work is to facilitate process mining operations for processes that employ more than a single system, where the logs of the involved systems may or may not be similarly

structured or connected. We proposed a method for merging these logs into a unified log and showed that different abstraction levels are required to achieve a full end-to-end flow analysis. In this section, we therefore delineate the current literature from three aspects: (a) pre-processing and merging of logs, (b) handling of different abstraction levels in process mining, and (c) end-to-end flow analysis of related processes.

Pre-processing and Log-merging.

Log-merging begins with the discovery of the relationships between separate datasets. Similar problems have been addressed in the context of database integration. Matching columns and discovering foreign keys in database integration (e.g., [53]), a common approach in this area, is based on the assumption that a unique identifier can be found that is common to all the involved schemas. Other approaches aim to discover correspondence based on similar values (e.g., [19]). Our solution goes beyond these approaches, as it establishes matches when no unique identifier is present or known, considering free text that may hold dissimilar values.

Log merging as a preparatory step to process mining has not been widely addressed in the literature. The challenge in merging logs of disparate systems is matching items that pertain to a single process execution. Approaches have been proposed for matching cases in different logs for the purpose of merging the logs based on a genetic algorithm [13] or on artificial immune system [14]. These have been extended in [15], to a generic log merging approach, where links are sought and configured between various data attributes that mark cases in the logs that are then merged. An underlying assumption of this approach is that each case in the subprocess relates to exactly one case in the main process; it thus does not address the more complex relationships of many-to-one or many-to-many, which are prevalent in real-life scenarios.

Finally, our previous work [43] introduced the matching technique we have discussed earlier, based on text similarity and temporal relations and applicable to all types of inter-log relationships. The current paper complements our previous work by addressing differences in granularity levels and suggesting patterns for flow analysis.

Handling different abstraction levels in process mining

The notion of different abstraction levels has received considerable attention in the literature. Generally, it follows two main streams, each addressing a different problem: that of information overload and that of dealing with varying granularity levels.

Dealing with information overload: This stream focuses on overcoming complexity, avoiding spaghetti models, and providing a meaningful business process model [45]. Two approaches characterize this stream: (a) hierarchical mining models that allow segmenting aligned traces to form representative groups of subprocesses, which reflect the hierarchy of the subprocesses, considering main execution patterns [51][36]; (b) clustering approaches, attempting to cluster events into subprocesses [8][9][26][29][31][37][48]. Clustering approaches emphasize the importance of abstracting event logs as a preprocessing phase,

handling many-to-many relations between activities and events, thus improving the discovery of the process model.

Varying granularity levels. This stream (to which our own work belongs) concerns entities of different granularity levels that may exist in the end-to-end process. Here the problem is not one of presentation or discovery of information but rather of semantically capturing different abstraction levels. These studies focus on process hierarchy, often encountered in connection to an *artifact-centric process approach* [16][34]. Artifact-centric processes, as their name implies, are centered around an artifact, which comprises both data and life-cycle models as Guard-Stage-Milestone (GSM) models [34], which delineate the natural hierarchy of embedded artifacts. Artifacts also enact and trigger other artifacts in a hierarchical manner, often involving multiple instances. The triggered artifacts might be managed separately by different information systems and could be documented in separate logs. Mining the artifact lifecycle is highly relevant to our work due to the possibility of many-to-many relationships between artifacts. In contrast to our approach, under artifact-centric approaches, the case is abandoned as a basis for mining [40][42]. Early research in this stream built on a single log in which all events are recorded. With this log, the effort was to relate each event to a relevant artifact, so artifact-related logs are created, facilitating mining that produced a GSM model of these artifacts. More recent studies attempted to discover artifact lifecycles from multiple logs exhibiting one-to-many relations [50][51]. It has been shown, however, that extracting events from multiple related instances into a single case, also known as *event divergence*, causes invalid models. Thus, proponents of this approach propose to extend the artifact-centric approach, offering a means by which to identify entities and artifacts as well as their interactions from relational data. These techniques require choosing a single level of abstraction, whereas the technique we suggest facilitates the combination of two abstraction levels [51].

Process mining techniques generally require forming event data according to an agreed upon format that defines the log's structure. The eXtensible Event Stream (XES) format [46] is a well-used standard in the area of process mining. It facilitates the management and manipulation of logs [10] thereby supplying traces that refer to the lifecycle of the traced entity (i.e., grouped events per case) [1][28]. However, XES files provide only one view and are not applicable for both high level (e.g., case view) and low level (e.g., instance view) perspectives. To this end, [22] present an approach that allows to store additional information linked to the event log in order to provide comprehensive process details that would enable extracting the data and mining it from two different perspectives: that of the data and that of the process. Their approach supports varying scopes and abstraction levels to the analyzed processes; yet it requires all the data to reside within a single database.

Techniques aimed at identifying business entities and uncovering dependencies between events enable the automated discovery of Business Process Model and Notation (BPMN) for

subprocesses [17], including multiple-instance scenarios [49], and their combination for a full end-to-end process discovery [48] . These approaches, however, require a clear notion of a case and a unique case identifier used among all subprocesses.

The log-merging algorithm proposed in our previous work [43] generated only the case view. It did not properly capture scenarios involving multiple instances as the end-to-end instance view was absent. In our current work, we enable generating a unified end-to-end log at both the case level and the instance level. This, we argue, is a necessity for a comprehensive process analysis as it offers a view of the process flow from different conceptual consolidations. Furthermore, mining aimed at the identification of process flow problems is simplified using our technique, since we maintain the original information of each log and event.

End-to-end flow analysis. Based on a correlation between logs, the identification of process flow problems is hardly discussed in the literature. To date, we are aware of only one existing approach [25]. While this approach addresses conformance checking of processes that comprise various subprocesses (proclets) with separate logs, it requires a-priori knowledge regarding the connection between the subprocesses. Thus, rather than merging logs before mining, the authors mine each subprocess separately, subsequently combining the process models.

In summary, the approach we present here has several advantages. First, as an extension of [43], it enables the merging of event logs that comprise many-to-many relationships and can accommodate free text that accompanies events. Second, using limited domain knowledge, it provides the two abstraction levels of both instance and case. Third, the resulting combination of abstraction levels enables mining that is targeted at identifying end-to-end process flow problems. This renders the hierarchical mining process easier as it bridges over the need of any unique techniques for discovering the information. Last, we provide patterns that, when discovered using process mining techniques, may indicate specific flow problems.

6. Discussion

The current work extends process mining research by addressing real-life complex situations that have not been addressed so far. Our contribution is as follows:

(1) Our log-merging technique addresses current limitations in the literature, namely, the ability to merge logs that reflect complex (one-to-many, many-to-many) inter-log relationships, lacking a single Case ID to connect them, and harnessing the latent power of free text in making the match. We thus offer a means to merge process logs recorded in disparate systems that maintains information from the original logs and thus reveals the full end-to-end process flow; this is an essential step for detecting problems that occur during the interaction between related processes.

(2) Using the complementing combination of both the case view and the instance view – made possible through the combination of our proposed algorithms – we facilitate the identification and diagnosis of specific problems in both simple and complex scenarios, pinpointing the spot in the process where they occur.
(3) Finally, through a systematic flow analysis we were able to reveal types of flow problems that might be detected in the end-to-end process flow and provided patterns that can be detected in a log accordingly.

In this paper, we have shown that the suggested approach performs well on both synthetic and real-life, practical logs. We have shown that the merging of related log files can reveal the full end-to-end process flow. Only then can flow anomalies in the main or the subprocess, as well as between instances of the subprocess be detected. This result is unique to our approach. Such problems cannot be detected when handling each of the logs separately, as is currently commonly practiced in process mining. Thus, an intelligent, comprehensive merging of related log files is an essential step in the identification of problematic issues that occur during the complete process life cycle. The two views, case and instance, resulting from our approach, offer a comprehensive view of the end-to-end process and its anomalies. The case view allows us to detect reasons for delays while the instance view points at flow anomalies, and specifically identification of three situation types: uninvoked subprocess; fragmented main process (when a main process seems to be ended abnormally); and fragmented subprocess (when a subprocess seems to be started abnormally). These flow anomalies emerge through corresponding patterns as indicated by the flow analysis we conducted.

Our approach has several limitations which should be acknowledged. First, corresponding cases are assumed to have a higher number of common words than unrelated cases. In other words, our merging algorithm assumes identical terminology is used in all logs, and does not consider possible differences that may exist between words used in the log text. Tools exist that can assist in overcoming this limitation [6]. Second, we require that relevant domain knowledge would be available a priori, indicating which entity (e.g., order, item) is at the instance level. In some scenarios, more than one entity may be considered as an instance (e.g., in our running example, both an item and a department could be considered an instance). Matching different entities would yield different results, dramatically affecting identified flow problems. Further research is needed to exploit this outcome as an opportunity for further analysis and insights. Third, given the complexity of O(MN), where M is the number of cases in the main log and N is the number of cases in the sub log, the consideration of free text reduces the efficiency of the algorithm, leading to a relatively long execution time due to the iterative calculation of text similarity between the cases. Furthermore, the calculation of text similarity depends on the number of words included in these cases, hence a substantial amount of free text in the log can impede the merge and slow it down. Specifically, let Wm be the maximal number of unique

words in a case in the main log and Ws be the maximal number of unique words in a case in the sub log, then each iteration entails Wm*Ws checks, setting the complexity of the algorithm at O(MNWmWs). Notwithstanding, since the investigation of performance and flow problems takes place only periodically, we do not consider execution time to be a critical factor. Thus, we posit that the algorithm may run over a lengthy or semi-lengthy period of time without causing critical delays. Fourth, we note that not all flow anomalies detected by our algorithm necessarily imply flow problems. Rather, they may indicate unexpected behavior that can be further investigated and explained. For example, it may indicate that data was modified during the process, thus hampering the identification of a match between corresponding traces of the main and subprocess. As this may imply data consistency and reliability problems, we encourage further research to refine flow problem analysis. Alternative interpretations of the discovered patterns are also possible. For example, an unexpected time gap between the main process and the subprocess might hamper their merge (failing to meet the required temporal relations). We would then observe pairs of subsequent cases, one missing a subprocess and the other missing the main process. If such a situation is observed frequently, an investigation of its reasons may be in place. Finally, the proposed merging was evaluated using synthetic logs as well as a single real-life log, which also served for demonstrating the flow analysis capabilities. This might be considered a limitation in terms of external validity. Although the anomaly patterns we explored are generic, it is yet to be determined whether they are applicable to diverse processes in different domains, and whether other types of anomalies exist in other domains.

7. Conclusions

The paper addresses a common reality, where processes include parts managed by different information systems, whose relations may not be simple (e.g., one-to-one). When attempting to analyze the flow of such processes, to use process mining techniques a single, unified log (of all the involved systems) is needed. Our work addresses this need and supports flow analysis by merging the respective logs, providing complementary case and instance views, and patterns for detecting flow anomalies in the end-to-end process. The main contributions of our work are three: (1) we have shown that both the case and instance views of the end-to-end process flow are required in order to truly understand the relationships between the merged log; (2) we offer algorithms that can unify event logs of different relationship types with high precision and recall rates even in the presence of free text and absence of uniform case identifier; (3) the merged log can be used to discover flow problems that cannot be found when handling each log separately.

We argue that large organizations, by their very nature, are expected to have complex, distributed systems that require unique handling capabilities. These are provided by our approach. The analysis we offer provides ample opportunity for business process tweaking and enhancement based on the comprehensive end-to-end flow perspective. This presents a major advantage compared to findings limited to the local views available hitherto as they relate only to individual subprocesses and thereby compromise the possibility for overall data mining. Moreover, our technique allows organizations to identify challenges with coherence, correlation, and synchronization among different parts of a process.

Another unique ability offered by our approach is the delivery of the end-to-end instance view at the lower level of a process in combination with the upper-level case view. This allows organizations to separately analyze the comprehensive process flow and the contained flows where processes comprise multiple items.

Future research calls for two directions. First, additional evaluation is required using real-life logs from different domains. Such information would be particularly beneficial in helping us better understand performance and flexibility properties of the suggested procedure. Second, a specialized visual representation of the end-to-end process based on the merged logs, comprising the case view and instance view together, can be developed. Extant process discovery techniques use visualizations that do not fully support the many-to-many relationship along process hierarchy. Thus, the full benefits associated with process visualization still need to be realized in scenarios involving complexly structured processes.

Acknowledgement: The research was partly supported by the Israel Science Foundation under grant agreement No. 856/13.

References

1. van der Aalst, W. M. P. (2011). Process mining: discovery, conformance and enhancement of business processes. Springer Science & Business Media.
2. van der Aalst, W. M. P., Adriansyah, A., de Medeiros, A. K. A., Arcieri, F., Baier, T., Blickle, T., & Pontieri, L. (2012, January). Process mining manifesto. In Business process management workshops (pp. 169-194). Springer Berlin Heidelberg.
3. van der Aalst, W. M. P., van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M. Workflow Mining: A Survey of Issues and Approaches. Data & Knowledge Engineering. 47, 237–267 (2003).
4. van der Aalst, W. M. P., Weijters, A.J.M.M. Process Mining: A Research Agenda. Computers in Industry. 53, 231–244 (2004).
5. van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H. M. W., Weijters, A. J. M. M., & van der Aalst, W. M. P. (2005, June). The ProM framework: A new era in process mining tool support. In *International Conference on Application and Theory of Petri Nets* (pp. 444-454). Springer Berlin Heidelberg.
6. Aggarwal, C. C., & Zhai, C. (2012). Mining text data. Springer Science & Business Media.
7. Allen, J. F. (1983). [a]Maintaining Knowledge about Temporal Intervals, Communications of the ACM, 26(11), 832-843, (1983).

8. Baier, T., & Mendling, J. Bridging abstraction layers in process mining by automated matching of events and activities. In Business Process Management (pp. 17-32). Springer Berlin Heidelberg (2013).
9. Bose, R. J. C., & van der Aalst, W. M. P. (2009). Abstractions in process mining: A taxonomy of patterns. In Business Process Management (pp. 159-175). Springer Berlin Heidelberg.
10. Buijs, J. C. A. M. (2010). Mapping data sources to XES in a generic way. Maters Thesis.
11. Caron, F., Vanthienen, J., & Baesens, B. (2013). Comprehensive rule-based compliance checking and risk management with process mining. Decision Support Systems, 54(3), 1357-1369.
12. Di Ciccio, C., Mecella, M., & Mendling, J. (2015). The Effect of Noise on Mined Declarative Constraints. In Data-Driven Process Discovery and Analysis (pp. 1-24). Springer Berlin Heidelberg.
13. Claes, J., & Poels, G.: Integrating computer log files for process mining: a genetic algorithm inspired technique. Advanced Information Systems Engineering Workshops. Springer Berlin Heidelberg, (2011).
14. Claes, J., & Poels, G. (2012, January). Merging computer log files for process mining: An artificial immune system technique. In Business Process Management Workshops (pp. 99-110). Springer Berlin Heidelberg.
15. Claes, J., & Poels, G. (2014). Merging event logs for process mining: A rule based merging method and rule suggestion algorithm. *Expert Systems with Applications*, *41*(16), 7291-7306.
16. Cohn, D., & Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 32(3), 3-9 (2009).
17. Conforti, R., Dumas, M., García-Bañuelos, L., La Rosa, M. Beyond tasks and gateways: discovering bpmn models with subprocesses, boundary events and activity markers. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 101–117. Springer, Heidelberg (2014).
18. Cook, J. E., Z. Du, C. Liu, and A. L. Wolf. Discovering models of behavior for concurrent workflows. Computers in Industry, 53(3):297-319, 2004.
19. Dasu, T., Johnson, T., Muthukrishnan, S., and Shkapenyuk, V. 2002. Mining database structure; or, how to build a data quality browser. In Proceedings of the 2002 ACM SIGMOD international conference on Management of data
20. Dijkman, R.M., Dumas, M., van Dongen, B. F., Kaarik, R., Mendling, J.: Similarity of Business Process Models: Metrics and Evaluation. Information Systems 36(2), 498–516 (2011).
21. Dumas, M., La Rosa, M., Mendling, J., & Reijers, H. A. (2013). Fundamentals of business process management (pp. I-XXVII). Berlin: Springer.
22. de Murillas, E. G. L., Reijers, H. A., & van der Aalst, W. M. P. (2016, June). Connecting Databases with Process Mining: A Meta Model and Toolset. In *International Workshop on Business Process Modeling, Development and Support* (pp. 231-249). Springer International Publishing.
23. Eid-Sabbagh, R. H., Dijkman, R., & Weske, M. (2012). Business process architecture: use and correctness. In Business Process Management (pp. 65-81). Springer Berlin Heidelberg.
24. Eid-Sabbagh, R. H., & Weske, M. (2013, January). Analyzing business process architectures. In Advanced Information Systems Engineering (pp. 208-223). Springer Berlin Heidelberg.
25. Fahland, D., de Leoni, M., van Dongen, B. F., & van der Aalst, W. M. P.: Conformance checking of interacting processes with overlapping instances. In Business Process Management (pp. 345-361). Springer Berlin Heidelberg (2011).
26. Ferreira, D., Zacarias, M., Malheiros, M., & Ferreira, P.: Approaching process mining with sequence clustering: Experiments and findings. In Business Process Management (pp. 360-374). Springer Berlin Heidelberg (2007).
27. Folino, F., Greco, G., Guzzo, A., & Pontieri, L. (2009, September). Discovering expressive process models from noised log data. In Proceedings of the 2009 international database engineering & applications symposium (pp. 162-172). ACM.
28. G¨unther, C.W.: XES Standard Definition. Fluxicon Process Laboratories (November 2009)
29. Greco, G., Guzzo, A., & Pontieri, L.: Mining hierarchies of models: From abstract views to concrete specifications. In Business Process Management (pp. 32-47). Springer Berlin Heidelberg. (2005).
30. G. Greco, A. Guzzo, L. Pontieri, and D. Sacca. Discovering expressive process models by clustering log traces. IEEE Transactions on Knowledge and Data Engineering, 18(8):1010–1027, 2006.
31. M. Hammer and J Champy, Reengineering the Corporation, Nicholas Brealey Publishing, London (1993).
32. Helal, I. M., Awad, A., & El Bastawissi, A. Runtime Deduction of Case ID for Unlabeled Business Process Execution Events.

33. J. Herbst and D. Karagiannis. Workflow mining with In WoLvE. Computers in Industry, 53(3):245–264, 2004.
34. Hull, R., Damaggio, E., De Masellis, R., Fournier, F., Gupta, M., Heath III, F. T., & Vaculin, R.: Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In Proceedings of the 5th ACM international conference on Distributed event-based system (pp. 51-62). ACM (2011).
35. de Leoni, M., Maggi, F. M., & van der Aalst, W. M. P. (2015). An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. Information Systems, 47, 258-277.
36. Li, J., Bose, R. J. C., & van der Aalst, W. M. P.: Mining context-dependent and interactive business process maps using execution patterns. In Business Process Management Workshops (pp. 109-121). Springer Berlin Heidelberg (2011).
37. Maggi, F. M., Slaats, T., & Reijers, H. A. (2014). The automated discovery of hybrid processes. In Business Process Management (pp. 392-399). Springer International Publishing.
38. Mezebovsky, D., Soffer, P., & Shimshoni, I. (2009). Pattern mining in system logs: opportunities for process improvement. In Enterprise, Business-Process and Information Systems Modeling (pp. 162-173). Springer Berlin Heidelberg.
39. Moghnieh, A., & Blat, J.: The potential of Recall and Precision as interface design parameters for information retrieval systems situated in everyday environments (2011).
40. Nigam, A., & Caswell, N. S.: Business artifacts: An approach to operational specification. IBM Systems Journal, 42(3), 428-445, (2003).
41. P´erez-Castillo, R., Weber, B., de Guzman, I.G.R., Piattini, M., Pinggera, J.: Assessing event correlation in non-process-aware information systems. Software and Systems Modeling, 1–23 (2012).
42. Popova, V., Fahland, D., & Dumas, M. (2015). Artifact lifecycle discovery. International Journal of Cooperative Information Systems, 24(01), 1550001.
43. Raichelson, L., & Soffer, P. (2014, September). Merging Event Logs with Many to Many Relationships. In Business Process Management Workshops (pp. 330-341). Springer International Publishing.
44. Rajaraman, A., & Ullman, J. D.: Mining of massive datasets. Cambridge University Press (2012).
45. Veiga, G. M., & Ferreira, D. R. (2009, September). Understanding spaghetti models with sequence clustering for ProM. In *International Conference on Business Process Management* (pp. 92-103). Springer Berlin Heidelberg.
46. Verbeek, H. M. W., Buijs, J. C., van Dongen, B. F., & van der Aalst, W. M. P. (2011). Xes, xesame, and prom 6. In Information Systems Evolution (pp. 60-75). Springer Berlin Heidelberg.
47. Verbeek, H. M. W., Buijs, J. C. A. M., van Dongen, B. F., & van der Aalst, W. M. P. (2010). Prom 6: The process mining toolkit. Proc. of BPM Demonstration Track, 615, 34-39.
48. WALES, N. S. (2014). Mining Processes with Multi-Instantiation: Discovery and Conformance Checking.
49. Wang, Y., Wen, L., Yan, Z., Sun, B., & Wang, J. (2015, October). Discovering BPMN Models with Subprocesses and Multi-instance Markers. In On the Move to Meaningful Internet Systems: OTM 2015 Conferences (pp. 185-201). Springer International Publishing.
50. Xixi Lu, Marijn Nagelkerke, Dennis van de Wiel, Dirk Fahland: Discovering Interacting Artifacts from ERP Systems. IEEE Trans. Services Computing 8(6): 861-873 (2015)
51. X. Lu, M. Nagelkerke, D. van de Wiel, D. Fahland. Discovering Interacting Artifacts from ERP Systems (Extended Version). BPM Center Report BPM-15-08, BPMcenter.org, 2015
52. Yzquierdo-Herrera, R., Silverio-Castro, R., & Lazo-Cortés, M.: Subprocess discovery: Opportunities for process diagnostics. In Enterprise Information Systems of the Future (pp. 48-57). Springer Berlin Heidelberg (2013).
53. Zhang, M., Hadjieleftheriou, M., Ooi, B. C., Procopiuc, C. M., and Srivastava, D. 2010. On multi-column foreign key discovery. Proc. VLDB Endow. 3, 1-2 (September 2010), 805-814.

Appendix 1.

Listing 1. Deconstruction Algorithm

Algorithm: Generating a flattened sub log which consist an individual case per instance; uses the following main variables and functions: *instanceList($trace$): an array of traces, so that each cell, instanceList[$trace$], is an array of instances related to that trace

```
1:  flattenedLog = {} // I start with an empty flattened log
2:  //review all cases related to the sub log
3:  For all trace ∈ subLog do
4:  |  // generate a list of instances associated to a case
5:  |  Set *instanceList(trace) == extractInstances(trace)
6:  end for
7:
8:  // generate flattened traces for all traces in the sub log
9:  For all trace ∈ subLog do
10: |  counter = 1 // Reinitialize the counter for every trace
11: |  // generate a flattened trace for every extracted instance
12: |  For all instance ∈* instanceList(trace) do
13: |  |  flattenedTrace = <> // I start with an empty flattened trace
14: |  |  // consider all events in the trace
15: |  |  For all event ∈ trace do
16: |  |  |  // generate new event from this event for current instance if needed
17: |  |  |  // compare current instance to current event
18: |  |  |  //positive match in one of two conditions:
19: |  |  |  //1. if the current event has no instance field
20: |  |  |  //2. if the current event's instance field contains the current instance
21: |  |  |  If ( #instance(event) == Null )|| (instance ∈ #instance(event))   then
22: |  |  |  |  //first, copy the current event to flattened event
23: |  |  |  |  flattenedEvent = event.clone()
24: |  |  |  |  //second, fix the case_id by suffixing the counter
25: |  |  |  |  #case id(flattenedEvent) = #case id(event) +'-'+counter
26: |  |  |  |  //third, fix the instance of the current single value
27: |  |  |  |  #instance(flattenedEvent) = instance
28: |  |  |  |  //fourth, add the flattened event to the flattened trace
29: |  |  |  |  flattenedTrace.add(flattenedEvent)
30: |  |  |  end if
31: |  |  end for
32: |  end for
33: |
34: |  //add the flattened trace to the flattened log
35: |  flattenedLog.add(flattenedTrace)
36: |  counter++ // Increment the counter for the next instance
37: end for
38:
39: return flattenedLog
```

Listing 2. Log-merging Algorithm

Algorithm: Merging of cross logs using match time and match score of attribute values uses the following main variables and functions: **word_set (trace)*: a set holding all the non-trivial values (that is, excluding stop words) of attributes in all events of a trace (bag of unique words); **Function *match_time* (trace1, trace2): checks the temporal relation between trace1 and trace2 and returns TRUE if they meet temporal requirements; ***Function *check_Match_Score* [word_set (trace1), word_set(trace2)]: returns an integer value Match_Score($\boldsymbol{MS_{trace1,trace2}}$) – the calculated similarity score of attribute values between trace1 and trace2; ****Merge(trace1, trace2): returns a new trace containing all events from both trace1 and trace2 where all events in the new trace are ordered by their timestamps (earliest first) and where the case IDs of all events will be set to the given case ID.

```
1:  //review all cases related to the main log
2:  For all mainTrace ∈ mainLog do
3:  |  Set *word_set (mainTrace) == extract all unique values for every event in main trace
4:  end for
```

5:
6: //review all traces related to the sub log
7: **For all** $subTrace \in$ subLog **do**
8: | **Set** **word_set (subTrace)* == extract all unique values for every event in sub trace
9: **end for**
10:
11: //review all traces related to the main log
12: **For all** $mainTrace \in mainLog$ **do**
13: | //review all traces related to the sub log
14: | **For all** $subTrace \in$ subLog **do**
15: | | //check if timelines of main case and sub case meets the requirements
16: | | //calculate match score between attribute values of main and sub cases
17: | | **If** ** $match\_time\ (mainTrace;\ subTrace)$ **then**
18: | | | **Set** ** $\boldsymbol{MS}_{\boldsymbol{mainTrace,subTrace}}$ == check_Match_Score
19: | | | (*word_set (mainTrace);word_set (subTrace)*)
20: | | **end if**
21: | **end for**
22: **end for**
23:
24: unifiedLog = {}
25: counter = 1 // initialize counter
26: //review all traces related to the main log
27: **For all** $mainTrace \in mainLog$ **do**
28: | // Set the new case id for all events in the unified trace
29: | // Here we assume that mainTrace[0] gives us the first event of the main trace
30: | unifiedTracesId = #case id(mainTrace[0]) + '-' + counter
31: |
32: | unifiedTrace = **** Merge(mainTrace,unifiedTraceId)
33: | unifiedLog.add(unifiedTrace)
34: | //review all traces related to the sub log
35: | **For all** $subTrace \in$ subLog **do**
36: | | //check all traces from sub log which hold sufficient matching score
37: | | **If** *** $\boldsymbol{MS}_{mainTrace,subTrace} != Null$ &&
38: | | | *** $\boldsymbol{MS}_{mainTrace,subTrace} \geq threshold$ **then**
39: | | |
40: | | | unifiedTrace = **** Merge(subTrace,unifiedTraceId)
41: | | | unifiedLog.add(unifiedTrace)
42: | | **end if**
43: | **end for**
44: **end for**
45: **return** *unifiedLog*

Listing 3. Conversion Algorithm

Algorithm: Generating a Unified Log File of an end-to-end Instance View; uses the following main variables and functions: *instanceList($trace$): an array of traces, so each cell, instanceList[$trace$], is an array of instances related to that trace

```
1:  InstanceViewLog = {} // I start with an empty instance view log
2:  //review all cases related to the unified log
3:  For all trace ∈ unifiedLog do
4:  |  // generate a list of instances associated to a case
5:  |  Set *instanceList(trace) == extractInstances(trace)
6:  end for
7:
8:  // generate flattened traces for all traces in the unified log
9:  For all trace ∈ unifiedLog do
10: |  counter = 1 // Reinitialize the counter for every trace
11: |  // generate a flattened trace for every extracted instance
12: |  For all instance ∈* instanceList(trace) do
13: |  |     flattenedTrace = <> // I start with an empty flattened trace
14: |  |
15: |  |    // consider all events in the trace
16: |  |    For all event ∈ trace do
17: |  |    |
18: |  |    |  // generate new event from this event for current instance if needed
19: |  |    |  // compare current instance to current event
20: |  |    |  //positive match in one of two conditions:
21: |  |    |  //1. if the current event has no instance field
22: |  |    |  //2. if the current event's instance field contains the current instance
23: |  |    |
24: |  |    |  If ( #instance(event) == Null )|| (instance ∈ #instance(event)) then
25: |  |    |  |
26: |  |    |  |  //first, copy the current event to flattened event
27: |  |    |  |  flattenedEvent = event.clone()
28: |  |    |  |  //second, fix the case_id by suffixing the counter
29: |  |    |  |  #case id(flattenedEvent) = #case id(event) +'-'+counter
30: |  |    |  |  //third, fix the instance of the current single value
31: |  |    |  |  #instance(flattenedEvent) = instance
32: |  |    |  |  //fourth, add the flattened event to the flattened trace
33: |  |    |  |  flattenedTrace.add(flattenedEvent);
34: |  |    |  |
35: |  |    |  end if
36: |  |    end for
37: |  end for
38: |
39: |  //add the flattened trace to the flattened log
40: |  InstanceViewLog.add(flattenedTrace);
41: |  counter++ // Increment the counter for the next instance
42: |
43: end for
44:
45: return InstanceViewLog
```