Conversation Extraction from Event Logs

Sébastien Salva, Laurent Provot and Jarod Sue
LIMOS - UMR CNRS 6158, Clermont Auvergne University, Aubière, France

Keywords: Event Log, Session, Conversation Extraction, Correlation.

Abstract: Event logs are more and more considered for helping IT personnel understand system behaviour or performance. One way to get knowledge from event logs is by extracting conversations (a.k.a. sessions) through the recovering of event correlations. This paper proposes a highly parallel algorithm to retrieve conversations from event logs, without having any knowledge about the used correlation mechanisms. To make the event log exploration effective and efficient, we devised an algorithm that covers an event log and builds the possible conversation sets w.r.t. the data found within the events. To limit the conversation set exploration and quicker recover good candidates, the algorithm is guided by an heuristic based upon the evaluation of invariants and conversation quality attributes. This heuristic also offers flexibility to users, as the quality and invariants can be adapted to the system context. We report experimental results obtained from 6 case studies and show that our algorithm has the capability of recovering the expected conversation sets in reasonable time delays.

1 INTRODUCTION

Log analysis gathers approaches and tools allowing to continuously extract knowledge from event logs. The benefits of knowledge extraction from event logs are substantial: they can be employed for security audits (Salva and Blot, 2020b), real-time anomaly detection (Zhang et al., 2019), or model learning (Conforti et al., 2016; Salva and Blot, 2020a).

Event logs are usually recorded from (complex) distributed systems made up of concurrent components, e.g., Web service compositions or Internet of things (IoT) systems. To retrieve what happens from the event logs of such systems, correlation mechanisms, e.g., execution trace identifiers, are employed to propagate context ids and keep track of the process contexts. Unfortunately, every company devises its own correlation mechanisms, therefore correlations used with distrusted systems are more and more complex to understand and retrieve. The problem strongly hampers the automatic analysis of event logs to get useful knowledge materialised here under the form of *conversations* (a.k.a. sessions), i.e. sequences of correlated events interchanged among different components that achieve a certain goal.

Event correlation has been widely studied in different kinds of domains, e.g., process mining, or event association mining. In short, many approaches try to recover conversations by mining frequent association rules in event logs, without using correlation mechanisms (Fu et al., 2012; Musaraj et al., 2010). Other works propose to recover conversations by using some correlation patterns (Conforti et al., 2016; Motahari Nezhad et al., 2011). In particular, Process spaceship (Motahari Nezhad et al., 2011) gathers a set of algorithms allowing to scan event logs and retrieve conversation sets. The event correlations are mined by using a sort of breadth search strategy over the parameter assignments found in events. It explores all the possible correlations over the domain of parameter assignments and prunes them with interestingness properties. The interesting conversation sets are found at the expense of time complexity.

Contribution: we propose another highly parallel algorithm to retrieve conversations from event logs, without having any knowledge about the used correlation mechanisms. To make the event log exploration effective and efficient, our algorithm is based upon a formalisation of the notion of correlation patterns and is guided by the quality of the generated conversations. As there is no consensus about what a relevant conversation should be, the conversation quality can be adapted to meet user needs and viewpoints. Our algorithm is based upon a strategy mixing the divide and conquer paradigm with the depth-search approach and a heuristic based upon the evaluation of invariants and conversation quality attributes. Both the strategy and heuristic allow to quicker find a first solution. Our algorithm can also return the conversation sets that meet quality attributes, and sort them

Salva, S., Provot, L. and Sue, J.
Conversation Extraction from Event Logs.
DOI: 10.5220/0010652300003064
In *Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2021) - Volume 1: KDIR*, pages 155-163
ISBN: 978-989-758-533-3; ISSN: 2184-3228

from the best to the lowest quality. In comparison to (Motahari Nezhad et al., 2011), our algorithm is devised to concentrate the exploration on the conversation sets having correct correlations and good quality, and not on the exploration of the correlation domain. We show that the worst complexity is reduced. Furthermore, our approach offers flexibility to users, as the quality and invariants can be adapted to express the user knowledge about the system or to meet application contexts. This paper also provides an empirical evaluation, which investigates the precision and recall of the conversation sets generated from 6 event logs generated by real IoT systems, along with the performance of our algorithm.

The paper is organized as follows: we provide some definitions and notations on events, correlations and conversations in Section 2. Our approach is presented in Section 3. The next section shows some experimental results. Section 5 discusses related work. Finally, Section 6 summarises our contributions and draws some perspectives for future work.

2 CORRELATIONS AND CONVERSATIONS

2.1 Preliminary Definitions

We denote $\mathcal{E}$ the set of events of the form $e(\alpha)$ with e a label and α an assignment of parameters in P. The concatenation of two event sequences σ_1, $\sigma_2 \in \mathcal{E}^*$ is denoted $\sigma_1.\sigma_2$. ε denotes the empty sequence. For the sake of readability, we also write $\sigma_1 \in \sigma_2$ when σ_1 is a (ordered) subsequence of the sequence σ_2. Events are partially ordered in event logs. This is expressed with these partial order relations:

- $<_t \subseteq \mathcal{E} \times \mathcal{E}$, which orders two actions according to their timestamps,
- $<_c \subseteq \mathcal{E} \times \mathcal{E}$, which orders two actions if the occurrence of the first action implies the occurrence of the second one,
- $<:=<_t \cup <_c$ is the transitive closure of $<_c$ and $<_t$.

We also use the following notations on events and sequences to make our algorithms more readable:

- $from(e(\alpha)) = c$ denotes the source of the event when available; $to(e(\alpha)) = c$ denotes the destination;
- $isReq(e(\alpha))$, $isResp(e(\alpha))$ are boolean expressions expressing the nature of the event;
- $A(\sigma) = \bigcup_{e(\alpha)\in\sigma} \alpha$ is the set of parameter assignments of σ.

2.2 Event Correlation and Conversation

The correlation mechanisms used from one system to another are seldom the same, but they often comply with some correlation patterns. Most of these patterns are introduced and discussed by (Barros et al., 2007). Correlation patterns always define the association of successive events into conversations by means of protocol information or event content. Given an event sequence $\sigma = e_1(\alpha_1)\dots e_k(\alpha_k) \in \mathcal{E}^*$, we formulate these patterns as follows:

- **Key based correlation:** an event $e(\alpha)$ is correlated with σ if all the events share the same keys or properties formulated by the same parameter assignment set: $\alpha \cap \alpha_1 \cap \dots \cap \alpha_k \neq \emptyset$;
- **Chained correlation:** $e(\alpha)$ is correlated with σ if $e(\alpha)$ shares some references with $e_k(\alpha_k)$: $\alpha \cap \alpha_k \neq \emptyset$;
- **Function based correlation:** this pattern is somehow a special case of the previous ones. A function $f : \mathcal{E} \to L$ firstly assigns to each event a label of the form "l:=label" in L according to the event parameter assignments. For instance, a function f can be designed to return company names from ip addresses. For an event $e(\alpha)$, we consider that α is completed with these label assignments. Then, the event correlation is performed with one of the previous patterns;
- **Time-based correlation:** $e(\alpha)$ is correlated with σ if it carries a time-based relationship with the events of σ. This pattern is somehow a special case of the previous one, in the sense that a label can be injected into an event w.r.t. a condition on time. A function $f : \mathcal{E} \to L$ assigns labels of the form "t:=l" to events according to timestamps. For example, considering that timestamps are real numbers, the function $f : \mathcal{E} \to L, f(e(\alpha)) = \{t := floor(time(e(\alpha))/T)\}$ provides the same labels to the events occurring in the same time lapse T.

An event correlation can be formulated with one of these patterns but also with expressions composed of pattern conjunctions or disjunctions. To make our algorithm readable, we write $e(\alpha)$ *correlates* σ if the event $e(\alpha)$ correlates with a sequence σ by such a correlation pattern-based expression.

We consider that correlations between pairs of successive events may change as different patterns might be used within the same conversation. These correlation changes are explicitly observed by the

successive sets of parameter assignments used. In reference to (OASIS Consortium, 2007), we call these sets of parameter assignments *correlation sets*. A conversation corresponds to an event sequence interchanged among components, whose events correlate by means of correlation sets. Finally, we call the set of parameters used in a correlation set, a *correlation key*:

Definition 1. *Let* $\sigma = e_1(\alpha_1)\dots e_k(\alpha_k) \in \mathcal{E}^*$.

- σ *is a conversation iff* $\forall 1 < i \leq k$: $e_i(\alpha_i)$ *correlates* $e_1(\alpha_1)\dots e_{i-1}(\alpha_{i-1})$
- $corr(\sigma) = \{cs_1, \dots, cs_{k-1}\}$ *denotes the set of correlation sets of* σ*, with* $cs_i \subseteq \alpha_i \cap \alpha_{i+1}$
- $K(\sigma)$ *is the set of correlation keys of* $corr(\sigma)$.
- $K(C) = \bigcup_{\sigma \in C} K(\sigma)$ *is the set of correlation keys of the conversation set* C.

3 THE APPROACH

Given an event log produced by a concurrent and distributed system, our algorithm aims at assembling events into conversations. We assume that events are ordered with the $<$ relation. When several log files are given, we assume that they can be assembled with $<_t$ or $<_c$. In particular, the causal order relation $<_c$ may help assemble two log files given by two systems whose internal clock values slightly differ. $<_c$ indeed helps order the actions $a_1(\alpha_1)$ in a first log that imply the occurrence of other actions $a_2(\alpha_2)$ in a second one. The analysis of the pairs $(a_1(\alpha_1), a_2(\alpha_2))$ helps compute the difference of time between these two systems.

Our approach begins by formatting the event log into an event sequence S of events of the form $e(\alpha)$ by means of regular expressions. Several techniques, e.g., (Vaarandi and Pihelgas, 2015; Messaoudi et al., 2018), can assist users in the mining of patterns or expressions from log files, which can be used to quickly derive regular expressions. Afterwards, our algorithm is devised to explore the possible correlations among the successive events of the sequence S, thus in a depth-wise way, while being efficiently guided by the construction of conversations and their consistencies. This notion of consistency is expressed by means of correlation patterns, *conversation set invariants* and *conversation set quality*. These two last notions are presented in the remainder of this section, then we introduce our conversation set extraction algorithm. Figure 1 illustrates a simple example of formatted events, which will be used as a running example.

/a0(id:=4) /a1(id:=5) ok1(id:=4,id1:=1) ok2(id:=5,id2:=1)
/buy1(item:=i,id1:=1) /login(a:=acc) /buy2(item:=i,id2:=1)
logged(a:=acc) ok3(id1:=1, content=done)
ok4(id2:=1, content:=done)

Figure 1: (Formatted) Event Sequence Example.

3.1 Conversation Set Invariants

As stated previously, an event $e(\alpha)$ complements a current conversation σ iff $e(\alpha)$ *correlates* σ. From this correlation notion, we derive properties that must hold (invariants) on a conversation set C. These invariants will allow our algorithm to stop the exploration of a candidate conversation set when they don't hold.

In accordance with the correlation patterns, an event must correlate with only one conversation σ in C with a unique correlation set: a correlation set cs of $corr(\sigma)$ cannot be empty, cs cannot be found in another conversation σ_2 of C. Besides, σ must have parameter assignments for building potential correlation sets; it must include parameter assignments that cannot be found in any other conversation σ_2. These invariants are formulated in the following proposition:

Proposition 2 (Conversation Set Invariants). *Let* C *be a conversation set and* $\sigma \in C$. *Inv stands for the set of conversation set invariants:*

- $\forall cs \in corr(\sigma) : cs \neq \emptyset$
- $\forall cs \in corr(\sigma), \forall \sigma_2 \in C \setminus \{\sigma\} : cs \cap A(\sigma_2) = \emptyset$
- $A(\sigma) \setminus \bigcup_{\sigma_2 \in C \setminus \{\sigma\}} A(\sigma_2) \neq \emptyset$

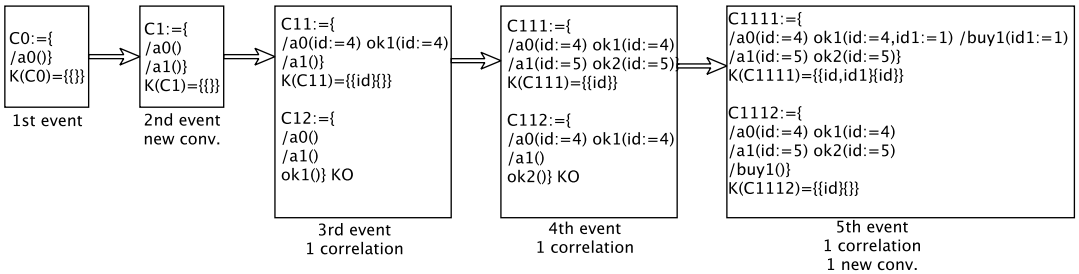
Additionally, other invariants can be defined to meet user preferences. For instance, the following invariant forbids the use of the parameters in NK to build correlation keys. The last invariant imposes conversations to start with a request.

- $\forall k \in K(\sigma) : k \cap NK = \emptyset$
- $\forall e_1(\alpha_1)\dots e_k(\alpha_k) \in C : isReq(e_1(\alpha_1))$

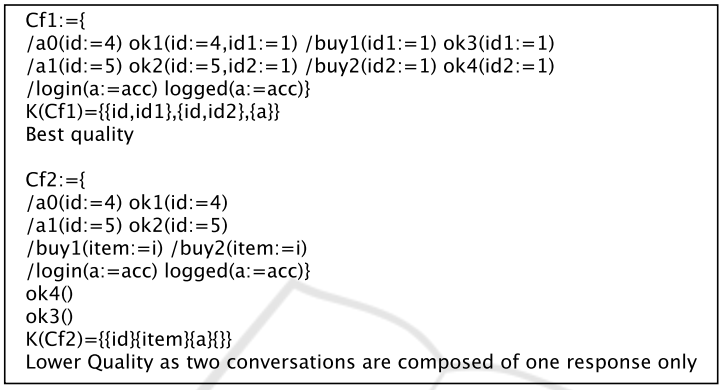
For readability, we denote that the conversations of a conversation set C meet conversation invariants with C *satisfies Inv*.

3.2 Conversation Set Quality

Our algorithm uses quality metrics as another way to limit the conversation set exploration, but also to prioritise this exploration among several conversation set candidates. We formulate a comprehensive quality metric of a conversation set C by means of a utility function for representing user preferences. We have chosen the technique *Simple Additive Weighting* (SAW) (Yoon and Hwang, 1995), which allows the interpretation of these preferences with weights. The



(a) First conversation sets generated from the sequance of Figure 1



(b) Final Conversation sets

Figure 2: Conversation sets after Steps 1-4 of Algorithm 18 and after the last step. Q is the conversation set quality.

following definition refers to quality metrics $M_i(C)$ over conversation sets, themselves calculated with metrics $m_i(\sigma)$ over conversations:

Definition 3 (Conversation Set Quality). *Let C be a conversation set. $Q(C)$ is a utility function defined as: $0 \leq Q(C) = \sum_{i=1}^{n} M_i(C).w_i \leq 1$ with $0 \leq M_i(C) = \frac{\sum_{\sigma \in C} m_i(\sigma)}{|C|} \leq 1$, $w_i \in [0;1]$ and $\sum_{i=1}^{k} w_i = 1$.*

The conversation quality metrics can be general or established with regard to a specific system context. Like invariants, our approach actually does not limit the metric set. We give below some examples implemented in our prototype tool. Two first metrics m_1 and m_2 evaluate whether a conversation σ follows the classical request-response exchange pattern (sender sends a request to receiver, ultimately returning a response). m_1 evaluates the ratio of requests in σ associated to some responses with $ReqwResp(\sigma)$. m_2 measures the ratio of responses following a prior request with $RespwReq(\sigma)$. We observed that when m_1 or m_2 are close to 0, this means that the event log may include a lot of noise, or that the event log is incomplete, or that the correlation sets are incorrect.

$$0 < m_1(\sigma) = \frac{|ReqwResp(\sigma)| + 1}{|Req(\sigma)| + 1} \leq 1 \qquad (1)$$

$$0 < m_2(\sigma) = \frac{|RespwReq(\sigma)| + 1}{|Resp(\sigma)| + 1} \leq 1 \qquad (2)$$

The metric m_3 examines whether σ is composed of correlated events, in other terms, whether σ has more than one event:

$$m_3(\sigma) = \begin{cases} 1 & \text{if } corr(\sigma) \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \qquad (3)$$

The metric m_4 measures the ratio of events that belong to a chain of events. An event $e(\alpha)$ sent by component c_1 to c_2 belongs to an event chain if $e(\alpha)$ is followed by another event sent by c_2 or nothing, and $e(\alpha)$ is preceded by an event sent to c_1 or nothing. These two notions are formulated with $f(e(\alpha))$ and $p(e(\alpha))$. $Events(\sigma)$ stands for the set of events of σ.

$$f(e(\alpha)) = \begin{cases} 1 & \text{if } \exists e_2(\alpha_2) : e(\alpha) < e_2(\alpha_2) \wedge \\ & to(e(\alpha)) = from(e_2(\alpha_2)) \text{ or} \\ & \nexists e_2(\alpha_2) : e(\alpha) < e_2(\alpha_2) \\ 0 & \text{otherwise} \end{cases}$$

$$p(e(\alpha)) = \begin{cases} 1 & \text{if } \exists e_2(\alpha_2) : e_2(\alpha_2) < e(\alpha) \wedge \\ & from(e(\alpha)) = to(e_2(\alpha_2)) \text{ or} \\ & \nexists e_2(\alpha_2) : e_2(\alpha_2) < e(\alpha) \\ 0 & \text{otherwise} \end{cases}$$

$$0 < m_4(\sigma) = \frac{\sum_{(e(\alpha)) \in \sigma} f(e(\alpha) + p(e(\alpha)))}{2|Events(\sigma)|} \leq 1 \qquad (4)$$

3.3 Conversation and Correlation Set Extraction Algorithm

Algorithm 1: Conversation and Correlation Set Extraction Parallel Algorithm.

```
input : Event sequence S , boolean first
output: Conversation set CS,
  struct PRIORITYTHREADPOOL
    PriorityTask List List
    run(): while there is a task in L do
        choose the Task with hightest Priority Q;
        run Task;
  end struct
  struct PRIORITYTASK,
    Conversation set C, int i, Priority Q, sequence σ
    run(): call FindCS(C, i, σ);
  end struct
Pool := PriorityThreadPool();
T := set of N sub-sequences σ uniformly extracted from S of length L
starting by a request;
foreach σ = e1(α1)...ek(αk) ∈ T do
    Add PriorityTask({e1(α1)}, 1, Q = 1, σ) to Pool;
C1,...,Cn:= Wait end of Pool;
Choose correlation key set K(C) among K(C1),...,K(Cn);
Extract CS from S with K(C);
```

Procedure: FindCS(C, i, σ).

```
1 : FindCS1:  FindCS(I): FindCS:  FindCS
2 if i ≤ k then
3     foreach σ1 = e1(α1)...ek(αk) ∈ C : ei(αi) correlates σ1 do
4         CS := P(αi ∩ αk)) \ {∅} ;
5         foreach cs ∈ CS do
6             σ2 := σ1.ei(αi);
7             corr(σ2) := corr(σ1) ∪ {cs};
8             C2 := C ∪ {σ2} \ {σ1};
9             if C2 satisfies Inv and Q(C2) ≥ T then
10                add PriorityTask(C2, i+1, Q(C2), σ) to Pool;
11    C3 := C ∪ {ei(αi)};
12    corr(ei(αi)) := ∅;
13    if C3 satisfies Inv and Q(C3) ≥ T then
14        add PriorityTask(C2, i+1, Q(C3), σ) to Pool;
15 else
16    return C;
17    if first and Q(C) ≥ T2 then
18        STOP Pool;
```

We can now present our Conversation and Correlation Set Extraction algorithm, given in Algorithm 1. It takes as input an event sequence S along with a boolean *first* determining whether the algorithm must stop after finding one conversation set that meet quality requirements. Algorithm 1 exploits two structures. PriorityThreadPool implements the thread pool paradigm to run tasks in parallel w.r.t. a set of available threads. The choice of the task to execute is guided by a priority Q, which is equal to the conversation set quality. These tasks are modelled with the PriorityTask structure, which holds a conversation set, an index i, a priority Q and an event sequence to explore. When a PriorityTask is executed by the PriorityThreadPool, the procedure $FindCS(C,i,\sigma)$ is called. Algorithm 1 somehow mimics human being by implementing the *divide and conquer* paradigm. It extracts N event sequences of length L in S and analyses them in parallel to quicker find the best correlation key sets. Given a sequence σ, it prepares a first task composed of the conversation set C equal to the first event of σ and supplies it to the thread pool. It results that $FindCS(\{e_1(\alpha_1)\}, i=2, \sigma)$ is called. Next, every event of σ are successively covered by recursively supplying a new task that calls the procedure *FindCS* with a new conversation set.

The procedure $FindCS(C,i,\sigma)$ takes the event $e_i(\alpha) \in \sigma$ and tries to find a conversation σ_1 in C such that $e_i(\alpha_i)$ *correlates* σ_1. If such a conversation exists, the procedure builds for every possible correlation set (line 5) a new conversation set C_2 with the new conversation $\sigma.e_i(\alpha_i)$. An additional conversation set C_3 is built to consider that the event $e_i(\alpha_i)$ might also be the beginning of a new conversation (line 11). For every new conversation set that meet conversation invariants and quality requirement Q(), a new PriorityTask is instantiated with the priority Q() (lines 10 and 14). The quality requirement is materialised by the thresholds T and $T2 \geq T$. The latter can be used to re-enforce the desired conversation set quality when the algorithm is stopped after finding one solution.

Consider the example of event sequence S of Figure 1. Some steps of Algorithm 1 and the final conversation sets are illustrated in Figure 2. Algorithm 1 starts with the first event a0 and creates a first conversation set $C1$. The next event /a1 cannot be correlated to /a0, hence a new conversation is begun. The 3rd event ok1 can correlate with /a0 with $\{id := 4\}$, $C1$ becomes $C11$. The event ok1 could also have been the first event of a new conversation in a new conversation set $C12$. But, the third invariant of Proposition 2 does not hold ($id := 4$ is the only assignment allowing to identify the conversation /a0, but the assignment is also found in the conversation ok1). Hence, $C12$ is not kept. The same situation happens with the 4th event ok2. The 5th event /buy1 can correlate with ok1 (conversation set C1111), but, at this stage, it may also be the first event of a new conversation (C112). Finally, two conversation sets are recovered from the sequence S by Algorithm 1, $Cf1$ and $Cf2$ given in Figure 2(b). With regard to conversation quality, $Cf1$ is the best candidate. $Q(Cf2)$ is lower than $Q(Cf1)$

because $Cf2$ contains two conversations composed of one response only.

Algorithm Complexity: At worst, the complexity of the procedure *FindCS* is exponential time. It explores conversation sets while covering the event sequence S. Given a conversation set C, it builds new conversation sets: 1) by completing a conversation with an event $e_i(\alpha_i)$ (lines 3-10) or 2) by creating a new event conversation (lines 11-14). 1) At worst, *FindCS* complements the conversations of C with $e_i(\alpha_i = \{p_1, \ldots, p_P\})$ in $2^P - 1$ different ways because there is at most $2^P - 1$ possible correlation sets in $\alpha \cap \{p_1, \ldots, p_P\}$ (line 4). With 1) and 2), Procedure *FindCS* builds 2^P new conversation sets from C at worst. While covering every event of S, the procedure covers $1 + (2^P) + \cdots + (2^P)^{(k-1)}$ conversation sets. Its complexity is then proportional to $M * (\frac{2^{kP}-1}{2^P-1})$ as 2^P is different from 1, with M the complexity for computing the quality of one conversation set. Even though the quality metrics may be different from one user to another, it sounds reasonable to estimate that the metric computation complexity is $O(k^2)$. The algorithm of Process spaceship (Motahari Nezhad et al., 2011) is double exponential time in the worst case. Hence the depth-wise strategy used in our algorithm offers a better time complexity.

In average, Algorithm 1 covers N sequences of length L, whatever the event log size k. Additionally, it relies on invariants and quality metrics to limit the conversation set space exploration. As a result, the average case complexity is much lower. This is confirmed by our experimentations presented in the next section.

4 EVALUATION

The experiments presented in this section aim to evaluate the capabilities of our algorithm in terms of effectiveness and performance through these questions:

- RQ1: can the approach extract relevant conversation sets from event logs? We evaluate the relevance of the conversation sets extracted by Algorithm 1 by assessing the accuracy of the extracted correlation key sets. This accuracy is studied with precision and recall. Precision is here the fraction of expected correlation key sets of conversations among the retrieved ones. Recall is the fraction of expected correlation key sets that were retrieved;
- RQ2: what is the performance of our algorithm? How does it scale with the size of the event log?

4.1 Empirical Setup

This study was conducted on 6 IoT systems integrating varied devices and gateways communicating over HTTP and UDP. We assembled and configured these systems from a set of 7 commercial devices (3 sensors, 2 gateways, 2 actuators). The behaviours of the gateway(s) after the receipt of data from the sensors differ in each configuration. We monitored these systems and collected event logs of about 2200 events, which themselves include 5 to 9 parameter assignments. We denote the logs $S1$ to $S6$. Furthermore, we implemented Algorithm 1 in Java. Source codes and event logs are available here[1].

4.2 RQ1: Can the Approach Extract Relevant Conversation Sets?

To study this question, we manually analysed the event logs S1 to S6 to determine the relevant correlation key sets, that is the correct and expected ones for every conversation. We observed that all the conversations are identified by means of a key-based correlation using the parameter session. Next, we applied Algorithm 1 on event logs with the parameters N=20, L=20 (20 sequences of 20 events were extracted by Algorithm 1). The quality threshold of Algorithm 1 was set to 80% (the conversation sets whose quality is lower are deleted). Finally, for every event log, we collected the correlation key sets of the generated conversations, and measured recall and precision. We also analysed how the precision is distributed over the N event sequences by measuring the ratio of occurrences of the expected correlation key sets, or in other terms, the ratio of event sequences providing a precision equal to 100% among the N sequences.

Table 1: Recall, Precision and Occurrence ratio of the expected correlation key sets.

	Correlation Key Set Recall	Correlation Key Set Precision	Occurrence Ratio of the Expected Results
$S1$	100%	81%	65%
$S2$	100%	76%	55%
$S3$	100%	80%	65%
$S4$	100%	100%	100%
$S5$	100%	100%	100%
$S6$	100%	90%	40%

Table 1 shows the results for S1 to S6. We deduce from Column 1 that Algorithm 1 always returns the expected correlation key sets with these event logs. This result comes from the fact that the quality metrics are suited to the event log contexts, i.e. the mes-

[1] https://github.com/sasa27/ConversationExtraction

sage passing protocols HTTP and UDP composed of request and responses.

The second column of Table 1 shows that Algorithm 1 has good precision, although it returns some unexpected results. After analysing them, we mostly observed that Algorithm 1 returned unexpected correlation key sets on account of sequences of successive UDP requests later followed by their successive responses. In these cases, the algorithm built correlation key sets composed of the expected parameter session, along with the parameters status, group, idx, or response because these ones are assigned to the same values between two successive requests or responses. In this case, the quality is equal to 100%. We believe that the definition of an additional metric favouring smallest correlation key sets can help lower the conversation qualities of these unexpected correlation key sets. When Algorithm 1 splits up an UDP request to its associated response into two conversations, conversation qualities are always lower to 100%, thanks to the metrics m_1, m_2, m_4.

When several correlation key sets are returned, the choice of the most relevant one by users can be guided by the ratio of occurrence of the expected correlation key sets (Column 3 of Table 1). For S1, S3, S4 and S5, this ratio indicates that the right correlation key set only was returned with most of the 20 event sequences. With S2 and S6, the ratios are lower, but there is no other correlation key set that has a higher ratio or close ratio. The ratios are lower with these event logs because they are composed of non communicating events (neither requests or responses) that lower conversation set qualities. Indeed, we observed that when there is an overpresence of this kind of events in the N event sequences, there is also an overpresence of requests not followed by responses (here, m_1 strongly lowered quality measurements). Extending the sequence length (parameter L) allows to increase the ratios.

In summary, Algorithm 1 provides good recall and precision with these event logs. It is worth noting that the capability of Algorithm 1 of finding accurate correlation key sets depends on quality metrics. These ones must be chosen w.r.t. the protocols used or the event types.

4.3 RQ2: What Is the Performance of Our Algorithm?

During our experiments, we observed that execution times strongly depend on the event log size but also on the conversation number, as Algorithm 1 checks whether invariants hold and computes quality metrics on conversation sets. Hence, to answer this question,

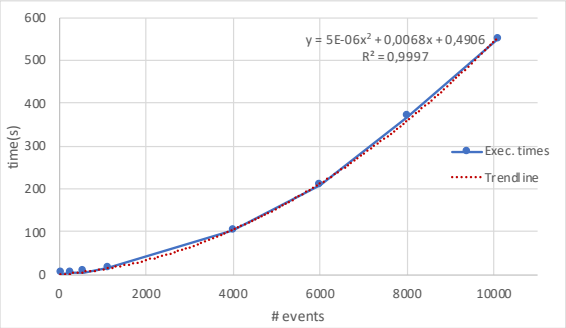


Figure 3: Execution times vs. event log sizes.

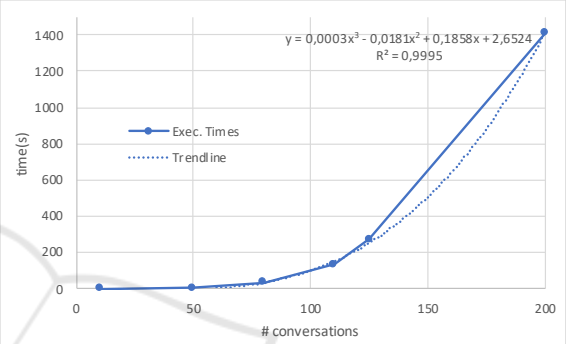


Figure 4: Execution times vs. conversation number.

we firstly studied how the tool scales with the size of the event logs by limiting the conversation number to 20. We took the 20 first conversations of S1 and augmented them using 40 to 10000 events. Additionally, we measured execution times with regard to the number of conversations in the event logs from 10 to 200 conversations of 2 events. Figures 3 and 4 depict execution time curves and tendency curves.

The execution time curve of Figure 3 follows a quadratic curve and reveals that Algorithm 1 performs well in practice. In comparison to the algorithm worst complexity, this can be explained by the fact that, in real logs, a new event does not correlate so many existing conversations (because it does not share common parameters with the last event of the conversation for instance) or because it breaks some conversation set invariants. On one hand, this limits the number of new conversation sets created. And on the other hand, it eliminates conversation sets that have reached a dead end. Overall, the ratio of new conversation sets over eliminated conversation sets stays low. It results that the algorithm is rather good in practice. Figure 4 depicts a cubic polynomial curve, which shows that execution times quicker increase with regard to the number of conversations. Here, we suspect a lack of optimisation within our current implementation. Indeed, the computation of invariant satisfiability (line

9 of Algorithm 1) is done by iterating over all the conversations of the conversation sets. Some investigations need to be conducted to try to reduce this number of iterations, perhaps on leveraging on invariant verifications that have already been checked on the previous calls of Procedure *FindCS*.

5 RELATED WORK

Event correlation has been widely studied in different kinds of domains, e.g., process mining, event association mining, or session recovery. Initially, some approaches restricted the problem of recovering conversations with assumptions. For instance, the correlation ids are assumed to be known in advance in (Kliger et al., 1995; Gaaloul et al., 2008).

Later, several papers (Fu et al., 2012; Liu and Liu, 2010; Serrour et al., 2008; Musaraj et al., 2010) presented techniques based upon the mining of association event rules among pairs of events. These rules can be seen as conversations. The advantage of these approaches is to not require any assumption on correlation patterns as these ones are not considered. Logmaster (Fu et al., 2012) generates event association rules with the computation of two event occurrence numbers: the support count, which is the recurring times of the preceding events which are followed by the posterior event, and the posterior count which is the recurring times of the posterior event that follows the preceding events. Liu et al. proposed an approach in (Liu and Liu, 2010) for discovering frequent correlation relationships by optimising the Web access pattern tree mining algorithm. Serrour et al. also proposed to correlate events by extracting frequent event correlation relationships, but they use graphs to express frequencies (Serrour et al., 2008). Conversations are then transformed into business processes. Their approach requires to know the senders or receivers of the events. The delta algorithm presented in (Musaraj et al., 2010) recovers correlations among pairs of events by using linear regression methods to derive the equations that describe the relationships that exist between the numbers of different message occurrences.

Other works use correlation patterns for recovering conversations as the correlation mechanism strongly reduces the amount of false positives when systems are made up of concurrent components. The approach given in (Dustdar and Gombotz, 2006) tries to identify conversations by heuristically setting a session duration threshold and then measuring the conversation quality. The session duration is updated by the approach until the conversation quality exceeds a given threshold. To reach this purpose, the approach assumes that a service cannot begins several conversations concurrently, and that the conversations are similar in terms of consumed services. These assumptions strongly limit its practical application.

The two papers (Conforti et al., 2016; Motahari Nezhad et al., 2011) present algorithms whose objectives and assumptions get closer to the ones of Algorithm 1. BPMN Miner (Conforti et al., 2016) is a tool specialised in the recovery of BPMN models. The novelty brought by BPMN Miner consists in detecting sub-conversations to later depict subprocesses. These conversations are obtained by splitting an event log into sub-logs by means of process instance identifiers. The algorithm supports one correlation pattern only (key based correlation). Then, it uses the TANE algorithm for the discovery of functional dependencies among events. When several candidate keys are available, it selects keys either with supervision or by choosing the lexicographically smallest candidate key. The latter builds a lot of incorrect conversation sets.

Process spaceship (Motahari Nezhad et al., 2011) gathers a set of algorithms allowing to correlate events of event logs and represent processes with views. The event correlations are mined by using a kind of breadth search strategy over the set of parameter assignments. The algorithms correlates events by considering all the possible atomic assignments, then conjunctions and finally disjunctions. In the meantime, this large set of possible correlation sets are pruned by means of 4 metrics, e.g., length of conversations, or occurrence of parameters used for correlations. Our algorithm uses another strategy, which aims at finding correlation sets while building conversations. Compared to Process spaceship, this can be considered as a depth search guided by heuristics based upon invariants and quality metrics. This strategy allows to quicker find a first solution. Our algorithm also has the capability of ordering conversation sets that meet quality requirements.

6 CONCLUSION

This paper has proposed the design and implementation of an algorithm for the recovery of conversation sets from event logs generated by concurrent and distributed systems. The algorithm explores the conversation set space that can be derived from an event log by implementing the divide and conquer paradigm. Furthermore, it is guided toward the most relevant solutions by means of conversation invariants and quality metrics. The latter can be adapted to define user

preferences. The algorithm either provides a first correlation key set that meets quality or returns a sorted list along with the respective conversations sets.

Our evaluation showed that despite using invariants and quality attributes, Algorithm 1 may still return several correlation key sets. In this case, the user has to choose one set with which conversations are finally extracted. We intend to reduce the need for supervision and increase precision by improving our approach with a decision-making algorithm. The latter will compute correlation key set scores, choose the most appropriate set, and finally, will automatically return one conversation set.

ACKNOWLEDGEMENT

Research supported by the French Project VASOC (Auvergne-Rhône-Alpes Region) https://vasoc.limos.fr/.

REFERENCES

Barros, A., Decker, G., Dumas, M., and Weber, F. (2007). Correlation patterns in service-oriented architectures. In Dwyer, M. B. and Lopes, A., editors, *Fundamental Approaches to Software Engineering*, pages 245–259, Berlin, Heidelberg. Springer Berlin Heidelberg.

Conforti, R., Dumas, M., García-Bañuelos, L., and La Rosa, M. (2016). Bpmn miner: Automated discovery of bpmn process models with hierarchical structure. *Information Systems*, 56:284–303.

Dustdar, S. and Gombotz, R. (2006). Discovering web service workflows using web services interaction mining. *Int. J. Bus. Process. Integr. Manag.*, 1:256–266.

Fu, X., Ren, R., Zhan, J., Zhou, W., Jia, Z., and Lu, G. (2012). Logmaster: Mining event correlations in logs of large-scale cluster systems. In *2012 IEEE 31st Symposium on Reliable Distributed Systems*, pages 71–80.

Gaaloul, W., Baïna, K., and Godart, C. (2008). Log-based Mining Techniques Applied to Web Service Composition Reengineering. *Service Oriented Computing and Applications*, 2(2-3):93–110.

Kliger, S., Yemini, S., Yemini, Y., Ohsie, D., and Stolfo, S. (1995). *A Coding Approach to Event Correlation*, pages 266–277. Springer US, Boston, MA.

Liu, L. and Liu, J. (2010). Mining web log sequential patterns with layer coded breadth-first linked wap-tree. In *2010 International Conference of Information Science and Management Engineering*, volume 1, pages 28–31.

Messaoudi, S., Panichella, A., Bianculli, D., Briand, L., and Sasnauskas, R. (2018). A search-based approach for accurate identification of log message formats. In *Proceedings of the 26th Conference*, ICPC '18, pages 167–177, New York, NY, USA. ACM.

Motahari Nezhad, H. R., Saint-Paul, R., Casati, F., and Benatallah, B. (2011). Event correlation for process discovery from web service interaction logs. *VLDB J.*, 20:417–444.

Musaraj, K., Yoshida, T., Daniel, F., Hacid, M.-S., Casati, F., and Benatallah, B. (2010). Message Correlation and Web Service Protocol Mining from Inaccurate Logs. In *IEEE International Conference on Web Services*, pages 259–266, Miami, Florida, United States. IEEE Computer Society.

OASIS Consortium (2007). Ws-bpel version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf.

Salva, S. and Blot, E. (2020a). Cktail: Model learning of communicating systems. In Ali, R., Kaindl, H., and Maciaszek, L. A., editors, *Proceedings of the 15th International Conference ENASE, Prague, Czech Republic, May 5-6, 2020*, pages 27–38. SCITEPRESS.

Salva, S. and Blot, E. (2020b). Verifying the application of security measures in iot software systems with model learning. In van Sinderen, M., Fill, H., and Maciaszek, L. A., editors, *Proceedings of the 15th International Conference ICSOFT, Lieusaint, Paris, France, July 7-9, 2020*, pages 350–360. ScitePress.

Serrour, B., Gasparotto, D. P., Kheddouci, H., and Benatallah, B. (2008). Message correlation and business protocol discovery in service interaction logs. In Bellahsène, Z. and Léonard, M., editors, *Advanced Information Systems Engineering*, pages 405–419, Berlin, Heidelberg. Springer Berlin Heidelberg.

Vaarandi, R. and Pihelgas, M. (2015). Logcluster - a data clustering and pattern mining algorithm for event logs. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 1–7.

Yoon, K. P. and Hwang, C.-L. (1995). Multiple attribute decision making: An introduction (quantitative applications in the social sciences).

Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., Xie, C., Yang, X., Cheng, Q., Li, Z., Chen, J., He, X., Yao, R., Lou, J.-G., Chintalapati, M., Shen, F., and Zhang, D. (2019). Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ESEC/FSE*, page 807–817, New York, NY, USA. Association for Computing Machinery.