Events Put into Context (EPiC)

Marcus Dees*¶, Bart Hompes†¶, Wil M.P. van der Aalst‡§¶

*Uitvoeringsinstituut Werknemersverzekeringen (UWV), Amsterdam, The Netherlands
† Artifex Consultancy, Eindhoven, The Netherlands
‡ RWTH Aachen University, Aachen, Germany
§ Fraunhofer Institute for Applied Information Technology
¶ Eindhoven University of Technology, Eindhoven, The Netherlands
marcus.dees@uwv.nl, bart.hompes@artifexconsultancy.nl, wvdaalst@pads.rwth-aachen.de

***Abstract*—Business process models can be (re)constructed using event data recorded during the process' execution. Similarly, event data can be used to verify conformance to prescribed behavior and to analyze and improve the underlying processes. However, not all events that are related to a process necessarily relate to its control-flow. Some events occur in the context of the process. In this work, we introduce the concept of *context events* to deal with these types of events. We show how distinguishing between contextual and control-flow events aids process discovery to obtain less complex process models. We demonstrate how visualizing context events on top of process models helps identify points in the process where context events occur often, aiding understanding. We analyze these benefits using two case studies involving real-life processes and event data.**

***Index Terms*—Process Mining, Complex Event Processing, Context events, Business process intelligence**

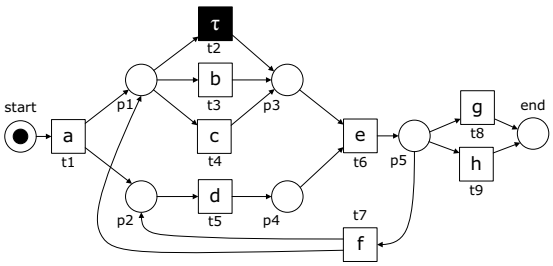
I. Introduction

A process comprises a series of activities performed in order to achieve a specific goal, such as creating a product or delivering a service. A process model describes the *control-flow* of a process, i.e., the order in which its activities should be executed. Organizations use process models in a normative and descriptive manner to define the boundaries of a process and to make sure the process yields the desired result. Modern processes are supported by information systems in which they leave digital footprints in the form of event data. Such events typically represent the occurrence of an instance of an activity. Analyzing event data is valuable because it can reveal previously unknown properties of processes. Process mining is a discipline that analyzes both event data and process models. It comprises three main use cases: process discovery, conformance checking, and process enhancement [1].

In many processes, there exist activities that, while bearing a relation to the process, are not part of its prescribed control-flow. These activities can occur at any time during the process execution. For example, lab tests in a hospital are part of the diagnostic process, but can be executed at any time during the process. Event data representing activities that may occur at any time during the process' execution have proven problematic for most discovery algorithms [2]. Additionally, such behavior complicates conformance checking [3]. In conclusion, it limits process understanding. Existing methods to obtain more precise process models exclude such *contextual behavior* during process discovery. However, by filtering out behavior, the discovered process model will not represent any of the excluded events and lack the complete picture.

It is our aim to both locate *contextual behavior* in a process and to represent it in a complete yet precise process model. To this end, we introduce the concept of *context events*. Context events are events that can be linked to cases in a process, but the activity associated with the event is not part of the prescribed control-flow for the process. As such, context events may *influence* the process, but do not *change* the control-flow state. Context events are a new class of events that are useful to be identified and handled separately from events that are related to the control-flow as defined by a process model.

Fig. 1 introduces a running example of a compensation request process to which we will relate the ideas presented in this paper. The process model for this example is taken from [1]. We use the alphabet a through h to represent the activities of the process. We also introduce three activities, x, y and z, that are related to the process but can not be mapped to an activity in the process' control-flow. These activities represent contact with the customer, which can occur at any given time in the process. There are few alternatives to represent the customer contact activities in our process model. Fig. 2a shows

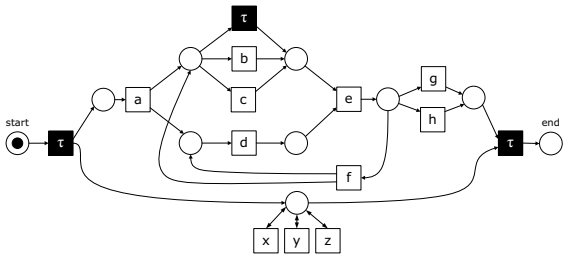


(a) Process model in Petri net notation.

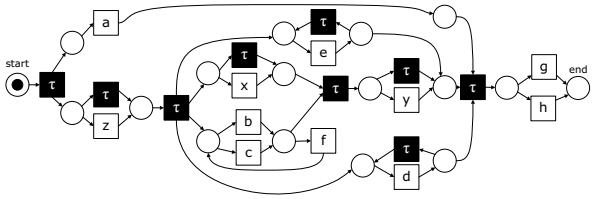
a.	register request	g.	pay compensation
b.	examine thoroughly	h.	reject request
c.	examine casually	h.	reject request
d.	check ticket	x.	incoming call
e.	decide	y.	incoming chat
f.	re-initiate request	z.	incoming complaint

(b) Legend for the transition labels of Fig. 1a

Fig. 1: Running example of a compensation request process.



(a) Fig. 1a with context activities added in a parallel branch.



(b) Discovered process model, using the *Inductive Visual Miner (IVM)* [4], based on a example dataset taken from [1] extended with context events x, y, and z for only 4 out of $1,391$ cases (0.29%).

Fig. 2: Two examples of how the running example can be extended with context activities x, y and z.

how they can be modelled in a parallel branch. This does not allow to pinpoint which activity occurs exactly where in relation to the other activities. Fig. 2b shows a discovered process model based on an example dataset taken from [1]. The dataset is extended with context events x, y, and z for only 4 of the $1,391$ cases (0.29%). The discovered model in Fig. 2b allows activity a to be executed after b,c and d, while in the dataset this never happens. Likewise, the number of occurrences of b plus c is no longer equal to the number of occurrences of d. This shows the influence of only a few events on a process discovery result. Clearly, neither alternative is satisfactory. A novel approach is required that captures the non-control-flow behavior and relates it to the process model.

The remainder of this paper is structured as follows. Section II discusses related work. Section III introduces preliminary definitions. Section IV explains our approach, while in Section V the approach is evaluated using two case studies. Section VI concludes the paper with suggestions for future work.

II. Related Work

Context is used in many different ways. In [1], the context in which events of a process are executed is divided into four categories: case, process, social, and external. These may influence the execution, performance or outcome of a process. In [5], Hompes et al. relate process, case, and activity context information to performance in order to identify possible causal relationships. The multi-perspective process explorer presented in [6] allows the user to visualize context information by selecting elements of a process model and comparing aggregated attribute values between the selected parts of the model. In [3], contextual information consists of control-flow events that occur in the neighborhood of a log pattern.

Our approach to distinguish control-flow events from context events could prove beneficial for other process mining approaches. In both [3] and [7], context is used during process discovery. However, no distinction is made between control-flow events and events that occur in the context of the process. Another way of improving the result of process discovery, is by filtering out undesired behavior from an event log before applying a discovery algorithm. An approach for this using information theory concepts is presented in [8], while in [9] the probability of an activity occurring in the context of other activities is used. Finally, a different approach to improving the result of discovery is the repair approach presented in [10]. A given process model is repaired to reflect the behavior present in an event log from the same process. Both model discovery and model repair techniques can benefit from our approach, as it helps reduce the number of activities and relations between those activities, hence reducing the complexity of the respective task, as shown in Fig. 2.

Visualization, used by many process mining techniques, can be improved by separating context information and control-flow information. As mentioned above, the multi-perspective process explorer [6] allows to aggregate the values of case and event attributes in the context of a process model. However, it does not allow to show contextual events in relation to a process model. Similarly, in [11], an artifact-centered approach is used to show multiple related (sub-)processes and how they interact with each other. Since context events are not necessarily related to one another, applying this approach in a process with contextual behavior does not lead to meaningful results. Work by de Leoni et al. creates movies in which the state of each process activity is calculated at each moment in time [12]. The result is plotted on top of a process model. However, there is no option to show events that are not part of the control-flow, i.e., that are not part of the process model in the movie. Only the number of unmapped events at any time during the process is shown in the process movie.

In [13], four questions are raised regarding the overlap between the fields of complex event processing (CEP) and process mining. Our work can be applied towards the challenge of process mining sensor events. Identifying control-flow-related sensor events, while keeping track of when other (contextual) sensor events take place, helps identify those sensor events that can be transformed into activities. Mandal et al. use an example from the logistics domain [14]. Goods are transported from a warehouse to a customer. While on route, incidents may happen that require recalculating the route. Knowing when incidents (context events) occur in relation to the transportation process (control-flow) facilitates the adaptation of the process, reducing the influence of incidents on the delivery time.

III. Preliminaries

The research reported in this paper builds on a body of existing research in process mining. Although our approach is generic and applicable to any modelling language (e.g., BPMN, EPC, etc.), we opt for Petri nets due to their simple and clear semantics. Our approach uses alignments of Petri nets and event logs for the localisation of context events.

Processes supported by information systems leave a digital footprint in event logs. Events have properties identifying different aspects of the event, such as a timestamp or the corresponding activity. Events from the same process instance are grouped into cases, and multiple cases form an event log.

Definition 1 (Event, Event attributes). *Let $\mathcal{E}$ be the universe of events, i.e., the universe of unique event identifiers. Let $\mathcal{Q}$ be the universe of event properties and let $\mathcal{V}$ be universe of event property values. Let $\pi \in \mathcal{Q} \rightarrow (\mathcal{E} \nrightarrow \mathcal{V})$ be the event property function. For any property $q \in \mathcal{Q}, \pi(q)$ $(denoted\ \pi_q)$ is a partial function mapping events onto values. If $\pi_q(e) = v$, then event $e \in \mathcal{E}$ has a property $q \in \mathcal{Q}$ and the value of this property is $v \in \mathcal{V}$. If $e \notin dom(\pi_q)$, then event e does not have property q and we write $\pi_q(e) = \perp$. Let $\mathcal{A}$ be the universe of activity labels, $label \in \mathcal{Q}$, $dom(\pi_{label}) = \mathcal{E}$, and $rng(\pi_{label}) = \mathcal{A}$, i.e., every event has a label property value.*

Definition 2 (Case, Trace, Event log). *A* case $c \in \mathcal{E}^*$ *is a sequence of events, i.e., $c = \langle e_1, e_2, \ldots, e_n \rangle$. A* trace $\sigma \in \mathcal{A}^*$ *is a sequence of activity labels. We define function $trace$ to project a case onto its trace, i.e., $trace : \mathcal{E}^* \rightarrow \mathcal{A}^*$ and $trace(c) = \langle \pi_{label}(e_1), \pi_{label}(e_s), \ldots, \pi_{label}(e_n) \rangle$. Let $\mathcal{L}$ be the universe of event logs. An* event log $L \in \mathcal{L}$ *is a set of cases, i.e., $L \subseteq \mathcal{E}^*$.*

Definition 3 (Petri net, Petri net edge). *Let $\mathcal{N}$ be the universe of Petri nets. Let $\mathcal{P}$ be the universe of places. Let $\mathcal{T}$ be the universe of transitions. Let $P \subseteq \mathcal{P}$ be a set of places, $T \subseteq \mathcal{T}$ be a set of transitions, and $F \subseteq (P \times T) \cup (T \times P)$ be a flow relation between places and transitions (and between transitions and places). A Petri net $N \in \mathcal{N}$ is a tuple $N = (P, T, F)$. A Petri net edge $f \in F$ is an input edge if $f \in P \times T$ and an output edge if $f \in T \times P$. We define function $label$ as a transition labeling function, i.e., $label \in \mathcal{T} \rightarrow \mathcal{A} \cup \{\tau\}$.*

Definition 4 (Marking, System net). *A marking M is a multiset of places, i.e., $M \in \mathbb{B}(\mathcal{P})$. Let $\mathcal{S}$ be the universe of system nets. A system net $S \in \mathcal{S}$ is a triplet (N, M_{init}, M_{final}) where $N \in \mathcal{N}$ is a Petri net, $M_{init} \in \mathbb{B}(P)$ is the initial marking, and $M_{final} \in \mathbb{B}(P)$ is the final marking.*

Figure 1a shows an example Petri net. Visually, a Petri net consists of squares, circles, and directed edges, which represent the transitions, places, and flow relations, respectively. Transitions represent process activities. The only exceptions are the *invisible* transitions labeled τ, which do not represent pieces of the process' work but are necessary to properly model some types of routing of the process. Places may contain tokens; whereas the structure of the Petri net never changes, tokens are created and consumed. A transition is enabled (the activity it represents is allowed to occur at the current state) if and only if at least one token exists in each input place of the transition. By firing (i.e., executing) a transition, a token is consumed from each input place and a token is produced for each output place. The state of a Petri net is uniquely determined by the distribution of tokens over places, which is denoted as its *marking*. For the Petri net in Fig. 1a, the initial marking is place *start*, and the final marking is place *end*. A complete firing sequence is a sequence of transitions leading from the initial marking to the final marking, indicating a complete execution of a process instance. The set of all complete firing sequences of a system net S is denoted by Ψ_S. For further information on Petri nets in relation to process mining, readers are referred to [1].

Conformance checking aims to verify whether observed behavior recorded in an event log matches behavior described by a process model. The notion of *alignments* provides a robust approach to conformance checking, which makes it possible to pinpoint the deviations causing nonconformity [15]. Building such alignments between an event log and a process model is not trivial, since the log may deviate from the model at an arbitrary number of places. We need to relate "moves" in the log to "moves" in the model. However, it may be that some of the moves in the log cannot be mimicked by the model and vice versa. In other words, not all traces in an event log may be reproducible by the corresponding Petri net, i.e., not all traces may correspond to a complete firing sequence.

Definition 5 (Alignment move). *Let $\mathcal{M}$ be the universe of alignment moves, i.e., the universe of unique alignment move identifiers. We define function λ as follows. Given $m \in \mathcal{M}$:*

- $\lambda_{event}(m) \in \mathcal{E} \cup \{\gg\}$,
- $\lambda_{label}(m) \in \mathcal{A} \cup \{\tau\}$,
- $\lambda_{trans}(m) \in \mathcal{T} \cup \{\gg\}$,

where $\gg$ identifies the absence of respectively an event or a transition, such that

- $\lambda_{event}(m) = \gg \implies \lambda_{trans}(m) \in \mathcal{T}$,
- $\lambda_{trans}(m) = \gg \implies \lambda_{event}(m) \in \mathcal{E}$,
- $\lambda_{event}(m) \neq \gg \implies \lambda_{label}(m) = \pi_{label}(\lambda_{event}(m))$,
- $\lambda_{trans}(m) \neq \gg \implies \lambda_{label}(m) = label(\lambda_{trans}(m))$.

Definition 6 (Alignment, Alignment set). *An alignment $\gamma \in \mathcal{M}^*$ is a finite sequence of alignment moves such that each alignment move appears only once, i.e., $1 \leq i < j \leq |\gamma| : \gamma(i) \neq \gamma(j)$. An alignment set is $\Gamma \subseteq \mathcal{M}^*$ such that each alignment move appears only once. $\mathcal{U}_\gamma$ is the universe of alignments and $\mathcal{U}_\Gamma$ the universe of alignment sets.*

For a given alignment move $m \in \mathcal{M}$, m is considered a synchronous move when $\lambda_{event}(m) \neq \gg \wedge \lambda_{trans}(m) \neq \gg$, a model move when $\lambda_{event}(m) = \gg$ and $\lambda_{label}(m) \neq \tau$, a log move when $\lambda_{trans}(m) = \gg$ and an invisible move when $\lambda_{label}(m) = \tau$. For every alignment $\gamma \in \mathcal{U}_\gamma$ of a system net $S \in \mathcal{S}$ and a case $c \in \mathcal{E}^*$, the projection on λ_{event} yields c, and the projection on λ_{trans} yields a path in Ψ_S.

Multiple alignments are possible for the same case and model. The aim is to find a complete alignment with a minimal number of deviations, also known as an *optimal alignment* [15]. For the sake of space, we assume here that all deviations (i.e., model moves for visible transitions and log moves) have unit cost. In [15], Aalst et al. show how this assumption can be removed. Different alignment techniques exist, which we generalize by the alignment function.

Definition 7 (Alignment function). *The alignment function $align \in \mathcal{L} \times \mathcal{S} \rightarrow \mathcal{U}_\Gamma$ aligns an event log with a system net and returns an alignment set consisting of one optimal alignment for each case in the event log.*

Clearly, different cases may have different (optimal) alignments as they contain different events and deviations. For example, Fig. 3 shows three possible alignments for trace $\langle a, e, b, d, g\rangle$ and the Petri net (system net) in Fig. 1a. Here, γ_1 contains two deviations: one log move (m_{12}) and one model move (m_{15}). Furthermore, γ_2 contains one invisible move (m_{22}), a model move (m_{23}), and two log moves (m_{25} and m_{26}). Lastly, γ_3 contains an invisible move (m_{32}), two model moves (m_{33} and m_{37}), and one log move (m_{34}). Since no alignment exists with less than two asynchronous moves for visible transitions, γ_1 is an optimal alignment for the trace.

IV. Putting Events into Context

In this paper, we argue that not all events should be treated equally. We distinguish two main types of events: control-flow events and context events. We aim to visualize the process model using both event types. The concept of alignments is used to identify where in a process model context events occur. This requires context events to be identified (Section IV-A) and that a mapping is created between the context events and the process model (Section IV-B).

A. Identifying Context Events

In order to identify different types of events, we introduce an event property $type \in \mathcal{Q}$ with $dom(\pi_{type}) = \mathcal{E}$. In other words, every event has an event type. The default value is *control-flow*. This implies that an event is part of the control-flow of the process. During analysis, an analyst may define other *event types* to relate to context events. For example, a purchasing process typically consists of activities such as placing an order, making a payment, receiving a request, and sending goods. These events would be of type *control-flow*. Other events may occur that may *influence* the process, but do not *change* the state of the process. For example, a customer may call to inquire the status of the order, may file a complaint, or may visit the website to check the current prices. These events can be defined as events of type *context*. Furthermore, multiple event types may be identified to distinguish different types of contextual events (e.g., *call*, *complaint*, *website visit*).

$$\gamma_1 = \left|\begin{array}{c|c|c|c|c|c} m_{11} & m_{12} & m_{13} & m_{14} & m_{15} & m_{16} \\ e_{11} & e_{12} & e_{13} & e_{14} & \gg & e_{15} \\ a & e & b & d & e & g \\ t_1 & \gg & t_3 & t_5 & t_6 & t_8 \end{array}\right|$$

$$\gamma_2 = \left|\begin{array}{c|c|c|c|c|c|c} m_{21} & m_{22} & m_{23} & m_{24} & m_{25} & m_{26} & m_{27} \\ e_{11} & \gg & \gg & e_{12} & e_{13} & e_{14} & e_{15} \\ a & \tau & d & e & b & d & g \\ t_1 & t_2 & t_5 & t_6 & \gg & \gg & t_8 \end{array}\right|$$

$$\gamma_3 = \left|\begin{array}{c|c|c|c|c|c|c|c} m_{31} & m_{32} & m_{33} & m_{34} & m_{35} & m_{36} & m_{37} & m_{38} \\ e_{11} & \gg & \gg & e_{12} & e_{13} & e_{14} & \gg & e_{15} \\ a & \tau & d & e & b & d & e & g \\ t_1 & t_2 & t_5 & \gg & t_3 & t_5 & t_6 & t_8 \end{array}\right|$$

Fig. 3: Three example alignments of trace $\langle a, e, b, d, g\rangle$ with the system net in Fig. 1a. Columns represent alignment moves. The rows represents the alignment move identifier, the event identifier, the label, and the transition, from top to bottom.

Process models describe the lifecycle of a single process instance, i.e., a case. As such, events that are not linked to a case cannot be logically represented on a process model. To analyze context events, they need to be included in an event log and linked to a case. A connection is to be made between the context events and the control-flow events. Such connections can be made based on shared event properties (e.g., order id) or on the time period in which the events occurred.

Figure 4 shows the alignments for the traces $\langle a, d, c, g, h\rangle$ and $\langle a, d, h\rangle$ with the Petri net in Fig. 1a. Context events labeled x, y, and z are inserted. Since there are no transitions in the Petri net labeled x, y, or z, these context events thus become log moves in the alignment. In order to distinguish log moves for context events (i.e., events of type *context*) from regular log moves (i.e., log moves for events of type *control-flow*), we introduce alignment move types. The type of the move equals the type of the event. If the move does not correspond to an event, the move is of type *control-flow*.

Definition 8 (Alignment move type). *Let $m \in \mathcal{M}$ be an alignment move. We extend function λ such that:*

- $\lambda_{event}(m) \neq \gg \implies \lambda_{type}(m) = \pi_{type}(\lambda_{event}(m))$
- $\lambda_{event}(m) = \gg \implies \lambda_{type}(m) =$ *control-flow*

$$\gamma_4 = \left|\begin{array}{c|c|c|c|c|c|c|c} m_{41} & m_{42} & m_{43} & m_{44} & m_{45} & m_{46} & m_{47} & m_{48} \\ e_{41} & e_{42} & e_{43} & e_{44} & e_{45} & \gg & e_{46} & e_{47} \\ a & d & x & c & y & e & g & h \\ t_1 & t_5 & \gg & t_4 & \gg & t_6 & t_8 & \gg \end{array}\right|$$

$$\gamma_5 = \left|\begin{array}{c|c|c|c|c|c|c|c} m_{51} & m_{52} & m_{53} & m_{54} & m_{55} & m_{56} & m_{57} & m_{58} \\ e_{51} & e_{52} & e_{53} & e_{54} & \gg & \gg & e_{55} & e_{56} \\ a & d & x & y & \tau & e & h & z \\ t_1 & t_5 & \gg & \gg & t_2 & t_6 & t_9 & \gg \end{array}\right|$$

Fig. 4: Example of alignments with context moves highlighted in gray. In γ_4 (trace $\langle a, d, x, c, y, g, h\rangle$), m_{43}, and m_{45} are of type context. In γ_5 (trace $\langle a, d, x, y, h, z\rangle$), m_{53}, m_{54}, and m_{58} are of type context.

B. Visualizing Context Moves

In existing literature, when projecting alignment data onto Petri nets, model moves are commonly associated with transitions and log moves with places [1]. As such, we have chosen to *associate context moves* (i.e., moves of type context) *to Petri net edges* as to avoid ambiguity about information mapped onto the process model. This design decision has several consequences that need to be addressed.

Firstly, to determine on which edges a context move should be plotted, we need to determine through which transitions a case travels, i.e., what its corresponding firing sequence is. Since context moves occur in-between control-flow moves, it is enough to determine the preceding and succeeding control-flow move in the alignment, for each context move. To do this, we define function *enrich*. Each context move is mapped to exactly one control-flow predecessor move and

one control-flow successor move. If the context move is the first (respectively last) move in the alignment, the predecessor (respectively successor) does not exist and we write $\gg$. Figure 5 shows the results of applying function *enrich* to γ_4 and γ_5.

Definition 9 (Enrich function). *We define function* $enrich$ *to enrich each context move in an alignment with it's preceding and succeeding control-flow alignment move, i.e.:* $enrich : \mathcal{U}_\gamma \rightarrow \mathcal{M} \times (\mathcal{M} \cup \{\gg\}) \times (\mathcal{M} \cup \{\gg\})$.

$$\begin{aligned} enrich(\gamma_4) = &\quad \{(m_{43}, m_{42}, m_{44}), (m_{45}, m_{44}, m_{46})\} \\ enrich(\gamma_5) = &\quad \{(m_{53}, m_{52}, m_{55}), (m_{54}, m_{52}, m_{55}), \\ &\quad (m_{58}, m_{57}, \gg)\} \end{aligned}$$

Fig. 5: Results of $enrich(\gamma_4)$ and $enrich(\gamma_5)$.

Secondly, since transitions might have multiple input and output edges, if we would associate a context move with every outgoing edge of the transition corresponding to the preceding control-flow move and every incoming edge of the transition corresponding to the succeeding control-flow move, we risk increasing the total number of context moves associated with the process model. To avoid this, we assign a weight to every context move. If a context move is mapped onto n edges, this weight is calculated as $\frac{1}{n}$.

Thirdly, the control-flow moves that precede or succeed a context move may be model moves, i.e., the labels represented by their corresponding transitions were expected in the model, but they did not occur in the event log. In this case, we cannot be entirely certain that the context move actually occurred between these two model transitions. We argue that, depending on whether only one of the surrounding control-flow moves in the alignment is a model move, or both, the certainty decreases. This is captured by the function $cert$. Given a context move, its predecessor, and its successor, the certainty is calculated as shown in Fig. 6.

$$cert(m_{cont}, m_{pred}, m_{succ}) = \begin{cases} 0.25 & \lambda_{trans}(m_{pred}) = \gg \wedge\ \lambda_{trans}(m_{succ}) = \gg \\ 0.50 & \lambda_{trans}(m_{pred}) = \gg \wedge\ \lambda_{trans}(m_{succ}) \neq \gg \\ 0.50 & \lambda_{trans}(m_{pred}) \neq \gg \wedge\ \lambda_{trans}(m_{succ}) = \gg \\ 1.00 & \lambda_{trans}(m_{pred}) \neq \gg \wedge\ \lambda_{trans}(m_{succ}) \neq \gg \end{cases}$$

Fig. 6: Calculation of the certainty that a context move has occurred in between two control-flow moves.

We choose this way of calculating weight and certainty for simplicity and ease of use. Other aspects of the model, the event log, or their alignment can be taken into account in order to calculate how certain one can be of a context move occurring between any two transitions of the model. For example, the prefix or postfix of the trace, or even the number of optimal alignments could be used.

Once the context moves are identified and enriched with their surrounding control-flow moves, they can be mapped onto a Petri net model. Algorithm 1 shows how context moves are mapped to Petri net edges. Context moves are plotted as *close as possible* to the location in the process model where they occurred, i.e., as close as possible to the transitions corresponding to the control-flow moves immediately before and after the context move. When a context move is positioned in between two transitions that have a route between them, then it is mapped on the two edges connecting the transitions in the Petri net. In Algorithm 1 this is expressed in the situation where *overlap* exists between the Petri net places related to both transitions. When no such overlap exists, the context move is plotted on all outgoing edges of the transition corresponding to the preceding control-flow move and on all incoming edges of the transition corresponding to the succeeding control-flow move. When the context move does not have a preceding control-flow move, i.e., it occurred before the first control-flow move, then the context move is plotted on the incoming edges of the transition corresponding to the succeeding control-flow move. Finally, when the context move does not have a succeeding control-flow move, it is plotted on the outgoing edges of the transition corresponding to the preceding control-flow move.

Algorithm 1: Map Context Moves onto Petri net edges

Input: $N = (P, T, F) \in \mathcal{N}$
Input: $\Gamma \in \mathcal{U}_\Gamma$
Output: $EF \subseteq F \times \mathcal{M} \times \mathbb{R} \times \mathbb{R}$

$EF \leftarrow \emptyset$
foreach $\gamma \in \Gamma$ **do**
  $EM \leftarrow enrich(\gamma)$
  foreach $(m_{cont}, m_{pred}, m_{succ}) \in EM$ **do**
    $ct \leftarrow cert(m_{cont}, m_{pred}, m_{succ})$
    $t_p \leftarrow \lambda_{trans}(m_{pred})$
    $t_s \leftarrow \lambda_{trans}(m_{succ})$
    $f_{out} \leftarrow \{(t_p, p_p) \in F\}$
    $f_{in} \leftarrow \{(p_s, t_s) \in F\}$
    $overlap \leftarrow \{(t_p, p_p) \in f_{out} \mid \exists (p_s, t_s) \in f_{in} \wedge p_s = p_p\} \cup \{(p_s, t_s) \in f_{in} \mid \exists (t_p, p_p) \in f_{out} \wedge p_p = p_s\}$
    if $|f_{out}| = 0 \wedge |f_{in}| > 0$ **then**
      foreach $f \in f_{in}$ **do**
        $EF \leftarrow EF \cup \{(f, m_{cont}, \frac{1}{|f_{in}|}, ct)\}$
    else if $|f_{out}| > 0 \wedge |f_{in}| = 0$ **then**
      foreach $f \in f_{out}$ **do**
        $EF \leftarrow EF \cup \{(f, m_{cont}, \frac{1}{|f_{out}|}, ct)\}$
    else if $|overlap| > 0$ **then**
      foreach $f \in overlap$ **do**
        $EF \leftarrow EF \cup \{(f, m_{cont}, \frac{1}{|overlap|}, ct)\}$
    else if $|overlap| = 0$ **then**
      foreach $f \in f_{in} \cup f_{out}$ **do**
        $EF \leftarrow EF \cup \{(f, m_{cont}, \frac{1}{|f_{in}|+|f_{out}|}, ct)\}$
return EF

Consider context move m_{43} of alignment γ_4 in Fig. 4. This move corresponds to event e_{43} (label x), and is positioned in between m_{42} (label d, transition t_5), and m_{44} (label c, transition t_4). Both m_{42} and m_{44} are synchronous alignment moves. This implies that the certainty that e_{43} occurred between t_5 and t_4 is equal to 1. In the Petri net (shown in Fig. 1a) there are no shared places between the outgoing edges from transition t_5 and the incoming edges from transition t_4. Following Algorithm 1, context event e_{43} is thus related to edge (p_2, t_5) and edge (p_1, t_4), both with a weight of $\frac{1}{2}$.

V. Evaluation

This section reports on an evaluation of our technique based on two case studies. The first case study, presented in Section V-A, is based on a hospital process in which patients are tested for sepsis. These tests can be executed at any time during the patients stay in the hospital. The second case study, in Section V-B, shows an insurance claim process in which customers can call the insurance company to ask questions regarding their claim. The approach has been implemented in ProM. [1]

A. Case Study Emergency Room Sepsis Process Discovery

This section illustrates how our technique can be beneficial for process discovery [1]. When selected events in the event log are marked as context events, they can be kept aside while executing the process discovery process. Using less activities in process discovery will lead to a less complex result. After a model is discovered, the context events can be plotted on top of this model, highlighting the locations where the context events are most likely to occur. Visualizing context events in this way can identify the best locations for modifying the process model by adding *context activities*, which relates to model repair.

The process used in the case study in this section was introduced by Mannhardt et al. in [16]. The data is publicly available through [17]. The data describes cases that follow a process starting in the emergency room of a hospital and ending when the patient is released from the hospital. The data focuses on activities to detect and handle sepsis. Sepsis is the presence in tissues of harmful bacteria and their toxins, typically through infection of a wound. The data consists of 16 activities leading to 15,214 events for 1,050 cases. The event log has 846 different execution paths (trace variants). Three activities are related to lab tests, i.e., *CRP*, *Lactic Acid* and *Leucocytes*. These activities can be performed anywhere during the process. When events for these activities are removed from the event log, we still have the same 1,050 cases, however, the number of variants is reduced to 182. A reduction of 664 variants by only removing these three activities highlights the complexity caused by considering those events as control-flow events. The same reduction in complexity can be observed when Fig. 7 is compared to Fig. 8. Both figures show the result of a process discovery analysis using the *Directly Follows Miner (DFM)* [18]. While the number of activities is reduced by only three, the number of connections between the transitions of the process model is reduced ten-fold. On the other hand, we have removed information about the execution of the lab tests and it is no longer visible when they can occur.

To provide insights into when the lab tests are executed, we use the hand-made Petri net presented in [16]. Next, we mark the three lab test events in the event log as context events. The context enriched event log is aligned with the model and finally, the lab test events are plotted on top of the model. The resulting enriched process model is shown in Fig. 9. The figure shows the sum of the weights of the

[1]ProM is an extensible tool that supports a wide variety of process mining techniques through plug-ins, see http://www.promtools.org. Our technique is part of the *ContextEvents* package.

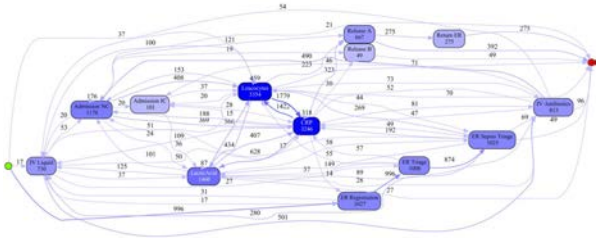


Fig. 7: Result of the DFM miner [18] on sepsis event log [17], using all events and the path slider is set to 80%.

Fig. 8: Result of the DFM miner [18] on sepsis event log [17], not using the lab events and the path slider is set to 80%.

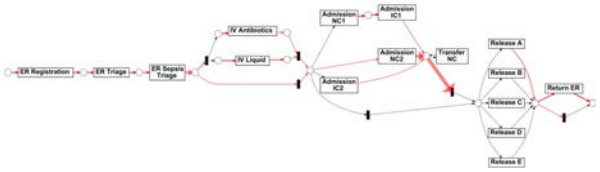


Fig. 9: Absolute context heatmap of sepsis process lab tests.

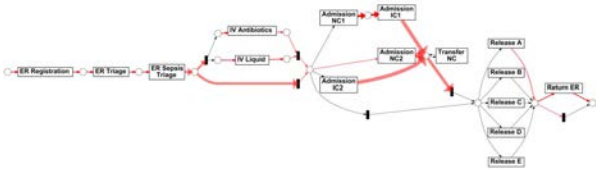


Fig. 10: Relative context heatmap of sepsis process lab tests.

context events on each edge, highlighted in a red color scale. An edge is colored black when the edge does not have any context events mapped to it. The width of the edge indicates the *sum of the weights of all context events* that are related to the edge. The red color intensity of an edge indicates the sum of the certainties of the context events related to that edge. We refer to this visualisation as an *absolute context heat map*. We use the term heatmap because the figure shows the magnitude of a phenomenon, i.e., the occurrence of context events. The variation in color gives a visual cue to the reader about how certain we are about the magnitude.

Fig. 9 shows that most lab tests are executed after *Admission NC2*. Since *Admission NC* is the activity with the highest frequency in the event log it is as expected that lab tests have a high occurrence rate at this point in the process model. We create a so-called *relative context heatmap* to put the cases with context events related to an edge in perspective of the total number of cases that traverse that edge, i.e., the percentage of cases with a context event related to that edge is calculated and plotted.

If we look at the *relative context heatmap*, presented in Fig. 10, a different insight emerges. Relatively, most lab tests

are executed after an admission to a normal or intensive care department, i.e., *Admission NC* and *Admission IC*. Lab tests also occur immediately following the *ER Sepsis Triage* activity when both *IV Antibiotics* and *IV Liquid* are not executed.

We can take this analysis a step further by filtering on cases for which a selected event occurs. In Fig. 11 the data set is filtered on cases having an event that is aligned to the *Admission IC1* transition, and in Fig. 12 only cases that have an event that is aligned to the *Admission IC2* transition. Fig. 11 shows that from *Admission NC1* to *Admission IC1* and immediately after *Admission IC1*, relatively often a lab test is executed. Respectively for 54% of cases after *Admission NC1* and 100% after *Admission IC1*. Fig. 12 shows that after *Admission IC2*, *always* a lab test is executed. Combining Fig. 11 and Fig. 12 reveals that after the admission to an intensive care department a lab test is always executed. Adding a mandatory labtest activity to the process model, immediately after both the *Admission IC* activities, would make it possible to check if the labtest is always executed or not.

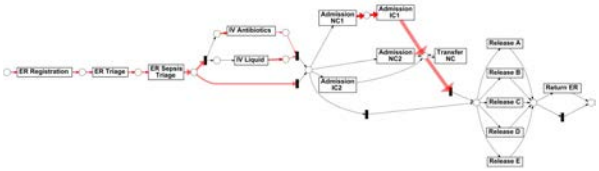


Fig. 11: Relative context heatmap of sepsis process lab tests, filtered on cases that have activity *AdmissionIC1*.

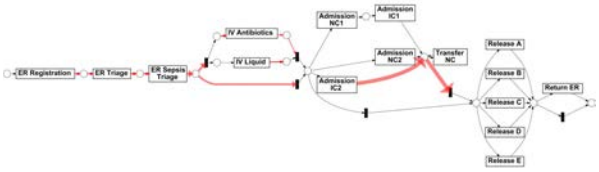


Fig. 12: Relative context heatmap of sepsis process lab tests, filtered on cases that have activity *AdmissionIC2*.

B. Case Study UWV Claim Process

UWV is the social security institute of the Netherlands and responsible for the implementation of a number of employee related insurances. The case study focuses on the unemployment benefits claim process of UWV. When employees become unemployed, they may be entitled to the benefits. Employees have to file a claim at UWV, which then decides whether they are entitled to benefits. When claims are accepted, employees receive benefits with a regular frequency until they find a new job or the maximum period for their entitlements is reached. UWV refers to employees who are making use of their services as customers, therefore we use the term customer in the remainder of the paper.

The unemployment benefits claim process starts at UWV when a claim is received. First, the customer is sent a change form, because the customer is obliged to notify UWV of any changes during the claim handling process. Next a check is performed whether all required information is available. If this is not the case, the customer can be requested to provide the missing information. The request can be done by two types of letters or a phone call. When the information is received, UWV registers a *Document IN* event. If needed a reminder can be send to the customer. When all information is received the decision is made. Finally, the customer is notified of the decision by the appropriate letter.

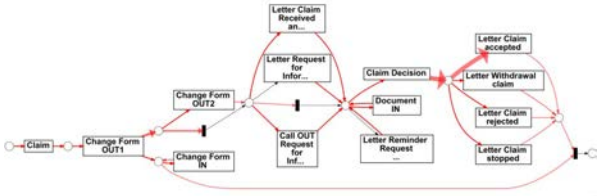


Fig. 13: Absolute context heatmap for UWV claim process.

Before, during and after the claim handling process the customer can contact the UWV call center. Every contact is registered as a *Call IN* event including a description of the questions asked by the customer. The process model used in this case study is obtained from a process specialist at UWV. Fig. 13 shows the absolute context heatmap for the UWV claim process. Incoming calls occur during the whole process execution as can be seen from almost all edges being colored red and not being thin. The most incoming calls occur after the decision is made and before the acceptance letter is sent. Most customers get accepted (on average 80%). The second most frequent location of calls is immediately after the first *Change Form OUT*, which occurs at the start of the process.

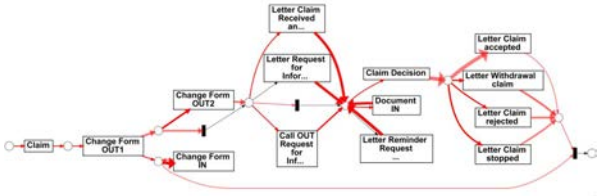


Fig. 14: Relative context heatmap for UWV claim process.

The relative context heatmap in Fig. 14 shows there are several locations that trigger similar amounts of incoming calls. The first location is again between the decision and the acceptance letter. Also, questions come in after information was requested from the customer by sending a letter. Interestingly, when information was requested by phone, a relatively smaller number of customers ask questions. This gives rise to the insight that contacting customers by phone could be beneficial for UWV to reduce the number of incoming calls. Next, just before receiving a *Change Form IN*, questions come in. Finally, after receiving a negative decision, withdrawing the claim or when the claim is stopped, more questions are triggered compared to when the claim is accepted. Providing a customer with adequate information regarding the decision could help reduce the number of customers calling UWV. This analysis shows that knowing when customers call is helpful when advising UWV on how the process could be improved from both the customer and UWV's perspective.

VI. Conclusion

The technique introduced in this paper uses existing process mining concepts such as event logs, process models, and alignments to introduce the concept of context events: events that are part of the event log and can be linked to cases in a process, but are not part of the process' control-flow. Highlighting such events on a process model separately from the control-flow activities provides added value by showing novel insights and facilitating process discovery.

The emergency room sepsis case study in Section V-A shows that keeping selected events aside while doing process discovery leads to less complex results. The events that are first kept aside, can, with our technique, be once more related to the process model. In this way the information contained in the events is used to improve the discovered model. In the UWV case study in Section V-B, customers can call UWV while their unemployment benefits claim is being processed. Identifying the locations in the process model when relatively the most customers call, gives input for improving the customer journey.

We foresee several directions for further research on this topic. For example, up until now, we have only looked at atomic events, i.e., events without duration. Instead, we could also look at durative events, i.e., events that have a start and a complete timestamp. These types of events identify a period in which they are active. Examples of these types of durative event are the caseload of a system and the happiness level of a customer during the process execution. One option of plotting these periods on top of a process model is to use a heatmap, reflecting the average caseload of the system or the average happiness of the customers when the process was executed. In this way, a more complex context can be related to the model.

In this paper, we have shown how to plot a single context event type in relation to a process model. Research could be directed to finding ways to plot multiple context classes on the same process model. This should be done in a way that is still comprehensible to the users of the visualization. A simple yet effective direction would be to use icons attached to the Petri net edges. Different icons may represent different event types.

Instead of having an analyst identify context events, the labeling could be done in a semi- or fully automatic way. One approach would be to use the techniques developed by Tax et al. [8] to identify so-called chaotic activities.

Another possible direction for future work is to use data attributes when plotting context events. In this way, context events can be conditionally plotted or guards could be derived indicating under which circumstances context events occur. Inversely, context events themselves can become input for explanatory data analysis. For example in the work work by de Leoni et al. [19] on process predictions. In the work by Van der Aalst et al. [20] on visualizing token flows using interactive performance spectra the context events can be used as a new classifier to explain the process behavior.

Finally, while we use calculation heavy alignments in our current approach, it is also possible to use the more performance friendly token-based replay.

Acknowledgment

Creating this paper would not have been possible without the help of Maikel Leemans.

References

[1] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[2] M. F. Sani, S. J. van Zelst, and W. M. P. van der Aalst, "Repairing outlier behaviour in event logs," in *Business Information Systems - 21st International Conference, BIS, Berlin, Germany, Proceedings*, 2018.

[3] X. Lu, D. Fahland, R. Andrews, S. Suriadi, M. T. Wynn, A. H. M. ter Hofstede, and W. M. P. van der Aalst, "Semi-supervised log pattern detection and exploration using event concurrence and contextual information," in *OTM 2017 Proceedings, Part I*, 2017, pp. 154–174.

[4] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs containing infrequent behaviour," in *Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers*, 2013, pp. 66–78.

[5] B. F. A. Hompes, J. C. A. M. Buijs, and W. M. P. van der Aalst, "A generic framework for context-aware process performance analysis," in *On the Move to Meaningful Internet Systems: OTM 2016 Conferences Proceedings*, 2016, pp. 300–317.

[6] F. Mannhardt, M. de Leoni, and H. A. Reijers, "The multi-perspective process explorer," in *Proceedings of the BPM Demo Session*, 2015.

[7] P. M. Dixit, H. M. W. Verbeek, J. C. A. M. Buijs, and W. M. P. van der Aalst, "Interactive data-driven process model construction," in *Conceptual Modeling - 37th International Conference, ER 2018, Xi'an, China, October 22-25, 2018, Proceedings*, 2018, pp. 251–265.

[8] N. Tax, N. Sidorova, and W. M. P. van der Aalst, "Discovering more precise process models from event logs by filtering out chaotic activities," *J. Intell. Inf. Syst.*, vol. 52, no. 1, pp. 107–139, 2019. [Online]. Available: https://doi.org/10.1007/s10844-018-0507-6

[9] M. F. Sani, S. J. van Zelst, and W. M. P. van der Aalst, "Repairing outlier behaviour in event logs using contextual behaviour," *Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model.*, vol. 14, pp. 5:1–5:24, 2018. [Online]. Available: https://doi.org/10.18417/emisa.14.5

[10] D. Fahland and W. M. P. van der Aalst, "Model repair - aligning process models to reality," *Inf. Syst.*, vol. 47, pp. 220–243, 2015. [Online]. Available: https://doi.org/10.1016/j.is.2013.12.007

[11] M. L. van Eck, N. Sidorova, and W. M. P. van der Aalst, "Discovering and exploring state-based models for multi-perspective processes," in *Business Process Management Conference Proceedings*, 2016.

[12] M. de Leoni, S. Suriadi, A. H. M. ter Hofstede, and W. M. P. van der Aalst, "Turning event logs into process movies: animating what has really happened," *Software and System Modeling*, vol. 15, no. 3, pp. 707–732, 2016.

[13] P. Soffer, A. Hinze, A. Koschmider, H. Ziekow, C. D. Ciccio, B. Koldehofe, O. Kopp, A. Jacobsen, J. Sürmeli, and W. Song, "From event streams to process models and back: Challenges and opportunities," *Information Systems*, 2017.

[14] S. Mandal, M. Hewelt, and M. Weske, "A framework for integrating real-world events and business processes in an iot environment," in *OTM 2017 Conferences Proceedings, Part I*, 2017, pp. 194–212.

[15] W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 182–192, 2012.

[16] F. Mannhardt and D. Blinde, "Analyzing the trajectories of patients with sepsis using process mining," in *29th International Conference on Advanced Information Systems Engineering (CAiSE)*, 2017, pp. 72–80.

[17] Mannhardt, F. (Felix), "Sepsis cases - event log," 2016. [Online]. Available: https://data.4tu.nl/repository/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460

[18] S. J. J. Leemans, E. Poppe, and M. T. Wynn, "Directly follows-based process mining: Exploration & a case study," in *International Conference on Process Mining, ICPM, Aachen, Germany*, 2019, pp. 25–32.

[19] M. de Leoni, W. M. P. van der Aalst, and M. Dees, "A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs," *Inform. Syst.*, vol. 56, 2016.

[20] W. M. P. van der Aalst, D. Tacke genannt Unterberg, V. Denisov, and D. Fahland, "Visualizing token flows using interactive performance spectra," 2020. [Online]. Available: https://www.youtube.com/watch?v=MkBQ_JXyiVs&feature=youtu.be