An Event Processing Platform for Business Process Management

Nico Herzberg, Andreas Meyer, and Mathias Weske
Business Process Technology Group
Hasso Plattner Institute at the University of Potsdam
Potsdam, Germany
{nico.herzberg, andreas.meyer, mathias.weske}@hpi.uni-potsdam.de

Abstract—The execution of business processes generates a lot of data comprising final process results as well as information about intermediate activities, both communicated as events. Automated process execution environments are centrally controlled by process engines that hold the connection between events and the processes they occur in. In contrast, in manual process execution environments, e.g., logistics, these events may not be correlated to the process they originate from. The correlation information is usually not present in the event but in so-called context data, which exists orthogonally to the corresponding process. However, in the areas of process monitoring and analysis, events need to be correlated to specific process instances. To close the gap between events occurring during process execution and required events with process correlation, we propose a framework that enriches recorded events with context data to create events correlated to processes, so-called process events.

Keywords—Business Process Management, Event, Event Processing, Process Monitoring, Process Analysis, Process Mining, Business Activity Monitoring

I. INTRODUCTION

Many companies face a competitive market environment nowadays and therefore are managed in a process-oriented fashion to be able to react quickly on market changes. Business process management (BPM) combines concepts, methods and techniques to support the design, administration, configuration, enactment, and analysis of these business processes [1]. Central to business process management are process models, as they explicitly describe the operations that need to be carried out to reach the companies’ business goals and are used, among others, for documentation, certification, and enactment. For improving these business process models, it is essential to monitor and analyze the execution of the processes as part of BPM to identify the weak points and potential improvements.

In recent years, the IT support of process execution received much attention and ranges from predominately rudimentary in very manual executing process environments to full-automated process execution by process engines. Whereas logging of events during full-automated process execution is provided by the process engine; in rather manual executing business process environments such a central logging mechanism is missing, although the execution of a process generates a lot of data. On the one hand, this data comprises the actual results of the business process as, for instance, reaching the final destination in a transportation scenario. On the other hand, many data arises from the path towards a process’ final result, e.g., information about landmarks passed during that transportation, and intermediate information about completing single process steps, e.g., loading and unloading the vehicle being used. This information is communicated by events. Unfortunately, these events may not be correlated to the specific process execution they belong to, because the process context may get lost during event capturing [2].

However, process monitoring and analysis rely on events being correlated to a particular business process execution for the techniques and methods applied there. One discipline in the field of process monitoring is business activity monitoring [3], which deals with real-time observation of processes including process progress and expected remaining execution times. Therefore, the main requirement is near-completeness with respect to events to enable confident computation of process progress and predictions and to provide more information for fine-grained process monitoring. In the area of process analysis, process mining is a very important discipline [4]–[7]. Process mining works with event logs – each being an ordered set of events – that are subject of several constraints. An event log needs to be complete with respect to events and correlated to a process. For instance, analyzing an event log being correlated to the same business process allows to derive a process model describing the behavior of this process [7]. If this event log is not complete, major activities may not be contained in the resulting process model.

In this paper, we introduce a framework that addresses the gap between events occurring during process execution and the correlation of them to the corresponding process. Additionally, we reason about the applicability by presenting an architecture, which allows to implement the framework as an continuation of [2]. The main benefits of this framework are bringing together data from manual executed processes with their process descriptions, i.e., process models, and broaden the event basis by making more event data accessible with the help of context data. This allows a more fine-grained process monitoring and analysis.

The remainder of the paper is as follows. Section II introduces a transportation scenario building the foundation to discuss the framework. Subsequently, we will briefly introduce fundamental notions before discussing the framework, illustrated by the scenario, in detail in Section IV. Section V describes the architecture followed by an case study assessing the framework. In Section VII, the related works follows before we conclude the paper in Section VIII.
II. Example

We will introduce the framework to enrich raw events, i.e., events occurring during process execution, towards process events, i.e., events being correlated to a process instance, along a running example from the logistics domain: Transportation of goods from Hamburg harbor terminal into a warehouse in Moscow. This business process contains three activities as presented in the corresponding process model in Fig. 1a with a sub-process in the middle, which handles the actual transportation utilizing a truck (see Fig. 1b). The sub-process Perform trucking utilizes a route plan specifying the way expected to be taken from the truck driver. The route plan can be understood as own process model consisting of a sequence of destinations to be reached. Happen when, for instance, entering Russia as in this scenario. Each sub-route specified in the plan is covered by one loop execution with activity Drive transportation leg until the driver reaches the final destination, in our case the warehouse in Moscow. Admittedly, the given route plan is preliminarily, because due to certain events as, for instance, traffic jam, blocked roads, or weather conditions, the driver might need to deviate from the plan. Before being able to start the transportation, the truck needs to be loaded in the terminal (first activity in Fig. 1a). After transportation, the truck is unloaded and cleared by the warehouse in Moscow (last activity). The described process is taken from a case study further described in Section VI.

The truck company handling this transportation owns three trucks and employs five drivers with each driver having an own on board unit with integrated navigation system, e.g., a smartphone, capable of sending information about the current location. For a transportation process, a driver gets assigned to a truck with respect to the truck usage plan, which lists the driver(s) using a truck in specific time frames (see TABLE I for a sketch).

The logistics company also wants to monitor and analyze the corresponding business processes during execution in real-time as well as in retrospective. Therefore, information about each execution — so called process instances — needs to be collected in terms of events. In the given scenario, the terminal communicates the completion of the truck loading to the logistics company by an electronic message, containing the transport ID, truck information, and data about the load. Further, the logistics organization gets information about the arrival of the truck at the warehouse, when the truck gets unloaded, and the clearance of the truck after completing the unloading. Additionally, the on board unit notifies about location updates by sending a message after each transport leg and after reaching the final destination. These events are correlated to a specific transportation, i.e., process instance, via the transport ID, because the events contain information about the used truck, which in turn is known to the process instance.

In Fig. 1, each node is assigned a life cycle specifying the state transitions an activity reaches during the execution of a process (cf. [1], [8] for the activity life cycle, which we extend to all nodes of the process model). These state transitions are (e)nabled describing the time an activity is ready to be executed, (b)egin describing the actual begin of the activity, and (t)erminated describing the completion of the activity. In the given scenario, activity Perform trucking is enabled as soon as the loading is complete. But this activity only begins, if the driver starts the engine and leaves the terminal.

Some of the state transitions in Fig. 1 are connected to an event, which originates either from the harbor terminal, the warehouse, or the on board unit of the driver. For instance, sending the message from the on board unit about reaching the destination triggers the event destination reached. We refer to these events as raw events.

We distinguish between events that are correlated to a concrete transportation and events that cannot be correlated. For instance, when the truck is loaded at the terminal a message is sent to the truck company electronically containing the transport ID, the time of loading completion, and information about the goods loaded. The event of messaging is represented by the raw event truck loaded and can be easily correlated to the concrete transportation process instance by the transport ID.

<table>
<thead>
<tr>
<th>TransportID</th>
<th>Timestamp Begin</th>
<th>Timestamp End</th>
<th>Driver</th>
<th>Truck</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport1</td>
<td>03.03.12 15:56</td>
<td>09.03.12 12:59</td>
<td>Helmut</td>
<td>Truck3</td>
</tr>
<tr>
<td>Transport58</td>
<td>24.08.12 08:37</td>
<td>30.08.12 18:07</td>
<td>Paul</td>
<td>Truck1</td>
</tr>
<tr>
<td>Transport59</td>
<td>21.09.12 07:58</td>
<td>23.09.12 08:01</td>
<td>Paul</td>
<td>Truck3</td>
</tr>
</tbody>
</table>
Analogously, a similar technique is used at the warehouse, where the begin and completion of loading is captured by scanning the bar code of the transport documents, containing the transport ID and the time when the scan was done. Thereby, the events truck arrived and truck unloaded are created.

Additionally, the organization receives many events from the driver of the truck that do not contain any information about the transport the driver performs, e.g., Paul reached Warsaw at 22:17 on September 21th, 2012. As each driver has its own on board unit and may use different trucks, the unit is not correlated with a specific truck but with the driver owning it. Following, the events originating from such on board unit cannot be correlated to a transportation based on the given information. Thus, the logistics organization does not have any information about the actual transportation of the goods from Hamburg to Moscow. In this paper, we discuss the steps to be taken to correlate events to the corresponding process such that this event information can be used for monitoring and analysis (see Section IV). In our example, the truck usage plan provides information needed by our framework to perform the correlation.

The following questions are, among others, the ones, the logistics company is interested in to answer with respect to an transportation as presented in this scenario: (i) When did the truck arrive at the warehouse parking lot? (ii) How long did the driver need to wait until the unloading could take place? (iii) How long was the trucking time from the terminal in Hamburg to the warehouse in Moscow? (iv) Which route did the driver actually take to reach the warehouse? In contrast to questions (i) to (iv) dealing with post-transportation analysis questions, question (v) deals with real-time information: Which portion of the way from origin to destination is already driven?

These questions cannot be answered using the information from events already correlated to the corresponding process instance. But after enriching the events from the driver’s on board unit with process instance information, the questions can be answered (see also Section VI).

III. PRELIMINARIES

Prior introducing the actual framework to enrich events with process information, we introduce the fundamental concepts utilized in the framework. Essential for our framework are events, which occur during the execution of business processes. They provide valuable insights about the corresponding execution, if they are firstly recorded and secondly processed for further usage.

Definition 1 (Event): An event $e$ is a real-world happening occurring in a particular point in time at a certain place in a certain context.

In the introduced scenario, finishing the loading at the terminal represents an event. This event is triggered in a particular point in time, i.e., when the loading of the truck is finished. The place maps to the location of the terminal and the context of the event comprises the information that the event correlates to a specific driver and that her truck got to be loaded by a specific terminal person. For example, the Truck3 driven by Paul is loaded by Harry at Terminal 2 of Hamburg harbor at 08:09 on September 21st, 2012.

Moreover, reaching a certain location on the path from the terminal to the warehouse also represents an event. Analogously, the event occurs when the destination is reached. The place maps to the current location of the truck, e.g., name of location, and the context of the event comprises the information who is driving the truck and that she follows the planned route to reach the warehouse. For example, the Truck3 driven by Paul operating the transport of three pallets of toys reaches Warsaw at 22:17 on September 21st, 2012.

Having the location data recorded probably does not provide insights into the execution, because these events may not be correlated to the actual transportation. Additional data is required to establish a connection between an execution and recorded events. Referring to our example, the additional data required to correlate the location events to the truck, and therefore the actual process instance (the transportation), is provided by the truck usage plan. Based on the name of the driver contained in the location event and its occurring time, the truck used for the transportation and the corresponding transport ID can be derived. To such additional data, we refer as context data. It exists orthogonal to a business process and contains the information to correlate events occurring during process execution to the corresponding process instance by providing the context in which the event occurs.

Each process instance relates to a process model, which describes all possible options to reach a desired business goal, e.g., transporting goods from Hamburg harbor terminal to a specific warehouse in Moscow. Such transportation comprises several steps as loading the truck, driving via several intermediate locations to the final destination as for instance a warehouse (see Fig. 1). The process instance describes one path through the process model, i.e., one of the options described in the process model is used in a process instance. A process model is defined as follows.

Definition 2 (Process Model): A process model is a tuple $M = (N,F)$, where $N$ is a finite set of nodes and $F \subseteq N \times N$ is the control flow relation.

The constructs used in a process model is subsumed by a number of nodes $N$ and their execution order $F$. These constructs are sufficient to cover commonly used process modeling notations, e.g., BPMN [9], EPC [10], and value chains [11]. In BPMN, which is used for process modeling in this paper, $N$ is partitioned into activities, gateways, and events, whereas $F$ represents the sequence flow among these nodes, i.e., the partial ordering. Thereby, sub-processes as shown in Fig. 1 are considered as activities. The BPMN events only capture a small subset of events occurring during process execution. In this paper, we do not implicitly catch the BPMN events. However, events are handled as all other nodes occurring in a process model such that the event can be caught explicitly through the modeler’s specification.

IV. FRAMEWORK

The framework introduced in this section combines the fundamental concepts described in Section III: Event, process model, and context data. Events occurring during process executions are correlated to the corresponding process instances, which relate to process models, by utilizing context data, which provides the context in which the event occurs. The
correlation comprises three phases: (i) the representation of the occurring event in an information system, (ii) the normalization of this represented event, and (iii) the actual correlation in the process event creation phase. These phases and the corresponding concepts utilized in each phase including their interdependencies to each other are presented in Fig. 2 and will be explained in this section. We distinguish the real-world happening event and an event captured by an information system called event object.

If a recognized real-world event is transferred into an information system, we refer to it as raw event. This capturing is done in the representation phase of our underlying framework. The event is represented by a bit stream or several strings in the database, for instance. As this phase works on the border between the real-world and their representation in information systems, the raw event builds the starting point for the correlation procedure. It is defined as follows.

**Definition 3 (Raw Event):** A raw event is a tuple \( E_R = (\text{raw\_data}) \) where \( \text{raw\_data} \) is the recorded data about the event occurrence. The form of \( \text{raw\_data} \) is not specified.

Each real-world event may be captured several times in different or even the same information system. Considering our transportation scenario, a load event occurs once the truck is loaded at origin, i.e., at the terminal. The representation of these events is handled in a database of the terminal information system landscape. Thus, the raw event in this case is a database record that looks as follows for example: \(<3989, \text{truck3, 2012-09-21T22:17:00, Warsaw}>\). The first field indicates the identifier of the record, the second one the identifier of the truck, and the third field the timestamp. The last three fields of the database entry relate to the kind of load indicating that three pallets of toys are loaded. Analogously, a location update event is raised after reaching a specified location, as mentioned above. This event is recognized by and following stored in the on board unit of the driver. In parallel, the location update event is also transmitted to a database of the logistics company collecting all transport events of all drivers and trucks. Thus, the real-world event is represented by two raw events simultaneously, on the on board unit device and in the database. The representation of the location update in the database looks as follows for example: \(<\text{Paul, 2012-09-21T22:17:00, Warsaw}>\), where the first field represents the on board unit identifier, the second one the timestamp, and the third the location.

Besides the raw event, there exist a second class of event objects, the structured events. While an raw event is unstructured and hard to interpret due to the missing form specification, the structured events have been processed to enable structured access to their content. By the normalization phase of our framework, a raw event respectively its \( \text{raw\_data} \) gets structured and interpreted by forming a normalized event according to its normalized event type definition. Each raw event that is required for further processing has to be transformed to a normalized event, which is a structured event also. However, some of the raw events may not be of interest for further processing such that they do not need to be transformed. For instance, an event is represented identically in different information systems, thus, another identical normalization is not required.

**Definition 4 (Normalized Event):** A normalized event is a tuple \( E_N = (\text{type}, \text{id}, \text{timestamp}, C) \) where \( \text{type} \) is the corresponding normalized event type, \( \text{id} \) is a unique identifier, \( \text{timestamp} \) is the point in time registered for the corresponding raw event, and \( C \) is the event content.

The event content \( C \) is the structured form of the \( \text{raw\_data} \) of the raw event. This data can be structured as key-value-pairs or in a tree structure, e.g., extensible markup language (XML). Each normalized event is an instance of a specific normalized event type, which is defined as follows.

**Definition 5 (Normalized Event Type):** A normalized event type is a tuple \( ET_N = (cd, bind) \), where \( cd \) is the description of the event content and \( bind \) is a function that points to a specific implementation to access the raw event data and to transform it.
with respect to the event content structure as defined by cd. 

The event content description cd formalizes the structure of the event content C of the normalized event \( E_N \), e.g., by defining the attributes (keys) or by an XML schema definition (XSD). The function bind describes in detail how the raw_data need to be accessed and how they need to be transformed to fulfill the event content structure as defined in the event content description cd. The binding function is some kind of a parser for getting requested data out of data sources holding raw events and knowing where exactly in the information system landscape this data could be accessed. Implementing the binding requires knowledge about the structure of the raw event data. The normalized event type definition for the normalized events about the location information stored in the database looks as follows:

```
1 locationUpdate_normalized.cd = {Driver, Location}
2 locationUpdate_normalized.bind = {Timestamp, Driver, Location}
3 FROM db.locations
4 INTO locationUpdate_normalized
5 WHERE id = rawEventId;
6 }
```

Listing 1. Definition of the normalized event type locationUpdate_normalized

In the example (see Listing 1), the binding defines that the raw event data could be found in a database table called db.locations. The extracted data, namely Timestamp, Driver, and Location, are selected to fill the timestamp attribute and the event content of the resulting normalized event. If the raw event data does not contain any timestamp information, because the time when the event occurred is not known or the time information got lost during transmission into the information system, the timestamp attribute needs to be determined by some other logic, e.g., by setting the time when the normalized event is created or the creation time of the raw event in the information system.

Using the normalized event type definition, the location update raw events can be normalized. Such a normalized event looks like \( N_{loc} = (\text{locationUpdate}_{normalized}, 6846, 2012-09-21T22:17:00, \{\text{Driver}=\text{Paul}, \text{Location}=\text{Warsaw}\}) \). 6846 is taken from the raw event as ID for this normalized event according the binding definition of the corresponding normalized event type (see Listing 1 line 6). Referring to the raw event origin from the warehouse the normalized event of type loadFinished content should contain the truck, and information about the load, i.e., quantity, unit, and product. Thus the normalized event for this kind of raw event will look like: \( N_{load} = (\text{loadFinished}_{normalized}, 9796, 2012-09-21T08:09:00, \{\text{Truck}=\text{Truck3}, \text{Quantity}=3, \text{Unit}=\text{pallet}, \text{Product}=\text{toys}\}) \). The ID 9796 is generated for this normalized event.

After normalizing the raw events, they can be enriched towards process events. This happens in the third and final phase of our framework, the process event creation phase. The additional information for enrichment is provided by context data as well as process models.

A process event gets assigned to a certain node in the process model, where it is desired to be recognized. However, positioning of such point on node level only is too coarse-grained, because it is often required to track the begin and the end of an activity for instance. Therefore, we assign a node life cycle to each node of the process model to allow a more fine-grained positioning of the points a process event is expected to happen. Fig. 3 adapts our scenario from Section II and assigns process events to points of the process model.

**Definition 6 (Node Life Cycle):** A node life cycle is a tuple \( L = (S, T) \), where \( S \) is a finite set of node states and \( T \subseteq S \times S \) is a finite set of node state transitions. Let \( M = (N, F) \) be a process model. There exists a function \( \varphi : N \rightarrow L \) that assigns a node life cycle to every node \( n \in N \) of \( M \).

State transitions are the elementary constructs that can be leveraged to position points in the process model where process events could be connected to. A point where a process event is expected on a state transition of a node is called process event monitoring point (PEMP), cf. [8]. The set of all node state transitions of a process model \( M = (N, F) \) is comprised by \( \bigcup_{n \in N} \{(n, t) | t \in T_{\varphi(n)}\} \), each of which could be potentially linked to a process event monitoring point. \( T_{\varphi(n)} \) represents all node state transitions \( t \in T \) returned by function \( \varphi(n) = (S, T) \) for node \( n \in N \).

Referring to Fig. 3, we assigned to each activity of the process model the same node life cycle consisting of the states enabled, running and terminated with the corresponding state transitions (enable, begin and terminate). The gateways get a node life cycle consisting of state transitions (enable) and (terminate) assigned, while events only have the state transition (terminate) once they get fired. In the scenario shown, we apply identical node life cycles to each kind of node, i.e., all activities have the same. However, the solution allows to apply individual node life cycles. A black dot represents a node state transition while the black edges between two black dots represent the node states. In sum, there exist 16 options to position a PEM.

**Definition 7 (Process Event Monitoring Point):** A process event monitoring point is a tuple \( PEMP = (M, n, t) \), where \( M \) is the process model it is contained in, \( n \in N \) is the node it is created for, and \( t \in T \) is the state transition within the node life cycle of \( n \) it is assigned to.

In the given example, we link six PEMPs to the process model. PEMP1 is linked to the terminate node state transition of activity Load truck. PEMP2 is linked to the activity Drive transport leg of the sub-process Perform Trucking. At PEMP3, we wait for several process events relating to it, because the activity is part of a loop structure indicating multiple transportation legs that usually exist in practice. At PEMP3/4, assigned to the terminate state transition of the second XOR-gateway, process events of different types are expected, one for process events indicating that the final destination is not reached, the other one indicating that it is reached. PEMP6 being linked to the end event of the sub-process handles process events also indicating that the final destination is reached. PEMP6 and PEMP7 are linked to the begin respectively terminate node state transitions of activity Unload truck. At the defined PEMPs, process events of specified process event types can be monitored. For one PEM, several process events of different types could be expected, see PEMP3/4 in Fig. 3 for an example.

**Definition 8 (Process Event Type):** A process event type is a tuple \( ET_P = (cd, P, X) \), where \( cd \) is the description of the event content, \( P \) is a set of references to PEMPs the process event type belongs to, and \( X \) is a finite set of extraction rules
that indicate (i) which and how normalized events are utilized to build the process event and (ii) how context data enriches the event information.

A process event type could be assigned to several PEMP\textsubscript{s}. For instance, reaching the final destination, i.e., process event\textsuperscript{9} of type PET\textsubscript{4/5}, is valid for two different PEMP\textsubscript{s}, namely PET\textsubscript{3/4} and PET\textsubscript{5}. If a process event type is connected to several PEMP\textsubscript{s}, for each PEMP a separate process event is created. Each process event is an instance of a specific process event type and is defined as follows.

Definition 9 (Process Event): A process event is a tuple \( E_P = (\text{type}, \text{id}, \text{timestamp}, m_i, n_i, t_i, C) \), where \text{type} is the corresponding process event type, \text{id} is a unique identifier, and \text{timestamp} is the point in time of the process event. \( m_i \) holds the reference to the process instance of the corresponding process model \( m \in M \) the process event is related to. \( n_i \) indicates the node instance and \( t_i \) indicates the instance of the node state transition this process event belongs to. \( C \) is the event content of the process event.

The extraction rules \( X \) given by the process event type point to specific normalized events and context data that are required to form process events. Furthermore, in the extraction rules, it is specified how the single attributes of the process event content, defined by content description \( cd \) of process event type, as well as how the general process event attributes, e.g., timestamp, are filled during process event creation. Within the extraction rules, the correlation of the process event to the corresponding process instance, activity instance, and node state transition instance is handled. Further, it is defined how the timestamp of the process event is computed and set. Either it can be taken over from the underlying normalized events directly, or it may need to be calculated when the normalized events utilized have different timestamps.

In the example transportation process, we define a process event type for the location updates of a concrete transport process instance. The process event type relates to PET\textsubscript{2} in Fig. 3 and looks as follows.

```plaintext
SELECT locationUpdate\_normalized\_Timestamp, 
      locationUpdate\_normalized\_Location, 
      truckUsagePlan\_transportID 
FROM locationUpdate\_normalized\_truckUsagePlan 
INTO locationUpdate\_process 
WHERE truckUsagePlan\_Driver = locationUpdate\_normalized\_Driver 
    AND truckUsagePlan\_Timestamp\_Begin < locationUpdate\_normalized\_Timestamp 
    AND truckUsagePlan\_Timestamp\_End > locationUpdate\_normalized\_Timestamp 
```

Listing 2. Definition of the process event type locationUpdate\_process, which maps to PET\textsubscript{2} in Fig. 3.

In Listing 2, the extraction rules define that the timestamp (is always required) and the location information of the utilized normalized event are used for creating the process event. Furthermore, the process ID, which is identical to the transport ID in our scenario, is determined with the help of the truck usage plan and the corresponding normalized event. The underlying normalized event is correlated to the process instance by looking at the driver’s name the location update comes from and in which time interval it appears (see Listing 2 lines 10 to 14). The determination of the activity instance is hidden and is set implicitly by increasing the counter by one each time the activity Derive transport leg is called.

Referring to our example of a transportation process, we expect several different process events. For instance, process events for the location update correlated to a process instance including the activity instance: \( P_{loc} = (\text{locationUpdate}_{process}, 984131, 2012-09-21T22:17:00, \text{Transport 59}, \text{Perform trucking 1}, \text{begin}) \). \( 984131 \) is a generated ID for the process event. Another process event is expected when the truck loading is finished. For this process event, we are interested in the timestamp information only, because we would like to use it for process progress monitoring. Thus, we do not define any event content information: \( P_{load} = (\text{loadFinished}_{process}, 9796, 2012-09-21T08:09:00, \text{Transport 59}, \text{Load Truck 1}, \text{terminate}) \). 9796 is the ID taken from the underlying normalized event and it is not generated.

Under the assumption that no time is consumed during the control flow between two activities, the enablement of an activity is determined by the termination node state of the
predecessor activity respectively activities. In Fig. 3, node state $e$ of the last activity \textit{Unload truck} is derived from node state $t$ of the preceding activity \textit{Perform trucking}. The assumption aligns with the BPMN specification [9]. However, this assumption may not hold for every process model, e.g., when only the main activities are modeled. In such environments, where time is consumed during the control flow from one activity to the successor activity, a PEMP can be defined that utilizes a process event information intended for another PEMP and add specific rules. In case a gateway exists between two activities, the semantic of the gateway provides the rule for determination of the node state enabled. For example, there might be an AND block before the activity \textit{Unload truck} such that the enablement must be derived from a set of events each stating the termination of a directly preceding activity. Assume, it takes ten minutes to prepare the truck for unloading after reaching the destination. Then, the node state enabled of the activity \textit{Unload truck} will be presented by a PEMP that utilizes process events expected at $PEMP_5$ and adds ten minutes.

V. Architecture

In the following, we will describe the architecture of an event processing platform, which implements the framework introduced above. Fig. 4 provides an overview about the main components of the event processing platform and their relations to each other. The platform comprises three main components: The importer, the query processor, and the core component. The importer works as adapter to get the events to be processed into the platform. The query processor uses an event processing engine of choice, in our case Esper [12], to provide the events requested. Such event processing engines provide, among others, the advantages that they are optimized regarding performance already and are well known by the community. The core component implements most functionality as described in the framework. This includes, among others, triggering and managing aggregation, composition, and correlation of events by the interfaced event processing engine. These components will be explained in detail below.

Additionally, the event processing platform contains a persistence layer encapsulating the database accesses from the functionality. Data needed in the platform can be distinguished in administrative and run time data. Administrative data comprises event types, queries, and process models. Run time data comprises events (raw, normalized as well as process events) and process instances. The event types describe the structure of normalized and process events that can be processed within the platform. Each event type needs to be defined here before it can be used for event processing. The queries can be used for data analysis and more importantly for event processing steps as described in the framework. More details are provided in Section V-C. The information about process models allows the definition of PEMP$s$ directly in the process model. During run time of the event processing platform, the incoming raw events as well the processed normalized events and process events are stored in the database for further processing or usage. To allow the correlation of events to corresponding process instances, information about them is also accessible for the event processing platform.

Finally, the platform provides two interfaces for access: An user interface for configuration, setup and manual usage and a web service interface for the (automated) usage by other information systems. Both interfaces allow full access to the platform and its three main components, i.e., event import as well as event retrieval.

In the remainder of this section, we will explain the platform’s main components importer, query processor, and core in detail.

A. Importer

The importer is used to import the events captured in the real-world into the platform. Thereby, the events may either arrive in a continuous stream as usual in business activity monitoring or be stored in event logs as known from the domain of process mining. Whichever input type is used, the importer takes all events, applies a user defined filter function, and normalizes the remaining ones with respect to the event content description and the binding function described in the framework. This information is received via a request to the core component. Finally, the importer initiates the storage of each normalized event with a request to the core component, which in turn utilizes the persistence layer to execute the request.

B. Query Processor

The query processor works as interface between the core component and the event processing engine of choice, e.g., Esper. The query processor receives a query, either created by the system automatically or created by the user, adapts them towards the specific needs of the engine in use, and initiates their execution there. The query result is received from the engine and forwarded to the core component that will handle the further usage of the results. The queries arriving in the query processor are used to handle the aggregation, composition, and correlation functionality in the core component.

Furthermore, the query processor forwards the events stored in the database to the utilized query processing engine to allow fast and independent query execution. At startup of the engine, the query processor crawls the database via the persistence layer and transfers all stored events to the engine. During run time of the engine, each newly stored event is transferred.
C. Core Component

The core component comprises the main functionalities needed to implement the framework described in Section IV. Thus, it contains the functionality to access the database via the persistence layer, to allow the process event creation, and to create the event queries, which get forwarded to the query processor. As aforementioned, each normalized event received from the importer is stored in the database and additionally, it is forwarded to the query processor, if the query processing engine is running.

Process event creation requires the aggregation and composition of normalized events as well as the correlation of the resulting events to process instances. To cope with these tasks, the core builds general event queries, which are passed to the event processing engine through the query processor for execution. The queries are build upon the specific extraction rules valid for a process event and are derived from the corresponding process model. Using the extraction rules and the results of the event queries, the process event is created and stored in the database.

Additionally, the user can build event queries, e.g., required for business activity monitoring and process mining applications, by using the respective user interface provided by the core component. The user specifies the information, she is interested in, explicitly in the process model and the core component returns the corresponding event queries that can be forwarded to the query processor.

VI. Case Study

Below, we describe the case study the running example introduced in Section II is taken from. While activity Perform Trucking is performed, two data sources provide information for this process fragment, the on board unit event log and the truck usage plan. The on board unit event log contains the name of the driver to identify the on board unit, a timestamp, and a location the driver was at that time, e.g., <21.09.12 22:17, Paul, Warsaw>. The truck usage plan contains the name of the driver who performs the transportation, the truck used for the transportation, as well as the transport ID for the corresponding transportation and its planned begin and end time, e.g., <Transport59, 21.09.12 07:58, 23.09.12 08:01, Paul, Truck3>.

In our case study, we used the process mining tool Disco\(^1\), more specifically its case analysis capabilities. We ran the tool for our raw events from the on board units. The cases based on the raw event data are represented by driver, e.g., in our raw event log, we have five drivers with each representing a single case. Thus, for every driver, we can mine a movement profile, but process mining targeting on the route of a certain transportation cannot be realized, because there are no identifiers about the different transportation executions. The events cannot be correlated to a certain transportation. Fig. 5a shows an excerpt of the case analysis for driver “Paul”, where no differentiation between the single transportations is possible. (b) shows the complete case analysis of transportation 59, where the transportation began in Hamburg, went over Berlin, Warsaw, and Minsk to the final destination Moscow. Note that the locations Minsk and Moscow in (a) and (b) are events of the same origin.

We loaded the raw event log of the on board unit events and the data about the truck usage into our event processing platform. The platform combines these different data sources by using aggregation rules and produces an event log containing process events. Process events in our example are the on board unit events enriched by the information about the truck and the transport ID. Using the resulting event log for process mining, shows a separation by transportation process. In Fig. 5b, the case analysis of transportation 59 is shown. One can see that the transportation began in Hamburg, went over Berlin, Warsaw, Minsk, and finally reaches Moscow. Further, the time between two destinations can be analyzed.

The process mining results shown in Fig. 5 indicate that processing and enriching raw events by our event processing platform delivers value to the case analysis in a process mining tool, e.g., Disco. Mining the raw events can bring up only movement profiles of the drivers, whereas mining of process events allows an analysis separated by transport process instances.

Combining the already existing process events from the terminal and warehouse, which are correlated to a transportation via the transport ID at event retrieval, and the enriched on board unit process events produced by the event processing platform,

\(^1\)http://www.fluxicon.com/disco/
a holistic view on the process execution of a transportation is possible. The analytical questions, mentioned in Section II, can be answered with the information available. (i) When did the truck arrive at the warehouse parking lot? This can be answered by looking at the termination state transition of the sub-process Perform Tracking, as shown in Fig. 1a respectively the end event Destination reached of the sub-process Perform Tracking, as shown in Fig. 1b. (ii) How long did the driver need to wait until the unloading could take place? This information can be calculated by taking the information at $PEMP_5$ = Destination reached and $PEMP_6$ = begin of the activity Unload truck into account, see Fig. 3. (iii) How long was the trucking time from the terminal in Hamburg to the warehouse in Moscow? This can be calculated by using the timestamp information of the corresponding process events relating the end of the activity Load truck and the process event relating to the end of the sub-process Perform trucking. (iv) Which route did the driver actually take to reach the warehouse? This can be answered by mining the produced process event log and analyzing the cases as shown in Fig. 5b. (v) Which portion of the way from origin to destination is already driven? Therefore, the process events about the location updates at $PEMP_2$ can be used and compared to the planned route.

With answering these questions raised by the motivating example, which could not be answered before with the raw data, we show that the approach delivers more insights into the process monitoring and analysis. Raw events as well as context data that were loosely coupled in the past are connected by the approach to provide a high-quality event basis. For some scenario related issues, other techniques, such as batch processing, can be applied; however, this could not establish the process and activity instance correlation. Using the introduced data with the event processing platform shows that there is an automated way to handle different data sources and bring them together to form process events.

VII. RELATED WORK

The event processing techniques used in this paper to create new structured events, such as normalized and process events, are related to complex event processing (CEP). [14] gives an overview about CEP and introduces several techniques and concepts of event processing, e.g., event patterns, rules, and event pattern languages. Such techniques and concepts are relevant for the creation of process events and could be applied in the process event creation phase. [15] defines terms related to CEP that are adopted for the presented approach. [16] discusses CEP as well and lists definitions for CEP-related terms, e.g., event type, that are used in this paper.

A major problem in event processing is the correlation of events to each other but also the correlation of normalized events to a process instance to form process events as presented in this paper. [17] introduces algorithms for the determination of correlation sets based on different attributes of events. They use atomic, conjunctive, and disjunctive correlation conditions but also heuristics to find correlating views.

The presented approach follows the paradigm of Extract, Transform, Load (ETL) known from the data warehouse domain [18]. The extraction in our approach is presented by the function $bind$ defined in the normalized event type to build normalized events, whereas the transformation is handled by bringing the extracted data into a defined structure of a normalized event. The event information is further transformed by creating the process events. Utilizing the process events represents the load step of ETL.

As motivated, the presented approach targets on a high-quality event basis for business process intelligence (BPI). BPI aims to process execution quality management by utilizing features like monitoring and analysis, prediction, control, and optimization [19], [20]. A lot of literature concerns the capturing and storing of business process execution data [20]–[22], however, most of them assume that every process step is recorded and thus, the resulting event log is complete. [19] proposes a reference architecture for BPI, consisting of an integration, a functional, and a visualization layer. With the presented approach, we target on the integration and combination of process execution data, process knowledge, and context data.

Process mining [5] is one application of business process intelligence [4] and profits from the presented framework as well. The created process events could be merged in an event log to allow the extraction of all kinds of process information, e.g., execution times and conformance checks to existing process models. Most process mining techniques assume a complete event log [7] that may not be existing because of the manual process execution and the non-process-aware information systems. This gap could be overcome by using the presented framework as pre-processing step for the existing process mining techniques [6].

Dahaynake et al. [3] present an overview of business activity monitoring (BAM) and categorize BAM systems in four classes: pure BAM, discovery-oriented BAM, simulation-oriented BAM, and reporting-oriented BAM. As these classes base on raw events, the presented approach could be applied in this field as well to enable BAM techniques and methods to provide valuable monitoring results by using process events as input.

Del-Río-Ortega et al. [23] introduced the concept of process performance indicators (PPI), the process related form of key performance indicators, to enable process evaluation. PPIs target on process measurements, e.g., time, costs, and occurrences. Our presented approach could be used as basis for the measurements and the concept of PPIs could be applied on top. This allows that the measurements can already be provided while the process instance is still running. Thus, violations of tolerance thresholds can be mitigated before the process instance failed a PPI, for instance.

VIII. CONCLUSION

We introduced a framework, which allows the transformation of raw events, presented as unstructured bit streams potentially without process correlation, via normalized events to process events. A normalized event represents raw events, which have been put into a schema predefined by the normalized event type. Process events have been enriched with information for correlation to process instances, each being composed of any positive number of normalized events. For the correlation, we reuse process event monitoring points (PEMPs) to capture events occurring at node state transitions.
Furthermore, we introduced an architecture, which allows the implementation of the framework for application in practice. Based on this architecture, a student team developed a prototype, which we then used to apply our framework to the presented case study from the logistics domain. Thereby, we prove that the quality of process mining results gets strongly improved by pre-processing the raw events with the event processing platform to get process events as input.

One main benefit of this framework is to bring together data from manual executed processes and their process descriptions, i.e., process models, to enable process monitoring and process analysis. This includes process mining techniques applied on the conditioned and correlated events provided by the framework’s implementation, e.g., for conformance checking. By applying the framework, also a broader event basis can be provided by making more event data accessible with the help of context data. This allows a more fine-grained process monitoring and analysis.

In future work, we plan to extend the framework with user management capabilities, because not every person or role is supposed to see all events being captured. Additionally, we will fully implement this framework.

ACKNOWLEDGMENT

Thanks to the student team comprising Michael Backmann, Susanne Buelow, Thomas Hille, Benjamin Ulm, and Tsun Yin Wong for many fruitful discussions about the architecture for the technical implementation of the framework and the actual development of the prototype we utilized for the case study.

REFERENCES