Extracting Event Data from Databases to Unleash Process Mining

Wil M.P. van der Aalst

[1] Architecture of Information Systems, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
e-mail:w.m.p.v.d.aalst@tue.nl
[2] International Laboratory of Process-Aware Information Systems, National Research University Higher School of Economics (HSE), 33 Kirpichnaya Street, Moscow, Russia.

Abstract. Increasingly organizations are using process mining to understand the way that operational processes are executed. Process mining can be used to systematically drive innovation in a digitalized world. Next to the automated discovery of the real underlying process, there are process-mining techniques to analyze bottlenecks, to uncover hidden inefficiencies, to check compliance, to explain deviations, to predict performance, and to guide users towards "better" processes. Dozens (if not hundreds) of process-mining techniques are available and their value has been proven in many case studies. However, process mining stands or falls with the availability of event logs. Existing techniques assume that events are clearly defined and refer to precisely one case (i.e. process instance) and one activity (i.e., step in the process). Although there are systems that directly generate such event logs (e.g., BPM/WFM systems), most information systems do not record events explicitly. Cases and activities only exist implicitly. However, when creating or using process models "raw data" need to be linked to cases and activities. This paper uses a novel perspective to conceptualize a *database view on event data*. Starting from a class model and corresponding object models it is shown that events correspond to the creation, deletion, or modification of objects and relations. The key idea is that *events leave footprints by changing the underlying database*. Based on this an approach is described that *scopes*, *binds*, and *classifies* data to create "flat" event logs that can be analyzed using traditional process-mining techniques.

1 Introduction

The spectacular growth of event data is rapidly changing the Business Process Management (BPM) discipline [2, 10, 20, 29, 35, 45, 53]. It makes no sense to focus on modeling, model-based analysis and model-based implementation *without* using the valuable information hidden in information systems [1]. Organizations are competing on analytics and only organizations that intelligently use the vast amounts of data available will survive [5].

Today's main innovations are intelligently exploiting the sudden availability of event data. Out of the blue, "Big Data" has become a topic in board-level discussions. The abundance of data will change many jobs across all industries. Just like computer science emerged as a new discipline from mathematics when computers became abundantly available, we now see the birth of *data science* as a new discipline driven by the torrents of data available in our increasingly digitalized world.[3] The demand for *data scientists* is rapidly increasing. However, the focus on data analysis should not obscure process-orientation. In the end, good processes are more important than information systems and data analysis. The old phrase "It's the process stupid" is still valid. Hence, we advocate the need for *process scientists* that will drive process innovations while exploiting the *Internet of Events* (IoE). The IoE is composed of:

- The *Internet of Content* (IoC): all information created by humans to increase knowledge on particular subjects. The IoC includes traditional web pages, articles, encyclopedia like Wikipedia, YouTube, e-books, newsfeeds, etc.
- The *Internet of People* (IoP): all data related to social interaction. The IoP includes e-mail, facebook, twitter, forums, LinkedIn, etc.
- The *Internet of Things* (IoT): all physical objects connected to the network. The IoT includes all things that have a unique id and a presence in an internet-like structure. Things may have an internet connection or be tagged using Radio-Frequency Identification (RFID), Near Field Communication (NFC), etc.
- The *Internet of Locations* (IoL): refers to all data that have a spatial dimension. With the uptake of mobile devices (e.g., smartphones) more and more events have geospatial attributes.

Note that the IoC, the IoP, the IoT, and the IoL partially overlap. For example, a place name on a webpage or the location from which a tweet was sent. See also Foursquare as a mixture of the IoP and the IoL.

It is not sufficient to just collect event data. The challenge is to exploit it for process improvements. Process mining is a new discipline aiming to address this challenge. *Process-mining techniques form the toolbox of tomorrow's process scientist.* Process mining connects process models and data analytics. It can be used:

- to automatically discover processes without any modeling (not just the control-flow, but also other perspectives such as the data-flow, work distribution, etc.),
- to find bottlenecks and understand the factors causing these bottlenecks,
- to detect and understand deviations, to measure their severity and to assess the overall level of compliance,
- to predict costs, risks, and delays,
- to recommend actions to avoid inefficiencies, and
- to support redesign (e.g., in combination with simulation).

Today, there are many mature process-mining techniques that can be directly used in everyday practice [1]. The uptake of process mining is

[3] We use the term "digitalize" to emphasize the transformational character of digitized data.

not only illustrated by the growing number of papers and plug-ins of the open source tool *ProM*, there are also a growing number of commercial analysis tools providing process mining capabilities, cf. *Disco* (Fluxicon), *Perceptive Process Mining* (Perceptive Software, before Futura Reflect and BPMone by Pallas Athena), *ARIS Process Performance Manager* (Software AG), *Celonis Process Mining* (Celonis GmbH), *ProcessAnalyzer* (QPR), *Interstage Process Discovery* (Fujitsu), *Discovery Analyst* (StereoLOGIC), and *XMAnalyzer* (XMPro).

Despite the abundance of powerful process-mining techniques and success stories in a variety of application domains[4], a limiting factor is the preparation of event data. The Internet of Events (IoE) mentioned earlier provides a wealth of data. However, these data are a not in a form that can be analyzed easily, and need to be extracted, refined, filtered, and converted to event logs first.

The starting point for process mining is an *event log*. Each event in such a log refers to an *activity* (i.e., a well-defined step in some process) and is related to a particular *case* (i.e., a *process instance*). The events belonging to a case are *ordered* and can be seen as one "run" of the process. Event logs may store additional information about events. In fact, whenever possible, process-mining techniques use extra information such as the *resource* (i.e., person or device) executing or initiating the activity, the *timestamp* of the event, or *data elements* recorded with the event (e.g., the size of an order).

If a BPM system or some other process-aware information system is used, then it is trivial to get event logs, i.e., typically the audit trail provided by the system can directly be used as input for process mining. However, in most organizations one encounters information systems built on top of database technology. The IoE depends on a variety of databases (classical relational DBMSs or new "noSQL" technologies). Therefore, we provide a *database view on event data* and assume that events leave footprints by changing the underlying database. Fortunately, database technology often provides so called "redo logs" that can be used to reconstruct the history of database updates. This is what we would like to exploit systematically.

Although the underlying databases are loaded with data, there are no *explicit* references to events, cases, and activities. Instead, there are tables containing records and these tables are connected through key relationships. Hence, the challenge is to convert tables and records into event logs. Obviously, this cannot be done in an automated manner.

To understand why process-mining techniques need "flat event logs" (i.e., event logs with ordered events that explicitly refer to cases and activities) as input, consider any process model in one of the mainstream process modeling notations (e.g., BPMN models, BPEL specifications, UML activity diagrams, and workflow nets). All of these notations present a diagram describing the life-cycle of an instance of the process (i.e., case) in terms of activities. Hence, all mainstream notations require the choice of a single process instance (i.e., case) notion. Notable exceptions are

[4] For example, `http://www.win.tue.nl/ieeetfpm/doku.php?id=shared:process_mining_case_studies` lists over 15 successful case studies in industry.

proclets [7] and artifacts [26], but these are rarely used and difficult to understand by end-users. Therefore, we need to relate raw event data to process instances using a single well-defined view on the process. This explains the requirements imposed on event logs.
In this paper, we focus on the problem of extracting "flat event logs" from databases. First, we introduce process mining in a somewhat more detailed form (Section 2). Section 3 presents *twelve guidelines for logging.* They point to typical problems related to event logs and can be used to improve the recording of relevant events. Although it is vital to improve the quality of logging, this paper aims to exploit the events hidden in existing databases. We use database-centric view on processes: the state of a process is reflected by the database content. Hence, events are merely changes of the database. In the remainder we assume that data is stored in a database management system and that we can see all updates of the underlying database. This assumption is realistic (see e.g. the redo logs of Oracle). However, how to systematically approach the problem of converting database updates into event logs? Section 4 introduces *class* and *object models* as a basis to reason about the problem. In Section 5 we show that class models can be extended with a so-called *event model.* The event model is used to capture changes of the underlying database. Section 6 describes a three-step approach (*Scope*, *Bind*, and *Classify*) to create a collection of flat event logs. The results serve as input for conventional process-mining techniques. Section 7 discusses related work and Section 8 concludes this paper.

2 Process Mining

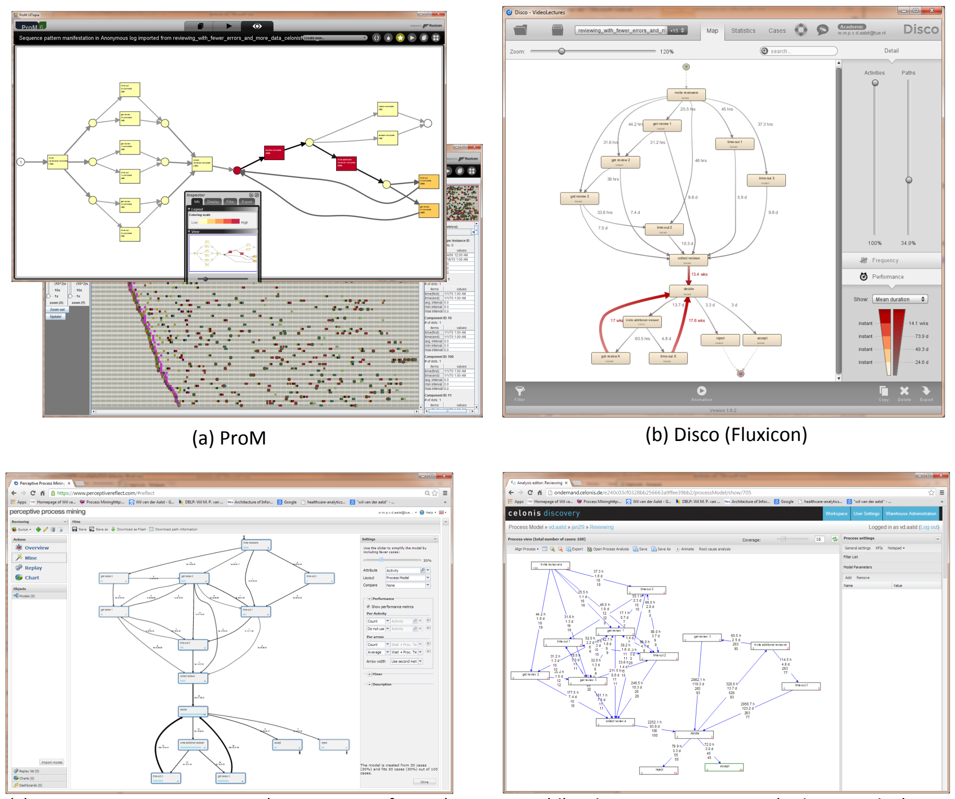
Process mining aims to *discover, monitor and improve real processes by extracting knowledge from event logs* readily available in today's information systems [1].
Normally, "flat" *event logs* serve as the starting point for process mining. These logs are created with a particular process and a set of questions in mind. An event log can be viewed as a multiset of *traces.* Each trace describes the life-cycle of a particular *case* (i.e., a *process instance*) in terms of the *activities* executed. Often event logs store additional information about events. For example, many process-mining techniques use extra information such as the *resource* (i.e., person or device) executing or initiating the activity, the *timestamp* of the event, or *data elements* recorded with the event (e.g., the size of an order). Table 1 shows a small fragment of a larger event log. Each row corresponds to an event. The events refer to two cases (654423 and 655526) and have additional properties, e.g., the registration for case 654423 was done by John at two past eleven on April 30th 2014 and the cost was 300 euro. An event may also contain transactional information, i.e., it may refer to an "assign", "start", "complete", "suspend", "resume", "abort", etc. action. For example, to measure the duration of an activity it is important to have a start event and a complete event. We refer to the *XES standard* [38] for more information on the data possibly available in event logs.
Flat event logs such as the one shown in Table 1 can be used to conduct four types of process mining [1].

Table 1. A fragment of an event log: each line corresponds to an event.

case id	timestamp	activity	resource	cost
654423	30-04-2014:11.02	register request	John	300
654423	30-04-2014:11.06	check completeness of documents	Ann	400
655526	30-04-2014:16.10	register request	John	200
655526	30-04-2014:16.14	make appointment	Ann	450
654423	30-04-2014:11.12	ask for second opinion	Pete	100
654423	30-04-2014:11.18	prepare decision	Pete	400
654423	30-04-2014:11.19	pay fine	Pete	400
655526	30-04-2014:16.26	check completeness of documents	Sue	150
655526	30-04-2014:16.36	reject claim	Sue	100
...	...	...	...	...

- The first type of process mining is *discovery*. A discovery technique takes an event log and produces a model without using any a priori information. Process discovery is the most prominent process-mining technique. For many organizations it is surprising to see that existing techniques are indeed able to discover real processes merely based on example behaviors stored in event logs.
- The second type of process mining is *conformance*. Here, an existing process model is compared with an event log of the same process. Conformance checking can be used to check if reality, as recorded in the log, conforms to the model and vice versa.
- The third type of process mining is *enhancement*. Here, the idea is to extend or improve an existing process model by directly using information about the actual process recorded in some event log. Whereas conformance checking measures the alignment between model and reality, this third type of process mining aims at changing or extending the a priori model. For instance, by using timestamps in the event log one can extend the model to show bottlenecks, service levels, and throughput times.
- The fourth type of process mining is *operational support*. The key difference with the former three types is that analysis is not done offline, but used to influence the running process and its cases in some way. Based on process models, either discovered through process mining or (partly) made by hand, one can check, predict, or recommend activities for running cases in an online setting. For example, based on the discovered model one can predict that a particular case will be late and propose counter-measures.

The *ProM* framework provides an open source process-mining infrastructure. Over the last decade hundreds of plug-ins have been developed covering the whole process-mining spectrum. *ProM* is intended for process-mining experts. Non-experts may have difficulties using the tool due to its extensive functionality. Commercial process-mining tools such as *Disco*, *Perceptive Process Mining*, *ARIS Process Performance Manager*, *Celonis Process Mining*, *QPR ProcessAnalyzer*, *Fujitsu Interstage Process Discovery*, *StereoLOGIC Discovery Analyst*, and *XMAnalyzer* are typically easier to use because of their restricted functionality. These



(a) ProM (b) Disco (Fluxicon)

(c) Perceptive Process Mining (Perceptive Software) (d) Celonis Process Mining (Celonis GmbH)

Fig. 1. Four screenshots of different tools analyzing the same event log.

tools have been developed for practitioners, but provide only a fraction of the functionality offered by *ProM*. Figure 1 shows four screenshots of process-mining tools analyzing the same event log.
In this paper, we neither elaborate on the different process-mining techniques nor do we discuss specific process-mining tools. Instead, we focus on the event data used for process mining.

3 Guidelines for Logging

The focus of this paper is on the input side of process mining: *event data*. Often we need to work with the event logs that happen to be available, and there is no way to influence what events are recorded and how they are recorded. There can be various problems related to the structure and quality of data [1, 19]. For example, timestamps may be missing or too coarse (only dates). Therefore, this paper focuses on the "input side of process mining". Before we present our database-centric approach, we introduce *twelve guidelines for logging*. These guidelines make no assumptions on the underlying technology used to record event data.

In this section, we use a rather loose definition of event data: events simply refer to "things that happen" and that they are described by *references* and *attributes*. *References* have a *reference name* and an *identifier* that refers to some object (person, case, ticket, machine, room, etc.) in the universe of discourse. *Attributes* have a *name* and a *value*, e.g., *age=48* or *time="28-6-2014 03:14:0"*. Based on these concepts we define our twelve guidelines. To create an event log from such "raw events" (1) we need to select the events relevant for the process at hand, (2) events need to be correlated to form process instances, (3) events need to be ordered using timestamp information, and (4) event attributes need to be selected or computed based on the raw data (resource, cost, etc.). Such an event log can be used as input for a wealth of process-mining techniques.

The guidelines for logging (**GL1**-**GL12**) aim to create a good starting point for process mining.

GL1 *Reference and attribute names should have clear semantics, i.e., they should have the same meaning for all people involved in creating and analyzing event data.* Different stakeholders should interpret event data in the same way.

GL2 *There should be a structured and managed collection of reference and attribute names.* Ideally, names are grouped hierarchically (like a taxonomy or ontology). A new reference or attribute name can only be added after there is consensus on its value and meaning. Also consider adding domain or organization specific extensions (see for example the extension mechanism of XES [38]).

GL3 *References should be stable (e.g., identifiers should not be reused or rely on the context).* For example, references should not be time, region, or language dependent. Some systems create different logs depending on the language settings. This is unnecessarily complicating analysis.

GL4 *Attribute values should be as precise as possible. If the value does not have the desired precision, this should be indicated explicitly (e.g., through a qualifier).* For example, if for some events only the date is known but not the exact timestamp, then this should be stated explicitly.

GL5 *Uncertainty with respect to the occurrence of the event or its references or attributes should be captured through appropriate qualifiers.* For example, due to communication errors, some values may be less reliable than usual. Note that uncertainty is different from imprecision.

GL6 *Events should be at least partially ordered. The ordering of events may be stored explicitly (e.g., using a list) or implicitly through an attribute denoting the event's timestamp.* If the recording of timestamps is unreliable or imprecise, there may still be ways to order events based on observed causalities (e.g., usage of data).

GL7 *If possible, also store transactional information about the event (start, complete, abort, schedule, assign, suspend, resume, withdraw, etc.).* Having start and complete events allows for the computation of activity durations. It is recommended to store activity references to be able to relate events belonging to the same activity instance. Without activity references it may not always be clear which events belong together, which start event corresponds to which complete event.

GL8 *Perform regularly automated consistency and correctness checks to ensure the syntactical correctness of the event log.* Check for missing references or attributes, and reference/attribute names not agreed upon. Event quality assurance is a continuous process (to avoid degradation of log quality over time).

GL9 *Ensure comparability of event logs over time and different groups of cases or process variants.* The logging itself should not change over time (without being reported). For comparative process mining, it is vital that the same logging principles are used. If for some groups of cases, some events are not recorded even though they occur, then this may suggest differences that do not actually exist.

GL10 *Do not aggregate events in the event log used as input for the analysis process.* Aggregation should be done during analysis and not before (since it cannot be undone). Event data should be as "raw" as possible.

GL11 *Do not remove events and ensure provenance. Reproducibility is key for process mining.* For example, do not remove a student from the database after he dropped out since this may lead to misleading analysis results. Mark objects as not relevant (a so-called "soft delete") rather than deleting them: concerts are not deleted - they are canceled, employees are not deleted - they are fired, etc.

GL12 *Ensure privacy without losing meaningful correlations.* Sensitive or private data should be removed as early as possible (i.e., before analysis). However, if possible, one should avoid removing correlations. For example, it is often not useful to know the name of a student, but it may be important to still be able to use his high school marks and know what other courses he failed. Hashing can be a powerful tool in the trade-off between privacy and analysis.

The above guidelines are very general and aim to improve the logging itself. The main purpose of the guidelines is to point to problems related to the input of process mining. They can be used to better instrument software.

After these general guidelines, we now change our viewpoint. We aim to exploit the hidden event data already present in databases. The content of the database can be seen as the current state of one or more processes. Updates of the database are therefore considered as the primary events. This database-centric view on event logs is orthogonal to the above guidelines.

4 Class and Object Models

Most information systems do not record events explicitly. Only process-aware information systems (e.g., BPM/WFM systems) record event data in the format shown in Table 1. To create an event log, we often need to gather data from different data sources where events exist only implicitly. In fact, for most process-mining projects event data need to be extracted from conventional databases. This is often done in an ad-hoc manner. Tools such as *XESame* [49] and *ProMimport* [34] provide some support, but still the event logs need to be constructed by querying the database and converting database records (row in tables) into events.
Moreover, the "regular tables" in a database only provide the *current state* of the information system. It may be impossible to see when a record was created or updated. Moreover, deleted records are generally invisible.[5] *Taking the viewpoint that the database reflects the current state of one or more processes, we define all changes of the database to be events.* Below we conceptualize this viewpoint. Building upon standard class and object models, we define the notion of an *event model*. The event model relates coherent set of changes to the underlying database to events used for process mining.
Section 5 defines the notion of an event model. To formalize event models, we first introduce and define class and object models.
A *class model* defines a set of classes that may be connected through relationships. UML class models [43], Entity-Relationship (ER) models [25], Object-Role Modeling (ORM) models, etc. provide concrete notations for the basic class model used in this paper.

Definition 1 (Unconstrained Class Model). *Assume V to be some universe of values (strings, numbers, etc.). An unconstrained* class model *is a tuple $UCM = (C, A, R, val, key, attr, rel)$ such that*
- *C is a set of class names,*
- *A is a set of attribute names,*
- *R is a set of relationship names ($C \cap R = \emptyset$),*
- *$val \in A \to \mathcal{P}(V)$ is a function mapping each attribute onto a set of values.*[6] *$V_a = val(a)$ is a shorthand and denotes the set of possible values of attribute $a \in A$,*
- *$key \in C \to \mathcal{P}(A)$ is a function describing the set of key attributes of each class,*
- *$attr \in C \to \mathcal{P}(A)$ is a function describing the set of additional attributes of each class ($key(c) \cap attr(c) = \emptyset$ for any class $c \in C$),*
- *$rel \in R \to (C \times C)$ is a function describing the two classes involved in a relation. Let $rel(r) = (c_1, c_2)$ for relationship $r \in R$: $rel_1(r) = c_1$ and $rel_2(r) = c_2$ are shorthand forms to obtain the two individual classes involved in the relationship.*

[5] Increasingly systems mark deleted objects as not relevant (a so-called soft delete) rather than deleting them. In this way all intermediate states of the database can be reconstructed. Moreover, marking objects as deleted instead of completely removing them from the database is often more natural, e.g., concerts are not deleted – they are canceled, employees are not deleted – they are fired, etc.

[6] $\mathcal{P}(X)$ is the powerset of X, i.e., $Y \in \mathcal{P}(X)$ if $Y \subseteq X$.

Figure 2 shows a class model with classes $C = \{c_1, c_2, \ldots, c_8\}$ and relationships $R = \{r_1, r_2, \ldots, r_8\}$. Classes and relationships also have longer names, e.g., c_1 is the class "concert hall". We will use the shorter names for a more compact discussion. In this example, each class has a singleton key, i.e., a single column serves as primary key. The keys are highlighted in Figure 2 (darker color). For example, $key(c_1) = \{hall\_id\}$ and $attr(c_1) = \{name\_of\_hall, address\}$ are the two additional (non-key) attributes of class c_1. $rel(r_4) = (c_5, c_2)$, i.e., relation r_4 relates tickets (c_5) to concerts (c_2). Figure 2 also shows cardinality constraints. These are not part of the unconstrained class model. Later we will define *constrained* class models (Definition 4). However, before doing so, we need to introduce some more notations.

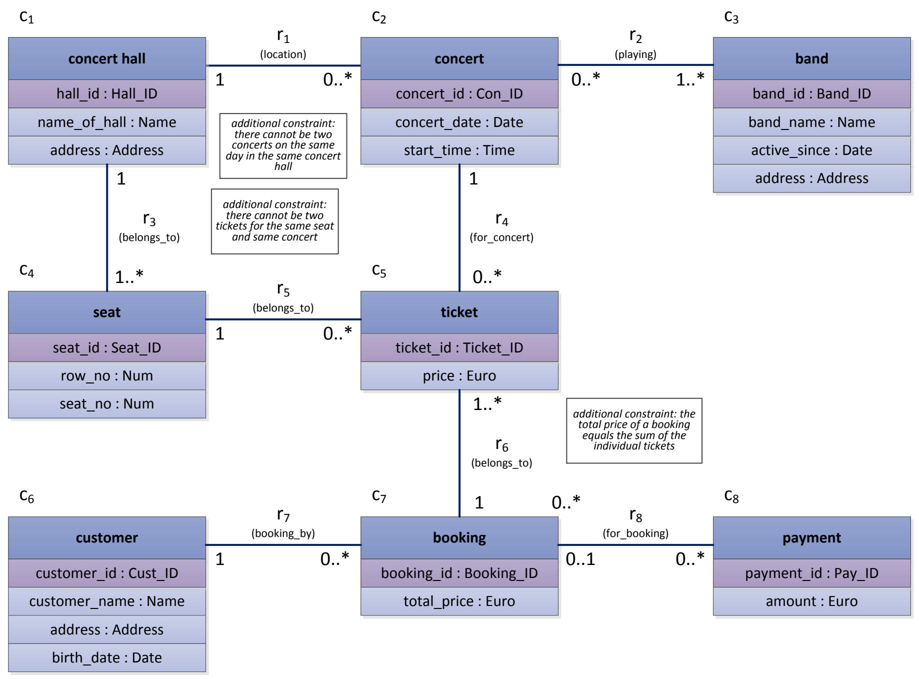


Fig. 2. Example of a constrained class model.

Definition 2 (Notations). *Let* $CM = (C, A, R, val, key, attr, rel)$ *be an (unconstrained) class model.*

– $M^{CM} = \{map \in A \not\rightarrow V \mid \forall_{a \in dom(map)}\ map(a) \in V_a\}$ *is the set of mappings,*[7]

[7] $f \in X \not\rightarrow Y$ is a partial function, i.e., the domain of f may be any subset of X: $dom(f) \subseteq X$.

- $K^{CM} = \{(c, map_k) \in C \times M^{CM} \mid dom(map_k) = key(c)\}$ *is the set of possible key values per class,*
- $A^{CM} = \{(c, map_a) \in C \times M^{CM} \mid dom(map_a) = attr(c)\}$ *is the set of possible additional attribute values per class,*
- $O^{CM} = \{(c, map_k, map_a) \in C \times M^{CM} \times M^{CM} \mid (c, map_k) \in K^{CM} \wedge (c, map_a) \in A^{CM}\}$ *is the set of objects,*
- $R^{CM} = \{(r, map_1, map_2) \in R \times M^{CM} \times M^{CM} \mid \exists_{c_1, c_2 \in C}\ rel(r) = (c_1, c_2) \wedge \{(c_1, map_1), (c_2, map_2)\} \subseteq K^{CM}\}$ *is the set of potential relations.*

A class model implicitly defines a collection of possible *object models*. Each class $c \in C$ may have multiple objects and each relationship $r \in R$ may hold multiple concrete object-to-object relations.

Definition 3 (Object Model). *Let $CM = (C, A, R, val, key, attr, rel)$ be an (unconstrained) class model. An* object model *of CM is a tuple $OM = (Obj, Rel)$ where $Obj \subseteq O^{CM}$ is a set of objects and $Rel \subseteq R^{CM}$ is a set of relations. $\mathcal{U}^{OM}(CM) = \{(Obj, Rel) \mid Obj \subseteq O^{CM} \wedge Rel \subseteq R^{CM}\}$ is the set of all object models of CM.*

The cardinality constraints in Figure 2 impose restrictions on object models. For example, a ticket corresponds to precisely one concert and each concert corresponds to any number of tickets (see annotations "1" and "0..*" next to r_4). Each ticket corresponds to precisely one booking and each booking refers to at least one ticket (see annotations "1" and "1..*" next to r_6). In our formalizations we abstract from the actual notation used to specify constraints. Instead, we assume a given set VOM of valid object models satisfying all requirements (including cardinality constraints).

Definition 4 (Constrained Class Model). *A* constrained *class model is a tuple $CM = (C, A, R, val, key, attr, rel, VOM)$ such that $UCM = (C, A, R, val, key, attr, rel)$ is an unconstrained class model and $VOM \subseteq \mathcal{U}^{OM}(UCM)$ is the set of* valid object models. *A valid object model $OM = (Obj, Rel) \in VOM$ satisfies all (cardinality) constraints including the following general requirements:*
- *for any $(r, map_{k1}, map_{k2}) \in Rel$ there exist c_1, c_2, map_{a1}, and map_{a2} such that $rel(r) = (c_1, c_2)$ and $\{(c_1, map_{k1}, map_{a1}), (c_2, map_{k2}, map_{a2})\} \subseteq Obj$, i.e., the referenced objects exist,*
- *for any $\{(c, map_k, map_{a1}), (c, map_k, map_{a2})\} \subseteq Obj$: $map_{a1} = map_{a2}$, i.e., keys are indeed unique.*

All notations defined for unconstrained class models are also defined for constrained class models. For any valid object model $OM \in VOM$ it is ensured that relations refer to existing objects and that there are not two objects in the same class that have the same key values. Moreover, all cardinality constraints are satisfied if $OM \in VOM$.
Definition 4 abstracts from the concrete realization of object and class models in a database. However, it is easy to map any class model onto a set of related tables in a conventional relational database system. To do this foreign keys need to be added to the tables or additional tables

need to be added to store the relationships. For example, one may add three extra columns to the table for c_5 ("ticket"): *concert_id* (for the foreign key relating the ticket to a concert), *seat_id* (for the foreign key relating the ticket to a seat), and *booking_id* (for the foreign key relating the ticket to a booking). These columns realize respectively r_4, r_5, and r_6. In the case of a many-to-many relationship an additional table needs to be added to encode the relations. In the remainder we abstract from the actual table structure, but it is obvious that the conceptualization agrees with standard database technology.

5 Events and Their Effect on the Object Model

Examples of widely used *DataBase Management Systems* (DBMSs) are *Oracle RDBMS* (Oracle), *SQL server* (Microsoft), *DB2* (IBM), *Sybase* (SAP), and *PostgreSQL* (PostgreSQL Global Development Group). All of these systems can store and manage the data structure described in Definition 4. Moreover, all of these systems have facilities to *record changes* to the database. For example, in the Oracle RDBMS environment, *redo logs* comprise files in a proprietary format which log a history of all changes made to the database. *Oracle LogMiner*, a utility provided by Oracle, provides methods of querying logged changes made to an Oracle database. Every *Microsoft SQL Server* database has a *transaction log* that records all database modifications. *Sybase IQ* also provides a transaction log. Such redo/transaction logs can be used to recover from a system failure. The redo/transaction logs will grow significantly if there are frequent changes to the database. In such cases, the redo/transaction logs need to be truncated regularly.
This paper does not focus on a particular DBMS. However, we assume that through redo/transaction logs we can monitor changes to the database. In particular, we assume that we can see when a record is inserted, updated, or deleted. Conceptually, we assume that we can see the *creation* of objects and relations (denoted by $\oplus$), the *deletion* of objects and relations (denoted by $\ominus$), and *updates* of objects (denoted by $\oslash$). Based on this we define the set of *atomic* and *composite event types.*

Definition 5 (Event Types). *Let $CM = (C, A, R, val, key, attr, rel, VOM)$ be a constrained class model. $ET_{atomic} = ET_{add,obj} \cup ET_{add,rel} \cup ET_{del,obj} \cup ET_{del,rel} \cup ET_{upd,obj}$ is the set of* atomic event types *composed of the following pairwise disjoint sets:*

- *$ET_{add,obj} = \{(\oplus, c) \mid c \in C\}$ are the event types for adding objects,*
- *$ET_{add,rel} = \{(\oplus, r) \mid r \in R\}$ are the event types for adding relations,*
- *$ET_{del,obj} = \{(\ominus, c) \mid c \in C\}$ are the event types for deleting objects,*
- *$ET_{del,rel} = \{(\ominus, r) \mid r \in R\}$ are the event types for deleting relations, and*
- *$ET_{upd,obj} = \{(\oslash, c) \mid c \in C\}$ are the event types for updating objects.*

$ET_{composite}(CM) = \mathcal{P}(ET_{atomic}) \setminus \{\emptyset\}$ is the set of all possible composite event types *of CM.*

The atomic event type $(\oplus, c_5)$ denotes the creation of a ticket and $(\oplus, r_8)$ denotes the linking of a payment to a booking. When updating the address of a customer, the atomic event type $(\oslash, c_6)$ is expected to occur.

When preparing for a new concert of an existing band in an existing concert hall, we may observe the composite event type $\{(\oplus, c_2), (\oplus, r_1), (\oplus, r_2)\}$, i.e., creating a new object for the concert and relating it to the existing concert hall and band.
The notion of atomic/composite event types naturally extends to concrete atomic/composite events. For an object creation event $(\oplus, c)$ we need to specify (map_k, map_a), i.e., the new key and additional attribute values. For deleting a relation $(\ominus, r)$ we need to specify (map_1, map_2), i.e., the key values of each of the two objects involved in the relation.

Definition 6 (Events). *Let $CM = (C, A, R, val, key, attr, rel, VOM)$ be a constrained class model. $E_{atomic} = E_{add,obj} \cup E_{add,rel} \cup E_{del,obj} \cup E_{del,rel} \cup E_{upd,obj}$ is the set of atomic events composed of the following pairwise disjoint sets:*
- *$E_{add,obj} = \{(\oplus, c, (map_k, map_a)) \mid (c, map_k, map_a) \in O^{CM}\}$,*
- *$E_{add,rel} = \{(\oplus, r, (map_1, map_2)) \mid (r, map_1, map_2) \in R^{CM}\}$,*
- *$E_{del,obj} = \{(\ominus, c, map_k) \mid (c, map_k) \in K^{CM}\}$,*
- *$E_{del,rel} = \{(\ominus, r, (map_1, map_2)) \mid (r, map_1, map_2) \in R^{CM}\}$, and*
- *$E_{upd,obj} = \{(\oslash, c, (map_k, map_a)) \mid (c, map_k, map_a) \in O^{CM}\}$.*

$E_{composite}(CM) = \mathcal{P}(E_{atomic}) \setminus \{\emptyset\}$ is the set of all possible composite events *of CM. $fprt \in E_{atomic} \rightarrow ET_{atomic}$ is a function computing the footprint of an atomic event: $fprt((x, y, z)) = (x, y)$ maps an atomic event $(x, y, z) \in E_{atomic}$ onto its corresponding type $(x, y) \in ET_{atomic}$. The footprint function is generalized to composite events, i.e., $fprt \in E_{composite} \rightarrow ET_{composite}$ such that $fprt(CE) = \{(x, y) \mid (x, y, z) \in CE\}$ for composite event CE.*

E_{atomic} is the set of atomic events. $E_{composite}(CM)$ is the set of non-empty composite events. *fprt* transforms atomic/composite events into the corresponding types. For example, $fprt((\oplus, r, (map_1, map_2))) = (\oplus, r)$.
An *event model* annotates a constrained class model with event types that refer to composite events. Figure 3 shows an event model that has seven events. Event en_3 models the deletion of a customer. The corresponding composite event type is $\{(\ominus, c_6)\}$. Event en_4 models the adding of a concert. The corresponding composite event type is $\{(\oplus, c_2), (\oplus, r_1), (\oplus, r_2)\}$.

Definition 7 (Event Model). *Let $CM = (C, A, R, val, key, attr, rel, VOM)$ be a constrained class model. An* event model *is a tuple $EM = (EN, type, VE)$ where*
- *EN is a set of event names,*
- *$type \in EN \rightarrow ET_{composite}(CM)$ is a function mapping each event name onto its composite event type,*
- *$VE \subseteq EN \times E_{composite}(CM)$ is the set of* valid events *such that for any $(en, CE) \in VE$: $fprt(CE) = type(en)$. Moreover, for any $en \in EN$ there exists a CE such that $(en, CE) \in VE$.*

Events should be of the right type and for each event name there is at least one valid event. Note that events may have varying cardinalities, e.g., one event may create five objects of the same class.
In Definition 7, we require $fprt(CE) = type(en)$. Alternatively, one could weaken this requirements to $\emptyset \neq fprt(CE) \subseteq type(en)$. This would allow

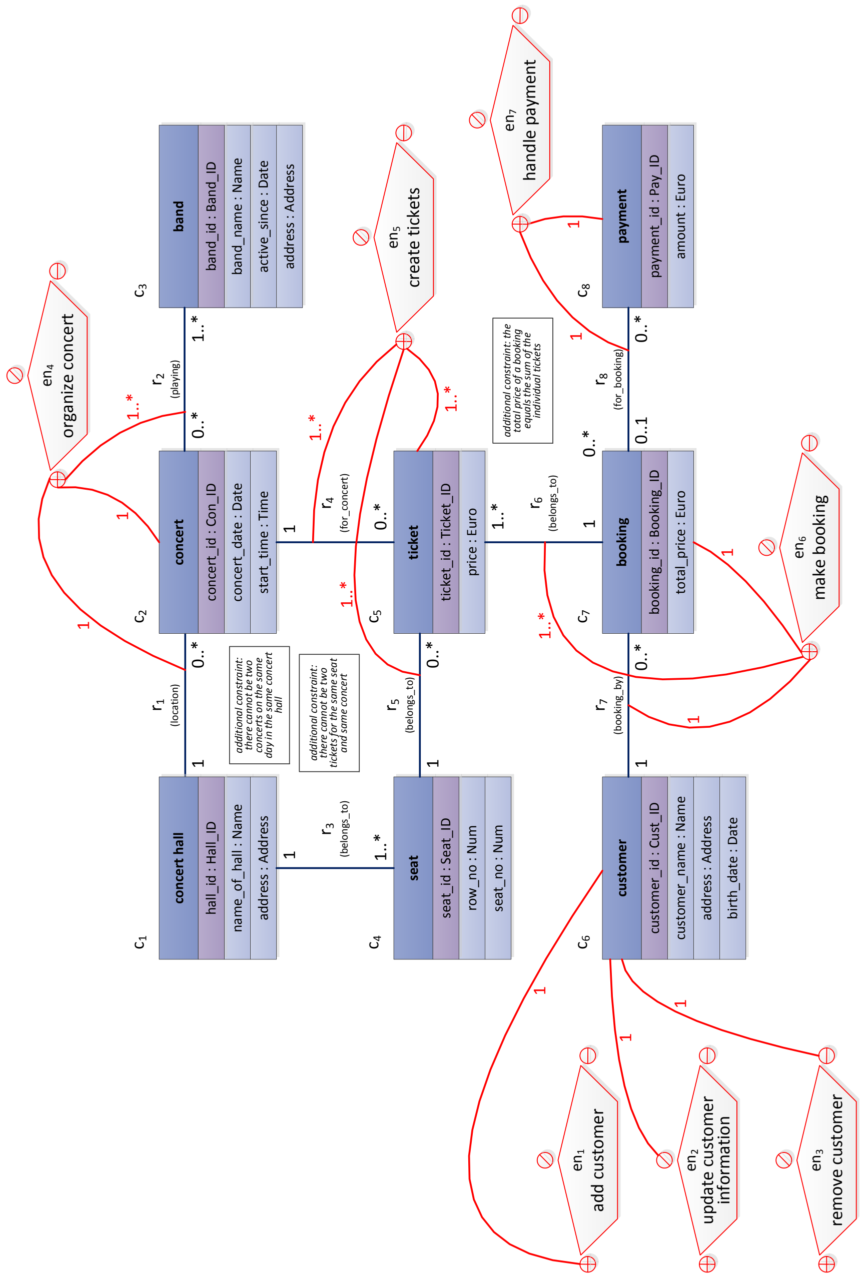


Fig. 3. Example of an event model.

for the omission of certain events, e.g., in case the object already exists it does not need to be created. Consider for example a new event en_8 with $type(en_8) = \{(\oplus, c_6), (\oplus, c_7), (\oplus, r_7)\}$ that creates a booking and the corresponding customer. If the customer is already in the database, the composite event cannot contain the creation of the customer object c_6. Instead of defining two variants of the same events (with or without creating a c_6 object), it may be convenient to define one event that allows for both variations. Case studies should show which requirement is more natural (strong versus weak event typing).
Here, we assume an event model to be given. The event model may be created by the analyst or extracted from the redo/transaction log of the DBMS. We also assume that event occurrences (defined next) can be related to events in the event model. Future work aims at providing support for the semi-automatic creation of event models and further investigating the relation with the redo/transaction logs in concrete systems like Oracle.
An *event occurrence* is specified by an event name en, a composite event CE, and a timestamp ts. A *change log* is a sequence of such event occurrences.

Definition 8 (Event Occurrence, Change Log). *Let $CM = (C, A, R, val, key, attr, rel, VOM)$ be a constrained class model and $EM = (EN, type, VE)$ an event model. Assume some universe of timestamps TS. $e = ((en, CE), ts) \in VE \times TS$ is an* event occurrence. *$EO(CM, EM) = VE \times TS$ is the set of all possible event occurrences. A* change log *$L = \langle e_1, e_2, \dots, e_n \rangle$ is a sequence of event occurrences such that time is non-decreasing, i.e., $L = \langle e_1, e_2, \dots, e_n \rangle \in (EO(CM, EM))^*$ and $ts_i \leq ts_j$ for any $e_i = ((en_i, CE_i), ts_i)$ and $e_j = ((en_j, CE_j), ts_j)$ with $1 \leq i < j \leq n$.*

Next we define the effect of an event occurrence, i.e., the resulting object model. If an event is not permissible, e.g., inserting an object for which an object with the same key already exists, the object model does not change.

Definition 9 (Effect of an Event). *Let $CM = (C, A, R, val, key, attr, rel, VOM)$ be a constrained class model and $EM = (EN, type, VE)$ an event model. For any two object models $OM_1 = (Obj_1, Rel_1)$ and $OM_2 = (Obj_2, Rel_2)$ of CM and event occurrence $e = ((en, CE), ts) \in EO(CM, EM)$, we denote $OM_1 \xrightarrow{e} OM_2$ if and only if*

- *$Obj_2 = \{(c, map_k, map_a) \in Obj_1 \mid (\ominus, c, map_k) \notin CE \wedge \forall_{map'} (\oslash, c, (map_k, map')) \notin CE\} \bigcup \{(c, map_k, map_a) \in O^{CM} \mid (\oplus, c, (map_k, map_a)) \in CE \vee (\oslash, c, (map_k, map_a)) \in CE\}$,*
- *$Rel_2 = \{(r, map_1, map_2) \in Rel_1 \mid (\ominus, r, (map_1, map_2)) \notin CE\} \bigcup \{(r, map_1, map_2) \in R^{CM} \mid (\oplus, r, (map_1, map_2)) \in CE\}$, and*
- *$\{OM_1, OM_2\} \subseteq VOM$.*

Event e is permissible *in object model OM, notation $OM_1 \xrightarrow{e}$, if and only if there exists an OM' such that $OM \xrightarrow{e} OM'$. If this is not the case, we denote $OM \overset{e}{\not\rightarrow}$, i.e., e is not permissible in OM. If an event is not permissible, it will fail and the object model will remain unchanged. Relation $\overset{e}{\Rightarrow}$ denotes the effect of event e. It is the smallest relation such that (a) $OM \overset{e}{\Rightarrow} OM'$ if $OM \xrightarrow{e} OM'$ and (b) $OM \overset{e}{\Rightarrow} OM$ if $OM \overset{e}{\not\rightarrow}$.*

The event occurrence $e = ((en, CE), ts)$ as a whole is successful or not. If $OM \overset{e}{\not\rightarrow}$, then nothing changes. The current definition of $OM_1 \overset{e}{\rightarrow}$ is rather forgiving, e.g., it allows for the deletion of an object that does not exist. It only ensures that the result is a valid object model, but relations $\overset{e}{\rightarrow}$ and $\overset{e}{\Rightarrow}$ can be made stricter if desired. Note that the atomic events in CE occur concurrently if e is successful, i.e., the events do not depend on each other.
Relation $\overset{e}{\Rightarrow}$ is deterministic, i.e., $OM_1 \overset{e}{\Rightarrow} OM_2$ and $OM_1 \overset{e}{\Rightarrow} OM_3$ implies $OM_2 = OM_3$.

Definition 10 (Effect of a Change Log). *Let $CM = (C, A, R, val, key, attr, rel, VOM)$ be a constrained class model, $EM = (EN, type, VE)$ an event model, and $OM_0 \in VOM$ the initial valid object model. Let $L = \langle e_1, e_2, \ldots, e_n \rangle \in (EO(CM, EM))^*$ be a change log. There exist object models $OM_1, OM_2, \ldots, OM_n \in VOM$ such that*

$$OM_0 \overset{e_1}{\Rightarrow} OM_1 \overset{e_2}{\Rightarrow} OM_2 \ldots \overset{e_n}{\Rightarrow} OM_n$$

Hence, change log L results in object model OM_n when starting in OM_0. This is denoted by $OM_0 \overset{L}{\Rightarrow} OM_n$.

The formalizations above provide operational semantics for an abstract database system that processes a sequence of events. However, the goal is not to model a database system. Instead, we aim to relate database updates to event logs that can be used for process mining. Subsequently, we assume that we can witness a change log $L = \langle e_1, e_2, \ldots, e_n \rangle$. It is easy to see atomic events. Moreover, various heuristics can be used to group events into composite events (e.g., based on time, session id, and/or user id). Definition 10 shows that this assumption allows us to reconstruct the state of the database system after each event, i.e., the object model OM_i resulting from e_i can be computed.

6 Approach: Scope, Bind, and Classify

Process-mining techniques require as input a "flat" event log and not a change log as described in Definition 10. Table 1 shows the kind of input data that process-mining techniques expect. Such a conventional flat event log is a collection of events where each event has the following properties:

- **Case id**: each event should refer to a case (i.e., process instance). If an event is relevant for multiple cases, it should be replicated when creating event logs.
- **Activity**: each event should be related to an activity. Events refer to activity instances, i.e., occurrences of activities in the corresponding process model.
- **Timestamp**: events within a case should be ordered. Moreover, timestamps are not just needed for the temporal order: they are also vital for measuring performance.
- Next to these mandatory attributes there may be all kinds of *optional* event attributes. For example:

- **Resource**: the person, machine or software component executing the event.
- **Type**: the transaction type of the event (start, complete, suspend, resume, etc.).
- **Costs**: the costs associated with the event.
- **Customer**: information about the person or organization for whom or which the event is executed.
- Etc.

Dedicated process-mining formats like XES or MXML allow for the storage of such event data. To be able to use existing process-mining techniques we need to be able to extract flat event logs and not a change log as defined in the previous section.

Let $\mathit{CM} = (C, A, R, \mathit{val}, \mathit{key}, \mathit{attr}, \mathit{rel}, \mathit{VOM})$ be a constrained class model, $\mathit{EM} = (\mathit{EN}, \mathit{type}, \mathit{VE})$ an event model, and $\mathit{OM}_0 \in \mathit{VOM}$ the initial valid object model. In the remainder we focus on the problem of converting a change log $L = \langle e_1, e_2, \ldots, e_n \rangle \in (\mathit{EO}(\mathit{CM}, \mathit{EM}))^*$ into a *collection of conventional events logs* that serve as input for existing process-mining techniques. Given an event occurrence $e_i = ((\mathit{en}_i, \mathit{CE}_i), \mathit{ts}_i)$, one may convert it into a conventional event by taking ts_i as timestamp and en_i as activity. However, an event occurrence needs to be related to zero or more cases and the change log may contain information about multiple processes. Hence, several decisions need to be made in the conversion process. We propose a three-step approach: (1) *scope* the event data, (2) *bind* the events to process instances (i.e., cases), and (3) *classify* the process instances.

6.1 Scope: Determine the Relevant Events

The first step in converting a change log into a collection of conventional events logs is to *scope* the event data. Which of the event occurrences in $L = \langle e_1, e_2, \ldots, e_n \rangle$ are relevant for the questions one aims to answer? One way to scope the event data is to consider a subset of event names $\mathit{EN}_s \subseteq \mathit{EN}$. Recall that EN are all event names in an event model. In Figure 3, $\mathit{EN} = \{\mathit{en}_1, \mathit{en}_2, \ldots, \mathit{en}_7\}$. Events may also be selected based on a time window (e.g., "all events executed after May 21st" or "all events belonging to cases that were complete in 2013") or the classes involved (e.g., "all events related to Metallica concerts").

6.2 Bind: Relate Events to Process Instances

Process models always describe lifecycles of instances. For example, when looking at any BPMN, EPC, or UML activity model there is the implicit notion of a process instance (i.e., case). The process model is instantiated once for each case, e.g., for an order handling process the activities always operate on a specific purchase order. The notion of process instances is made explicit in process-aware information systems, e.g., Business Process Management (BPM) and Workflow Management (WfM) systems. However, in most other systems the instance notion is implicit. Moreover, the instance notion selected may depend on the questions one would like to answer. Consider for example Figure 3. Possible instance notions are concert, ticket, booking, customer, band, concert hall, seat, and payment. One could construct a process describing the lifecycle of tickets. Such a lifecycle is different from the lifecycle of a concert or booking. One could even consider discovering the lifecycle of chairs in a concert hall by taking seat IDs as process instances.

Technically, we need to define a set of process instances PI (cases) and relate events to these instances: $bind \subseteq VE_s \times PI$ with $VE_s = \{(en, CE) \in VE \mid en \in EN_s\}$ the subset of the valid events selected (without timestamps). Let $pi \in PI$ be a process instance and $e_i = ((en_i, CE_i), ts_i)$ an event occurrence: event e_i belongs to case pi if $((en_i, CE_i), pi) \in bind$. Note that $bind$ is a relation and not a function. This way the same event occurrence may yield events in different process instances. For example, the cancelation of a concert may influence many bookings.

Relation $bind$ allows us to associate events to cases. This, combined with the timestamps and activity names, enables the construction of event logs.

6.3 Classify: Relate Process Instances to Processes

After scoping and binding, we have a set of events related to process instances. Since we can reconstruct the object model before and after each event occurrence, we can add all kinds of optional element attributes. Hence, we can create a conventional event log with a rich set of attributes. However, as process-mining techniques mature it becomes interesting to compare different groups of process instances [3]. Instead of creating one event log, it is often insightful to create multiple event logs. For example, to compare the booking process for two concerts we create two event logs and compare the process-mining results.

To allow for *comparative process mining*, process instances are classified using a relation $class \subseteq PI \times CL$ with CL the set of classes. Consider for example the study process of students taking a particular course. Rather than creating one process model for all students, one could create (1) a process model for students that passed and a process model for students that failed, (2) a process model for male students and a process model for female students, or (3) a process model for Dutch students and a process model for international students. Note that $class \subseteq PI \times CL$ does not require a strict partitioning of the process instances, e.g., a case may belong to multiple classes.

In [3], the notion of *process cubes* was proposed to allow for comparative process mining. In a process cube events are organized using different dimensions. Each cell in the process cube corresponds to a set of events that can be used to discover a process model, to check conformance, or to discover bottlenecks. Process cubes are inspired by the well-known OLAP (Online Analytical Processing) data cubes and associated operations such as slice, dice, roll-up, and drill-down [24]. However, there are also significant differences because of the process-related nature of event data. For example, process discovery based on events is incomparable to computing the average or sum over a set of numerical values. Moreover, dimensions related to process instances (e.g. male versus female

students), subprocesses (e.g. group assignments versus individual assignments), organizational entities (e.g. students versus lecturers), and time (e.g. years or semesters) are semantically different and it is challenging to slice, dice, roll-up, and drill-down process-mining results efficiently.
As mentioned before, we deliberately remain at the conceptual level and do not focus on a particular DBMS. However, the "scope, bind, and classify" approach allows for the transformation of database updates into events populating process cubes that can be used for a variety of process-mining analyses.

7 Related Work

The reader is referred to [1] for an introduction to process mining. Alternatively, one can consult the Process Mining Manifesto [36] for best practices and the main challenges in process mining. Next to the automated discovery of the underlying process based on raw event data, there are process-mining techniques to analyze bottlenecks, to uncover hidden inefficiencies, to check compliance, to explain deviations, to predict performance, and to guide users towards "better" processes. Dozens (if not hundreds) of process-mining techniques are available and their value has been proven in many case studies. For example, dozens of process discovery [1, 9, 11, 16, 32, 18, 22, 23, 27, 33, 39, 48, 51, 52] and conformance checking [6, 13, 14, 15, 21, 28, 33, 41, 42, 47, 50] approaches have been proposed in literature. However, this paper is not about new process-mining techniques but about getting the event data needed for all of these techniques. We are not aware of any work systematically transforming database updates into event logs. Probably, there are process-mining case-studies using redo/transaction logs from database management systems like Oracle RDBMS, Microsoft SQL server, IBM DB2, or Sybase IQ. However, systematic tool support seems to be missing.
The binding step in our approach is related to topic of event correlation which has been investigated in the (web) services [4]. In [8] and [17] various interaction and correlation patterns are described. In [44] a technique is presented for correlating messages with the goal to visualize the execution of web services. Also Nezhad et al. [40] developed techniques for event correlation and process discovery from web service interaction logs.
Most closely related seem to be the work on artifact-centric process mining [12, 30, 31], process model repositories [46], event log extraction ([49, 34]), and process cubes [3]. However, none of these approaches define an event model on top of a class model.

8 Conclusion

To drive innovation in an increasingly digitalized world, the "process scientist" needs to have powerful tools. Recent advances in process mining provide such tools, but cannot be applied easily to selections of the Internet of Events (IoE) where data is heterogeneous and distributed. Process

mining seeks the "confrontation" between real event data and process models (automatically discovered or hand-made). The fifteen case studies listed on the web page of the IEEE Task Force on Process Mining [37] illustrate the applicability of process mining. Process mining can be used to check conformance, detect bottlenecks, and suggest process improvements. However, the most time-consuming part of process mining is not the actual analysis. Most time is spent on locating, selecting, converting, and filtering the event data. The twelve *guidelines for logging* presented in this paper show that the input-side of process mining deserves much more attention. Logging can be improved by better instrumenting systems. However, we can also try to better use what is already there and widely uses: database systems. This paper focused on supporting *the systematic extraction of event data from database systems.*
Regular tables in a database provide a view of the actual state of the information system. For process mining, however, it is interesting to know when a record was created, updated, or deleted. *Taking the viewpoint that the database reflects the current state of one or more processes, we define all changes of the database to be events.* In this paper, we conceptualized this viewpoint. Building upon class and object models, we defined the notion of an event model. The event model relates changes to the underlying database to events used for process mining. Based on such an event model, we defined the "scope, bind, and classify" approach that creates a collection of event logs that can be used for comparative process mining.
In this paper we only conceptualized the different ideas. A logical next step is to develop tool support for specific database management systems. Moreover, we would like to relate this to our work on process cubes [3] for comparative process mining.

Acknowledgements

This work was supported by the Basic Research Program of the National Research University Higher School of Economics (HSE) in Moscow.

References

[1] W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes.* Springer-Verlag, Berlin, 2011.

[2] W.M.P. van der Aalst. Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, pages 1–37, 2013. doi:10.1155/2013/507984.

[3] W.M.P. van der Aalst. Process Cubes: Slicing, Dicing, Rolling Up and Drilling Down Event Data for Process Mining. In M. Song, M. Wynn, and J. Liu, editors, *Asia Pacific Conference on Business Process Management (AP-BPM 2013)*, volume 159 of *Lecture Notes in Business Information Processing*, pages 1–22. Springer-Verlag, Berlin, 2013.

[4] W.M.P. van der Aalst. Service Mining: Using Process Mining to Discover, Check, and Improve Service Behavior. *IEEE Transactions on Services Computing*, 6(4):525–535, 2013.

[5] W.M.P. van der Aalst. Data Scientist: The Engineer of the Future. In K. Mertins, F. Benaben, R. Poler, and J. Bourrieres, editors, *Proceedings of the I-ESA Conference*, volume 7 of *Enterprise Interoperability*, pages 13–28. Springer-Verlag, Berlin, 2014.

[6] W.M.P. van der Aalst, A. Adriansyah, and B. van Dongen. Replaying History on Process Models for Conformance Checking and Performance Analysis. *WIREs Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.

[7] W.M.P. van der Aalst, P. Barthelmess, C.A. Ellis, and J. Wainer. Proclets: A Framework for Lightweight Interacting Workflow Processes. *International Journal of Cooperative Information Systems*, 10(4):443–482, 2001.

[8] W.M.P. van der Aalst, A.J. Mooij, C. Stahl, and K. Wolf. Service Interaction: Patterns, Formalization, and Analysis. In M. Bernardo, L. Padovani, and G. Zavattaro, editors, *Formal Methods for Web Services*, volume 5569 of *Lecture Notes in Computer Science*, pages 42–88. Springer-Verlag, Berlin, 2009.

[9] W.M.P. van der Aalst, V. Rubin, H.M.W. Verbeek, B.F. van Dongen, E. Kindler, and C.W. Günther. Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting. *Software and Systems Modeling*, 9(1):87–111, 2010.

[10] W.M.P. van der Aalst and C. Stahl. *Modeling Business Processes: A Petri Net Oriented Approach*. MIT press, Cambridge, MA, 2011.

[11] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[12] ACSI. Artifact-Centric Service Interoperation (ACSI) Project Home Page. `www.acsi-project.eu`.

[13] A. Adriansyah, B. van Dongen, and W.M.P. van der Aalst. Conformance Checking using Cost-Based Fitness Analysis. In C.H. Chi and P. Johnson, editors, *IEEE International Enterprise Computing Conference (EDOC 2011)*, pages 55–64. IEEE Computer Society, 2011.

[14] A. Adriansyah, B.F. van Dongen, and W.M.P. van der Aalst. Towards Robust Conformance Checking. In M. zur Muehlen and J. Su, editors, *BPM 2010 Workshops, Proceedings of the Sixth Workshop on Business Process Intelligence (BPI2010)*, volume 66 of *Lecture Notes in Business Information Processing*, pages 122–133. Springer-Verlag, Berlin, 2011.

[15] A. Adriansyah, N. Sidorova, and B.F. van Dongen. Cost-based Fitness in Conformance Checking. In *International Conference on Application of Concurrency to System Design (ACSD 2011)*, pages 57–66. IEEE Computer Society, 2011.

[16] R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, volume 1377 of *Lecture Notes in Computer Science*, pages 469–483. Springer-Verlag, Berlin, 1998.

[17] A. Barros, G. Decker, M. Dumas, and F. Weber. Correlation Patterns in Service-Oriented Architectures. In M. Dwyer and A. Lopes, editors, *Proceedings of the 10th International Conference on Fundamental Approaches to Software Engineering (FASE 2007)*, volume 4422 of *Lecture Notes in Computer Science*, pages 245–259. Springer-Verlag, Berlin, 2007.

[18] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383. Springer-Verlag, Berlin, 2007.

[19] R.P. Jagadeesh Chandra Bose, R. Mans, and W.M.P. van der Aalst. Wanna Improve Process Mining Results? It's High Time We Consider Data Quality Issues Seriously. In B. Hammer, Z.H. Zhou, L. Wang, and N. Chawla, editors, *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2013)*, pages 127–134, Singapore, 2013. IEEE.

[20] J. vom Brocke and M. Rosemann, editors. *Handbook on Business Process Management*, International Handbooks on Information Systems. Springer-Verlag, Berlin, 2010.

[21] T. Calders, C. Guenther, M. Pechenizkiy, and A. Rozinat. Using Minimum Description Length for Process Mining. In *ACM Symposium on Applied Computing (SAC 2009)*, pages 1451–1455. ACM Press, 2009.

[22] J. Carmona and J. Cortadella. Process Mining Meets Abstract Interpretation. In J.L. Balcazar, editor, *ECML/PKDD 210*, volume 6321 of *Lecture Notes in Artificial Intelligence*, pages 184–199. Springer-Verlag, Berlin, 2010.

[23] J. Carmona, J. Cortadella, and M. Kishinevsky. A Region-Based Algorithm for Discovering Petri Nets from Event Logs. In *Business Process Management (BPM2008)*, pages 358–373, 2008.

[24] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM Sigmod Record*, 26(1):65–74, 1997.

[25] P.P. Chen. The Entity-Relationship Model: Towards a unified view of Data. *ACM Transactions on Database Systems*, 1:9–36, Jan 1976.

[26] D. Cohn and R. Hull. Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. *IEEE Data Engineering Bulletin*, 32(3):3–9, 2009.

[27] J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.

[28] J.E. Cook and A.L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology*, 8(2):147–176, 1999.

[29] M. Dumas, M. La Rosa, J. Mendling, and H. Reijers. *Fundamentals of Business Process Management*. Springer-Verlag, Berlin, 2013.

[30] D. Fahland, M. De Leoni, B. van Dongen, and W.M.P. van der Aalst. Behavioral Conformance of Artifact-Centric Process Models. In A. Abramowicz, editor, *Business Information Systems (BIS*

2011), volume 87 of *Lecture Notes in Business Information Processing*, pages 37–49. Springer-Verlag, Berlin, 2011.

[31] D. Fahland, M. De Leoni, B. van Dongen, and W.M.P. van der Aalst. Many-to-Many: Some Observations on Interactions in Artifact Choreographies. In D. Eichhorn, A. Koschmider, and H. Zhang, editors, *Proceedings of the 3rd Central-European Workshop on Services and their Composition (ZEUS 2011)*, CEUR Workshop Proceedings, pages 9–15. CEUR-WS.org, 2011.

[32] W. Gaaloul, K. Gaaloul, S. Bhiri, A. Haller, and M. Hauswirth. Log-Based Transactional Workflow Mining. *Distributed and Parallel Databases*, 25(3):193–240, 2009.

[33] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens. Robust Process Discovery with Artificial Negative Events. *Journal of Machine Learning Research*, 10:1305–1340, 2009.

[34] C. Günther and W.M.P. van der Aalst. A Generic Import Framework for Process Event Logs. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops, Workshop on Business Process Intelligence (BPI 2006)*, volume 4103 of *Lecture Notes in Computer Science*, pages 81–92. Springer-Verlag, Berlin, 2006.

[35] A.H.M. ter Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation: YAWL and its Support Environment.* Springer-Verlag, Berlin, 2010.

[36] IEEE Task Force on Process Mining. Process Mining Manifesto. In *BPM Workshops*, volume 99 of *Lecture Notes in Business Information Processing.* Springer-Verlag, Berlin, 2011.

[37] IEEE Task Force on Process Mining. Process Mining Case Studies. `http://www.win.tue.nl/ieeetfpm/doku.php?id=shared:process_mining_case_studies`, 2013.

[38] IEEE Task Force on Process Mining. XES Standard Definition. www.xes-standard.org, 2013.

[39] A.K. Alves de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.

[40] H.R. Montahari-Nezhad, R. Saint-Paul, F. Casati, and B. Benatallah. Event Correlation for Process Discovery from Web Service Interaction Logs. *VLBD Journal*, 20(3):417–444, 2011.

[41] J. Munoz-Gama and J. Carmona. A Fresh Look at Precision in Process Conformance. In R. Hull, J. Mendling, and S. Tai, editors, *Business Process Management (BPM 2010)*, volume 6336 of *Lecture Notes in Computer Science*, pages 211–226. Springer-Verlag, Berlin, 2010.

[42] J. Munoz-Gama and J. Carmona. Enhancing Precision in Process Conformance: Stability, Confidence and Severity. In N. Chawla, I. King, and A. Sperduti, editors, *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, pages 184–191, Paris, France, April 2011. IEEE.

[43] OMG. Unified Modeling Language, Infrastructure and Superstructure (Version 2.2, OMG Final Adopted Specification), 2009.

[44] W. De Pauw, M. Lei, E. Pring, L. Villard, M. Arnold, and J.F. Morar. Web Services Navigator: Visualizing the Execution of Web Services. *IBM Systems Journal*, 44(4):821–845, 2005.

[45] M. Reichert and B. Weber. *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer-Verlag, Berlin, 2012.

[46] M. La Rosa, H.A. Reijers, W.M.P. van der Aalst, R.M. Dijkman, J. Mendling, M. Dumas, and L. Garcia-Banuelos. APROMORE: An Advanced Process Model Repository. *Expert Systems With Applications*, 38(6):7029–7040, 2011.

[47] A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008.

[48] M. Sole and J. Carmona. Process Mining from a Basis of Regions. In J. Lilius and W. Penczek, editors, *Applications and Theory of Petri Nets 2010*, volume 6128 of *Lecture Notes in Computer Science*, pages 226–245. Springer-Verlag, Berlin, 2010.

[49] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. XES, XESame, and ProM 6. In P. Soffer and E. Proper, editors, *Information Systems Evolution*, volume 72 of *Lecture Notes in Business Information Processing*, pages 60–75. Springer-Verlag, Berlin, 2010.

[50] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens. A Robust F-measure for Evaluating Discovered Process Models. In N. Chawla, I. King, and A. Sperduti, editors, *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, pages 148–155, Paris, France, April 2011. IEEE.

[51] A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.

[52] J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process Discovery using Integer Linear Programming. *Fundamenta Informaticae*, 94:387–412, 2010.

[53] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag, Berlin, 2007.