# Extracting Object-Centric Event Logs to Support Process Mining on Databases

Guangming Li[1], Eduardo González López de Murillas[1], Renata Medeiros de Carvalho[1], and Wil M.P. van der Aalst[1,2]

[1] Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.
{g.li.3,e.gonzalez,r.carvalho,w.m.p.v.d.aalst}@tue.nl
[2] RWTH Aachen University, Aachen, Germany.

**Abstract.** Process mining helps organizations to investigate how their operational processes are executed and how these can be improved. Process mining requires event logs extracted from information systems supporting these processes. The eXtensible Event Stream (XES) format is the current standard which requires a case notion to correlate events. However, it has problems to deal with object-centric data (e.g., database tables) due to the existence of one-to-many and many-to-many relations. In this paper, we propose an approach to extract, transform and store object-centric data, resulting in eXtensible Object-Centric (XOC) event logs. The XOC format does not require a case notion to avoid flattening multi-dimentional data. Besides, based on so-called object models which represent the states of a database, a XOC log can reveal the evolution of the database along with corresponding events. Dealing with object-centric data enables new process mining techniques that are able to capture the real processes much better.

## 1 Introduction

Process mining represents a set of techniques which are widely used to extract insights from event data generated by information systems. The starting point for process mining techniques is formed by event logs. The XES log format [1] is widely employed to generate event logs. In general, a XES log consists of a collection of traces. A trace describes the life-cycle of a particular case (i.e., a process instance) in terms of the activities executed. Process-aware information systems (e.g., BPM/WFM systems) which assume an explicit case notion to correlate events, provide such logs directly.

However, the information systems one encounters in most organizations are *object-centric*. Some examples of these systems are Customer Relationship Management (CRM) and/or Enterprise Resource Planning (ERP) which provide business functions such as procurement, production, sales, delivery, finance, etc. Examples are the ERP/CRM suites from vendors such as SAP (S/4HANA), Oracle (E-Business Suite), Microsoft (Dynamics 365), and Salesforce (CRM). There are also some free and open source alternatives such as Dolibarr and Odoo. A common feature of these systems is that they are built on top of database technology, i.e., they contain hundreds of tables covering customers, orders, deliveries, etc. Figure 1 shows a fragment of data generated by the Dolibarr ERP system. This is an example of object-centric data since object instances are explicitly recorded (e.g., orders, which can be abstracted as objects) while

events related to the underlying process are implicitly recorded by other means, e.g., through redo logs (cf. Table 1). Approaches to extract XES logs from databases with the aid of ontology [3] or redo logs [5] have been previously proposed.

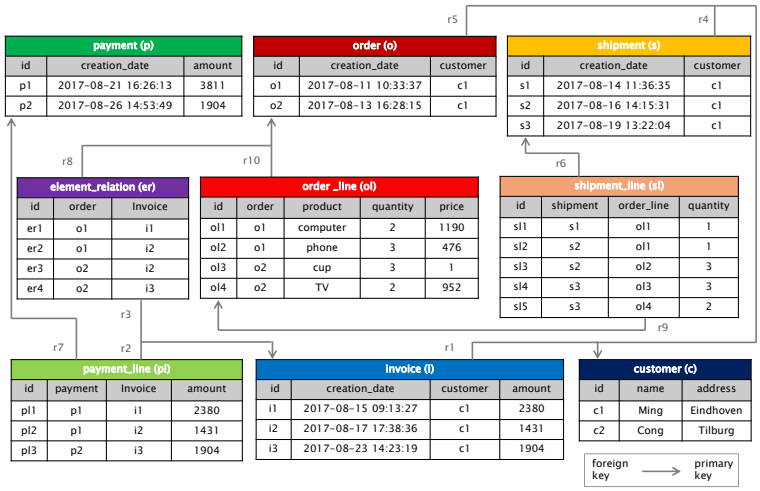


**Fig. 1.** A fragment of data generated by a real ERP system.

Although there exist approaches to extract XES logs from object-centric data, the constraints set by this format do not match the nature of the data at hand. The following challenges have been identified when applying the XES format to object-centric data:

- *It is required to identify the case id for the whole process*. Database object-centric data lacks a single explicit definition of case notion. On the contrary, because of the existence of multiple classes of objects (tables) and relations (foreign keys), object-centric data can be correlated in many ways. Each of these correlations may correspond to different processes affecting the same data, or different views on the same process (customer vs. provider point of view). In the case of XES, a case notion has to be defined beforehand to proceed with the extraction, requiring one dedicated event log for each perspective to be analyzed.
- *The quality of the input data gets compromised*. If we straightjacket object-centric data into XES logs, the input data with one-to-many and many-to-many relations is flattened into separate traces, in which events referred to by multiple cases are duplicated. This forced transformation introduces unnecessary redundancy and leads to problems such as data convergence and divergence [9].[3]
- *Interactions between process instances get lost*. Traces in XES logs only describe the evolution (i.e., lifecycle) of one type of process instance and they are typically considered in isolation. Due to the lack of interactions between process instances, XES logs cannot provide a whole view to indicate the state of a system.

[3] Data Convergence means the same event is related to multiple process instances at once. Data divergence means that multiple events of the same activity are related to one process instance, i.e., the same activity is performed multiple times for the same process instance (cf. Figure 4).

– *The data perspective is only secondary*.[4] The XES format focuses on the behavioral perspective, considering any additional information as trace or event attributes. In this sense, the information not directly related to the events is discarded (records, data objects, etc.), which weakens the data perspective of the original system.

To face the problems mentioned above, we propose a novel log format to organize object-centric data from databases, resulting in *eXtensible Object-Centric (XOC*[5]*)* logs. This log format provides an evolutionary view on databases based on the idea that a log is a list of events and each event refers to an *object model* (cf. Section 2.1) representing the state of the database just updated by the event. Process mining takes logs as input, and this log format transforms the raw data from databases into event logs, which paves a road to analyze databases in a process mining approach.

Compared with XES, the XOC format has the following contributions. It correlates events based on objects rather than a case notion, which solves the challenge of identifying a case id. Besides, without the constraints set by the case notion, XOC logs can better deal with one-to-many and many-to-many relations. Additionally, more powerful concepts such as *object models* are employed to provide a whole view of the state of the database as well as the interactions between instances. Moreover, all columns in a record can be abstracted as attributes of an object (denoting the record) to enrich the data perspective. To validate the log format, we present an approach to automatically transform object-centric data into XOC logs. In resulting logs, an event type is defined on the information system level (e.g., "create order") rather than the database level (e.g., "insert an order record" and "insert an order line record") [5], which makes extracted events more intuitive for users. Besides these contributions, XOC logs enable new process mining techniques to further solve the problems mentioned above. Discovering data-aware process models like the *object-centric behavioral constraint (OCBC)* models [8], to better reveal the complex interactions between the behavioral and data perspectives[4] (cf. Figure 5), checking conformance for deviations which cannot be detected by conventional methods [14] and predicting future events and objects according to a discovered OCBC model (cf. Section 4) are some examples of enabled techniques.

The remainder is organized as follows. In Section 2, we describe the XOC log format. Section 3 proposes an approach to extract XOC logs from databases, and Section 4 demonstrates an implementation of the approach and a comparison between XOC logs and XES logs. Section 5 reviews the related work while Section 6 concludes the paper.

## 2 Object-Centric Event Log

Process mining techniques take event logs as input. In order to apply these techniques to object-centric data, we propose a novel log format to organize such data in this section.

### 2.1 Object-Centric Data

Increasingly, organizations are employing object-centric information systems, such as ERP and CRM, to deal with their transactions. In this paper, we use the term "object"

[4] The behavioral perspective refers to the control-flow of events while the data perspective refers to data attributes.

[5] XOC is pronounced as /sɒk/.

to abstract data elements (e.g., records in database tables) generated by information systems. In this sense, object-centric systems refer to systems which record the transactions of the same category (e.g., orders) in the same table (e.g., the "order" table) based on the relational database technology. Accordingly, the data generated by such systems are called object-centric data. Figure 1 shows an example of object-centric data.

Objects are grouped in classes and have some attributes. For example, a record in the "order" table (e.g., the first row) can be considered as an object of class "order". Each value (e.g., "c1") in the record can be considered as an attribute of the object. Besides, there exist class relationships between classes, which corresponds to dependency relations between tables. For instance, there is a class relationship between class "order" and class "order line", indicated by one of foreign keys of table "order_line", which refers to the primary key of table "order".

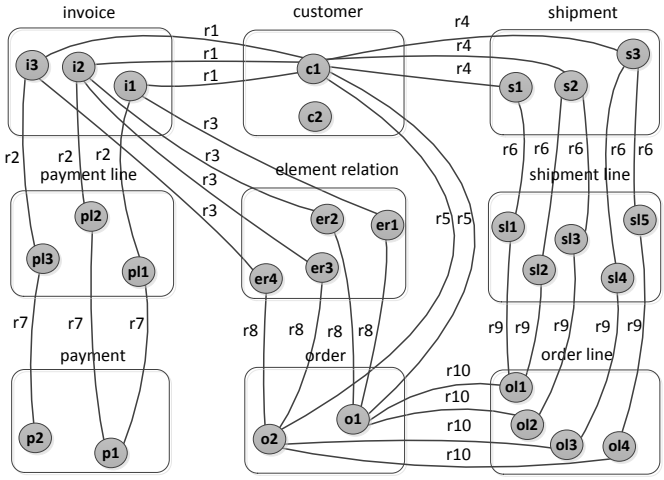


**Fig. 2.** An example of an object model.

**Definition 1 (Object Model).** *Let $\mathcal{U}_O$ be the universe of* objects*, $\mathcal{U}_C$ be the universe of* classes*, $\mathcal{U}_{RT}$ be the universe of* class relationships*, $\mathcal{U}_{Attr}$ be the universe of* attribute names *and $\mathcal{U}_{Val}$ be the universe of* attribute values*. $C \subseteq \mathcal{U}_C$ is a set of classes and $RT \subseteq \mathcal{U}_{RT}$ is a set of class relationships. An object model is a tuple $OM = (Obj, Rel, class, objectAttr)$, where*

- *$Obj \subseteq \mathcal{U}_O$ is a set of objects,*
- *$Rel \subseteq RT \times Obj \times Obj$ is a set of object relations,*
- *$class \in Obj \rightarrow C$ maps objects onto classes, and*
- *$objectAttr \in Obj \rightarrow (\mathcal{U}_{Attr} \not\rightarrow \mathcal{U}_{Val})$ maps objects onto a partial function assigning values to some attributes.*

*$\mathcal{U}_{OM}$ is the universe of object models.*

An object model consists of objects, object relations and two functions indicating the class and attributes of each object. In the database context, an object model represents the state of a database at some moment, where objects correspond to records in tables and relations correspond to dependencies between records.

Figure 2 shows an object model $OM = (Obj, Rel, class, objectAttr)$ which represents the state of the database indicated in Figure 1. The 28 objects are depicted as grey dots. There are two objects $o1$ and $o2$ belonging to class "order" , i.e., $class(o1) = class(o2) = order$. There are three object relations corresponding to class relationship $r1$, i.e., $(r1, c1, i1) \in Rel$, $(r1, c1, i2) \in Rel$ and $(r1, c1, i3) \in Rel$. The object $i1$ has an attribute named "creation_date" with a value "2017-08-15 09:13:27", i.e., $objectAttr(i1)(\text{“}creation\_date\text{”}) = \text{“}2017-08-15\ 09:13:27\text{”}$. Note that, the attributes of objects are not shown in the graph.

### 2.2 Database Changes

Unlike process-aware information systems (e.g., BPM/WFM systems), object-centric information systems keep transactions in "regular tables" in databases. A record in a table indicates the current state of an instance. It is difficult to reveal all the events impacting this record, e.g., the first record in "order_line" table tells nothing about events.

Fortunately, database technology often provides so-called *redo logs* (as shown in Table 1) that record the history of database changes. Similarly, most SAP systems provide two *change tables*, i.e., CDHDR (recording all database changes) and CDPOS (recording details of each change in CDHDR) as shown in Table 2. If redo logs and change tables are not available, then domain knowledge is needed to obtain changes, i.e., timestamps/dates in particular columns refer to the creation of the row or to changes of values. Using domain knowledge, these timestamps can reconstruct database changes.

**Table 1.** A fragment of a redo log in which each line corresponds to a change in the database.

| # | Time | Redo |
|---|---|---|
| 1 | 2017-08-11 10:33:37 | insert into "order" ("id","creation_date","customer") values ("o1","2017-08-11 10:33:37","c1") |
| 2 | 2017-08-11 10:33:37 | insert into "order_line" ("id","order","product","quantity","price") values ("ol1","o1","computer","2","1190") |
| 3 | 2017-08-11 10:33:37 | insert into "order_line" ("id","order","product","quantity","price") values ("ol2","o1","phone","3","476") |

**Table 2.** A fragment of the CDHDR table (left) and CDPOS table (right).

| HeadID | Date | Time | Op |
|---|---|---|---|
| 100 | 2017-08-11 | 10:33:37 | I |
| 101 | 2017-08-11 | 10:33:37 | I |
| 102 | 2017-08-11 | 10:33:37 | I |
| 103 | 2017-08-13 | 16:28:15 | I |
| 104 | 2017-08-14 | 11:36:35 | I |

| ItemID | HeadID | TabName | Key | FieldName | Op | NewValue | OldValue |
|---|---|---|---|---|---|---|---|
| 1 | 100 | order | o1 | id | I | o1 | none |
| 2 | 100 | order | o1 | creation_date | I | 2017-08-11 10:33:37 | none |
| 3 | 100 | order | o1 | customer | I | c1 | none |
| 4 | 101 | order_line | ol1 | id | I | ol1 | none |
| 5 | 101 | order_line | ol1 | order | I | o1 | none |

Each change corresponds to one execution of an SQL sentence, i.e., one event on the database level. In this paper, the extracted logs cater to people who operate information systems and may be unfamiliar with databases. Therefore, the extracted events are on the information system level rather than the database level. In other words, an example of an event is a "create order" operation rather than an execution of "insert an order record", which makes extracted logs more intuitive and understandable. Note that, an event may correspond to multiple changes, e.g., the three changes in Table 1 correspond to one "create order" event, and these changes may have the same timestamp (i.e., it is not necessary that each change has a unique timestamp). In real applications, the derived changes may cover multiple processes. For a given process, we can scope the changes using tricks such as based on a time window or classes involved [13].

### 2.3 eXtensible Object-Centric (XOC) Log Format

Based on database changes and the current state of a database, it is possible to restore all previous states of the database. The idea of the XOC format is that one event represents one operation on the information system and corresponds to an object model providing a *snapshot of the system just after this operation*. Besides, each event has an event type to indicate the executed activity (e.g., "create order"), and refers to some objects modified by the event.

**Table 3.** A fragment of the XOC log extracted from the motivating example data in Figure 1.

| Index | Event | Event Type | References | Object Model | |
|---|---|---|---|---|---|
| | | | | Objects | Relations |
| 1 | $co1$ | $create\ order\ (co)$ | $\{o1, ol1, ol2\}$ | $\{c1, c2, o1, ol1, ol2\}$ | $\{(r5, c1, o1), (r10, o1, ol1), (r10, o1, ol2)\}$ |
| 2 | $co2$ | $create\ order\ (co)$ | $\{o2, ol3, ol4\}$ | $\{c1, c2, o1, ol1, ol2, o2, ol3, ol4\}$ | $\{(r5, c1, o1), (r10, o1, ol1), (r10, o1, ol2), (r5, c1, o2), (r10, o2, ol3), (r10, o2, ol4)$ |
| 3 | $cs1$ | $create\ shipment\ (cs)$ | $\{s1, sl1\}$ | $\{c1, c2, o1, ol1, ol2, o2, ol3, ol4, s1, sl1\}$ | $\{(r5, c1, o1), (r10, o1, ol1), (r10, o1, ol2), (r5, c1, o2), (r10, o2, ol3), (r10, o2, ol4), (r4, c1, s1), (r6, s1, sl1), (r9, ol1, sl1)\}$ |
| 4 | $ci1$ | $create\ invoice\ (ci)$ | $\{i1, er1\}$ | $\{c1, c2, o1, ol1, ol2, o2, ol3, ol4, s1, sl1, i1, er1\}$ | $\{(r5, c1, o1), (r10, o1, ol1), (r10, o1, ol2), (r5, c1, o2), (r10, o2, ol3), (r10, o2, ol4), (r4, c1, s1), (r6, s1, sl1), (r9, ol1, sl1), (r1, c1, i1), (r3, i1, er1), (r8, o1, er1)\}$ |
| 5 | $cs2$ | $create\ shipment\ (cs)$ | $\{s2, sl2, sl3\}$ | $\{c1, c2, o1, ol1, ol2, o2, ol3, ol4, s1, sl1, i1, er1, s2, sl2, sl3\}$ | $\{(r5, c1, o1), (r10, o1, ol1), (r10, o1, ol2), (r5, c1, o2), (r10, o2, ol3), (r10, o2, ol4), (r4, c1, s1), (r6, s1, sl1), (r9, ol1, sl1), (r1, c1, i1), (r3, i1, er1), (r8, o1, er1), (r4, c1, s2), (r6, s2, sl2), (r9, ol1, sl2), (r6, s2, sl3), (r9, ol2, sl3)\}$ |

**Definition 2 (eXtensible Object-Centric Event Log).** *Let $\mathcal{U}_E$ be the universe of* events *and $\mathcal{U}_{ET}$ be the universe of* event types*. An eXtensible Object-Centric (XOC)* event log *is a tuple $L = (E, act, refer, om, \prec)$, where*

- *$E \subseteq \mathcal{U}_E$ is a set of events,*
- *$act \in E \to \mathcal{U}_{ET}$ maps events onto event types,*
- *$refer \in E \to \mathcal{P}(\mathcal{U}_O)$ relates events to sets of objects,*
- *$om \in E \to \mathcal{U}_{OM}$ maps an event to the object model just after the event occurred,*
- *$\prec \subseteq E \times E$ defines a total order on events.*

*$\mathcal{U}_L$ is the universe of XOC event logs.*

Table 3 shows a XOC log example. The "Event" column specifies the set of events while the "Index" column indicates the total order on events. The last four columns show the corresponding event type, object references (i.e., modified objects) and object model (i.e., objects and relations) of each event, respectively. Note that, the information of objects (e.g., attribute values) is left out in Table 3. In some cases, object models may increase dramatically since an object model contains unchanged contents of the previous one and new contents. This problem can be solved by some storage tricks, e.g., only storing updated information, which falls out of the scope of this paper.

## 3 Extracting Object-Centric Event Logs from Databases

In this section, we propose an approach to extract XOC logs from databases based on object-centric data (i.e., database tables) and database changes.

### 3.1 Formalizations of Object-Centric Data

This subsection provides a formal definition to describe object-centric data (i.e., database tables), which refers to the notations in [13, 5].

**Definition 3 (Data Model).** *Let $V \subseteq \mathcal{U}_{Val}$ be a set of values. A data model is a tuple $DM = (C, A, classAttr, val, PK, FK, classPK, classFK, keyRel, keyAttr, refAttr)$ such that*

- *$C \subseteq \mathcal{U}_C$ is a set of classes,*
- *$A \subseteq \mathcal{U}_{Attr}$ is a set of attribute names,*
- *$classAttr \in C \to \mathcal{P}(A)$ maps each class onto a set of attribute names,*[6]
- *$val \in A \to \mathcal{P}(V)$ maps each attribute onto a set of values,*
- *$PK$ is a set of primary keys and $FK$ is a set of foreign keys, where PK and FK are assumed to be disjoint in this paper (i.e., $PK \cap FK = \emptyset$) and $K = PK \cup FK$,*
- *$classPK \in C \to PK$ maps each class onto a primary key,*
- *$classFK \in C \to \mathcal{P}(FK)$ maps each class onto a set of foreign keys,*
- *$keyRel \in FK \to PK$ maps each foreign key onto a primary key,*
- *$keyAttr \in K \to \mathcal{P}(A)$ maps each key onto a set of attributes, and*
- *$refAttr \in FK \times A \not\to A$ maps each pair of a foreign key and an attribute onto an attribute from the corresponding primary key. That is, $\forall k \in FK : \forall a, a' \in keyAttr(k) : (refAttr(k, a) \in keyAttr(keyRel(k)) \wedge (refAttr(k, a) = refAttr(k, a') \Longrightarrow a = a')$.*

*$\mathcal{U}_{DM}$ is the universe of data models.*

A *data model* describes the structure of the object-centric data. More precisely, a class represents a table while an attribute represents a column in a table. Function $classAttr$ specifies the attributes of a class and function $val$ indicates possible values for an attribute. $classPK$ and $classFK$ define the primary key (PK) and foreign keys (FKs) for each table, respectively. Given a FK, $keyRel$ indicates its referred PK.

Take the tables in Figure 1 as an example. $C = \{p, o, s, er, ol, sl, pl, i, c\}$ and $A = \{id, creation\_date, amount, ..., address\}$. $classAttr(p) = \{id, creation\_date, amount\}$ and $val(amount) = \mathbb{N}$ where $p \in C$ (i.e., the "payment" table).

**Definition 4 (Records).** *Let $DM = (C, A, classAttr, val, PK, FK, classPK, classFK, keyRel, keyAttr, refAttr)$ be a data model and $M^{DM} = \{map \in A \not\to V \mid \forall a \in dom(map) : map(a) \in val(a)\}$ be the set of mappings of $DM$. $R^{DM} = \{(c, map) \in C \times M^{DM} \mid dom(map) = classAttr(c)\}$ is the set of all records of $DM$ if $\forall (c, map), (c, map') \in R^{DM} : (\forall a \in keyAttr(classPK(c)) : map(a) = map'(a)) \Rightarrow map = map'$.*

Given a data model, Definition 4 formalizes the possible records of the model. For instance, the first record in the "payment_line" table can be formalized as $(c, map)$ where $c = pl$ , $dom(map) = \{id, payment, invoice, amount\}$, and $map(id) = pl1$, $map(payment) = p1$, $map(invoice) = i1$, $map(amount) = 2380$.

**Definition 5 (Data Set).** *An object-centric data set is a tuple $DS = (DM, RS)$ where $DM \in \mathcal{U}_{DM}$ is a data model and $RS \subseteq R^{DM}$ is a set of records of $DM$. $\mathcal{U}_{DS}$ is the universe of data sets.*

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

### 3.2 Formalizations of Database Changes

After formalizing the data perspective (i.e., data model and records), this subsection formalizes the changes (i.e., adding, updating, or deleting records) in databases.

**Definition 6 (Change Types).** *Let $DM = (C, A, classAttr, val, PK, FK, classPK, classFK, keyRel, keyAttr, refAttr)$ be a data model. $CT^{DM} = CT^{DM}_{add} \cup CT^{DM}_{upd} \cup CT^{DM}_{del}$ is the set of change types composed of the following pairwise disjoint sets:*

- *$CT^{DM}_{add} = \{(\oplus, c, A') \mid c \in C \wedge A' \subseteq classAttr(c) \wedge keyAttr(classPK(c)) \subseteq A'\}$ are the change types for adding records of $DM$,*
- *$CT^{DM}_{upd} = \{(\oslash, c, A') \mid c \in C \wedge A' \subseteq classAttr(c) \wedge keyAttr(classPK(c)) \cap A' = \emptyset\}$ are the change types for updating records of $DM$, and*
- *$CT^{DM}_{del} = \{(\ominus, c, A') \mid c \in C \wedge A' = classAttr(c)\}$ are the change types for deleting records of $DM$.*

*$\mathcal{U}_{CT}$ is the universe of change types.*

**Definition 7 (Changes).** *Let $DM$ be a data model, $R^{DM}$ be the set of records of $DM$, $CT^{DM} = CT^{DM}_{add} \cup CT^{DM}_{upd} \cup CT^{DM}_{del}$ be the set of change types and $map_{null} \in \emptyset \rightarrow V$ be a function with the empty set as domain. $CH^{DM} = CH^{DM}_{add} \cup CH^{DM}_{upd} \cup CH^{DM}_{del}$ is the set of changes composed of the following pairwise disjoint sets:*

- *$CH^{DM}_{add} = \{(\oplus, c, A', map_{old}, map_{new}) \mid (\oplus, c, A') \in CT^{DM}_{add} \wedge map_{old} = map_{null} \wedge (c, map_{new}) \in R^{DM}\}$,*
- *$CH^{DM}_{upd} = \{(\oslash, c, A', map_{old}, map_{new}) \mid (\oslash, c, A') \in CT^{DM}_{upd} \wedge (c, map_{old}) \in R^{DM} \wedge (c, map_{new}) \in R^{DM}\}$, and*
- *$CH^{DM}_{del} = \{(\ominus, c, A', map_{old}, map_{new}) \mid (\ominus, c, A') \in CT^{DM}_{del} \wedge (c, map_{old}) \in R^{DM} \wedge map_{new} = map_{null}\}$.*

A *change* $(op, c, A', map_{old}, map_{new})$ corresponds to an SQL sentence, i.e., adding, updating or deleting a record in a table. Its *change type* $(op, c, A')$ indicates which table (i.e., $c$) the record is in, which columns (i.e., $A'$) of the record are impacted and how the record is impacted (indicated by $op$, i.e., adding, updating or deleting), while its old and new mappings (i.e., $map_{old}$ and $map_{new}$) specify the contents of the record before and after the change, respectively. Note that, $A'$ provides more power to produce different change types, i.e., changes impacting different attributes in the same table can be clarified into different change types, which is not considered in [5].

**Definition 8 (Change Occurrence, Change Log).** *Let $DM$ be a data model and $CH^{DM}$ be the set of changes of $DM$. Let $TS$ be the universe of timestamps. $CO^{DM} = CH^{DM} \times TS$ is the set of all possible change occurrences of $DM$. A change log $CL = \langle co_1, co_2, ..., co_n \rangle \in (CO^{DM})^*$ is a sequence of change occurrences such that time is non-decreasing, i.e., $ts_i \leqslant ts_j$ for any $co_i = (ch_i, ts_i)$ and $co_j = (ch_j, ts_j)$ with $1 \leqslant i < j \leqslant n$. $\mathcal{U}_{CL}$ is the universe of change logs.*

A *change occurrence* $co = (ch, ts)$ represents a change $ch$ happened at $ts$. It corresponds to one row in the redo log or CDHDR table. A *change log* consists of a list of change occurrences which are sorted by timestamps such that time is non-decreasing. Note that, *change logs* record behavior on the database level (e.g., an execution of "insert an order record") while XOC logs record behavior on the information system level (e.g., a "create order" operation) (cf. Section 2.2).

**Definition 9 (Effect of a Change).** *Let $DM$ be a data model. $CH^{DM}$ is the set of changes of $DM$ and $ch = (op, c, A', map_{old}, map_{new}) \in CH^{DM}$ is a change. $DS_{old} = (DM, RS_{old})$ and $DS_{new} = (DM, RS_{new})$ are two data sets. $DS_{new}$ is generated after the change $ch$ on $DS_{old}$ and denote $DS_{old} \xrightarrow{ch} DS_{new}$ if and only if*

- $RS_{new} = \{(c', map') \in RS_{old} \mid (c', map') \neq (c, map_{old})\} \cup \{(c, map_{new}) \mid op \neq \ominus\}$ *or*
- $RS_{old} = \{(c', map') \in RS_{new} \mid (c', map') \neq (c, map_{new})\} \cup \{(c, map_{old}) \mid op \neq \oplus\}$

**Definition 10 (Effect of a Change Log).** *Let $DM$ be a data model and $CL = \langle co_1, co_2, ..., co_n \rangle \in \mathcal{U}_{CL}$ be a change log. There exist data sets $DS_0$,$DS_1$,...,$DS_n \in \mathcal{U}_{DS}$ such that $DS_0 \xrightarrow{co_1} DS_1 \xrightarrow{co_2} DS_2 ... \xrightarrow{co_n} DS_n$. Hence, change log CL results in data set $DS_n$ when starting in $DS_0$. This is denoted by $DS_0 \xrightarrow{CL} DS_n$.*

Given a data set $DS_{old} = (DM, RS_{old})$, a change results in a new data set $DS_{new}$ through adding, updating or deleting a record in the record set $RS_{old}$ of $DS_{old}$ (cf. Definition 9). Similarly, a change log results in a data set $DS_n$ through orderly accumulating the effects of all changes on the initial data set $DS_0$, indicated by Definition 10.

### 3.3 Formalizations of Extracting Logs

A XOC log $(E, act, refer, om, \prec)$ consists of events, event types, object references, object models and event ordering. For extracting events, one needs to specify some event types based on domain knowledge which indicates the relations between event types (classifying behavior at the information system level) and change types (classifying behavior at the database level)(cf. Section 2.2). An example of the knowledge is that the "create order" event type consists of two change types, i.e., "insert one order record" and "insert one or more order line records", since a click on the "create order" button inserts one record in the "order" table and one or more records in the "order_line" table.

**Definition 11 (Cardinalities).** $\mathcal{U}_{Card} = \{X \in \mathcal{P}(\mathbb{N}) \setminus \{\emptyset\}\}$ *defines the universe of all cardinalities. A cardinality (an element of $\mathcal{U}_{Card}$) specifies a non-empty set of integers.*

**Definition 12 (Event Types).** *Let $DM$ be a data model and $CT^{DM}$ be the set of change types of $DM$. $ET^{DM} = \{et \in \mathcal{P}(\mathcal{U}_{Card} \times CT^{DM}) \setminus \{\emptyset\} \mid \forall (card_1, ct_1), (card_2, ct_2) \in et : (ct_1 = ct_2 \Rightarrow card_1 = card_2)\}$ is the set of event types of $DM$.*

An *event type* is defined as a set of tuples of a *cardinality* and a change type, where the cardinality describes the quantitative relation between the event type and the change type. For example, the "create order" event type is denoted as $\{(\{1\}, (\oplus, o, A'_1)), (\{1..*\}, (\oplus, ol, A'_2))\}$ which means a "create order" event adds precisely one record in the "order" ($o$) table, and at least one record in the "order_line" ($ol$) table. Definition 12 specifies a concrete realization for event types while $\mathcal{U}_{ET}$ (cf. Definition 2) only abstractly defines the universe.

**Definition 13 (Events).** *Let $DM$ be a data model, $CO^{DM}$ be the set of change occurrences of $DM$ and $ET^{DM}$ be the set of event types of $DM$. $E^{DM} = \{e \in$*

$(CO^{DM})^* \setminus \{\emptyset\} \mid \forall (ch_i, ts_i), (ch_j, ts_j) \in e : (i < j \Rightarrow ts_i \leqslant ts_j)\}$ *is the set of events of* $DM$. *Function* $possibleE \in ET^{DM} \rightarrow \mathcal{P}(E^{DM})$ *returns possible events of an event type such that* $possibleE(et) = \{e \in E^{DM} \mid \forall((op, c, A', map_{old}, map_{new}), ts) \in e : (\exists card \in \mathcal{U}_{card} : (card, (op, c, A')) \in et) \wedge \forall(card', (op', c', A'')) \in et : |\{((op', c', A'', map'_{old}, map'_{new}), ts') \in e\}| \in card'\}$.

An event is a non-empty sequence of change occurrences such that time is non-decreasing. Function $possibleE$ gives all possible events corresponding to one event type. For instance, $e = \langle((\oplus, o, A, map_{old}, map_{new}), ts), ((\oplus, ol, A', map'_{old}, map'_{new}), ts)\rangle$ is a possible event of the event type "create order" ($co$), i.e., $e \in possibleE(co)$. Definition 13 specifies a concrete realization for events.

**Definition 14 (Extracting Event Types).** *Let* $DM$ *be a data model.* $E \subseteq E^{DM}$ *is a set of events.* $ET \subseteq ET^{DM}$ *is a predefined set of event types where* $\forall et_1, et_2 \in ET : possibleE(et_1) \cap possibleE(et_2) = \emptyset$. *Function* $extractET \in E \rightarrow ET$ *maps an event to an event type such that* $extractET(e) = et$ *where* $e \in possibleE(et)$.

Given a predefined set of event types whose possible events are disjoint, function $extractET$ maps an event to an event type in the set. In this paper, we assume there exists exactly one possible type in the predefined set for the event.

**Definition 15 (Mapping Record into Object).** *Let* $DM = (C, A, classAttr, val, PK, FK, classPK, classFK, keyRel, keyAttr, refAttr)$ *be a data model and* $R^{DM}$ *be the set of records of* $DM$. *Function* $extractO \in R^{DM} \rightarrow \mathcal{U}_O$ *maps a record* $(c, map)$ *to an object such that* $extractO((c, map)) = (c, map_k) \in C \times M^{DM}$ *where*

- $dom(map_k) = keyAttr(classPK(c))$, *and*
- $\forall a \in dom(map_k) : map_k(a) = map(a)$.

Function $extractO$ filters in the mapping for attributes corresponding to the primary key of a record, resulting in an object. This function specifies a concrete realization, i.e., a tuple $(c, map_k)$, for each object in $\mathcal{U}_O$ (cf. Definition 1).

**Definition 16 (Mapping Data Set into Object Model).** *Let* $DS = (DM, RS)$ *be a data set where* $DM = (C, A, classAttr, val, PK, FK, classPK, classFK, keyRel, keyAttr, refAttr)$ *is a data model and* $RS$ *is a set of records. Function* $extractOM \in \mathcal{U}_{DS} \rightarrow \mathcal{U}_{OM}$ *maps a data set to an object model such that* $extractOM(DS) = (Obj, Rel, class, objectAttr)$ *where*

- $Obj = \{extractO(r) \mid r \in RS\}$,
- $Rel = \{(rt, extractO((c, map)), extractO((c', map')) \mid rt = (c, pk, c', fk) \in C \times PK \times C \times FK \wedge (c, map) \in RS \wedge (c', map') \in RS \wedge pk = classPK(c) \wedge fk \in classFK(c') \wedge keyRel(fk) = pk \wedge \forall a \in keyAttr(fk) : map'(a) = map(refAttr(fk, a))\}$,
- $\forall (c, map_k) \in Obj : class((c, map_k)) = c$, *and*
- $\forall (c, map_k) \in Obj : objectAttr((c, map_k)) = map$ *where* $(c, map) \in RS$ *and* $extractO((c, map)) = (c, map_k)$.

Function $extractOM$ maps a data set into an object model. More precisely, if each attribute value corresponding to a foreign key of a record $r$ is equal to the value of the attribute identified by function $refAttr$ in another record $r'$, there exists an object relation between $extractO(r)$ and $extractO(r')$.

**Definition 17 (Transforming Change Log Into Event Sequence).** *Function* $extractES \in \mathcal{U}_{CL} \rightarrow (\mathcal{U}_E)^*$ *extracts an event sequence from a change log such that* $\forall CL = \langle co_1, co_2, ..., co_n \rangle \in \mathcal{U}_{CL} : extractES(CL) = \langle e_1, e_2, ..., e_m \rangle$ *where*

- $\{co \in CL\} = \{co' \mid co' \in e_i \wedge e_i \in \langle e_1, e_2, ..., e_m \rangle\}$, *and*
- $\forall e_i, e_j \in \langle e_1, e_2, ..., e_m \rangle : (\forall (ch, ts) \in e_i, (ch', ts') \in e_j : i < j \Rightarrow ts < ts')$.

*A reverse function* $restoreES \in (\mathcal{U}_E)^* \rightarrow \mathcal{U}_{CL}$ *which restores an event sequence to a change log such that* $restoreES(\langle e_1, e_2, ..., e_m \rangle) = \langle co_1, co_2, ..., co_n \rangle$.

Function $extractES$ transforms a change log (i.e., a change occurrence sequence) into an event sequence by grouping change occurrences at the *same time* into an event without modifying the order of these change occurrences. Note that, it is possible to group change occurrences based on *domain knowledge* instead of *time*, which provides more freedom for extracting events (e.g., overlapping events). On the contrary, function $restoreES$ transforms an event sequence into a change occurrence sequence, i.e., $restoreES(\langle e_1, e_2, ..., e_m \rangle) = \langle co_1, co_2, ..., co_n \rangle$ if and only if $extractES(\langle co_1, co_2, ..., co_n \rangle) = \langle e_1, e_2, ..., e_m \rangle$, .

**Definition 18 (Extracting XOC Event Log).** *Function* $extractL \in \mathcal{U}_{DS} \times \mathcal{U}_{CL} \rightarrow \mathcal{U}_L$ *extracts a XOC event log from a data set and a change log such that* $\forall DS \in \mathcal{U}_{DS}, CL \in \mathcal{U}_{CL} : extractL(DS, CL) = (E, act, refer, om, \prec)$ *where*

- $E = \{e \in \langle e_1, e_2, ..., e_m \rangle\}$ *where* $\langle e_1, e_2, ..., e_m \rangle = extractES(CL)$,
- $act = extractET$,
- $\forall e_i \in E : refer(e_i) = \{o \mid \exists((op, c, A', map_{old}, map_{new}), ts) \in e_i : o = extractO(c, map_{new}) \wedge op \neq \ominus\} \cup \{o \mid \exists((\ominus, c, A', map_{old}, map_{new}), ts) \in e_i : o = extractO(c, map_{old})\}$,
- $\forall e_i \in E : om(e_i) = OM_i$ *such that* $OM_i \overset{CL'}{\rightarrow} extractOM(DS)$ *where* $CL' = restoreES(tl^{m-i}(\langle e_1, e_2, ..., e_m \rangle))$ *and* $\langle e_1, e_2, ..., e_m \rangle = extractES(CL)$,[7] *and*
- $\prec = \{(e_i, e_j) \mid e_i \in E \wedge e_j \in tl^{m-i}(\langle e_1, e_2, ..., e_m \rangle)$ *where* $\langle e_1, e_2, ..., e_m \rangle = extractES(CL)$.

Function $refer$ is obtained through identifying the impacted (added, updated or deleted) objects by each event. The data set indicates the object model (i.e., $extractOM(DS)$) corresponding to the last event (i.e., $e_m$). In order to obtain the object model corresponding to one event $e_i$, we get (i) its suffix event sequence $tl^{m-i}(\langle e_1, e_2, ..., e_m \rangle)$ (i.e., $\langle e_{i+1}, e_{i+2}, ..., e_m \rangle$), (ii) the corresponding change sequence $CL'$ through $extractES$ and (iii) the object model $OM_i$ through $\rightarrow$.

## 4 Implementation

Our approach has been implemented in the "XOC Log Generator" Plugin in *ProM 6 Nightly builds*, which takes tables (from a database or csv files) as input and automatically generates a XOC log. For instance, taking the motivating data as input, this plugin generates an event log with 10 events, the first five ones of which are shown in Table 3.[8]

A XOC log reveals the evolution of a database along with the events from its corresponding information system. As shown in Figure 3, after the occurrence of an event

[7] $tl^k(\sigma)$ means to get the last k elements of a sequence $\sigma$.

[8] The involved data and more real life examples can be found at http://www.win.tue.nl/ocbc/.

(denoted by a black dot), its corresponding object model (denoted by a cylinder) is updated by adding, updating or deleting objects (highlighted in red). Besides, XOC logs provide methods to correlate events by objects. For instance, two events are correlated if they refer to (indicated by dotted lines) two related objects, e.g., $co1$ and $cs1$ are correlated since $co1$ refers to $ol1$, $cs1$ refers to $sl1$ and $ol1$ is related to $sl1$.

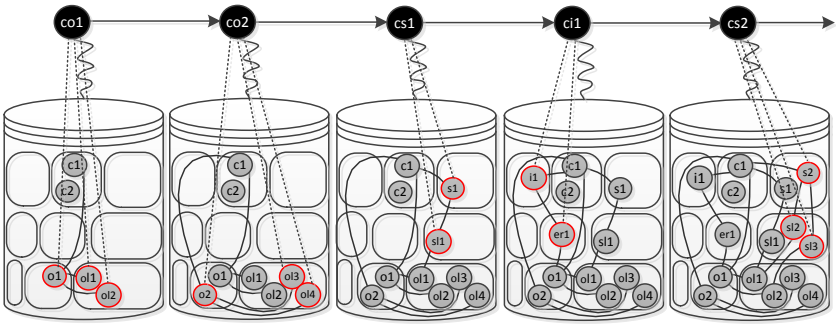


**Fig. 3.** A XOC log (in Table 3) revealing the evolution of a database.

Figure 4 shows the comparison between the XES log and the XOC log derived from the motivating data in Figure 1. More precisely, Figure 4(a) reveals the behavioral perspective of the XOC log after correlating events and Figure 4(b) shows two cases $o1$ and $o2$. Our log is able to represent one-to-many and many-to-many relations by removing the case notion, while the XES log splits events based on a straightjacket case id. The split performed in the XES log results in two separated cases without interactions in between as well as leading to convergence (e.g., $cp1$ appears in two cases) and divergence (e.g., multiple instances of the activity $cp$ appear in $o2$) problems.

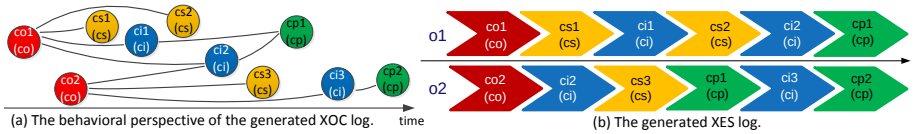


**Fig. 4.** A comparison which indicates XOC logs can avoid data convergence and divergence.

Besides the advantages over the existing log format, XOC logs open a door for other techniques. Based on XOC logs, we can discover *object-centric behavioral constraint (OCBC)* models [8], as shown in Figure 5. Figure 6 shows parts of three example models discovered from the generated XES log in Figure 4(b), which displays the problems caused by data convergence and divergence. In (a) we see that the negative relation denoted by $d5$ with a dotted line cannot be discovered in the declare model, since in case $o2$, the "create payment" event $cp1$ is wrongly related to its subsequent "create invoice" event $ci3$. The discovered directly follow graph (DFG) in (b) indicates that "create invoice" happens 4 times, violating the real frequency of 3 times due to the duplication of $ci2$. In (c) we observe that, since the multiple instance of activities "create invoice" and "create payment" cannot be distinguished, two loops with two implicit transitions are discovered, resulting in an imprecise Petri net. In contrast, since XOC logs are

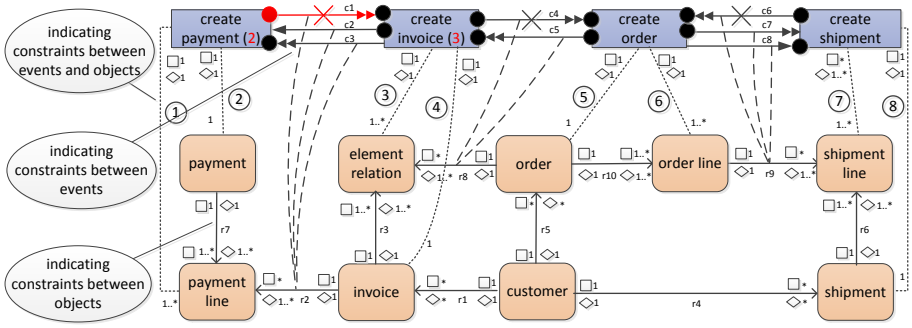


**Fig. 5.** The OCBC model discovered from the generated XOC log.

not affected by convergence and divergence, the discovered OCBC model (cf. Figure 5) can show the correct frequencies (i.e., numbers in red) and the missing negative relation (i.e., $c1$)[9], avoiding the problems confronted by these three models. Besides, compared with existing modeling notations, such as the models in Figure 6, OCBC models can show the data perspective, the behavioral perspective and the interactions in between in a single diagram. Moreover, since XOC logs can correlate events by objects and have a richer data perspective, they can be used to *check conformance* for more implicit deviations by considering multiple instances and data objects [14]. Additionally,since the discovered OCBC model indicates constraints in/between objects and events, it is possible to *predict* future events and objects based on observed ones.

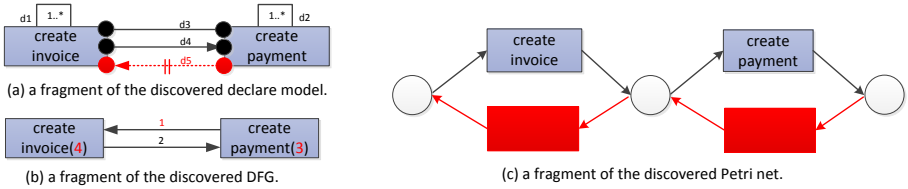


**Fig. 6.** Problems suffered by three kinds of models discovered from the generated XES log.

## 5 Related Work

In this section, we review the existing event log formats and introduce some researches which extract event logs from databases.

**Event Log Formats**. Event logs serve as the input for many process mining techniques. The XES log format [1] which stands for *eXtensible Event Stream (www.xes-standard.org)* is the de facto exchange format for process mining. This format is supported by tools such as ProM and implemented by OpenXES, an open source java library for reading, storing and writing XES logs. The XES format cannot deal well with object-centic data. [4] proposes a meta model to abstract the object-centric data and

[9] The notation for the negative relation in OCBC models is different from that in declare models.

redo logs into different types of entities such as attributes, objects, relations and events. The meta model covers both behavioral and data perspectives and builds a bridge to connect databases with process mining. This meta model is not a concrete log format, but transforms the object-centric data into "bricks" which can constitute logs to enable process mining techniques. Our XOC log format combines the "bricks" from [4] in an object-centric way, i.e., it focuses more on the data perspective unlike the XES format. Objects are replacing a case notion to correlate events, which makes XOC logs able to deal with one-to-many and many-to-many relations. Moreover, through defining object models, a XOC log reveals the evolution of a database through time.

**Extracting Event Logs**. In order to obtain event logs, researches propose a number of techniques and tools. A popular tool is XESame[10], which is used to convert databases into XES event logs. [13] conceptualizes a database view over event data based on the idea that events leave footprints by changing the underlying database, which are captured by redo logs of database systems. Triggered by this idea, [5] proposes a method to extract XES logs from databases by identifying a specific trace id pattern. [3] proposes an approach to flexibly extract XES event logs from relational databases, by leveraging the ontology-based data access paradigm and [2] provides a tool (*onprom*) to extract event logs from legacy data based on an ontology. In order to obtain logs from non-process-aware information systems, [11] correlates events into process instances using similarity of their attributes. [7] provides an overview of the decisions that impact the quality of the event logs constructed from database data. [6] proposes an analysis system which allows users to define events, resources and their inter-relations to extract logs from SAP transactions. [12] addresses merging logs produced by disintegrated systems that cover parts of the same process by choosing a "main" process. Artifact-centric approaches [10, 9] try to extract artifacts (i.e., business entities) and address the possibility of many-to-many relationships between artifacts. Compared with the existing approaches, our approach outputs object-centric logs, i.e., XOC logs, rather than process-centric logs, i.e., XES logs. The main advantage of object-centric logs is the new kinds of analyses that they enable. Data-aware process model discovery [8], and new conformance checking techniques [14] that exploit data relations are examples of approaches directly applicable to XOC logs. Another important contribution is that our approach supports the abstraction from low level database events (like the ones obtained in [5]) to high level events (e.g., from the original information system).

## 6 Conclusion

In this paper we proposed an approach to extract event logs from object-centric data (i.e., database tables), using the eXtensible Object-Centric (XOC) log format. The XOC format provides a process mining view on databases. Compared with existing log formats, such as XES, it has the following advantages:

- By removing the case notion and correlating events with objects, XOC logs can perfectly deal with one-to-many and many-to-many relations, avoiding convergence and divergence problems and displaying interactions between different instances.
- An object in XOC logs contains as much information as its corresponding record in the database. By extending the data perspective, XOC logs retain the data quality.

[10] http://www.processmining.org/xesame/start

– The object model of an event represents a snapshot of the database just after the event occurrence. Based on this idea, the log provides a view of the evolution of the database, along with the operations which triggered changes in the database.

Besides, XOC logs serve as a starting point for a new line of future techniques. Based on experiments implemented on the generated XOC logs, it is possible to discover OCBC models (cf. Figure 5) to describe the underlying process in an object-centric manner [8]. Additionally, taking a XOC log and an OCBC model as input, many deviations which cannot be detected by existing approaches, can be revealed by new conformance checking techniques [14]. Moreover, prediction of future events and objects is also enabled according to the constraints indicated by the discovered OCBC model. Note that, the OCBC discovery, conformance checking and prediction are just examples of potential applications of XOC logs. It is possible to propose more techniques based on XOC logs, e.g., discovering other types of data-aware process models.

## References


1. IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams. *IEEE Std. 1849-2016*, pages i–48, 2016.
2. D. Calvanese, T.E. Kalayci, M. Montali, and S. Tinella. Ontology-Based Data Access for Extracting Event Logs from Legacy Data: The onprom Tooland Methodology. In *BIS 2017*, pages 220–236. Springer, 2017.
3. D. Calvanese, M. Montali, A. Syamsiyah, and W.M.P. van der Aalst. Ontology-Driven Extraction of Event Logs from Relational Databases. In *BPM 2015 workshops*, pages 140–153. Springer, 2015.
4. E. González López de Murillas, H.A. Reijers, and W.M.P. van der Aalst. Connecting Databases with Process Mining: A Meta Model and Toolset. In *International workshop on business process modeling, development and support*, pages 231–249. Springer, 2016.
5. E. González López de Murillas, W.M.P. van der Aalst, and H.A. Reijers. Process Mining on Databases: Unearthing Historical Data from Redo Logs. In *International Conference on Business Process Management*, pages 367–385. Springer, 2015.
6. J.E. Ingvaldsen and J.A. Gulla. Preprocessing Support for Large Scale Process Mining of SAP Transactions. In *BPM 2007 workshops*, pages 30–41. Springer, 2007.
7. M. Jans and P. Soffer. From Relational Database to Event Log: Decisions with Quality Impact. In *International Workshop on Quality Data for Process Analytics*. Springer, 2017.
8. G. Li, R.M. de Carvalho, and W.M.P. van der Aalst. Automatic Discovery of Object-Centric Behavioral Constraint Models. In *BIS 2017*, pages 43–58. Springer, 2017.
9. X. Lu, M. Nagelkerke, D. van de Wiel, and D. Fahland. Discovering Interacting Artifacts from ERP Systems. *IEEE Transactions on Services Computing*, 8(6):861–873, 2015.
10. A. Nigam and N.S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.
11. R. Prez-Castillo and et al. Assessing Event Correlation in Non-Process-Aware Information Systems. *Software & Systems Modeling*, 13(3):1117–1139, 2014.
12. L. Raichelson and P. Soffer. Merging Event Logs with Many to Many Relationships. In *BPM 2014 workshops*, pages 330–341. Springer, 2014.
13. W.M.P. van der Aalst. Extracting Event Data from Databases to Unleash Process Mining. In *BPM-Driving innovation in a digital world*, pages 105–128. Springer, 2015.
14. W.M.P. van der Aalst, G. Li, and M. Montali. Object-Centric Behavioral Constraints. Corr technical report, arXiv.org e-Print archive, 2017. Available at https://arxiv.org/abs/1703.05740.