Journal of Intelligent Information Systems (2023) 61:835–857
https://doi.org/10.1007/s10844-023-00799-9

RESEARCH

A generic approach to extract object-centric event data from databases supporting SAP ERP

Alessandro Berti[1] · Gyunam Park[1] · Majid Rafiei[1] · Wil M.P. van der Aalst[1]

Received: 13 January 2023 / Revised: 22 May 2023 / Accepted: 23 May 2023 /
Published online: 30 June 2023

Abstract

Process mining provides a collection of techniques to gain insights into business processes by analyzing event logs. Organizations can gain various insights into their business processes by using process mining techniques. Such techniques use event logs extracted from relational databases supporting the business process as input. However, extracting event logs is challenging due to the size of the data, and it remains ad-hoc. Existing commercial tools partly support the extraction of event logs, but they are proprietary and focus on the mainstream processes such as Purchase-To-Pay (P2P) and Order-To-Cash (O2C). Moreover, the extracted event logs suffer from well-known deficiency, convergence, and divergence issues. For example, due to convergence events are unintentionally duplicated causing unreliable or confusing performance diagnostics. In this paper, we propose an approach to extract event logs while avoiding the aforementioned issues. More in detail, we extract object-centric event logs by using an abstraction layer of the database, called Graph of Relationships (GoRs), designing blueprints with domain knowledge, and converting the database and blueprint into object-centric event logs.We fully implemented the proposed approach, which can extract object-centric event logs from SAP ERP systems, and evaluate the utility and scalability of the proposed approach.

Keywords Object-centric event data · Graph event data · Database event log extraction · SAP ERP

1 Introduction

Process mining provides a collection of techniques for gaining insights into business processes by analyzing event logs. Such event logs are extracted from the database of information systems supporting the business process (de Murillas et al., 2015) . However, the task of extracting event logs logs is often challenging due to the size and complexity of the underlying databases.

✉ Alessandro Berti
a.berti@pads.rwth-aachen.de

Extended author information available on the last page of the article

We identify four different steps for the extraction of an event log from a database: *process identification*, *table selection*, *extraction*, and *preprocessing* (Berti et al., 2021) . First, process identification aims to identify a collection of business processes supported by information systems and choose a target business process to analyze. Next, the table selection aims to select a set of tables to extract for analyzing the business process. Third, the extraction of an event log aims to extract event logs by formulating data-retrieving queries and assigning a case identifier to correlate a group of events. Note that process mining traditionally assumes a *single case notion*, e.g., patient in a healthcare process (Diba et al., 2020) . Finally, the preprocessing aims to prepare the event log that is used by different process mining techniques.

Each of the aforementioned phases is challenging. First of all, process identification is not trivial since a database contains records of hundreds of different business processes. Despite knowing the target process to analyze, inferring the set of tables related to the target process is challenging, e.g., a database supporting an SAP ERP system contains hundreds of thousands of tables. Furthermore, formulating querying statements, e.g., SQL queries, requires a huge amount of domain knowledge. In this paper, we aim to support each step of the event log extraction.

The event log extracted by existing techniques assumes that an event is associated with a single case identifier. However, such event logs have *deficiency*, *convergence*, and *divergence* issues (van der Aalst, 2019) . The selection of a case notion may lead to the unintentional removal of events (deficiency) or the unintentional duplication of events (convergence). Also, causalities may get lost leading to spaghetti-like process models (convergence).Recently object-centric event logs have been introduced as a more natural way to represent event data extracted from real-life databases (van der Aalst and Berti, 2020) . An event in object-centric event logs can be associated with several objects of different object types. In this paper, we aim to extract object-centric event logs from relational databases to avoid the issues that a traditional event log has.

To summarize, this paper introduces an approach to extracting object-centric event logs by supporting process analysts in each step of the event log extraction. To that end, we first introduce an abstraction of the database and a representation of the main relationships at the database level, called *Graph of Relationships (GoRs)*. The abstraction layer supports the process identification and table selection steps. Next, we introduce *blueprints* that specify how the data of selected tables are associated with the events of object-centric event logs to support the extraction step.

Moreover, we present an implementation on top of a database supporting SAP ERP. We introduce the list of queries needed to retrieve: i) the graph of relationships (nodes and edges); ii) the rows of the selected set of tables, which are needed to apply the blueprint. We evaluate that the execution time of such queries is acceptable, and the extracted event logs provide insightful knowledge of the business process.

The contribution in Berti et al. (2021) exploits the relational structure of SAP in order to build a Graph of Relationships (GoR) that is useful for identifying the set of tables of interest for a process. After the selection of the set of tables, an object-centric event log is extracted from the relational database supporting the SAP instance. The contribution of the current paper extends (Berti et al., 2021) by:

- Clearly defining the abstraction at the database level and connecting the GoR to the underlying table entries.
- Introducing a database abstraction.
- Describing in detail the database abstraction used on SAP.

- Introducing the concept of blueprint to define different activity/timestamp/related object concepts on each table entry.

Moreover, the tool support has been improved, thanks to the usage of graph databases, especially in the process identification and selection phases.

The rest of the paper is organized as follows. Section 2 introduces the concept of object-centric event logs and corresponding process mining approaches. Section 3 defines the database abstraction used throughout the paper, along with the translation to an object-centric event log. Section 4 shows how the database abstraction can be built on top of SAP ERP instances, introduces the tool support, and proposes an assessment of the methods. Section 5 presents the related work. Eventually, Section 6 concludes the paper.

2 Object-centric process mining

Object-centric process mining techniques have been developed in the last years, considering the intrinsic relationships between the objects (i.e., orders, invoices) and the events (i.e., the creation of an invoice) recorded in the database. Three different types of relationships are used by object-centric process mining techniques: event-to-object relationships (e.g., the creation of an invoice is related to an invoice and some orders), object-to-object relationships (e.g., an order is related to the corresponding order items) and event-to-event relationships (e.g., events of invoice approval executed in the same batch).

The current standard for the storage of object-centric event logs is the OCEL standard http://www.ocel-standard.org/, which allows the storage of the events and the objects (along with their attributes), along with the event-to-object relationships (Ghahfarokhi et al., 2021) . A formal definition of OCEL is proposed in Def. 2 (after the introduction of some universes in Def. 1). Two different implementations of the standard are provided (JSON-OCEL and XML-OCEL), which are based on popular file formats.

Definition 1 (Universes) $\mathbb{U}_\Sigma$ is the universe of all the strings; $\mathbb{U}_e$ is the universe of event identifiers; $\mathbb{U}_{act}$ is the universe of activities; $\mathbb{U}_{timest}$ is the universe of timestamps; $\mathbb{U}_{att}$ is the universe of attribute names; $\mathbb{U}_{val}$ is the universe of attribute values; $\mathbb{U}_{typ}$ is the universe of attribute types; $\mathbb{U}_o$ is the universe of object identifiers; $\mathbb{U}_{ot}$ is the universe of object types; $\mathbb{U}_{dom}$ is the universe of attribute domains; $\mathbb{U}_{transact}$ is the universe of database transactions.

An example object-centric event log is contained in Table 1. There, a tabular notation is used, where each row is an event of the object-centric event log. The first row is related to the event with the identifier e_1.

Definition 2 (Object-Centric Event Log) An object-centric event log is a tuple $L = (E, AN, AV, OT, O, \pi_{act}, \pi_{time}, \pi_{vmap}, \pi_{omap}, \pi_{otyp}, \pi_{ovmap}, \leq)$ such that:

- $E \subseteq \mathbb{U}_e$ is a set of event identifiers.
- $AN \subseteq \mathbb{U}_{att}$ is a set of attribute names.
- $AV \subseteq \mathbb{U}_{val}$ is a set of attribute values (with the requirement that $\mathrm{AN} \cap \mathrm{AV} = \emptyset$).
- $OT \subseteq \mathbb{U}_{ot}$ is a set of object types.
- $O \subseteq \mathbb{U}_o$ is a set of object identifiers.
- $\pi_{act} : E \rightarrow \mathbb{U}_{act}$ is a function associating an event (identifier) to its activity.
- $\pi_{time} : E \rightarrow \mathbb{U}_{timest}$ is a function associating an event (identifier) to a timestamp.
- $\pi_{vmap} : E \rightarrow (AN \nrightarrow AV)$ is a function associating an event (identifier) to its attribute value assignments.

Table 1 Informal representation of the events of an OCEL

id	activity	timestamp	order	item	package	customer	resource	price
e_1	*place order*	*2022-07-09 08:20:01.527+01:00*	$\{o_1\}$	$\{i_1, i_2, i_3\}$	$\emptyset$	$\{c_1\}$	*John*	*200.0*
e_2	*confirm order*	*2022-07-10 09:23:01.527+01:00*	$\{o_1\}$	$\emptyset$	$\emptyset$	$\emptyset$	*Jane*	*302.0*
e_3	*check availability*	*2022-07-10 17:10:08.527+01:00*	$\{o_1\}$	$\{i_1\}$	$\emptyset$	$\emptyset$	*Marc*	*125.0*
...	...	...	...	...	...	...	...	

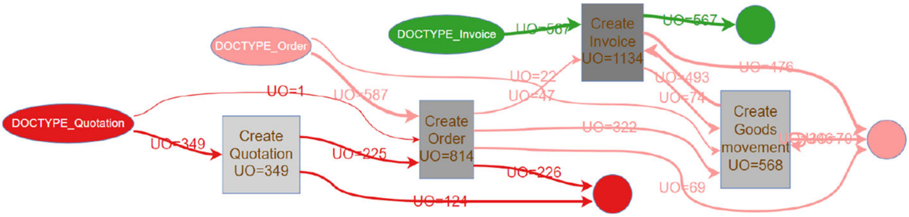

Fig. 1 Object-centric directly-follows graph of an Order-to-Cash process discovered using the OC-PM tool

- $\pi_{omap} : E \rightarrow \mathcal{P}(O)$ is a function associating an event (identifier) to a set of related object identifiers.
- $\pi_{otyp} \in O \rightarrow OT$ assigns precisely one object type to each object identifier.
- $\pi_{ovmap} : O \rightarrow (AN \not\rightarrow AV)$ is a function associating an object to its attribute value assignments.
- $\leq$ is a total order (i.e., it respects the antisymmetry, transitivity, and connexity properties).

We can see that the following properties are associated with the event e_1 in the table: $\pi_{act}(\mathrm{e}_1)$ = place order, $\pi_{time}(\mathrm{e}_1) =$ 2022-07-09 08:20:01.527+01:00, $\pi_{omap}(\mathrm{e}_1) = \{\mathrm{o}_1, \mathrm{i}_1, \mathrm{i}_2, \mathrm{i}_3, \mathrm{c}_1\}$, so the event is associated to an object of type order (o_1) three objects of type item ($\mathrm{i}_1, \mathrm{i}_2, \mathrm{i}_3$) and one object of type customer (c_1). Moreover, $\pi_{vmap}(\mathrm{e}_1)(\mathrm{resource}) =$ John and $\pi_{vmap}(\mathrm{e}_1)(\mathrm{price}) = 200.0$.

Some tools are available for object-centric process mining. In particular, the OC-PM tool https://www.ocpm.info provides a rich set of object-centric process mining features: ingestion/exporting of object-centric event logs in the OCEL standard format (JSON-OCEL and XML-OCEL); flattening the object-centric event logs into traditional event logs with the choice of a case notion; advanced preprocessing features (filtering, sampling); discovering object-centric process models: object-centric directly-follows graphs (Berti and van der Aalst, 2022) , object-centric Petri nets (van der Aalst and Berti, 2020) , and object-centric BPMN models; conformance checking on object-centric event logs based on declarative and temporal constraints (log skeleton, temporal profile); exploration of the events/objects of the object-centric event log; machine learning (anomaly detection, correlation analytics, advanced conformance checking).

In particular, Fig. 1 shows an example of object-centric directly-follows multigraph of an Order-to-Cash process containing three different object types (quotation, order, and invoice). The event-to-object relationships are exploited to show in the model the activities in which the interaction between different object types occurs. For example, the activity *Create Order* involves objects of type quotation and order, and we can understand that the quotation is placed before the order. We can also see that the lifecycle of the objects of type quotation starts with the activity *Create Quotation* and either terminates (when the quotation is not converted to order) or is completed with the activity *Create Order*.

Object-centric process mining is also supported by commercial software. For example, the Celonis vendor recently introduced the *Process Sphere* feature, which allows "to analyze and visualize the complex relationships between events and objects across interconnected processes"[1] and can ingest object-centric event logs in the OCEL standard.

[1] https://shorturl.at/azOT9.

3 Approach

In this section, we introduce an approach for extracting object-centric event logs from relational databases. First, the proposed approach supports the table selection step (i.e., identifying a set of tables necessary to extract OCELs) by providing *Graph of Relationships (GoRs)*. A GoR graphically represents the key tables and object types of a database that are related to a specific process of interest. Process analysts can explore the graph to select the essential tables. Next, the approach supports the extraction step (i.e., extracting OCELs) in two phases. First, we provide the concept of *blueprints* that allow for the extraction of primitive events. Eventually, we automatically translate relational databases and blueprints into OCELs.

3.1 Step 1: Supporting table selection

In this section, we introduce the concept of GoRs. A GoR highlights the key tables and object types of a database of an information system, allowing one to select a set of interconnected tables corresponding to a process in the information system. The visual information provided by a GoR helps the process identification and table selection steps.

To define the GoR and support the following extraction steps, we need to propose an abstraction of the concept of database (Def. 3), containing both the relational structure (the tables, along with their attributes and interconnections) and the data objects (along with their interconnections). This is independent of the specific technology (MySQL, Oracle, MSSQL) and implementation (in some information systems, such as SAP ERP, the coherence of the schema is maintained at the application level; moreover, the database updates could be stored as REDO logs, in-table versioning, or as separate change tables (de Murillas et al., 2015)). In the current paper we do not cover the translation from a specific database concept to such abstraction (except for SAP ERP).

Definition 3 (Database Abstraction)A database is a tuple $DB = (T, O, OT, A, D, TR,$ te, tr, otyp, attdomain, transact, primary, foreign) such that:

- $T \subseteq \mathbb{U}_{table}$ is a set of tables (identifiers).
- $O \subseteq \mathbb{U}_{o}$ is a set of objects (identifiers).
- $OT \subseteq \mathbb{U}_{ot}$ is a set of object types.
- $A \subseteq \mathbb{U}_{att}$ is a set of attribute names.
- $D \subseteq \mathbb{U}_{dom}$ is a set of attribute domains.
- $TR \subseteq \mathbb{U}_{transact}$ is a set of transactions.
- te : $T \rightarrow \mathcal{P}(O \times T \times \mathbb{U}_{timest} \times (A \nrightarrow \mathbb{U}_{value}))$ is the *table entries* function associating to a table name a set of table entries, each identified by an object identifier, a reference table, a (creation) timestamp, and an attribute map. We denote $TE = O \times T \times \mathbb{U}_{timest} \times (A \nrightarrow \mathbb{U}_{value})$ as the set of table entries.
- tr : $T \times T \rightarrow \mathcal{P}(O \times O)$ is the *tables relationships* function associating to each couple of tables a set of tuples of objects that are related. Given $t_1, t_2 \in T$, $\text{tr}(t_1, t_2) \subseteq \text{AE}(t_1) \times \text{AE}(t_2)$, where $AE(t) = \{\pi_1(y) \mid y \in \text{te}(t)\}$.
- otyp : $O \rightarrow OT$ associates to every object (identifier) the corresponding object type.
- attdomain : $A \rightarrow D$ associates to every attribute name the corresponding domain.
- transact : $TE \nrightarrow TR$ associates to some table entries a transaction.
- primary : $T \rightarrow \mathcal{P}(A)$ associates to every table a set of attributes that are primary keys for the given table.

- foreign : $T \rightarrow \mathcal{P}(A)$ associates to every table a set of attributes that are foreign keys for the given table.

An important point is that the table entries introduced in Def. 3 are not the events (or the objects) of the object-centric event log. Instead, each entry could correspond to zero, one, or many events given the definition of a *blueprint* (Def. 5). The tables relationship function tr associates objects that are related in the database. The relationship could be implemented on the underlying database in many different ways (foreign keys, attributes matching without explicit foreign keys, ...), however, tr is independent from such implementation and just reports the related objects.

To build the abstraction introduced in Def. 3, we need to extract the set of tables, objects, columns/attributes along with their type, and the primary/foreign key attributes. Moreover, we assume that we can extract an object type for every object in the database. A less common assumption is recording the transaction executed against the data.

We introduce as an example for the previous definition an abstraction of a database with three tables, the third one hosting the changes operated to the objects of the other two tables:

- $T = \{t_1, t_2, \mathrm{changetab}\}$
- $O = \{o_{11}, o_{12}, o_{21}, o_{22}\}$
- $OT = \{\mathrm{ot}_1, \mathrm{ot}_2\}$
- $A = D = \emptyset$
- $TR = \{\mathrm{TRANSACT1}\}$
- $te(t_1) = \{\mathrm{ent}_{11} = (o_{11}, t_1, 2007 - 04 - 05, \mathrm{vmap}_{11}), \mathrm{ent}_{12} = (o_{12}, t_1, 2007 - 04 - 07, \mathrm{vmap}_{12})\}$
- $te(t_2) = \{\mathrm{ent}_{21} = (o_{21}, t_2, 2007 - 04 - 10, \mathrm{vmap}_{21}), \mathrm{ent}_{22} = (o_{22}, t_2, 2007 - 04 - 11, \mathrm{vmap}_{22})\}$
- $te(\mathrm{changetab}) = \{\mathrm{ent}_{31} = (o_{11}, t_1, 2007 - 04 - 15, \mathrm{vmap}^2_{11}), \mathrm{ent}_{32} = (o_{21}, t_2, 2007 - 04 - 20, \mathrm{vmap}^2_{21})\}$
- $tr(t_1, t_2) = \{(o_{11}, o_{21}), (o_{12}, o_{22})\}$
- $\mathrm{otyp}(o_{11}) = \mathrm{otyp}(o_{12}) = \mathrm{ot}_1$
- $\mathrm{otyp}(o_{21}) = \mathrm{otyp}(o_{22}) = \mathrm{ot}_2$
- $\mathrm{transact}((o_{11}, t_1, 2007 - 04 - 05, \mathrm{vmap}_{11})) = \mathrm{TRANSACT1}$

In this abstraction example, the table t_1 has two entries (the object identifiers of such entries are o_{11} and o_{12} respectively, and their object type is ot_1), and the table t_2 has two entries (the object identifiers of such entries are o_{21} and o_{22}, and their object type is ot_2). Moreover, the table changetab hosts some changes happening to the entries of the other two tables. The first change is related to the entry with object identifier o_{11} of the table t_1, while the second change is related to the entry with object identifier o_{21} of the table t_2. In the proposed abstraction, the object o_{11} is in relationship with o_{21}, while o_{12} is in relationship with o_{22} (however, we do not specify why they are in a relationship). Moreover, the transaction used to create the first entry of t_1 is *TRANSACT1*.

Def. 4 derives the GoR from the database abstraction.

Definition 4 (Graph of Relationships (GoR))Given a database DB as in Def. 3, the graph of relationships is defined as the tuple $GoR(DB) = (T, A, D, OT, TR, R_{T,T}, R_{T,A}, R_{A,D}, R_{T,OT}, R_{T,TR}, R_{T,PR}, R_{T,FR})$ where:

- T is a set of tables (identifiers).
- A is a set of attribute names.
- D is a set of attribute domains.

- OT is a set of object types.
- TR is a set of transactions.
- $R_{T,T} = \{(t_1, t_2) \in \text{dom}(\text{tr}) \mid |\text{tr}(t_1, t_2)| \geq 1\}$ is the set of connections between tables.
- $R_{T,A} = \{(t, a) \in T \times A \mid \exists_{y \in \text{te}(t)} a \in \text{dom}(\pi_4(y))\}$ is the set of connections between tables and attributes.
- $R_{A,D} = \{(a, d) \in A \times D \mid \text{attdomain}(a) = d\}$ is the set of connections between attribute names and domains.
- $R_{T,OT} = \{(t, ot) \in T \times OT \mid \exists_{y \in \text{te}(t)} \text{otyp}(\pi_1(y)) = ot\}$ is the set of connections between tables and object types.
- $R_{T,TR} = \{(t, \text{tra}) \in T \times TR \mid \exists_{y \in \text{te}(t)} y \in \text{dom}(\text{transact}) \wedge \text{transact}(y) = \text{tra}\}$ is the set of connections between tables and transactions.
- $R_{T,PR} = \{(t, \text{pr}) \in T \times A \mid a \in \text{primary}(t)\}$ is the set of connections between tables and primary keys.
- $R_{T,FR} = \{(t, \text{fr}) \in T \times A \mid a \in \text{foreign}(t)\}$ is the set of connections between tables and foreign keys.

The GoR helps in identifying the tables related to the different processes supported by the information systems. For example, in SAP the tables related to the Procure-to-Pay process (P2P) are interconnected in the GoR. To identify them, we could, for example, start from an object type (EINKBELEG), that is connected to a set of tables of interest for a given process (see Fig. 3) or from a subset of tables, which is then expanded following the edges of the GoR.

3.2 Step 2: Formulating blueprints

Next, we introduce *blueprints* associating a table to a set of *primitive events*. A primitive event consists of an activity, a timestamp, and a set of related objects, which describes *what* happened, *when* it happened, and *which* is the set of (business) objects involved in the action. For instance, a blueprint can map an entry (i.e., a table entry describing that on 1989-02-20 in a table called "BIRTHS", and the attributes of such entry are BABYNAME corresponding to Alex, and NURSE corresponding to Lucia) to an event (i.e., an event that a baby is born on such date and with two related objects (baby Alex and nurse Lucia)). However, a database entry can also be associated with zero, or more than one, events. Since a database entry is always related to a single object, this means that the blueprint could associate several events for each object.

Definition 5 (Blueprint) Given a database DB as in Def. 3, a blueprint is a function bluep : $T \rightarrow \mathcal{P}(\mathbb{U}_{act} \times \mathbb{U}_{timest} \times \mathcal{P}(O))$, that associates to the tables a set of primitive events having an activity, a timestamp and a set of related objects.

The information obtained by applying the blueprint can be used to create an object-centric event log, which can be analyzed with object-centric process mining tools. This will be introduced in Subsection 3.3.

In the following two subsections, we introduce two different types of blueprints, with the first not requiring any domain knowledge from the user, and the second requiring domain knowledge.

3.2.1 Basic blueprint

In this subsection, we introduce a *basic blueprint* that automatically associates a primitive event to the table entries. This is done by exploiting the timestamp of insertion of the entry (that is defined on the given abstraction), the object directly related to the entry, and the other objects related to the given object in the database. Moreover, an activity is associated with the table entry depending on the type of table. In Def. 6, we introduce a characterization of some types of tables.

Definition 6 (Types of Tables) Given a database DB as in Def. 3, we identify different types of tables:

- Change Tables (CT ⊆ T): tables recording changes happening in other tables. If t ∈ CT, then $t \notin \{\pi_2(y) \mid y \in \mathrm{te}(t)\}$.
- Transaction Tables (TT ⊆ T): non-change tables recording different transactions executed on their entries. Moreover, every entry is associated with a transaction. If $t \in TT$, then

$$|\{\mathrm{transact}(y) \mid y \in \mathrm{te}(t) \cap \mathrm{dom}(\mathrm{transact})\}| \ \geq 1 \ \wedge \\ \forall_{y \in \mathrm{te}(t)} y \in \mathrm{dom}(\mathrm{transact})$$

- Object tables (OT ⊆ T): tables not falling in any of the previous categories.

Concerning SAP ERP, we could mention a table for every type: *CDHDR* is a generic change table; *RBKP* contains the transactions executed to verify the invoices (hence it is a transaction table), *VBAK* contains information on different sales order documents without reporting the transaction (hence it is an object table). In Def. 7, the basic blueprint is introduced. We see that entries of a change table are associated with the label of the table on which the change is applied; entries of a transaction table are associated with the executed transaction; entries of a creation table are associated with the label of the table. Variations of the basic blueprint are possible, for example considering the fields that were inserted/deleted/updated during the change.

Definition 7 (Basic Blueprint) Given a database DB as in Def. 3, and a labeling function label : $T \rightarrow \mathbb{U}_\Sigma$ (which can be the identity function), the basic blueprint is defined as $B_{bas} : T \rightarrow \mathcal{P}(\mathbb{U}_{act} \times \mathbb{U}_{timest} \times \mathcal{P}(O))$ such that for $t \in T$, $B_{bas}(t) = \{(a(y), ts(y), robj(y)) \mid y \in \mathrm{te}(t)\}$ where:

- $ts(y) = \pi_3(y)$
- $robj(y) = \{o \in O \mid o = \pi_1(y) \ \vee \ \exists_{(t_1,t_2) \in T \times T} (\pi_1(y), o) \in \mathrm{tr}(t_1, t_2) \vee (o, \pi_1(y)) \in \mathrm{tr}(t_1, t_2)\}$
- $$a(y) = \begin{cases} \text{“Changed”} \ \oplus \ \mathrm{label}(\pi_2(y)) & \text{if } t \in CT. \\ \text{“Executed”} \ \oplus \ \mathrm{transact}(y) & \text{if } t \in TT. \\ \text{“Created”} \ \oplus \ \mathrm{label}(t) & \text{if } t \in OT. \end{cases}$$

In the example presented previously for Def. 3, the basic blueprint would return the following primitive events:

- For t_1:
  - $a(\mathrm{ent}_{11})$ = “Created t1”, $ts(\mathrm{ent}_{11}) = 2007 - 04 - 05$, $robj(\mathrm{ent}_{11}) = \{o_{11}, o_{21}\}$
  - $a(\mathrm{ent}_{12})$= “Created t1”, $ts(\mathrm{ent}_{12}) = 2007 - 04 - 07$, $robj(\mathrm{ent}_{12}) = \{o_{12}, o_{22}\}$
- For t_2:
  - $a(\mathrm{ent}_{21})$ = “Created t2”, $ts(\mathrm{ent}_{21}) = 2007 - 04 - 10$, $robj(\mathrm{ent}_{21}) = \{o_{11}, o_{21}\}$

  - $a(\text{ent}_{22})$ = "Created t2", $ts(\text{ent}_{22}) = 2007 - 04 - 11$, $robj(\text{ent}_{22}) = \{o_{12}, o_{22}\}$

- For changetab:

  - $a(\text{ent}_{31})$ = "Changed t1", $ts(\text{ent}_{21}) = 2007 - 04 - 15$, $robj(\text{ent}_{21}) = \{o_{11}, o_{21}\}$
  - $a(\text{ent}_{32})$ = "Changed t2", $ts(\text{ent}_{22}) = 2007 - 04 - 20$, $robj(\text{ent}_{22}) = \{o_{11}, o_{21}\}$

We note that the set of related objects for ent_{31} and ent_{32} obtained using the basic blueprint is suboptimal. Indeed, only o_{11} should be associated with the primitive event obtained from ent_{31}, and only o_{12} should be associated with the primitive event obtained from ent_{32}.

3.2.2 Blueprint with domain knowledge

The basic blueprint introduced in the previous subsection works on the database abstraction without any further domain knowledge. However, there are situations in which the basic blueprint cannot capture the correlation between entries reported in different tables. A very simple example is the situation in which an order is placed along with three items. This would create one entry in the "orders" table and three entries in the "items" table. These four different entries would be captured as four different primitive events using the basic blueprint. If we are able to provide domain knowledge in the process, i.e., explicitly correlating the four entries of the "orders" and "items" table, we would be able to capture the correct primitive event.

We introduce the concept of domain knowledge in Def. 8. In this case, a view is used to group the related entries of the database. Primitive events can be obtained from such groups by considering as activity the concatenation of the names of the tables of the related entries, as timestamp an aggregation of the timestamps of the related entries[2], and as a set of related objects the union between the objects involved in the related entries.

Definition 8 (Blueprint with Domain Knowledge) Given a database DB as in Def. 3, TE as the set of table entries in DB, and a labeling function label : $T \rightarrow \mathbb{U}_{\Sigma}$ (which can be the identity function), let $VIEW(DB) \subseteq \mathcal{P}(TE)$ be a collection of related entries specified by the domain knowledge. Let $t \in T$ be a table (which is arbitrarily used to represent the view). The blueprint $B_{domk} : T \rightarrow \mathcal{P}(\mathbb{U}_{act} \times \mathbb{U}_{timest} \times \mathcal{P}(O))$ is defined such that:

- $B_{domk}(t') = \emptyset$ if $t' \neq t$
- $B_{domk}(t) = \{(a(v), ts(v), robj(v)) \mid v \in \text{VIEW}\}$ where:
  - $a(v) =$ "Updated" $\bigoplus_{y \in v} \text{label}(\pi_2(y))$ is the concatenation of the names of the tables of the related entries.
  - $ts(v) = \min_{y \in v} \pi_3(y)$ is the minimum of the timestamps of the related entries.
  - $robj(v) = \{\pi_1(y) \mid y \in v\}$ is the union of the object identifiers of the related entries.

In the example presented previously for Def. 3, we could provide the following domain knowledge of the relationships between the entries:

$$\text{VIEW}(DB) = \{v1 = \{\text{ent}_{11}, \text{ent}_{21}\}, v2 = \{\text{ent}_{12}, \text{ent}_{22}\},\\ v3 = \{\text{ent}_{31}\}, v4 = \{\text{ent}_{32}\}\}$$

In this situation, using the blueprint with domain knowledge, we obtain the following primitive events:

[2] An example could be the insertion of a purchase order in SAP. The order document is first created and then filled with the items. This process could take some time for the operator, which could be 10 minutes. In this case, the creation of the order starts at 09:00:00 and is completed at 09:10:00. Since we need to choose a timestamp for the aggregated entry, we can choose 09:00:00 or 09:10:00.

- $a(v1) =$ “Updated t1 t2”, $ts(v1) = 2007-04-05$, $robj(v1) = \{o_{11}, o_{21}\}$
- $a(v2) =$ “Updated t1 t2”, $ts(v2) = 2007-04-07$, $robj(v2) = \{o_{12}, o_{22}\}$
- $a(v3) =$ “Updated t1”, $ts(v3) = 2007-04-15$, $robj(v3) = \{o_{11}\}$
- $a(v4) =$ “Updated t2”, $ts(v4) = 2007-04-20$, $robj(v4) = \{o_{12}\}$

3.3 Step 3: Extracting object-centric event logs

In the previous subsection, the concept of blueprints has been introduced, which allows the extraction of the information (activity, timestamp, set of related objects) about the events recorded by the information system supported by the database. Here, we build an object-centric event log on top of the basilar information extracted by the blueprint, associating additional attributes at the object level. This is the final step of our approach, which translates the contents of the database into the object-centric event log. Object-centric event logs can be analyzed with any application supporting object-centric process mining (for example, OCPM https://www.ocpm.info).

The conversion to an object-centric event log is described in Def. 9. We see that for every object, we pick as attribute map the attribute map of the related entry with the greatest timestamp. This is done to pick the latest version of the object.

Definition 9 (Conversion to Object-Centric Event Log) Given a database DB as in Def. 3, a blueprint $B : T \rightarrow \mathcal{P}(\mathbb{U}_{act} \times \mathbb{U}_{timest} \times \mathcal{P}(O))$, the object-centric event log $L = (E, AN, AV, OT, O, \pi_{act}, \pi_{time}, \pi_{vmap}, \pi_{omap}, \pi_{otyp}, \pi_{ovmap}, \leq)$ is defined in which:

- $AN = A$.
- $AV = \{\text{im}(\pi_4(y)) \mid \exists_{t \in T}\ y \in \text{te}(t)\}$.
- $\pi_{otyp} = \text{otyp}$.
- For any $o \in O$, $\pi_{ovmap}(o) = \pi_4(\text{argmax}_{\pi_3}\{y \mid \exists_{t \in T}\ y \in \text{te}(t)\ \wedge\ \pi_1(y) = o\})$
- For $t \in T$ and $(a, ts, robj) \in \text{B}(t)$, we define an event e with the following properties:
  - $\pi_{act}(e) = a$
  - $\pi_{time}(e) = ts$
  - $\pi_{omap}(e) = robj$

We may be interested in applying Def. 9 only to a subset of tables of the database. This is because, generally, we are not interested in generating an object-centric event log from the entire database, but focusing on the tables related to a process (e.g., in SAP ERP we may be interested in the P2P, O2C, and inventory management tables). For this, we introduce a projection function at the database level in Def 10.

Definition 10 (Projecting Database) Given a database DB as in Def. 3 and a subset of tables $T' \subseteq T$, we define the projected database $DB' = (T', O, OT, A, D, TR, \text{te}', \text{tr}_{|T' \times T'}, \text{otyp}, \text{attdomain}, \text{transact}', \text{primary}_{|T'}, \text{foreign}_{|T'})$ for which given any $t, t_1, t_2 \in T$:

$$\text{te}'(t) = \begin{cases} \{y \in \text{te}(t) \mid \pi_2(y) \in T'\} & \text{if } t \in T' \\ \emptyset & \text{otherwise} \end{cases}$$

$$\text{transact}' = \text{transact}_{|\bigcup_{t \in T'} \text{te}'(t)}$$

4 Implementation on SAP ERP

In this section, we aim to extract an object-centric event log related to a process supported by the SAP ERP system. This is done by building the database abstraction and then using the information contained in the corresponding GoR to select the tables related to a given process (O2C, P2P), perform preprocessing by limiting the allowed values for some attributes, and eventually extract the object-centric event log.

4.1 Constructing the database abstraction

Here, we build a GoR on top of an instance of SAP ERP. This is done automatically (without user interaction) in three different steps:

- Extraction of the set of nodes.
- Extraction of the set of arcs.
- Association of a set of table entries.
- Definition of the relationships between the different object identifiers.

Extraction of the set of nodes: the different categories of nodes, and some queries that can be used to extract them, are presented in Table 2. In particular, five different categories of nodes are identified: *tables* (such as EKKO, RBKP, BKPF), *transactions* (such as MIRO, MR1M, VA21N), *attributes* (such as BELNR, which is the invoice number, and GJAHR, which is the fiscal year), *domains* (such as a timestamp (DATUM), organizational resource (USNAM)), and *object types* (for example, EINKBELEG corresponds to the purchase order documents and VERKBELEG corresponds to the sales order document).

Extraction of the set of arcs: The different categories of arcs, and some queries that can be used to extract them, are presented in Table 3. In particular, seven different categories of arcs are identified: *attribute arcs* (connecting the tables to some attributes that aren't primary or foreign keys for the given table), *primary key arcs* (connecting the tables to some attributes that are primary keys, but not foreign keys, for the given table), *foreign key arcs* (connecting the tables to some attributes that are foreign keys), *domain arcs* (connecting the attributes to the corresponding domain(s)), *object type arcs* (connecting the tables to the corresponding object type(s)), *transaction arcs* (connecting the tables to the corresponding transaction(s)), *relationship arcs* (connecting a table to another related table, i.e., two tables having a couple of related objects).

Table entries association: here, we associate each table with its entries. A strategy for this is to consider each row of each table in SAP as a different entry. In this case, the attributes of the table entry and the transactions executed against it depend only on the attributes of each row.

Table 2 SQL queries to extract different categories of nodes of the GoR built on top of SAP ERP

Node Type	SQL query
tables	SELECT DISTINCT TABNAME FROM DD02L
transactions	SELECT DISTINCT TCODE FROM TSTCT
attributes	SELECT DISTINCT FIELDNAME FROM DD03L
domains	SELECT DISTINCT DOMNAME FROM DD03L
object types	SELECT DISTINCT OBJECT FROM TCDOB

Table 3 SQL queries to extract different categories of arcs of the GoR built on top of SAP ERP

Category	SQL query
attribute arcs	SELECT TABNAME,FIELDNAME FROM DD03L WHERE KEYFLAG!='X' AND CHECKTABLE=' '
primary key arcs	SELECT TABNAME,FIELDNAME FROM DD03L WHERE KEYFLAG='X' AND CHECKTABLE=' '
foreign key arcs	SELECT TABNAME,FIELDNAME FROM DD03L WHERE CHECKTABLE!=' '
domain arcs	SELECT DISTINCT FIELDNAME,DOMNAME FROM DD03L
object type arcs	SELECT DISTINCT TABNAME,OBJECT FROM TCDOB
transaction arcs	SELECT DISTINCT TABNAME,TCODE FROM CDHDR a JOIN CDPOS b ON a.MANDANT = b.MANDANT AND a.CHANGENR = b.CHANGENR
relationship arcs	SELECT DISTINCT TABNAME,CHECKTABLE FROM DD03L

Relationships between different object identifiers: some tables in the graph can define relationships between object identifiers. This is done by connecting all the object identifiers referenced by the foreign keys of the rows of such a table.

4.2 Process identification and selection

The database abstraction built in the previous section is the starting point for the extraction of an object-centric event log from an SAP ERP instance. SAP ERP contains different processes (for example, Order-to-Cash and Procure-to-Pay) involving different tables. Therefore, a subset of tables needs to be selected to extract an object-centric event log. A strategy to choose the subset of tables is:

- Extracting the GoR from the database.
- Starting from a representative table for the process (for example, EKKO for Procure-to-Pay) or to the tables belonging to a given object type (for example, EINKBELEG is associated with different tables of the Procure-to-Pay process, including EKKO, EKET, EKPA).
- Expanding the set of tables based on the relationships in the GoR.

The expansion step visits the relationships arcs having as the target node one of the tables included in the initial set. Then, every source table of such arcs is included in the expanded set of tables. This expansion step can be repeated several times, increasing the size of the set of tables progressively.

Here, the user needs to choose the representative tables and evaluate which tables of the proposed expansion could be interesting for the extraction.

4.3 Preprocessing the table entries

The preprocessing step helps to reduce the number of entries associated with the tables of the provided set by checking the values allowed by some attributes. For example, in a Procure-to-Pay process, we might be interested in considering only the orders having a given material (in this case, the attribute is MATNR).

This step is fully manual (e.g., the user selects the values admitted for the attributes).

4.4 Extracting an object-centric event log

The object-centric event log can be extracted as described in Def. 9 with the provision (from the user) of a blueprint.

4.5 Tool support - extraction from SAP

After presenting the implementation of the database abstraction on top of SAP ERP, we present the tool *Interactive SAP Explorer*. The tool encodes the database abstraction of SAP in a labeled property graph inserted inside a graph database. Then, a web interface is provided that permits the exploration of the relational structure of the SAP instance, the identification of the most important processes, and the creation of a list of tables for extraction. The list of tables is eventually provided to another component of the tool which creates an object-centric event log from such a list of tables.

The process identification and selection (see Fig. 2) is implemented as follows:

- The elements of the relational structure of SAP that are important for the definition of a set of classes related to a given process are imported inside a graph database (Neo4J).
  - A graph database permits a faster exploration of the neighboring entities to a given concept because the connections are referenced inside the node object.
  - The chosen graph database (Neo4J) provides efficient implementations of layout algorithms, which can be executed on a significant amount of nodes/edges to provide an understandable graphical representation of the relational structure in SAP.
- Then, the identification process can be started. The first step is to identify an object type of interest (for example, the *purchase orders* and *sales orders*). This is directly connected, in the relational structure of SAP, to a set of tables (*purchase orders* are connected to the tables *EKKO*, *EKPO*, *EKPA*, *EKET*, *EKKN*).
- The next step is expanding the aforementioned set of tables. Starting from the initial set of tables, we identify the tables connected to the initial tables via the relational structure. The union of these tables contains the set of events regarding a process in SAP. For

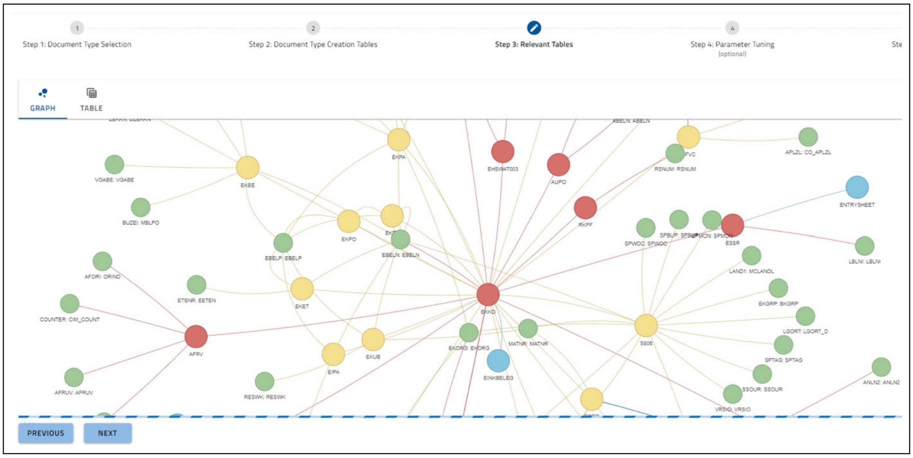

Fig. 2 GoR visualized using the tool support *Interactive SAP Explorer*

example, by expanding the tables related to the *purchase orders* object type, we get a set of tables including purchase requisitions (*EBAN*), goods/invoice receipts (*EKBE*), accounting documents (*BKPF*), and other tables containing the events of the P2P process in SAP.

The process extraction component, which uses the approach described in Berti et al. (2021), aims to extract an object-centric event log out of the SAP system based on the relevant tables identified in the previous step. There is no need to specify any SQL query.

- A pre-processing step is performed to restrict the extraction to the desired configuration.
- The extraction of the object-centric event log is performed, with an output following the OCEL specification http://www.ocel-standard.org/.

The source codes of the different components of the tool are available in the following repositories:

- *Layer of web services that can be run on IIS*: this component can be downloaded at https://github.com/Javert899/interactive-extractor-from-sap-main/tree/main/Backend-C%23/SAPExtractorAPI.
- *Angular web application*: this component can be downloaded at https://github.com/Javert899/interactive-extractor-from-sap-main/tree/main/Frontend/InteractiveSAPExtractor.
- *Python web services for the extraction of the object-centric event log*: this component can be downloaded at https://github.com/Javert899/sap-extractor.

Note that there is a dependency on non-open source UI components which need to be licensed to a single user. Therefore, the application is not directly runnable from the aforementioned source repositories. Also, the extractor requires the availability of an SAP ECC instance supported by the Oracle database and the installation of the Neo4J graph database, which is released under a proprietary license.

The existing version of the tool can connect only to an SAP ECC instance supported by the Oracle database. Despite this being a popular option, this limits the possibility to apply the extractor in a generic setting. The extractor needs different components to run. This is architecturally complicated and, therefore, highly dependent on the functioning of existing queries/connectors on different versions of the software.

Our extractor overcomes the following limitations of existing SAP extractors; they are process-specific, they rely on traditional event logs, and suffer from convergence/divergence issues. However, there are remaining limitations, including the fairly basic definition of activity/timestamp concepts. The choice of the graph database to navigate the relational structure of SAP is advantageous in terms of performance. After the selection of a set of tables, the extraction of an object-centric event log is left to the Python component, which executes many SQL queries to load the information needed in memory. Therefore, the extractor is limited by the amount of memory of the client.

4.6 Assessment - SAP ERP

In this section, we assess the proposed method's utility and scalability in encoding the relational structure and extracting object-centric event logs from SAP ERP.

Table 4 Extraction of the different nodes of the GoR

Category	Number of Nodes	Query Time(s)
tables	662161	0.43 s
transactions	103060	0.80 s
attributes	934546	4.05 s
domains	115484	2.29 s
object types	1885	0.15 s

For every category, we report the number of nodes and the time in seconds needed for the extraction

4.6.1 Scalability

In this subsection, we assess the scalability of the proposed approach. We saw in Section 4 that the process identification and selection, the pre-processing, and also the extraction of the object-centric event logs are quite straightforward when the database abstraction is built. Here, some measurements are done to assess the time needed to build the database abstraction, which is the main source of computational complexity.

For our experiments, we used an educational instance of SAP ERP with demonstration data. While the number of different documents contained in this instance is limited, the relational structure is quite complete. Therefore, the number of nodes/edges is quite representative. In the experimental results, we do not report the time of insertion in the graph (database) but focus on the times needed to extract the information from the relational database supporting SAP ERP, and on the pre-processing time needed to identify the connections between the concepts before the insertion in the graph database.

Tables 4 and 5 assess the retrieval of the nodes/edges of the GoR. The extraction of the table-to-table relationships needs postprocessing since it is extracted in our implementation by associating the tables sharing a foreign key with a given table, and all the pairwise relationships between the tables are considered. For these queries and postprocessing operations, the execution time is very good, and, therefore, the database abstraction is identified in a reasonable time (24 seconds in our experiment).

Table 6 considers some tables that are important for the Procure-to-Pay process (purchase requisitions, purchase orders, invoices, and payments) and the time needed to extract the entries from such tables. The number of entries, and the execution time, are influenced by the educational nature of our instance. Table 7 shows how much time is needed to extract

Table 5 Extraction of the arcs of the GoR

Arc Type	Number of Arcs	Query Time(s)	Postprocessing Time(s)
attribute arcs ($R_{T,A}$)	9628155	6.71 s	0.0 s
domain arcs ($R_{A,D}$)	1311489	9.64 s	0.0 s
object type arcs ($R_{T,OT}$)	4803	0.12 s	0.0 s
relationship arcs ($R_{T,T}$)	1671478	6.83 s	16.28 s
primary key arcs ($R_{T,PR}$)	412183	6.71 s	0.0 s
foreign key arcs ($R_{T,FR}$)	1931064	1.73 s	0.0 s

For every category, we report the number of arcs, the time in seconds needed for the extraction, and the postprocessing time

Table 6 Extraction of the table entries for some tables in our educational instance in SAP ERP

Table	Description	Num. Entries	Query Time(s)
EBAN	Purchase Requisitions (Master)	4281	0.18 s
EBKN	Purchase Requisitions (Detail)	1042	0.15 s
EKKO	Purchase Orders (Master)	16214	0.20 s
EKPO	Purchase Orders (Detail)	30985	0.25 s
RBKP	Invoices (Master)	5720	0.18 s
RSEG	Invoices (Detail)	14802	0.22 s
BKPF	Payments (Master)	664021	0.41 s
BSEG	Payments (Detail)	1844532	1.25 s

For every table, we report the number of extracted entries and the time needed for the query

the relationships between object identifiers given the table containing such relationships. For example, EKPO relates purchase requisitions and purchase orders, and RSEG relates invoices and purchase orders. The majority of the time is spent finding the relationships between the object identifiers; therefore, this is the biggest bottleneck in the overall process.

4.6.2 Quality of the database extraction

The initial step of the extraction is the process identification and selection step. In this step, starting from an object type, a set of interconnected tables is identified, which is eventually used to extract the object-centric event log. Here, we assess the set of interconnected tables found starting from the purchasing order type (EINKBELEG). This object type is related to the Procure-to-Pay process. Hence, we expect that in this step also the tables related to the purchase requisitions, the invoices, and the payments are found. The subgraph of the GoR represented in Fig. 3, shows that (as expected) the purchasing order type is interconnected to tables of other object types, including invoices, payments, and purchase requisitions.

As we run the tool on our educational SAP ERP instance, the process identification step identifies the object types described in Table 8. Proceeding with the subsequent table selection and extraction phases, we are able to connect every object type to a set of tables, and extract events from such tables.

The main issue is the naming of the activities if the basic blueprint is applied. Purchase orders are associated indistinctly with the same activity; invoices are associated with the

Table 7 Extraction of the relationships between object identifiers defined in some tables of our educational instance in SAP ERP

Table	Description	Num. Relationships	Query Time(s)	Postprocessing Time(s)
EBKN	Purchase Requisitions (Detail)	8964	0.15 s	0.06 s
EKPO	Purchase Orders (Detail)	748750	0.25 s	3.72 s
RSEG	Invoices (Detail)	157086	0.22 s	0.58 s
BSEG	Payments (Detail)	2323792	1.25 s	137.79 s

For every table, we report the number of relationships and the query/postprocessing times

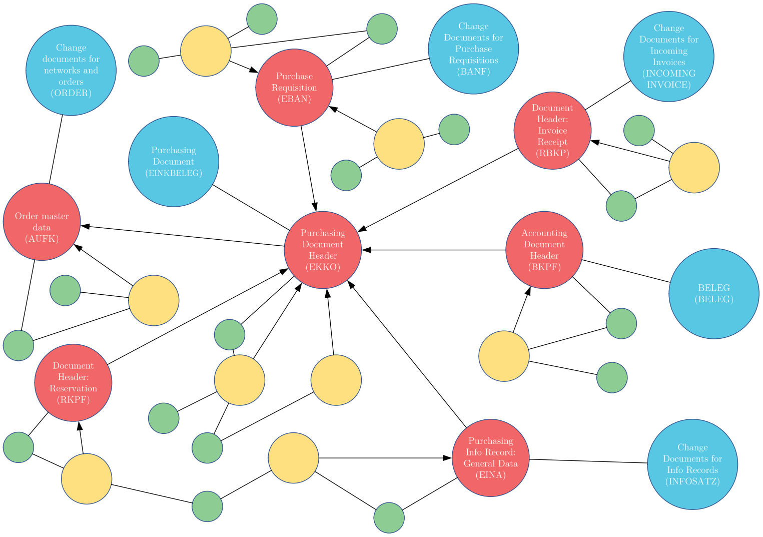
Change documents for networks and orders (ORDER)
Purchase Requisition (EBAN)
Change Documents for Purchase Requisitions (BANF)
Change Documents for Incoming Invoices (INCOMING INVOICE)
Document Header: Invoice Receipt (RBKP)
Purchasing Document (EINKBELEG)
Order master data (AUFK)
Purchasing Document Header (EKKO)
Accounting Document Header (BKPF)
BELEG (BELEG)
Document Header: Reservation (RKPF)
Purchasing Info Record: General Data (EINA)
Change Documents for Info Records (INFOSATZ)

Fig. 3 Subgraph showing the tables related to the EINKBELEG object type (purchase orders)

transaction executed on the corresponding document; changes are associated with the set of fields that are changed without comparing their values. Therefore, the naming of the activities is of minor quality in comparison to a more ad-hoc extraction.

4.6.3 Extracted logs - O2C and P2P

In this section, we show some logs extracted on an educational instance of SAP ERP related to the O2C process (*VERKBELEG* object type) and the P2P process (*EINKBELEG* object type). The two object-centric event logs are in the JSON-OCEL implementation and can be retrieved at the addresses:

- https://www.ocpm.info/o2c.jsonocel for the O2C process.
- https://www.ocpm.info/p2p.jsonocel for the P2P process.

We can visualize the two event logs inside the OC-PM tool https://www.ocpm.info/. In particular, we provide the following links:

- https://www.ocpm.info/ocel.html?ocel=o2c.jsonocel for the O2C process.
- https://www.ocpm.info/ocel.html?ocel=p2p.jsonocel for the P2P process.

We see in Fig. 4 the document flow between different sales order documents[3], and in Fig. 5 different stages of the purchasing process, including the creation of a purchase requisition (taken from the table *EBAN*), the subsequent creation of a purchase order (taken from the table *EKKO*) and invoice (taken from the table *RBKP*). A change activity (*Change KEY*) has been recorded for the orders.

In general, the naming of the extracted activities is less understandable than the naming conventions used in commercial SAP extractors. However, this comes for free with the basic blueprint.

[3] In this case, C, Q, R, J, U, M, 3 are identifiers of document types, which have been reported as raw identifiers by the current version of the extractor.

Table 8 Processes identified on the educational SAP ERP instance

Object type	Number of events	Number of tables	Description of the process
VERKBELEG	412228	11	Management of Sales Document (O2C)
IFLO	133263	3	PM: Functional Location
EINKBELEG	95490	6	Management of Purchasing Document (P2P)
VTBFHA	77200	9	Treasury: Transaction
EQUI	66827	4	Equipment change document objects
STUE	65430	10	Object list change documents
BUPA_BUP	45810	6	Business partner: Use of BUP
PROJ	26067	5	Project structure plan (PSP)
FTR_TCORT_CO	6586	8	Treasury: Correspondence
IMAK	4080	5	Appropriation requests with variant
EHFND_REGLIST	1447	4	Regulatory List Revision
FARR_CONTRACT	521	4	Financial Accounting Revenue Recognition

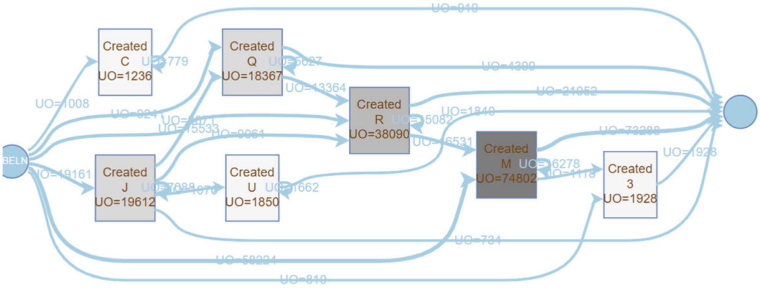

Fig. 4 Object-centric directly follows graph discovered on top of the log extracted for the O2C process

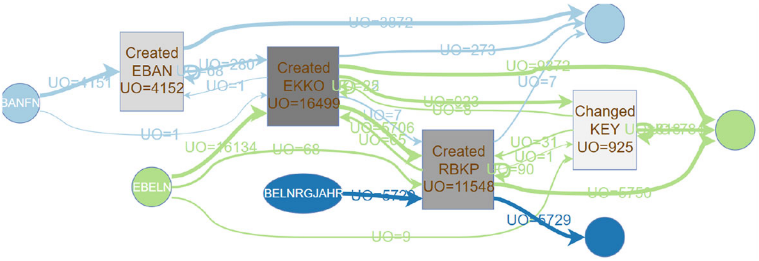

Fig. 5 Object-centric directly follows graph discovered on top of the log extracted for the P2P process

5 Related work

In this section, we present some related work on the extraction and analysis of event data from SAP ERP. This has been done using traditional event logs (with convergence/diverge issues) or artifact-centric/object-centric paradigms.

Extraction of Traditional Event Logs from SAP ERP: SAP is an interesting system for process mining since its widespread usage by companies and the unstructuredness of the supported processes. Hence, several process mining publications targeted the extraction of data from SAP ERP. In Ingvaldsen and Gulla (2007), a method is proposed for the extraction and transformation of event logs from SAP ERP, which involves the manual specification of a meta-model defining how events, resources, and their relationships are stored. The method has been applied to SAP systems provided by Norwegian Agricultural and Marketing Cooperative and Nidar. Some limitations exist: although all the information (transactional, master, and ontological data) needed to extract meaningful process models is available, the transactions are not mapped directly to the tasks. Moreover, it was not possible to map the extracted transaction flow to the processes in the SAP reference model. In de Murillas et al. (2019), a meta-model that can ingest the contents of a relational database and provide the possibility to easily specify queries to produce an event log is described. The meta-model can be used on a database supporting SAP ERP. However, this leads to the generation of traditional event logs having convergence/divergence issues (van der Aalst, 2019) .

Extraction of Object-Centric Event Data from SAP ERP: Some approaches have been proposed to avoid the drawbacks of using traditional event logs. In Lu et al. (2015), the construction of artifact-centric models on top of SAP ERP is proposed, along with an implementation in the popular ProM 6.x framework. An artifact-centric model considers both the lifecycle of an artifact (purchase order document, invoice document, payment document) and the interaction between different artifacts. Some limitations exist: the approach requires some non-trivial manual steps, and the discovery phase is limited to two artifacts. In Berti et al. (2021), a method to extract object-centric event logs starting from SAP ERP is proposed, which is the foundation of the current paper. The proposed prototypal software is limited by an in-memory approach and customization options. Moreover, some fundamental details (the construction of the GoR and the extraction of the relationships between the table entries) have been omitted for space reasons.

Process Mining Case Studies on top of SAP ERP: SAP ERP stores interesting but company-critical data. Therefore, few case studies in applying process mining on top of SAP ERP data have been proposed. In Fleig et al. (2018b), an application of process mining to the Order-to-Cash and Procure-to-Pay processes of a manufacturing company is proposed. An application of process mining to the Procure-to-Pay and Accounts Payable processes is proposed in Stolfa et al. (2013); Stephan et al. (2021). The warehouse management process is considered in ER et al. (2015). Also, Fleig et al. (2018a) discusses the implementation of a decision support system, supported by process mining, for the standardization of ERP systems.

Graph-Based Analyses of SAP ERP: The graph-based nature of event data is exploited in Esser and Fahland (2021). Traditional (and object-centric) event logs can be encoded in a graph database. This allows for queries that are unfeasible on top of relational databases since edges are first-class entities in graph databases. An application to ERP systems (BPI Challenge 2019 log) is proposed. The contribution in Fahland (2022) further exploits the graph- and object-based nature of event data to build event knowledge graphs. This data structure allows us to naturally model behavior over multiple entities as a network of events.

6 Conclusion

In this paper, we introduced an approach for the extraction of an object-centric event log from a relational database. This overcomes three challenges in the generation of such an event log: i) the identification of the processes contained in the database; ii) the identification of the tables of interest for such processes; iii) the generation of the events (activity, timestamp, set of related objects). An implementation of the approach has been done on top of an educational instance of SAP ERP. In particular, a tool is offered to perform process identification, selection, and extract an object-centric event log from SAP ERP. While commercial extractors have advanced activity concepts and greater scalability in comparison to the provided implementation, they support only mainstream processes and need customization if the given process is executed slightly differently, in contrast to our technique that can support automatically many different processes. Moreover, having object-centric event logs as outputs helps to avoid deficiency/convergence/divergence issues and have a more natural expression of the event data. The assessment shows that the techniques proposed in the paper can be used in reasonable time on an educational instance of SAP ERP. The challenge of the extraction lies in the identification of a good set of tables covering a target process. Starting from an initial set of tables connected to an object type, subsequent user interaction, and exploration of the graph of relationships are needed to expand such a set. For example, in a procure-to-pay process, the user initially selects the purchasing document (EINKBELEG) type, and then he gets the possibility to select other object types (the invoice). Also, the definition of the translation of the table entries to events (blueprint) influences the quality of the resulting event log. The paper proposes a basic blueprint that can be applied to any type of table (a change table, a transaction table, or a record table) but does not ensure an optimal naming for the activities, and a blueprint based on domain knowledge. Overall, finding criteria to define blueprints of good quality is an open research topic. The domain knowledge of the user is therefore required in choosing the tables of interest for a given process (starting from the information represented in the GoR), preprocessing the information, and defining the blueprint[4]. While the implementation and evaluation are mainly focused on the SAP ERP system, the method can be applied in principle to relational databases (provided that the abstraction in Def. 4 can be computed). A limitation of our approach is the lack of a direct translation from a database concept to the given database abstraction. We only showcase the translation of an SAP ERP schema (implemented on the Oracle relational database). While we expect that the translation can be implemented on many other databases/schemas, the translation of the current variety of database concepts (temporal, bi-temporal) could not be included in the scope of the current paper. Moreover, the method does not assure to choose a complete set of tables for a given process, or to avoid uninteresting tables, and the responsibility is left to the user.

Acknowledgements We thank the Alexander von Humboldt (AvH) Stiftung.

Author Contributions Alessandro Berti worked on the tool support, the experimental setting, and the first version of the text of the paper, while Gyunam Park, Majid Rafiei, and Wil van der Aalst contributed to reviewing the paper and improving its quality.

Funding Alexander von Humboldt (AvH) Stiftung. Open Access funding enabled and organized by Projekt DEAL.

Availability of supporting data Not Applicable.

[4] A difference from Berti et al. (2021) is that there only an automatic blueprint was defined.

Declarations

Ethical Approval Not Applicable.

Competing Interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Berti, A., & van der Aalst, W. M. P. (2022). OC-PM: analyzing object-centric event logs and process models. CoRR. https://doi.org/10.48550/arXiv.2209.09725. arXiv:2209.09725

Berti, A., Park, G., & Rafiei, M., et al. (2021). An event data extraction approach from SAP ERP for process mining. In: J. Munoz-Gama, & X. Lu (Eds.), *Process Mining Workshops - ICPM 2021 International Workshops, Eindhoven, The Netherlands, October 31 - November 4, 2021, Revised Selected Papers, Lecture Notes in Business Information Processing*, (vol 433, pp. 255-267). Springer, New York City. https://doi.org/10.1007/978-3-030-98581-3_19

de Murillas, E. G. L., Reijers, H. A., & van der Aalst, W. M. P. (2019). Connecting databases with process mining: a meta model and toolset. *Software & Systems Modeling 18*(2), 1209–1247. https://doi.org/10.1007/s10270-018-0664-7

de Murillas, E. G. L., van der Aalst, W. M. P., & Reijers, H. A. (2015). Process mining on databases: Unearthing historical data from redo logs. In: H. R. Motahari-Nezhad, J. Recker, & M. Weidlich (Eds.), *Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31- September 3, 2015, Proceedings, Lecture Notes in Computer Science*, (vol 9253, pp. 367–385). Springer, New York City. https://doi.org/10.1007/978-3-319-23063-4_25

Dibam, K., Batoulis, K., & Weidlich, M., et al. (2020). Extraction, correlation, and abstraction of event data for process mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 10*(3). https://doi.org/10.1002/widm.1346

ER, M., Astuti, H. M., & Wardhani, I. R. K. (2015). Material movement analysis for warehouse business process improvement with process mining: A case study. In: J. Bae, S. Suriadi, & L. Wen (Eds.), *Asia Pacific Business Process Management- Third Asia Pacific Conference, AP-BPM 2015, Busan, South Korea, June 24-26, 2015, Proceedings, Lecture Notes in Business Information Processing*, vol 219, pp. 115-127. Springer, New York City. https://doi.org/10.1007/978-3-319-19509-4_9

Esser, S., & Fahland, D. (2021). Multi-dimensional event data in graph databases. *Journal on Data Semantics 10*(1-2), 109–141. https://doi.org/10.1007/s13740-021-00122-1

Fahland, D. (2022). Process mining over multiple behavioral dimensions with event knowledge graphs. In: W. M. P. van der Aalst, & J. Carmona (Eds.), *Process Mining Handbook, Lecture Notes in Business Information Processing*, (vol 448, pp. 274-319). Springer, New York City. https://doi.org/10.1007/978-3-031-08848-3_9

Fleig, C., Augenstein, D., & Maedche, A. (2018a). Designing a process miningenabled decision support system for business process standardization in ERP implementation projects. In: M. Weske, M. Montali, I. Weber, et al. (Eds.), *Business Process Management Forum - BPM Forum 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings, Lecture Notes in Business Information Processing*, (vol 329, pp. 228-244). Springer, New York City. https://doi.org/10.1007/978-3-319-98651-7_14

Fleig, C., Augenstein, D., & Maedche, A. (2018b). Process mining for business process standardization in ERP implementation projects - an SAP S/4 HANA case study from manufacturing. In: W.M.P. van der Aalst, F. Casati, R. Conforti, et al (Eds.) *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018 co-located with 16th International Conference on Business Process Management*

(BPM 2018), Sydney, Australia, September 9-14, 2018, CEUR Workshop Proceedings, (vol 2196, pp. 149-155). CEUR-WS.org, Aachen. http://ceur-ws.org/Vol-2196/BPM_2018_paper_31.pdf

Ghahfarokhi, A. F., Park, G., & Berti, A., et al. (2021). OCEL: A standard for objectcentric event logs. In: L. Bellatreche, M. Dumas, P. Karras, et al. (Eds.) *New Trends in Database and Information Systems - ADBIS 2021 Short Papers, Doctoral Consortium and Workshops: DOING, SIMPDA, MADEISD, MegaData, CAoNS, Tartu, Estonia, August 24-26, 2021, Proceedings, Communications in Computer and Information Science*, (vol 1450, pp. 169-175). Springer, New York City. https://doi.org/10.1007/978-3-030-85082-1_16

Ingvaldsen, J. E., & Gulla, J. A. (2007). Preprocessing support for large scale process mining of SAP transactions. In: A. H. M. ter Hofstede, B. Benatallah, & H. Paik (Eds.), *Business Process Management Workshops, BPM 2007 International Workshops, BPI, BPD, CBP, ProHealth, RefMod, semantics4ws, Brisbane, Australia, September 24, 2007, Revised Selected Papers, Lecture Notes in Computer Science*, (vol 4928, pp. 30-41). Springer, New York City. https://doi.org/10.1007/978-3-540-78238-4_5

Lu, X., Nagelkerke, M., & van de Wiel, D., et al. (2015). Discovering interacting artifacts from ERP systems. *IEEE Transactions on Services Computing 8*(6), 861–873. https://doi.org/10.1109/TSC.2015.2474358

Stephan, S., Lahann, J., & Fettke, P. (2021). A case study on the application of process mining in combination with journal entry tests for financial auditing. In: *54th Hawaii International Conference on System Sciences, HICSS 2021, Kauai, Hawaii, USA, January 5, 2021*. (pp. 1-10) ScholarSpace, Denver. https://hdl.handle.net/10125/71314

Stolfa, J., Kopka, M., & Stolfa, S., et al. (2013). An application of process mining to invoice verification process in SAP. In: A. Abraham, P. Krömer, V. Snásel (Eds.), *Innovations in Bio-inspired Computing and Applications- Proceedings of the 4th International Conference on Innovations in Bio-Inspired Computing and Applications, IBICA 2013, August 22 -24, 2013 - Ostrava, Czech Republic, Advances in Intelligent Systems and Computing*, (vol 237, pp. 61-74). Springer, New York City. https://doi.org/10.1007/978-3-319-01781-5_6

van der Aalst, W. M. P. (2019). Object-centric process mining: Dealing with divergence and convergence in event data. In: P. C. Ölveczky ,& G. Salaün (Eds.), *Software Engineering and Formal Methods - 17th International Conference, SEFM 2019, Oslo, Norway, September 18-20, 2019, Proceedings, Lecture Notes in Computer Science*, (vol 11724. pp. 3-25). Springer, New York City. https://doi.org/10.1007/978-3-030-30446-1_1

van der Aalst, W. M. P., & Berti, A. (2020). Discovering object-centric petri nets. *Fundam Informaticae 175*(1-4), 1- 40. https://doi.org/10.3233/FI-2020-1946

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Alessandro Berti[1] · Gyunam Park[1] · Majid Rafiei[1] · Wil M.P. van der Aalst[1]

Gyunam Park
gnpark@pads.rwth-aachen.de

Majid Rafiei
majid.rafiei@pads.rwth-aachen.de

Wil M.P. van der Aalst
wvdaalst@pads.rwth-aachen.de

[1] Process and Data Science Group, RWTH Aachen University, Ahornstrasse 55, Aachen 52074, NRW, Germany