Extracting Event Data from Document-Driven Enterprise Systems

Diego Calvanese[1,2], Mieke Jans[3,4](✉), Tahir Emre Kalayci[5], and Marco Montali[1]

[1] Free University of Bozen-Bolzano, 39100 Bolzano, Italy
{calvanese,montali}@inf.unibz.it
[2] Umeå University, 90187 Umeå, Sweden
diego.calvanese@umu.se
[3] Hasselt University, 3500 Hasselt, Belgium
mieke.jans@uhasselt.be
[4] Maastricht University, 6200, MD Maastricht, Netherlands
mj.jans@maastrichtuniversity.nl
[5] Virtual Vehicle Research GmbH, 8010 Graz, Austria
emre.kalayci@v2c2.at

Abstract. The preparation of input event data is one of the most critical phases in process mining projects. Different frameworks have been developed to offer methodologies and/or supporting toolkits for data preparation. One of these frameworks, called OnProm, relies on sophisticated semantic technologies to extract event logs from relational databases. The toolkit consists of a series of general steps, meant to work on arbitrary, legacy databases. However, in many settings, the input database is not a legacy one but is structured with conceptually understandable object types and relationships that can be effectively employed to support business users in the extraction process. This is, for example, the case for document-driven enterprise systems. In this paper, we focus on this class of systems and propose a guided approach, erprep, to support a group of business and technical users in setting up OnProm with minimal effort. We demonstrate the approach in a real-life use case.

Keywords: Data preparation · event log extraction · ERP systems · Ontology-based event modeling

1 Introduction

Many business processes are supported by an information system, recording large amounts of data about the underlying process. Extracting useful knowledge and insights from this process data is the purpose of the field of process mining. Process mining has brought forward various algorithms and techniques that are used to discover, analyze, and improve business processes. A process mining project requires the availability of event data in a standardized format, the IEEE

XES standard. Although the standard is a good format to represent process logs, different perspectives on the data can be taken, and it is often hard to understand how to identify and extract events from legacy data sources. Hence, different event logs can be extracted from the same system [8]. Once a perspective has been chosen, the extraction is typically handled via coding, e.g., through ETL-scripts for relational databases. This is error-prone, does not directly reflect the taken perspective, and requires to write new code from scratch whenever one wants to change perspective when analyzing the data (e.g., when changing the case notion).

To mitigate these issues, a novel approach, called OnProm, has recently been put forward to facilitate and semi-automate this log extraction process in the case of relational databases [4,5], leveraging the *ontology-based data access* paradigm [21]. Event data extraction in OnProm consists of three phases: *(1)* creating a domain ontology to conceptually capture the relevant concepts and relations of the domain of interest, *(2)* mapping this domain ontology to the underlying database structure, and *(3)* annotating the domain ontology to indicate where to find cases, events, and their attributes. While these three phases are free of procedural code, they may still be very demanding as they intensively rely on human knowledge and expertise, due to the intrinsic complexity of the extraction process, and the complete generality of the approach, which does not make any assumption on how data is structured in the underlying relational database.

The goal of this work is to study these three phases in the setting where the underlying database is a so-called *document-driven enterprise system*, like an ERP-system. In such systems, the database is organized around conceptually understandable document objects and their mutual relationships, with specific modeling structures to store events. This leads to the following research question: *how can one simplify the definition of an* OnProm *pipeline in the case of document-driven enterprise systems?*

We answer this question by introducing the erprep guided approach, which supports domain and IT experts in the incremental, semi-automated definition of a suitable domain ontology and the corresponding mappings, and helps them in easily defining case- and event-annotations depending on the perspective of interest. We show through a case study in the manufacturing domain that, by applying erprep, we can indeed smoothly (in terms of procedure) and feasibly (in terms of performance) extract multiple XES event logs that, together, provide a suitable basis to conduct process mining analyses.

2 Related Work

To guide projects aiming to improve process performance or compliance to rules and regulations, the PM^2 methodology has been developed [7]. This methodology defines the stages that a successful process mining project entails and it describes the accompanied activities to these stages. Stages 1, 2, and 3 (Planning – Extraction – Data processing) represent the labor-intensive preparation of the core project (which starts at Stage 4). Although these stages are presented as clear-cut different stages in the PM^2 methodology, it is difficult to disentangle the activities related to these stages. With much less research conducted on these stages, important research problems remain unaddressed, like how to effectively extract and process your data as event log [8].

The availability of a (high-quality) event log in a standardized format is essential for every process mining project, but most information systems are currently not capable of automatically producing the required kind of event logs. Hence it is paramount to identify novel methods for improving the log extraction process. Some work has been done on the topic of extracting event logs in the context of specific environments (like SAP) or domains (like accounting) [10,16].

In general, process mining requires a clear notion of a case. However, many database systems that provide the input for process mining analyses are document-oriented: they associate a chain of documents to one process execution. Selecting one specific document as case identifier is therefore providing only a single perspective on the process. To avoid the problems of *data divergence* and *convergence* that arise when forcing event data from these systems to adopt a single case notion, artifact-centric process mining has been proposed as a more generic solution [6,14,17]. This approach tries to solve the aforementioned problem by discovering the life-cycles of interrelated data objects (i.e., artifacts) and their interactions. The authors of [13] take another approach and present the eXtensible Object-Centric (XOC) event log format, which does not require a case notion, hereby avoiding the flattening of reality into an event log.

Two key frameworks that tackle the manual, labor-intensive aspect of event data extraction are [5,8]. The authors of [8] introduce a meta-model and toolchain to connect databases with processes through bridging concepts. Data from three source environments (redo logs, SAP-style change tables, and in-table version storage) are transformed into a common representation and missing values are automatically inferred where possible. The authors envision this model to assist in multi-perspective event log building in a more intuitive way and provide a solution for extracting multiple event logs without having to restart from scratch when changing perspective. The other framework is called OnProm [4,5] and is briefly recalled in the next section.

3 The OnProm Approach

The OnProm approach [4,5] aims at the semi-automatic extraction of event logs from various kinds of legacy information systems, reflecting different process-related views on the same data, and consequently supporting analysts in the application of process mining along multiple perspectives. The approach leverages the *ontology-based data access* (OBDA) paradigm [21] and is based on the use of an ontology that captures the semantics of the domain of interest.

Intuitively, an ontology represents the domain of interest in terms of a set of *classes* (which denote sets of objects), *object properties* (which connect pairs of objects), and *data properties* (which connect objects to values), and captures domain knowledge by means of suitable axioms. Ontologies in OnProm are formalized using the lightweight description logic *DL-Lite*$_{\mathcal{A}}$ [2], belonging to the

DL-Lite family [3][1]. In OnProm, in order to facilitate the understanding of the ontology and the annotation of ontology elements (see below), we rely on the tight correspondence between the *DL-Lite* family and conceptual data models (see, e.g., [2]), and actually represent *DL-Lite*$_{\mathcal{A}}$ ontologies by means of UML Class Diagrams. Specifically, OnProm considers a fragment of UML Class Diagrams that allows for specifying *(i)* classes and *(ii)* class hierarchies (specified through ISA and generalizations), *(iii)* binary associations (corresponding to object properties) with *(iv)* multiplicity constraints (of the form `0..1`, `1`, `*`, and `1..*`), and *(v)* class attributes (corresponding to data properties) [4].

In OBDA, the domain ontology is linked to the underlying legacy data through declarative (as opposed to procedural) mappings that specify how the ontology elements (i.e., classes and properties) are to be populated from the data in the sources. In line with the OBDA paradigm, we assume that the legacy data sources are relational databases, typically equipped with integrity constraints, notably keys and foreign keys. To represent database schemas, we resort here to a simple box notation for tables, where we show the name of the table and its columns, and where we indicate referential integrity constraints using ER crow-foot notation (to distinguish one-to-many and one-to-one dependencies).

Considering a *DL-Lite*$_{\mathcal{A}}$ ontology $\mathcal{O}$ (represented as a UML Class Diagram) and a relational schema of data sources, consisting of a set $\mathcal{R} = \{R_1, \ldots, R_n\}$ of relation schemas, an *OBDA Specification* is a triple $\mathcal{S} = \langle \mathcal{O}, \mathcal{M}, \mathcal{R} \rangle$, where $\mathcal{M}$ is a set of *(OBDA) mappings* [2]. To formalize such mappings, we refer to a (countably infinite) set $\mathcal{V}$ of values (i.e., strings, numbers, dates, timestamps, etc.) that may be stored in the data source relations, and to a (countably infinite) set $\mathcal{F}$ of function symbols. These are exploited to construct the identifiers of the ontology objects, by applying a function symbol to data values extracted from the sources through the mappings. Specifically, each such identifier is a term of the form $f(v_1, \ldots, v_k)$, where $f \in \mathcal{F}$ is a function symbol of arity k and $v_1, \ldots, v_k$ are values in $\mathcal{V}$. We call such terms *object terms*. Then, each mapping in $\mathcal{M}$ has the form $Q_{sql}(\vec{x}) \rightsquigarrow \Phi(\vec{y}, \vec{t})$, where *(i)* $Q_{sql}(\vec{x})$ is an arbitrary SQL query over schema $\mathcal{R}$, having the (non-empty) set $\vec{x}$ of answer variables, *(ii)* $\vec{y} \subseteq \vec{x}$, *(iii)* $\vec{t}$ is a set of object terms, each of the form $f(\vec{z})$, where $f \in \mathcal{F}$ and $\vec{z} \subseteq \vec{x}$, and *(iv)* $\Phi(\vec{y}, \vec{t})$ is a set of atoms over the variables $\vec{y}$ and the object terms $\vec{t}$. Each such atom has as predicate symbol a class or a data/object property of the ontology $\mathcal{O}$. Intuitively, each mapping creates a link between the database and the ontology, expressing how instances of the involved classes/properties are obtained from the answers of queries posed over the database. Function symbols from $\mathcal{F}$, applied to the values retrieved from the data sources, are used to construct the object terms that act as identifiers of objects populating the ontology (For the formal semantics, we refer interested readers to [2].)

Example 1. Consider the portion of the domain ontology in Fig. 6 consisting of the class **SalesDoc** and its subclass **Order**. Consider further just the `VBAK` table

[1] The *DL-Lite* family provides the formal counterpart of the lightweight ontology language OWL 2 QL, standardized by the W3C [15].

in the database schema in Fig. 1, storing sales documents identified through the combination of the columns `MANDT` and `VBELN`, and where entries with `AUART` = `'ZOR'` denote orders. Taking this into account, we can use the following two mappings to link the `VBAK` table to **SalesDoc** and **Order**, respectively:

```
SELECT MANDT, VBELN FROM VBAK      ⇝   SalesDoc(sd(MANDT, VBELN))

SELECT MANDT, VBELN FROM VBAK
WHERE AUART='ZOR'                   ⇝   Order(sd(MANDT, VBELN))
```

Notice that in both mappings, in the right-hand side we have chosen to use the binary function symbol sd, applied to the answer variables `MANDT` and `VBELN` of the SQL query in the left-hand side, to construct object terms denoting sales documents. The mappings specify that these sales documents become instances of the **SalesDoc** and **Order** classes. For example, if the `VBAK` table contains the two tuples (`c25`, `d362`, `ZCN`) and (`c25`, `d571`, `ZOR`), the mappings generate the two instances sd(`c25`, `d362`) and sd(`c25`, `d571`) of **SalesDoc**, of which the latter is also an instance of **Order**. ◁

As mentioned, OnProm relies on the availability of a domain ontology and of the mappings to the underlying data. Producing these two artifacts might be very time-consuming, as they encode human knowledge and expertise about the underlying data sources. However, one can make use of well-established methodologies for ontology design [9] and of sophisticated tools that facilitate the construction of mappings even in complex real-world scenarios[2].

Finally, in OnProm, the domain ontology is graphically annotated to single out, among all possible alternatives, which classes and properties define the desired notions of case object, events, and their relevant attributes. This method aims to let domain experts focus solely on the high-level process characteristics, while the connection with the underlying data is handled by the OBDA system. Notably, the OnProm approach is particularly effective for multi-perspective process mining, since changing the notion of case only requires changing the (graphical) annotation of the domain ontology, without the need to revise the entire extraction process. Case and event annotations follow the OnProm notation [4], as shown in Fig. 6 for our running example (and discussed in Step 4 in the next section).

4 The erprep Guided Approach

The erprep guided approach provides a series of steps for a group of experts collaborating in a process mining project. The approach is inspired by the procedure in [11] and systematically explores it as an intermediate layer between the group of users and OnProm, providing guidance in taking decisions on the log structure in a conscious manner [12]. Following the PM^2 methodology [7], we assume the group contains people covering three main roles/competencies:

[2] See, e.g., the tools developed by Ontopic, https://ontopic.ai/.

(i) domain expertise of the organizational domain and the main questions to be answered through process mining; *(ii) data engineering* to access and query the information systems used by the organization to (implicitly) store events and process-related data; *(iii) analytics/process mining expertise* regarding analysis techniques and tools.

The main purpose of erprep is to drive extracting event data from document-driven enterprise systems in agreement with the questions the group wants to find an answer for, thus relieving the group from manual, ad-hoc data preparation procedures. We therefore need to better clarify what a document-driven enterprise system is. It is a relational database tracking the evolution of documents and related objects, obeying to three assumptions.

– Every document type is represented as a pair of tables, respectively storing information about the main (material) documents of that type, and the parts of those documents. A typical example is that of order and order lines, which typically coexist in the same material document (but may later be altered in different moments, through different activities). The document-part table points to the document table via a foreign key.
– Every document table is linked, directly or indirectly, to the other (document and non-document tables), hence navigational queries can be expressed to retrieve related documents and objects.
– Timestamps are used to denote the evolution of documents and other objects. Given a table C denoting an object or document type, a timestamp can be added in one of the two following ways:
  • *Column timestamp*, added directly as one or more columns of C to mark the transition of its instances from one phase to another; different timestamp columns (or groups of columns) can be used to mark different *phase transitions* of instances of C.
  • *History-table timestamp*, where the timestamp belongs to a separate *history-table* (or bundle of history-tables) listing all the changes applied to instances of C, indicating which column was affected, how, and *when.*[3]

Document-driven enterprise systems abstractly characterize the typical structure of ERP systems, such as SAP.

Example 2. Consider the database schema of Fig. 1, showing a fragment of the order-to-cash tables of SAP, which are used by the international company subject of the real case study reported in Sect. 5. The structure of the schema is a typical structure similarly adopted in other ERP systems, and uses 6 tables and 4 referential integrity constraints to capture the evolution of sales/billing documents and their lines. Specifically, VBAK and VBAP store sales documents and their lines, related via $\texttt{Ref}_1$; VBRP stores billing document lines; VBFA stores document flows, where each entry relates a sales document line with its next document, which sometimes is indeed a billing document (this depends on the complex join condition attached to $\texttt{Ref}_3$).

[3] History-tables closely relate to the notion of *redo logs* in databases, previously studied within process mining in [8].

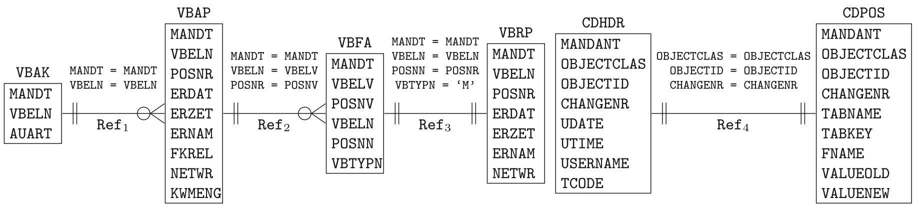

Fig. 1. Excerpt of the SAP order-to-cash database schema, in a Belgian installment. Naming is kept as in the original schema.

Both column and history-table timestamps are present. For example, the creation of a sales order line is traced via the combination of the ERDAT (date) and ERZET (time) columns of the VBAP table (cf. Figure 1), which is in fact the same table used to store order lines. The additional columns ERNAM, NETWR, and KWMENG in the same table respectively indicate the person who created the line, and its initial price and quantity.

The fact that a customer changes the quantity of a line is instead separately traced in the CDHDR and CDPOS history-tables from Fig. 1. Each entry in CDHDR indicates that change number CHANGENR occurred at date UDATE and time UTIME for the concept OBJECTCLAS, and in particular for its entry identified by OBJECTID. A corresponding entry in table CDPOS (where the correspondence is given by table, entry identifier, and change number) indicates the specifics of the change, in particular that the change is about modifying the value for the column FNAME from VALUEOLD to VALUENEW for table TABNAME. ◁

With this premise at hand, the four steps of the erprep procedure are:

1. *Data Understanding:* definition of the process mining questions/cornerstones and decision of which relevant tables to use for the procedure;
2. *Conceptual Understanding:* identification and documentation of which document/object/relationship types are represented in these tables;
3. *Activity Elicitation:* elicitation of the main activities of interest;
4. *Perspective, Granularity, and Scoping:* identification of a suitable notion of case and of the relevant events pertaining the elicited activities.

These steps are depicted in Fig. 2, together with their relationship to the OnProm approach. The first three steps are used to semi-automatically generate the domain ontology and mappings required by OnProm, whereas the last one guides the user in the semi-automatic generation of annotations used by OnProm to generate the desired XES log. We next detail the four steps of erprep, using a running example that covers an excerpt of the use case reported in Sect. 5.

Step 1: Data Understanding. This step corresponds to the classical initialization step of any process mining project [7]. The domain expert narrates in own words the process steps that are perceived as the cornerstones of the process. These are often a combination of activities and statuses. At the same time, the domain expert needs to explicitly formulate the key questions driving the

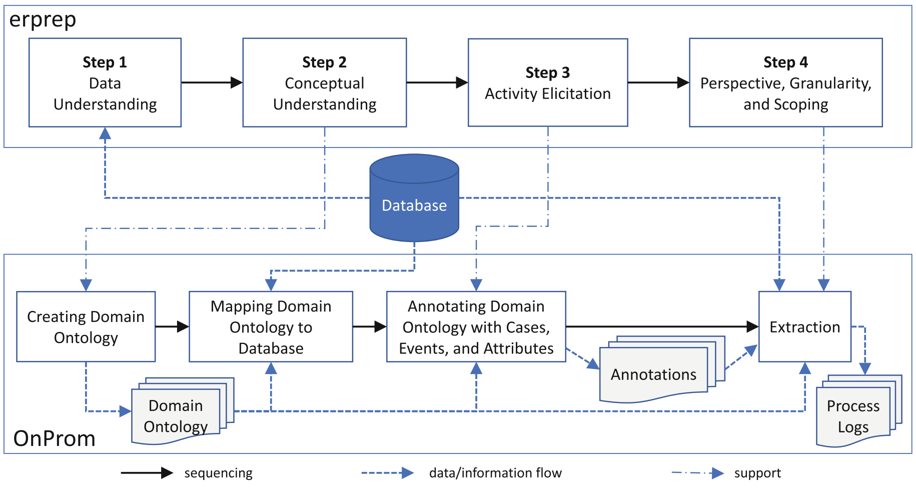
erprep
Step 1 Data Understanding
Step 2 Conceptual Understanding
Step 3 Activity Elicitation
Step 4 Perspective, Granularity, and Scoping
Database
Creating Domain Ontology
Mapping Domain Ontology to Database
Annotating Domain Ontology with Cases, Events, and Attributes
Extraction
Domain Ontology
Annotations
Process Logs
OnProm
sequencing
data/information flow
support

Fig. 2. The four steps of the erprep procedure and their relationship to OnProm.

process mining analysis. The data engineer then assists in identifying the tables that capture the timestamps of the cornerstones. These tables, the foreign keys between them, and the relevant fields are displayed in a relational schema so as to understand the underlying relations. This step provides the anchor for selecting out of thousands of tables of the database the set $\mathcal{R}$ of relevant tables that are part of the OBDA Specification $\langle\mathcal{O}, \mathcal{M}, \mathcal{R}\rangle$ to be provided (together with annotations) as input to OnProm.

Step 2: Conceptual Understanding. This step focuses on the conceptual understanding of the selected documents and related concepts. The purpose is to understand how the following key concepts are stored therein: *(i)* documents and their parts; *(ii)* document flows, mutually linking documents in a logical sequence; *(iii)* other relevant classes and associations. The focus here is on structural aspects (classes, attributes, and associations), while dynamic aspects related to activities executed on these structural elements are handled later.

Before entering in the description of the sub-steps, we describe the two main information sources used to identify classes, associations, and documents/document-parts, in corresponding *tabs*, that is, forms structured as depicted in Fig. 3. These tabs are filled in by the data engineer, on the basis of interaction with the domain expert. A *class tab* describes a relevant class whose instances can be obtained from an underlying table, possibly using a filter on its entries. Identifiers of objects in this class have to be consequently constructed from one or multiple columns from the table, where the default choice is that of selecting the primary key of the table. Also each relevant attribute of the class is obtained by taking the value of one or more columns (in the latter case, the semantics is that of concatenation). Sub-classes of the considered class could also implicitly be identified from the table, by looking into a so-called *discriminator column* whose values indicate to which sub-class the corresponding entry

Class ⟨*cla-name*⟩

DESCRIPTION	⟨*text*⟩		
TABLE	⟨*tab-name*⟩		
FILTER	⟨*sql-filter*⟩		
OBJECT ID	⟨*col-name*⟩$^{+}$		
ATTRIBUTES	NAME	DESCR.	COLS
	⟨*att-name*⟩	⟨*text*⟩	⟨*col-name*⟩$^{+}$
	...	...	
SUBCLASSES	⟨*col-name*⟩		
	SUBCLASS	COL VALUE	
	⟨*cla-name*⟩	⟨*text*⟩	
	...	...	

(a) Class tab template

Association ⟨*rel-name*⟩

DESCRIPTION	⟨*text*⟩	
	CLASS	MULT.
SOURCE	⟨*cla-name*⟩	⟨*min-max*⟩
TARGET	⟨*cla-name*⟩	⟨*min-max*⟩
LINK	⟨*sql-nav*⟩	

(b) Association tab template

Main document ⟨*cla-name*⟩

CLASS TAB

Document Part ⟨*cla-name*⟩

CLASS TAB	
LINK	⟨*sql-nav*⟩

(c) Document tab template

Fig. 3. Templates of documentation tabs for document classes and other classes/associations; the document tab uses twice the class tab (once for the main document, once for its parts). Sections can be omitted if not needed. Symbols inside angles indicate the entry type: ⟨*text*⟩ free text; ⟨*cla-name*⟩, ⟨*rel-name*⟩, and ⟨*att-name*⟩ class, association, and attribute names (at the conceptual level); ⟨*tab-name*⟩ and ⟨*col-name*⟩ table and column names (at the database level); ⟨*sql-filter*⟩ and ⟨*sql-nav*⟩ "SQL notes", sketching filters over single tables and join/navigational queries relating multiple tables; ⟨*min-max*⟩ multiplicity constraints à la UML (`0..1`, `1`, `*`, `1..*`).

belongs to; this tackles the common pattern where a class hierarchy is mapped to a single database table. An example of usage is given in Fig. 4.

An *association tab* describes a relevant association between two classes, previously described using class tabs. Such classes are linked through a so-called *navigation expression* that relates the two corresponding class tables. This is done in two possible ways: either by defining a full-fledged query that relates the two tables via joins and filters, or as a sequence of one or more referential constraints of the database schema. In the former case, the participating multiplicities of the source and target classes to the association have to be defined manually (possibly validating them using the data stored in the database). In the latter case, the multiplicities are instead directly obtained by considering the concatenation of the constraints: if, by moving from the source to the target, all referential constraints indicate mandatory participation, the source multiplicity has 1 as lower bound, 0 otherwise; similarly if, by moving from the source to the target, all referential constraints are functional, the source multiplicity has 1 as upper bound, * otherwise. The same line of reasoning applies from the target to the source, to define the target multiplicity.

In this phase, queries and filters have not to be thought as full SQL queries, but more as SQL "notes" used by the data engineer.

Step 2.1: Identification of documents and their flows. The main entry point for Step 2 is to start from the elicitation of the *document classes* of interest, identifying their document- and document-part- tables. To do so, one compiles a *document tab* that, as shown in Fig. 3(c), consists of three sections: *(i)* a *main document* section, defined using a class tab as described above; *(ii)* a *document*

Main document SalesDoc

DESCRIPTION	Doc. for sales-specific info	
TABLE	VBAK	
OBJECT ID	MANDT,VBELN	
SUBCLASSES	AUART	
	SUBCLASS	COL VALUE
	Order	ZOR
	DebitNote	...
	CreditNote	...

Document part SalesDocLine

DESCRIPTION	Entry for sales of a single item		
TABLE	VBAP		
FILTER	FKREL='a'		
OBJECT ID	MANDT,VBELN,POSNR		
ATTRIBUTES	NAME	DESCR.	COLS
	lineNo	line number	POSNR
LINK	VBAP->Ref$_1$->VBAK		

Fig. 4. Document tab for sales document in our running example.

part section, also defined using a class tab; *(iii)* a *link* section, containing a navigation expression to relate all parts of the same document to their unique, main document. The navigation expression must induce a one-to-many association from the main document to its parts – which can be directly checked if the navigation expression is a sequence of referential constraints.

Example 3. By applying Step 2.1 to the database schema of Fig. 1, we encode the knowledge described in Example 2 into a document tab for sales document, instantiating the template in Fig. 3(c) into Fig. 4. The tab shows that sales documents are identified through the combination of the columns MANDT (identifying the SAP client) and VBELN (identifying a specific sales document within a SAP client), and that the AUART column acts as discriminator column to identify the three types of sales documents existing in the system; for example, VBAK entries with AUART = 'ZOR' denote orders (as we had anticipated in Example 1). The tab further captures the knowledge that only VBAP entries with FKREL = 'a' (sales document lines that are relevant for billing) should be included in the event log. Each sales document line is identified by combining its MANDT and VBELN columns (actually referencing the two corresponding VBAK columns via Ref$_1$) together with POSNR, which denotes the line number. Ref$_1$ also provides the navigation for linking sales document lines to their sales documents. ◁

Step 2.2: Identification of additional classes and associations. Additional, relevant classes and associations can be defined similarly to documents and their parts, until all the tables isolated in Step 1 are inspected. Notice that not all the tables should be promoted to classes, as some of them are actually used to establish associations. This is for example the case of the VBFA table in Fig. 1, which is used to implicitly link different documents in a flow.

Step 2.3: Generation of core domain ontology and mappings. All the tabs filled in Steps 2.1 and 2.2 are used to automatically generate a domain ontology $\mathcal{O}$ that describes the domain of interest at a pure conceptual level, and constitutes a further component of the OBDA Specification $\langle \mathcal{O}, \mathcal{M}, \mathcal{R} \rangle$ to be used for data

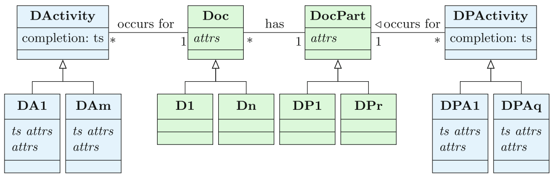

Fig. 5. Domain ontology fragment for document **Doc**, its parts, and related activities.

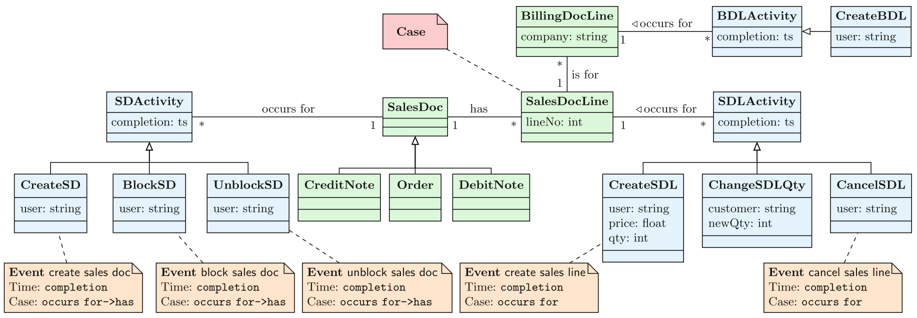

Fig. 6. Annotated domain ontology for the SAP schema fragment of Fig. 1. The green part denotes what is obtained after Step 2, the blue part the addition obtained after Step 3, and the annotations what done in Step 4. (Color figure online)

extraction via OnProm. Such an ontology will be later expanded with additional components. This is done by simply fetching all classes, subclasses, attributes, associations, and multiplicities introduced in the different documentation tabs. The green, central part of Fig. 5 shows the generic contribution that a specific document tab provides to the ontology $\mathcal{O}$ (the presence or absence of subclasses there depends on what indicated in the tab), while the central, green part of Fig. 6 shows the ontology generated for Example 3.

Documentation tabs also provide a basis to semi-automatically derive mappings $\mathcal{M}$ to link the database schema $\mathcal{R}$ isolated in Step 2 to the ontology $\mathcal{O}$. In this respect, each class tab provides the basis for generating one mapping to populate the instances of the class, using the table as a target for the selection of values, and what indicated in the OBJECT ID entry to construct objects from the retrieved database values. The same mechanism, using filters based on the discriminator column, is used to populate the subclasses. Also, every attribute entry leads to a dedicated mapping. In addition to the mapping for its two classes, each document tab leads to a further mapping, used to connect document part instances to their main document instance based on what indicated in the LINK entry. Finally, similar mappings are generated based on the association tabs.

Example 4. The document tab in Fig. 4 leads to the generation of the mappings that we have anticipated in Example 1 (where we have shown only one subclass mapping for brevity), together with the following mapping:

Activities

NAME	TABLE	COMPLETION TIME	TARGET CLASS	FILTER	LINK	OTHER ATTRS		
						NAME	DESCR.	QUERY
⟨*text*⟩	⟨*tab-name*⟩	⟨*col-name*⟩$^+$	⟨*cla-name*⟩	– or ⟨*sql-filter*⟩	– or ⟨*sql-nav*⟩	⟨*att-name*⟩	⟨*text*⟩	⟨*sql-nav*⟩
						...	...	...
...	...	...	...	...	...	...		

Fig. 7. Template of the activity tab for defining relevant activities; each entry depends on whether the timestamp is retrieved by column or by history-table.

`SELECT MANDT, VBELN, POSNR FROM VBAP WHERE FKREL=‘a’`
$\rightsquigarrow$ **SalesDocLine**(*sdl*(MANDT, VBELN, POSNR)), **lineNo**(*sdl*(MANDT, VBELN, POSNR), POSNR),
has(*sd*(MANDT, VBELN), *sdl*(MANDT, VBELN, POSNR))

Note how this last mapping populates at the same time the class **SalesDocLine**, its data property **lineNo**, and the object property **has**, which connects instances of **SalesDoc** to their sales document lines. ◁

Step 3: Activity Elicitation. Step 3 enriches the ontology $\mathcal{O}$ (and the mappings $\mathcal{M}$) obtained at the end of Step 2 with all the relevant activities that are executed on the documents and objects instantiating the classes mentioned therein. This is done by iterating over all document/document-part types and other classes that may be subject to activities. For each such class, the data engineer identifies the relevant timestamp columns in the database schema, considering the two patterns of column and history-table timestamps. For simplicity of exposition, we concentrate here on timestamps marking the completion of activities. Timestamp attributes are used to decide, together with the domain expert, which activities should be modeled.

Step 3.1: Turn timestamps into activities. This step singles out all relevant activities, and populates an overall activity tab following the template in Fig. 7. Each activity is described there by a name, the table containing the timestamp of interest, and a selection of columns providing that timestamp (more columns may be needed if, e.g., the date and time are stored separately, as it happens in our running example). In addition, the target class C containing the instances that are the subject of the activity is provided. Here, the two patterns of column and history-table timestamps have to be considered. In the case of column timestamp, no further details need to be given, as each instance of C will go through that activity whenever a timestamp in the selected column(s) is given. In the case of history-table timestamp, instead, since the table contains different changes for the same or different classes, and may be related to additional history tables, two information units have to be provided: a filter indicating under which circumstances a change actually describes an occurrence of the activity of interest, and a navigation expression to explain how entries in the history table relate to instances of C. In both cases, additional attributes (such as the resource responsible for the activity) can be declared.

Example 5. We continue our running example by eliciting two activities for sales document lines, following the description in Example 2. The creation of a sales order line can be simply tracked by looking into the columns `ERDAT` and `ERZET` of the `VBAP` table - according to the column timestamp pattern. A change of sales order line quantity is instead tracked following the history-table pattern, specifically by filtering the `CDHDR` table on those entries where `TCODE` = '`VA02`', `OBJECTCLAS` = '`VERKBELEG`', and whose corresponding entry in `CDPOS` (obtained by navigating $\texttt{Ref}_4$) is so that `FNAME` = '`KWMENG`', with the new quantity retrieved also from `CDPOS` by selecting `VALUENEW`. To link occurrences of this activity to corresponding lines, one can define a complex navigation expression connecting `CDHDR` to `VBAP`, using the aforementioned columns. ◁

Step 3.2: Extension of domain ontology and mappings. The core ontology $\mathcal{O}$ obtained at the end of Step 2 is automatically extended using the activity tab produced in Step 3.1. Specifically, every class **C** that has at least one activity gets associated via a one-to-many association to a corresponding **CActivity** class, denoting a generic activity, and equipped with the completion timestamp as attribute. Every entry in the activity tab filled for **C** in Step 3.1 becomes a dedicated subclass of **CActivity**, possibly carrying additional attributes. The result of this extension for a generic document is shown in the blue part of Fig. 5, whereas the result in the context of our running example is shown in the blue part of Fig. 6 (including the two activities elicited in Example 5).

Additional mappings linking the underlying database to such activity classes are semi-automatically devised, following again the procedure of Step 2.3.

Step 4: Perspective, Granularity, and Scoping. In this final step, the domain and the process mining experts finally concentrate only on the domain ontology $\mathcal{O}$, without considering anymore the specific storage mechanisms provided by the underlying database. The purpose is to define the perspective and scope of the analysis, identifying which notion of case best serves this purpose, and which events should be included (or not). Thanks to the ontology annotation mechanism provided by OnProm to identify cases and events, this final step is performed directly using OnProm in a guided way.

Step 4.1: Perspective and granularity selection. Based on the questions elicited in Step 1, the domain expert indicates which document provides the main angle for the analysis. This is not enough, as one has to consequently indicate the granularity of the analysis, either choosing the main document class, or the class for document-parts, as case notion. Depending on this choice, there may be *convergence* and *divergence* issues that the two experts have to be aware of [1]. Specifically, if the main document class is selected, all events related to its lines will be merged together in a single overall sequence, making it impossible to distinguish which events refer to the same line. On the other hand, if the document part class is selected, all events related to the main document will be replicated in the traces of all its parts. Thanks to the explicit annotation scheme provided by OnProm, and the fact that association multiplicities are explicitly shown in the ontology, these aspects are clearly communicated to the user group.

For example, Fig. 6 indicates that the decision has been to declare the sales document line as case class. This implies that all activities attached to the main sales document are replicated in each line trace, and the activities of multiple billing documents referring to the same line are merged in the same line trace.

Step 4.2: Scoping and definition of events. The very last step consists in selecting which activities are in the scope of the analysis, providing corresponding event annotations for those activities that are kept. As required by OnProm, every event must come with an event label (that is provided manually), an indication of the associated timestamp, and a navigation expression to indicate how the event is related to its reference case(s). The latter two information units are instead directly defined using what produced in previous steps of erprep, in particular by using the completion timestamps available in the abstract activity classes, and by exploiting the associations generated in Steps 3 and 4. The latter requires the intervention of the expert group to manually disambiguate those cases where multiple associations exist between two classes.

Figure 6 shows event annotations reflecting that the domain expert is interested in the creation and cancellation of sales lines, in the creation, blocking and unblocking of their parent sales documents, without considering the billing flow.

At this stage, OnProm has all the necessary information to automatically generate an XES event log from the database, reflecting the decisions taken. Alternative logs for different perspectives and scopes can be easily obtained by re-executing Step 4 with different annotations.

5 Case Study

To select an appropriate evaluation strategy for erprep, we applied the *framework for evaluation in design science* of [20] and selected the 'Human Risk & Effectiveness' evaluation strategy. This led to a case study evaluation of erprep in a formative-naturalistic setting, following [18,19]. Ideally, an evaluation benchmarks the new approach against a baseline setting where the approach has not been followed. However, this is nearly impossible in a naturalistic setting that involves industry, as it would require the case company to build several event logs in the traditional, labor-intensive way first, only to redo the exercise differently.

By applying erprep, we aim to answer the following questions: *(1)* Can the three envisioned phases of the OnProm approach be smoothly applied in a real-life setting? *(2)* Is performance playing a role in the feasibility of erprep?

To test these, we collaborated with an international manufacturing company that uses SAP. One of the authors interacted with the company. This researcher has expertise in process mining and event log building, but no prior experience with OnProm, thus representing a process mining expert that is knowledgeable of erprep, but not of ontology nor mapping design. The lack of knowledge on the OnProm approach by this person is key to this study. This set-up allowed us to evaluate whether erprep is capable of employing domain knowledge on both the process and the event data structure in its traditional form, not manipulated towards the OnProm approach. Orthogonal to the OnProm steps, the same researcher also executed the data extraction on premises of the company. The other authors are experts in the OnProm approach and toolchain.

The case study was conducted following the steps described in Sect. 4. In particular, the running example covers a fragment of what was conducted during the case study. The main difference is that, in the case study, the focus has been on the entire order-to-cash process, which requires to consider additional tables, and in turn a wider repertoire of documents, including documents related to accounting and delivery. Since compliance to financial regulations is of interest to the company, the accounting document was selected as case notion. However, by choosing the accounting document (the invoice) as case, it was not possible to answer some of the identified key questions, since they would require a different view on the process. Hence, a second event log was generated by taking the sales document perspective, which only required to repeat Step 4 of erprep a second time. Building these logs using erprep resulted in a smooth process, with a single workshop to gather the domain knowledge and the key questions.

The result was directly encoded into an ontology and mappings, following erprep without the need of taking further decisions. This led to the direct generation of 21 mappings linking the database to the ontology. Using also the annotations, OnProm automatically transformed them into 251 mappings linking the database to the XES concepts of trace, event, and event attributes. Two logs based on the result of erprep were generated on a computer with an Intel i7-7820HQ CPU and 64 GB of RAM. The input data were stored in a PostgreSQL database on the same machine, setting 48 GB as the maximum heap size for log extraction. We obtained the following indicators, witnessing feasibility; *(i)* using the sales document perspective, 238 806 traces, 1 345 269 events, and 10 489 169 attributes were extracted in 241 s, resulting in a log of 338 MB; *(ii)* using the invoice perspective, 247 051 traces, 1 742 127 events, and 10 497 414 attributes were extracted in 362 s, resulting in a log of 457 MB.

6 Conclusion

We have defined a guided approach, called erprep, to help expert groups in extracting event logs from relational databases of document-driven, enterprise information systems. The approach relies on OnProm as a technical pipeline for data extraction, while simplifying the set up of such pipeline. We have evaluated erprep on a real-world industrial use case, answering relevant research questions that show feasibility in an industrial setting.

The next, natural step is to make erprep able to generate object-centric logs, such as those conforming to the recent OCEL log format. This is directly implementable, considering a recent extension of OnProm to provide object-centric log annotations and the capability of generating OCEL logs [22].

References

1. Aalst, W.M.P.: Object-centric process mining: dealing with divergence and convergence in event data. In: Ölveczky, P.C., Salaün, G. (eds.) SEFM 2019. LNCS, vol. 11724, pp. 3–25. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30446-1_1

2. Calvanese, D., et al.: Ontologies and databases: the *DL-Lite* approach. In: Tessaris, S., et al. (eds.) Reasoning Web 2009. LNCS, vol. 5689, pp. 255–356. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03754-2_7
3. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. of Autom. Reason. **39**(3), 385–429 (2007)
4. Calvanese, D., Kalayci, T.E., Montali, M., Santoso, A.: OBDA for Log extraction in process mining. In: Ianni, G., Lembo, D., Bertossi, L., Faber, W., Glimm, B., Gottlob, G., Staab, S. (eds.) Reasoning Web 2017. LNCS, vol. 10370, pp. 292–345. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61033-7_9
5. Calvanese, D., Kalayci, T.E., Montali, M., Tinella, S.: Ontology-based data access for extracting event logs from legacy data: the onprom tool and methodology. In: Abramowicz, W. (ed.) BIS 2017. LNBIP, vol. 288, pp. 220–236. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59336-4_16
6. Cohn, D., Hull, R.: Business artifacts: a data-centric approach to modeling business operations and processes. IEEE Bull. Data Eng. **32**(3) (2009)
7. van Eck, M.L., Lu, X., Leemans, S.J.J., van der Aalst, W.M.P.: PM^2: a process mining project methodology. In: Zdravkovic, J., Kirikova, M., Johannesson, P. (eds.) CAiSE 2015. LNCS, vol. 9097, pp. 297–313. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19069-3_19
8. González López de Murillas, E., Reijers, H.A., van der Aalst, W.M.P.: Connecting databases with process mining: a meta model and toolset. Softw. Syst. Model. 18(2) (2019)
9. Guarino, N., Welty, C.A.: An overview of OntoClean. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies. International Handbooks on Information Systems, pp. 151–171. Springer, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24750-0_8
10. Ingvaldsen, J.E., Gulla, J.A.: Preprocessing support for large scale process mining of SAP transactions. In: ter Hofstede, A., Benatallah, B., Paik, H.-Y. (eds.) BPM 2007. LNCS, vol. 4928, pp. 30–41. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78238-4_5
11. Jans, M., Soffer, P.: From relational database to event log: decisions with quality impact. In: Teniente, E., Weidlich, M. (eds.) BPM 2017. LNBIP, vol. 308, pp. 588–599. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74030-0_46
12. Jans, M., Soffer, P., Jouck, T.: Building a valuable event log for process mining: an experimental exploration of a guided process. Enterp. Inf. Syst. **13**(5) (2019)
13. Li, G., de Murillas, E.G.L., de Carvalho, R.M., van der Aalst, W.M.P.: Extracting object-centric event logs to support process mining on databases. In: Mendling, J., Mouratidis, H. (eds.) CAiSE 2018. LNBIP, vol. 317, pp. 182–199. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92901-9_16
14. Lu, X., Nagelkerke, M., v. d. Wiel, D., Fahland, D.: Discovering interacting artifacts from ERP systems. IEEE Trans. Serv. Comput. 8(6) (2015)
15. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language profiles (second edition). W3C Recommendation, W3C (2012). https://www.w3.org/TR/owl2-profiles/
16. Mueller-Wickop, N., Schultz, M.: ERP event log preprocessing: timestamps vs. accounting logic. In: vom Brocke, J., Hekkala, R., Ram, S., Rossi, M. (eds.) i. LNCS, vol. 7939, pp. 105–119. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38827-9_8

17. Nooijen, E.H.J., van Dongen, B.F., Fahland, D.: Automatic discovery of data-centric and artifact-centric processes. In: La Rosa, M., Soffer, P. (eds.) BPM 2012. LNBIP, vol. 132, pp. 316–327. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36285-9_36
18. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Emp. Softw. Eng. **14**(2) (2008)
19. Shull, F., et al.: Replicating software engineering experiments: addressing the tacit knowledge problem. In: Proceedings of International Symposium on Empirical Software Engineering (2002)
20. Venable, J., Pries-Heje, J., Baskerville, R.: FEDS: a framework for evaluation in design science research. Eur. J. Inf. Syst. **25**(1) (2016)
21. Xiao, G., et al.: Ontology-based data access: a survey. In: Proceedings of IJCAI (2018)
22. Xiong, J., Xiao, G., Kalayci, T.E., Montali, M., Gu, Z., Calvanese, D.: Extraction of object-centric event logs through virtual knowledge graphs (extended abstract). In: Proceedings of DL. CEUR, vol. 3263. CEUR-WS.org (2022)