Ontology-Driven Extraction of Event Logs from Relational Databases

Diego Calvanese[1], Marco Montali[1(✉)], Alifah Syamsiyah[1], and Wil M.P. van der Aalst[2]

[1] Free University of Bozen-Bolzano, Bolzano, Italy
{calvanese,montali}@inf.unibz.it, alifah.syamsiyah@stud-inf.unibz.it
[2] Eindhoven University of Technology, Eindhoven, The Netherlands
w.m.p.v.d.aalst@tue.nl

Abstract. Process mining is an emerging discipline whose aim is to discover, monitor and improve real processes by extracting knowledge from event logs representing actual process executions in a given organizational setting. In this light, it can be applied only if faithful event logs, adhering to accepted standards (such as XES), are available. In many real-world settings, though, such event logs are not explicitly given, but are instead implicitly represented inside legacy information systems of organizations, which are typically managed through relational technology. In this work, we devise a novel framework that supports domain experts in the extraction of XES event log information from legacy relational databases, and consequently enables the application of standard process mining tools on such data. Differently from previous work, the extraction is driven by a conceptual representation of the domain of interest in terms of an ontology. On the one hand, this ontology is linked to the underlying legacy data leveraging the well-established ontology-based data access (OBDA) paradigm. On the other hand, our framework allows one to enrich the ontology through user-oriented *log extraction annotations*, which can be flexibly used to provide different log-oriented views over the data. Different data access modes are then devised so as to view the legacy data through the lens of XES.

Keywords: Multi-perspective process mining · Log extraction · Ontology-based data access · Event data

1 Introduction

Process mining aims to *discover, monitor and improve real processes by extracting knowledge from event logs* readily available in today's information systems [13]. Dozens (if not hundreds) of process-mining techniques are available and their value has been proven in various case studies [9]. Process mining techniques can be used to discover the real process, to detect deviations from

A. Syamsiyah—Supported by the European Master's Program in Computational Logic.

M. Reichert and H.A. Reijers (Eds.): BPM Workshops 2015, LNBIP 256, pp. 140–153, 2016.
DOI: 10.1007/978-3-319-42887-1_12

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

Case id	Timestamp	Activity	Resource	Cost
654423	30-04-2014:11.02	register request	John	300
654423	30-04-2014:11.06	check completeness of documents	Ann	400
655526	30-04-2014:16.10	register request	John	200
654423	30-04-2014:11.18	prepare decision	Pete	400
...	...	...	...	...

some normative process, to analyze bottlenecks and waste, and to predict flow times [13]. Normally, "flat" *event logs* serve as the starting point for process mining [13,14]. These logs are created with a particular process and a set of questions in mind. An event log can be viewed as a multiset of *traces*. Each trace describes the life-cycle of a particular *case* (i.e., a *process instance*) in terms of the *activities* executed. Often event logs store additional information about events. E.g., many process-mining techniques use extra information such as the *resource* (i.e., person or device) executing or initiating the activity, the *timestamp* of the event, or *data elements* recorded with the event (e.g., the size of an order). Table 1 shows a small fragment of a larger event log. Each row corresponds to an event. The events refer to two cases (654423 and 655526) and have additional properties, e.g., the registration for case 654423 was done by John at 11:02 on 30 April 2014 and the cost was 300€. An event may also contain transactional information, i.e., it may refer to an "assign", "start", "complete", "suspend", "resume", "abort", etc. action. For example, to measure the duration of an activity it is important to have a start event and a complete event. We refer to the *XES standard* [10] for more information on the data possibly available in event logs. See [14] for logging guidelines and details on getting event data from databases using redo logs.

It is apparent that a *condition sine qua non* for the application of process mining is the availability of faithful event logs, adhering to accepted standards (such as XES) and guaranteeing a certain quality level [15]. In many real-world settings, though, such event logs are not explicitly given, but are instead implicitly represented inside legacy information systems of organizations, which are typically managed through relational technology. This calls for the need of suitable methodologies, techniques and tools for *extracting event logs from relational databases*. This problem is extremely challenging, as pointed out in Chap. 4 of [13], which overviews the different problems encountered when extracting event data. On the one hand, this extraction process spans across several levels of abstractions: from the high-level, namely the domain-independent notions which are characterized at the conceptual level by the so-called *domain ontology*, and coming down to the concrete level at which data are effectively stored. On the other hand, there is no such a notion of "single" event log, but multiple event logs can be obtained by focusing on the dynamics of different domain entities. For example, in many applications there is not a single instance (case) notion. This is

addressed in the context of *artifact-centric process mining* [5].[1] Various tools for event log extraction have been proposed, e.g., XESame [16] and ProMimport [8]. Moreover, commercial tools like Disco make it easy to convert a CSV or Excel file into a XES log. In [14] it is shown how event data can be extracted from the redo-logs of a database. However, none of the tools and approaches actually puts the domain ontology in the loop. As a result, the extraction is often ad-hoc, data is duplicated for different views, and the semantics of the resulting event log cannot be traced back. Furthermore, the extraction cannot be driven by experts of the domain who do not have any technical knowledge about the underlying information systems and concrete storage mechanisms. Some work has been done on *semantically annotated event logs* [4]. However, these approaches do not consider the extraction of event data. Their focus is on exploiting ontological information during analysis.

In this work, we overcome these issues by proposing a novel framework that supports domain experts in the extraction of XES event log information from legacy relational databases, and consequently enables the application of standard process mining tools on such data. Differently from previous work, the extraction is driven by a conceptual representation of the domain of interest in terms of an ontology. This ontology is linked to the underlying legacy data leveraging the well-established *ontology-based data access* (OBDA) paradigm [1,11]. In this way, domain experts can focus on the ontological level only, while the connection with the underlying data is automatically managed by the OBDA system. Notably, after more than a decade of foundational investigation [3], OBDA systems relying on lightweight description logics [1] are now subject to extensive implementation efforts [2,12], so as to make them able to manage huge amounts of data [7]. To leverage OBDA in the context of event log data extraction and access, our framework allows one to enrich the ontology through user-oriented *log extraction annotations*, which can be flexibly used to provide different log-oriented views over the data. Once these annotations are specified, we show how it is possible to automatically construct a direct link from the raw relational data sources to a general-purpose ontology that captures the XES standard. This, in turn, provides the basis for the process mining algorithms to extract this information either by materializing it explicitly, or by accessing it on-demand. The framework has been implemented in a prototype ProM[2] plug-in that relies on the state-of-the-art OBDA system Ontop[3]. The full code with a tutorial and examples, is available at http://tinyurl.com/op6y82s.

2 Problem Overview

To describe the problem we want to attack and introduce the main technical challenges towards its solution, we informally discuss a real-world example. John, the owner of a conference submission web portal, is interested in applying process mining techniques on the historical system data. His goal is to better

[1] Cf. also the EU ACSI Project: www.acsi-project.eu.
[2] http://www.processmining.org/prom/start.
[3] http://ontop.inf.unibz.it/.

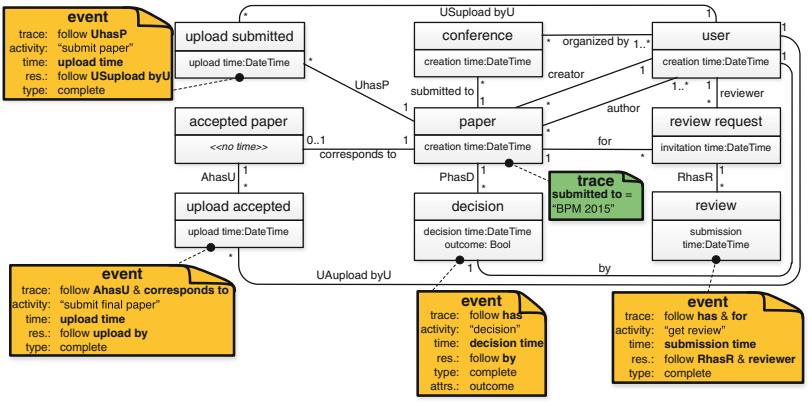

Fig. 1. Domain ontology of the conference submission portal in UML, with annotations to track the dynamics of BPM 2015 papers

LOGIN

ID	User
5	Alifah
6	Marco
2	Diego
3	Wil

LOGINSTATS

User	CT	LastAccess
5	15.10.14	15.03.15
6	17.10.14	20.03.15
2	18.10.14	20.03.15
3	18.10.14	18.03.15

PUB

ID	Title	CT	User	Conf	Type	Status
151	Mining	09.11.14	5	5	FP	R01
127	Monitoring	10.02.15	3	32	SP	A03
945	OBDA	13.03.15	2	21	FP	S02
724	BPM	15.03.15	3	56	FM	A02

Fig. 2. Excerpt of a possible instance of the conference submission information system

understand how the different users of the system actually use it, and consequently take strategic decisions on how to restructure the portal and improve its functionalities.

As it is typical in contemporary organizations, the relevant knowledge used by John and the other members of his company to *understand* the application domain, is captured through a conceptual schema (such as a UML class diagram). We call such a schema a *domain ontology*. This ontology provides a high-level, shared conceptual view of the domain. In John's case, it contains the relevant concepts and relations that must be captured in order to manage a conference. A fragment of this knowledge is shown in Fig. 1 (for the moment, we ignore the colored annotations). However, the actual data are not maintained at this level of abstraction, but are instead stored inside an underlying relational information system. Figure 2 provides the excerpt of a possible relational database keeping track of papers and authors. At this level, data are hardly understandable by John. First, the vocabulary and the organization of the data radically depart from that of the domain ontology, due to design and implementation choices made by the IT staff of the company. Second, internal codes with an implicit semantics are employed, like in the *Status* column of the PUB table (which tracks whether a publication has been submitted, reviewed, accepted, ...) or in the *Type* column of the same table (which tracks whether the

publication is a full/short paper, a front matter, or other). This so-called *impedance mismatch* is a challenging problem that has been thoroughly investigated in the field of intelligent data access and integration [11].

When John wants to apply process mining techniques on this complex information system, he does not only face the impedance mismatch problem, but also the equally challenging problem of "process-orientation": the underlying data must be understood through a conceptual lens that is different from the domain ontology, and that focuses on the process-related notions of trace, event, resource, timestamp, and so on. In other words, John needs to extract an *event log* that explicitly represents the dynamics John wants to analyze. In this paper, we consider XES as the reference standard for representing event logs. This problem becomes even more difficult if one considers that, in general, a plethora of different event logs may be extracted from the same data, by changing perspective and by focusing on the evolution of different entities. For example, John could decide to analyze his data by following the submission and review of papers within or across conferences, or he could focus on users and the operations they execute to submit and review papers.

In this light, supporting John requires to solve three technical problems:1. How can John overcomes the impedance mismatch between the domain ontology and the underlying data? 2. How can John captures the connection of the domain ontology and the representation of an event log, depending on the dynamics he wants to track? 3. How can John finally obtains a view of the low-level data in terms of a corresponding event log? In this work, we tackle this overarching problem by resorting to a novel combination of techniques coming from intelligent data access and integration, extended and adapted to the case of process mining and flexible extraction of multi-dimensional event logs from raw relational data. To attack the first problem, we resort to the well-established OBDA framework, which allows one to link the raw data to the domain ontology and overcome the impedance mismatch [1,11]. To tackle the second challenge, we define an event log ontology that mirrors XES, and provide an annotation language to the user, which makes it possible

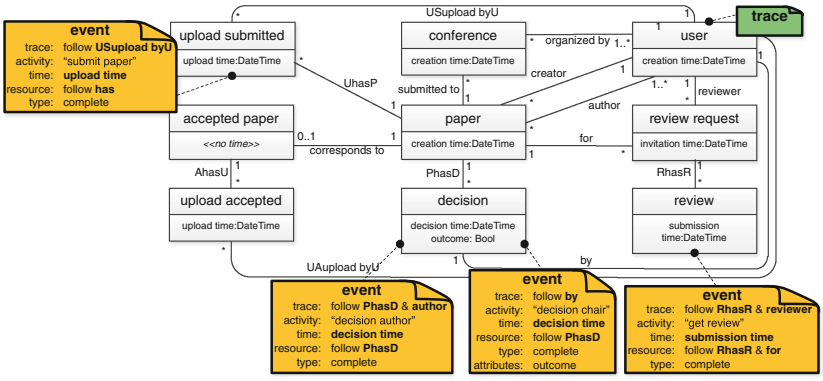

Fig. 3. Domain ontology annotations to track the dynamics of users

to capture semantic links between the constitutive (combinations of) elements in the domain ontology, and corresponding elements in XES. John could employ this annotation language to enrich the domain ontology of Fig. 1, ultimately producing the two schemas illustrated in Fig. 1 itself and in Fig. 3. In particular, Fig. 1 declares that each trace is related to the evolution of a single paper submitted to the BPM 2015 conference, and that meaningful events in a trace are paper submissions, reviews, final decisions, and upload of the camera-ready version. Figure 3 declares instead that each trace tracks the operations of a user, and that meaningful operations are paper submissions, reviews, and final decisions (to be listed both in the trace of the person chair who took the decision, and the paper creator who received it). The third problem is finally solved by automatically establishing a direct bridge from the low-level relational data to the event log ontology, in a way that is completely transparent to the user. The user can then access the event log with different modalities, and apply process mining without knowing how traces, events, and attributes are concretely stored in the underlying information system.

3 Preliminaries

We introduce some necessary background material, namely the description logic (DL) *DL-Lite*$_\mathcal{A}$ and the ontology-based data access (OBDA) framework. To capture domain ontologies, we use the *DL-Lite*$_\mathcal{A}$ language [1]. This allows for specifying *concepts*, representing sets of (abstract) objects, *roles*, representing binary relations between objects, and *attributes*, representing binary relations between objects and (domain) values. The syntax of *concept expressions* (B) and *role expressions* (R) in *DL-Lite*$_\mathcal{A}$ is:

$$B \longrightarrow N \mid \exists R \mid \delta(U) \qquad\qquad R \longrightarrow P \mid P^-$$

Here, N, P, and U respectively denote a *concept name*, a *role name*, and an *attribute name*, and P^- denotes the *inverse* of role P. The concept $\exists R$, also called *unqualified existential restriction*, denotes the *domain* of a role R, i.e., the set of objects that R relates to some object. Notice that $\exists P^-$ actually denotes the *range* of role P. Similarly, the concept $\delta(U)$ denotes the *domain* of an attribute U.

A *DL-Lite*$_\mathcal{A}$ *ontology* is a pair $(\mathcal{T}, A)$, where $\mathcal{T}$ is a TBox, i.e., a finite set of *TBox assertions*, and A is an Abox, i.e., a finite set of *ABox assertions*. *DL-Lite*$_\mathcal{A}$ TBox assertions have the following form:

$B_1 \sqsubseteq B_2$	$R_1 \sqsubseteq R_2$	$U_1 \sqsubseteq U_2$	(funct R)
$B_1 \sqsubseteq \neg B_2$	$R_1 \sqsubseteq \neg R_2$	$U_1 \sqsubseteq \neg U_2$	(funct U)

In the first three columns, assertions of the first row denote *inclusions* between concepts, roles, and attributes, respectively; assertions of the second row denote *disjointness*. Assertions of the last column denote *functionality* on roles and attributes, i.e., that every object in the domain of R/U is related via R/U to at most one other object/value.

DL-Lite$_\mathcal{A}$ ABox assertions are used to express extensional knowledge about specific objects and values in the domain of interest. They have the form $N(t)$, $P(t, t')$, or $U(t, v)$, where t and t' denote individual objects and v denotes a value.

The semantics of *DL-Lite*$_\mathcal{A}$ is given in [1]. Interestingly, *DL-Lite*$_\mathcal{A}$ TBoxes are suitable to formally capture the semantics of UML class diagrams (with the exception of *covering constraints* in a class hierarchy) [1]. Consequently, whenever we talk about a *DL-Lite*$_\mathcal{A}$ domain ontology, we can always imagine that the intensional knowledge of such an ontology can be modelled and graphically rendered in UML.

Example 1. Let *Paper* and *User* be *DL-Lite*$_\mathcal{A}$ concepts, *creator* and *author* roles, and *pCT* and *uCT* attributes (corresponding to the creation time of a paper and of a user respectively). The following *DL-Lite*$_\mathcal{A}$ TBox captures a portion of the UML domain ontology shown in Fig. 1:

$$\begin{array}{rrr} \exists creator \sqsubseteq Paper & \exists author \sqsubseteq Paper & \delta(pCT) \sqsubseteq Paper \\ \exists creator^- \sqsubseteq User & \exists author^- \sqsubseteq User & Paper \sqsubseteq \delta(pCT) \\ Paper \sqsubseteq \exists creator & Paper \sqsubseteq \exists author & \delta(uCT) \sqsubseteq User \\ (\mathsf{funct}\ creator) & & User \sqsubseteq \delta(uCT) \end{array}$$

The first column captures the semantics of the *creator* UML association, where the first two rows capture the typing of the association, the third row the fact that every paper must have a creator, and the fourth that every paper has at most one creator. Collectively, the last two assertions capture the 1 cardinality of the association from the perspective of the paper class. The second column captures the semantics of the *author* UML association. The third column instead deals with the creation time attributes for papers and users. ■

To interact with the domain ontology, we make use of *queries*. As typical in DLs, to *query* a *DL-Lite*$_\mathcal{A}$ ontology we make use of conjunctive queries (CQs) and union thereof (UCQs). CQs are first-order queries that corresponds to the well-known SPJ (select-project-join) queries in SQL. Syntactically, we specify UCQs using SPARQL, the standard ontology query language for the Semantic Web.

Ontology-Based Data Access. In an OBDA system, a relational database is connected to an ontology that represents the domain of interest by a mapping, which explicitly accounts for the impedance mismatch by relating database values with values and (abstract) objects in the ontology (c.f. [1]).

Technically, we consider a countably infinite set $\mathcal{V}$ of values and a set Λ of function symbols, each with an associated arity. Intuitively, function symbols are used to construct an abstract object in the ontology from a combination of values in the underlying database. We also define the set $\mathcal{C}$ of constants as the union of $\mathcal{V}$ and the set $\{f(d_1, \ldots, d_n) \mid f \in \Lambda \text{ and } d_1, \ldots, d_n \in \mathcal{V}\}$ of *object terms*.

Formally, an OBDA system is a structure $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$, where: *(i)* $\mathcal{R} = \{R_1, \ldots, R_n\}$ is a database schema, constituted by a finite set of relation schemas; *(ii)* $\mathcal{T}$ is a *DL-Lite*$_\mathcal{A}$ TBox; *(iii)* $\mathcal{M}$ is a set of mapping assertions, each of the

form $\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{y}, \vec{t})$, where: (a) $\vec{x}$ is a non-empty set of variables, (b) $\vec{y} \subseteq \vec{x}$, (c) $\vec{t}$ is a set of object terms of the form $f(\vec{z})$, with $f \in \Lambda$ and $\vec{z} \subseteq \vec{x}$, (d) $\Phi(\vec{x})$ is an arbitrary SQL query over $\mathcal{R}$, with $\vec{x}$ as output variables, and (e) $\Psi(\vec{y}, \vec{t})$ is a set of atoms over the variables $\vec{y}$ and the object terms $\vec{t}$, whose predicate symbols are atomic concepts, atomic roles, and attributes of $\mathcal{T}$. Without loss of generality, we use the special symbol $\mathsf{val}/1$ to map values from the relational layer to the range of attributes in the semantic layer.

Each mapping assertion creates a link between the database and the ontology, expressing how instances of the involved concepts/roles are obtained from the answers of queries posed over the database. We ground this definition to our running example.

Example 2. Consider an OBDA framework devised to link the database shown in Fig. 2 to the ontology in Fig. 1. Suitable mapping assertions must be devised so as to interconnect these two information layers. We consider the definition of *User* and *Paper*, with their creation times. One could argue that a paper can be extracted from the PUB table, by considering only those entries that have type FP or SP (respectively denoting full and short papers). This is captured by the following mapping assertion ($\mathsf{p}/1$ constructs a publication object from its identifier in the database):

$$\texttt{SELECT } ID, CT \texttt{ FROM PUB WHERE } CT\texttt{='FP' OR } CT\texttt{='SP'} \\ \rightsquigarrow \{Paper(\mathsf{p}(ID)), pCT(\mathsf{p}(ID), \mathsf{val}(CT))\}$$

A user and his/her system creation time are extracted by joining LOGIN and LOGINSTATS tables:

$$\texttt{SELECT } L.User, S.CT \texttt{ FROM LOGIN } L, \texttt{ LOGINSTATS } S \texttt{ WHERE } L.ID\texttt{=}S.User \\ \rightsquigarrow \{User(\mathsf{u}(L.User)), pCT(\mathsf{u}(L.User), \mathsf{val}(S.CT))\}$$

where $\mathsf{u}/1$ constructs a user object from its username in the database. ■

A UCQ q over an OBDA system $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ is simply a UCQ over $\mathcal{T}$. To compute the answers of q over $\mathcal{O}$ wrt a database instance D over $\mathcal{R}$, two approaches can be followed. In the first approach, which we call *materialization*, query answering consists of two steps. In the first step, an ABox is explicitly materialized starting from D and by applying the mapping assertions in a forward way. In particular, the ABox generated from D by a mapping assertion $m = \Phi(x) \rightsquigarrow \Psi(y, t)$ in $\mathcal{M}$ is $m(D) = \bigcup_{v \in eval(\Phi, D)} \Psi[x/v]$, where $eval(\Phi, D)$ denotes the evaluation of the SQL query Φ over D. Then, the ABox generated from D by the mapping $\mathcal{M}$ is $\mathcal{M}(D) = \bigcup_{m \in \mathcal{M}} m(D)$. In the second step, the query is posed directly over the domain ontology $\langle \mathcal{T}, \mathcal{M}(D) \rangle$.

In the second approach, which we call *on-demand*, the data are kept in D and no further information is explicitly materialized. Instead, the input query q is subject to a multi-step reformulation approach, which: *(i)* compiles away $\mathcal{T}$; *(ii)* unfolds the components of the obtained query by applying $\mathcal{M}$. This produces a corresponding SQL query that can be posed directly over the underlying database [11]. The answers are then computed and returned in the form of ontological objects. Notably, the two approaches yield the same answer, and can be therefore be interchangeably applied.

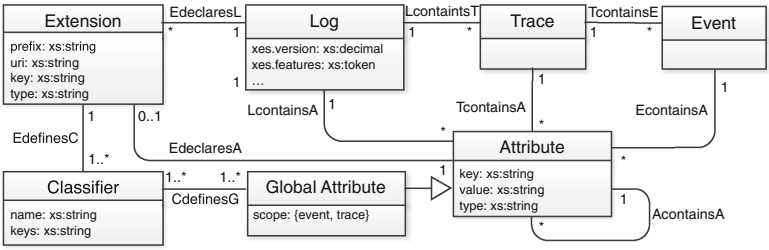

Fig. 4. General event ontology

4 Log Extraction Framework

Our log extraction process is organized in two phases:

1. the *design phase*, in which a domain expert specifies how to view the domain ontology using the lens of an event ontology for XES;
2. the *data access phase*, in which the result of the first phase is applied over actual data, so as to obtain a corresponding event log that is then accessed by domain experts or process mining algorithms.

In this section, we focus on the design phase. Data access is discussed in Sect. 5. The purpose of the design phase is to support a domain expert in the annotation of a domain TBox given as input, so as to properly link it to the event ontology that captures XES. Notice that the domain expert is not required, in this phase, to employ actual data, nor to have any specific knowledge about how such data are stored inside the company information system.

Specifically, we discuss how XES has been modeled as a *DL-Lite*$_\mathcal{A}$ TBox, and then focus on the language and modeling language to annotate the domain TBox, implicitly establishing a link with the XES TBox.

The XES Event Log TBox. We carefully analyzed the documentation of XES [10], and consequently derived the *DL-Lite*$_\mathcal{A}$ TBox $\mathcal{X}$, rendered in Fig. 4 as a UML class diagram. This TBox is fixed once and for all in our framework, and does not depend on the modeled domain. Beside the standard XES elements of *Trace*, *Event*, and *Attribute*, we also consider the following standard extensions: *(i) Concept* extension (to assign a name to events); *(ii) Time* extension (to assign a timestamp to events); *(iii) Lifecycle* extension (to link the event type to the XES transactional model); *(iv)* (optionally) *Organizational* extension (to link an event to its responsible resource).

Annotations. Annotations are defined by a domain expert to link a domain-specific *DL-Lite*$_\mathcal{A}$ TBox $\mathcal{T}$ to the generic XES TBox $\mathcal{X}$. Intuitively, they take the form reported in Figs. 1 and 3. Technically, a *log annotation* over $\mathcal{T}$, written $\mathcal{L}_\mathcal{T}$, is a pair $\langle \Phi_{tr}(t,l), \mathcal{E} \rangle$, where $\Phi_{tr}(t,l)$ is a *trace annotation*, i.e., a SPARQL query expressing how to extract traces (and their corresponding log) from $\mathcal{T}$, and $\mathcal{E}$ is a set of *event-attribute annotations*. Each event-attribute annotation is a tuple $\langle \Phi_{ev}(\vec{e},t), \Phi_{ts}(ts,\vec{e}), \Phi_n(n,\vec{e}), [\Phi_a^1(v_1,a_1,c_1,\vec{e}), \ldots, \Phi_a^k(v_k,a_k,c_k,\vec{e})] \rangle$ of SPARQL

queries over $\mathcal{T}$, where: *(i)* query $\Phi_{ev}(\vec{e}, t)$ is an *event annotation*, i.e., a SPARQL query extracting events from $\mathcal{T}$, together with the trace they belong to; *(ii)* queries $\Phi_{ts}(ts, \vec{e})$ and $\Phi_{n}(n, \vec{e})$ bind events to the two mandatory attributes of timestamp and activity name (and are consequently called *timestamp* and *name annotations*); *(iii)* each query $\Phi^{i}_{a}(v_i, a_i, c_i, \vec{e})$ (with $i \in \{1, \ldots, k\}$) is an *attribute annotation* that extracts the value(s) v_i of an attribute of type a_i and with key c_i, binding it/them to event $\vec{e}$. Attribute annotations are optional. Among all possible attributes, they can be used to extract the responsible resource for an event, or its transactional lifecycle type (cf. the XES *Organizational* and *Lifecycle* extensions). Obviously, different guidelines and constraints apply to the different queries in $\mathcal{L}_{\mathcal{T}}$. We discuss this issue in the following.

Note that each query in $\mathcal{L}_{\mathcal{T}}$ is centred around a concept/role/attribute of $\mathcal{T}$. Hence, it is straightforward to ideally "attach" an annotation to the corresponding element in $\mathcal{T}$.

Trace Annotations. A trace focuses on the evolution of an entity/object over time. In Fig. 1, the focus is on the evolution of papers, whereas in Fig. 3 it is on users. In general, a trace can be annotated only by selecting a concept in $\mathcal{T}$, possibly adding further conditions on which members of such concept give raise to a trace. Technically, this means that in $\Phi_{tr}(t, l)$, t must be bound to a concept of $\mathcal{T}$. For example, the trace annotation in Fig. 1 can be formalized as:

```
SELECT ?t "BPM15-papers-log"
WHERE {?t a:Paper; :submittedTo ?conf.
       ?conf a:Conference. FILTER regex(?conf,"BPM 2015","i").}
```

where `"BPM15-papers-log"` identifies the log l we are constructing.

Event Annotations. Event annotations differ from trace annotations in two respects. First of all, any element of $\mathcal{T}$ may be subject of an event annotation: not only concepts, but also relations and attributes. The arity of the corresponding SPARQL query then depends on which element is targeted: if it is a concept, than a single event variable will be used; it if is a relation or an attribute, two variables will be employed, matching with the involved subject and object. Second, to be a valid target, such an element must be "timed", i.e., either directly or undirectly univocally associated to exactly one timestamp attribute, and have a unique name. To check whether this property is guaranteed or not, one could pose its corresponding SPARQL query over the input OBDA system, and verify that, for each event, exactly one timestamp and one name is returned. By referring again to Fig. 1, the indication that a submission upload is an event is captured by:

```
SELECT ?e ?t WHERE {?e a:UploadSubmitted; :has? t. ?t a:Paper.}
```

Note that event annotations always have event and trace variables as distinguished variables, establishing a correspondence between the returned events and their traces. Also notice that a single variable is used for the event, since it is extracted from a concept.

Timestamp and Name Annotations. We consider two mandatory attributes: a timestamp and an activity name. Both of them are distinguished variables in

the SPARQL query together with its event. As pointed out before, each event is required to be associated with exactly one timestamp and exactly one name. Obviously, in general many different possibilities exist. For example, in Fig. 1 it is apparent that the submission time associated to a submission upload may be the upload time of the upload itself, or the creation time borrowed either from *User* by navigating the *uploadBy* relation, or from *Paper* by navigating the *has* relation. As for the activity name, the most typical situation is the one in which the name is directly bound to an explicitly specified string. However, in general one could construct such a name by navigating through $\mathcal{T}$ and even composing the name by concatenating together strings and query results. As for the timestamp, three annotation patterns typically emerge. We discuss each of them.

1. Concept with single timestamp. The timestamp attests that the concept is directly timed, and therefore the concept itself is marked as an event. This is the case, e.g., for the submission upload in Fig. 1.
2. Concept with pre-defined multiple timestamps. This pattern emerges whenever there are objects that flow through different "phases" over time. For example, a cart may be associated to a pre-defined set of timestamps, such as the creation time, the finalization time, and the payment time. Each of such timestamps represent a specific phase transition, and some of such timestamps may be null (this is the case, e.g., for a cart that has been created but not finalized). Each phase transition could be singled out as a relevant event, by annotating the corresponding timestamp attribute.
3. Variable CRUD timestamps. This situation arises whenever a certain concept/relation is subject to many different operations over time, whose number is not known a priori. For example, objects may have *create*, *read*, *update*, *delete* actions related to them, and there can be any number of actions related to the same object, e.g., multiple updates. Similarly, there may also be *create* and *delete* actions for relations. If we imagine to have an "operation table", containing the different operation types and the corresponding object identifiers (each denoting the object over which an operation is applied), then such a table will be annotated as an event, whose name depends on the operation type.

Optional Attribute Annotations. Optional attributes are annotated by using specific SPARQL queries having, as distinguished variables: *(i)* the attribute type, *(ii)* its key, *(iii)* its value, *(iv)* the variables of the corresponding event.

5 XES Log Extraction and Access

In Sect. 4, we have shown how a domain expert can annotate a domain *DL-Lite*$_\mathcal{A}$ TBox so as to express how to conceptually identify traces, events and their attributes starting from the domain concepts, relations and attributes. We now show how such annotations can be employed so as to make the framework operational. We assume to have, as input, not only a domain TBox, but also an entire OBDA system, previously prepared to link the domain TBox with the corresponding underlying relational database. This process is completely

independent from process mining-related interests. Hence, the overall input for our log extraction framework is an OBDA system $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$. After the design phase, we also have a log annotation $\mathcal{L}_\mathcal{T}$ for $\mathcal{T}$. To make the framework operational and effectively get the data contained in a concrete instance D of the company database $\mathcal{R}$, we proceed in two steps: *(i)* The annotations are used to automatically create a new OBDA system that links the company database $\mathcal{R}$ to the XES TBox $\mathcal{X}$, according to the semantics of annotations in $\mathcal{L}_\mathcal{T}$. *(ii)* This OBDA system is exploited to access the concrete data in D through the conceptual lens of $\mathcal{X}$, following the materialization or the on-demand paradigm (cf. Sect. 3 (Ontology-Based Data Access)). We focus on both aspects, discussing how they have been implemented using ProM as process mining infrastructure, OpenXES as reference implementation for XES, and Ontop for OBDA.

Automatic Generation of Schema-to-XES Mapping Assertions. Given an OBDA system $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ and a log annotation $\mathcal{L}_\mathcal{T}$ for $\mathcal{T}$, we automatically construct a new OBDA system $\mathcal{O}_{log} = \langle \mathcal{R}, \mathcal{X}, \mathcal{M}_{log} \rangle$ that directly links schema $\mathcal{R}$ to the XES TBox $\mathcal{X}$. This is done by iterating through the annotations present in $\mathcal{L}_\mathcal{T}$, and by transforming each annotation into a corresponding mapping assertion in $\mathcal{M}_{log}$. Intuitively, this is done as follow. Let $\Phi(\vec{x})$ be the SPARQL query associated to one of the annotations in $\mathcal{L}_\mathcal{T}$. We first reformulate $\Phi(\vec{x})$ as a corresponding SQL query $Q(\vec{x})$ directly posed over $\mathcal{R}$. We then construct a mapping assertion of the form $Q(\vec{x}) \rightsquigarrow \Psi(\vec{y}, \vec{t})$, where Ψ is a set of atoms over $\mathcal{X}$, built according to the semantics of the annotation. Since we have 5 kinds of annotations (for trace, event, event timestamp, event activity name, event attributes), 5 corresponding translation rules must be provided do as to generate such mapping assertions. As an example, we consider the case of the trace annotation $\Phi_{tr}(t, l)$ in $\mathcal{L}_\mathcal{T}$. The corresponding mapping assertion to be inserted in $\mathcal{M}_{log}$ is:

$$Q(t,l) \rightsquigarrow \left\{ \begin{array}{l} \mathit{LcontainsT}(\mathsf{log}(l), \mathsf{tr}(t)),\ \mathit{TcontainsA}(\mathsf{tr}(t), \mathsf{attr}(t)), \\ \mathit{key}_{\mathit{Attribute}}(\mathsf{attr}(t), \texttt{"concept:name"}), \\ \mathit{value}_{\mathit{Attribute}}(\mathsf{attr}(t), t),\ \mathit{type}_{\mathit{Attribute}}(\mathsf{attr}(t), \texttt{"literal"}) \end{array} \right\}$$

where $Q(t, l)$ is the SQL query corresponding to $\Phi_{tr}(t, l)$, and specific unary function symbols are used to construct the abstract objects of log, trace, and attribute, out from the flat values for t and l. The first line defines the relationships between the log $\mathsf{log}(l)$ and the trace $\mathsf{tr}(t)$, as well as between $\mathsf{tr}(t)$ and an attribute generated for it. Note that there is no need to explicitly assert the concepts to which these three objects belong, as all relations are typed in $\mathcal{X}$. The features of the trace attribute are fixed in the second and third line, which model that value t is a literal that constitutes the name of $\mathsf{tr}(t)$.

Data Access. Once the OBDA system $\langle \mathcal{R}, \mathcal{X}, \mathcal{M}_{log} \rangle$ has been obtained, the data contained in $\mathcal{R}$ (D henceforth) can be "viewed" and accessed through the lens of the XES ontology $\mathcal{X}$ thanks to $\mathcal{M}_{log}$. We support in particular two access modes, which have been effectively implemented. The first mode is the *XES log materialization* mode, and consists in concretely materializing the actual event log in the form of the ABox $\mathcal{M}_{log}(D)$, using Ontology-Based Data Access procedure. This ABox is then automatically serialized into an XML file that

is fully compliant with the XES standard. Multiple XES logs can be seamlessly obtained by just changing the annotations. This mode has been implemented as a ProM 6 plug-in. It currently supports only a textual specification of the ontology and the annotations, but we are working on a GUI that exposes the domain TBox as a UML class diagram, and allows the user to visually annotate it.

The second mode is the *on-demand access*. With this approach, do not use $\mathcal{M}_{log}$ to concretely materialize the log, but we maintain the data in D, and reformulate queries posed over $\mathcal{X}$ as SQL queries directly posed over D. In this light, the XES log only "virtually" exists: no redundant copy of the data is created, and log-related information is directly fetched from D. Since the caller does not perceive any difference when adopting this strategy or the other one, process mining algorithms can seamlessly exploit both techniques without changing a line of code. This mode has been realized by providing a new implementation of the OpenXES interface, used to access XES logs from JAVA. The implementation combines the on-demand OBDA approach with the well known design pattern of *lazy loading* [6], which intutively indicates to defer the initialization of an object to when it is needed. In particular, the JAVA side does not really maintain in memory the whole log, but when a portion of the log is needed by the requester, it is lazily constructed by issuing a query to the underlying database.

6 Conclusions

We have proposed a novel methodology and technology to flexibly extract event logs from legacy, relational data sources, by leveraging the ontology-based data access paradigm. This is especially useful for multi-perspective process mining, since event logs reflecting different views of the same data can be obtained by just changing the ontology annotations. Our framework enables the materialization of event logs from legacy data, or the possibility of maintaining logs virtual and fetch log-related information on-demand. We are currently following three lines of research: *(i)* application of the framework to real-world case studies; *(ii)* improvement of the framework with visuali interfaces; *(iii)* benchmarking of the different data access strategies.

References

1. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R.: Ontologies and databases: the *DL-Lite* approach. In: Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M.-C., Schmidt, R.A. (eds.) Reasoning Web. LNCS, vol. 5689, pp. 255–356. Springer, Heidelberg (2009)
2. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The Mastro system for ontology-based data access. Semantic Web J. **2**(1), 43–53 (2011)
3. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: DL-Lite: tractable description logics for ontologies. In: Proceedings of AAAI (2005)
4. Alves de Medeiros, A.K., van der Aalst, W.M.P., Pedrinaci, C.: Semantic process mining tools: core building blocks. In: Proceedings of ECIS (2008)

5. Fahland, D., De Leoni, M., van Dongen, B., van der Aalst, W.M.P.: Many-to-many: some observations on interactions in artifact choreographies. In: Proceedings of ZEUS (2011)
6. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley (2003)
7. Giese, M., et al.: Scalable end-user access to big data. In: Rajendra, A. (ed.) Big Data Computing. CRC (2013)
8. Günther, C.W., van der Aalst, W.M.P.: A generic import framework for process event logs. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 81–92. Springer, Heidelberg (2006)
9. IEEE Task Force on Process Mining. Process mining case studies (2013). http://tinyurl.com/ovedwx4
10. IEEE Task Force on Process Mining. XES standard definition (2013). http://www.xes-standard.org/
11. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. In: Spaccapietra, S. (ed.) Journal on Data Semantics X. LNCS, vol. 4900, pp. 133–173. Springer, Heidelberg (2008)
12. Rodríguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-based data access: *Ontop* of databases. In: Alani, H., et al. (eds.) ISWC 2013, Part I. LNCS, vol. 8218, pp. 558–573. Springer, Heidelberg (2013)
13. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Heidelberg (2011)
14. van der Aalst, W.M.P.: Extracting event data from databases to unleash process mining. In: Proceedings of BPM. Springer (2015)
15. van der Aalst, W.M.P., et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012)
16. Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: XES, XESame, and ProM 6. In: Soffer, P., Proper, E. (eds.) CAiSE Forum 2010. LNBIP, vol. 72, pp. 60–75. Springer, Heidelberg (2011)