Connecting Databases with Process Mining: A Meta Model and Toolset

E. González López de Murillas[1,3], H.A. Reijers[1,2], and W.M.P van der Aalst[1]

[1] Department of Mathematics and Computer Science
Eindhoven University of Technology, Eindhoven, The Netherlands
[2] Department of Computer Science
VU University Amsterdam, Amsterdam, The Netherlands
[3] Lexmark Enterprise Software, Gooimeer 12, 1411DE Naarden, The Netherlands
{e.gonzalez,h.a.reijers,w.m.p.v.d.aalst}@tue.nl

Abstract. Process Mining techniques require event logs which, in many cases, are obtained from databases. Obtaining these event logs is not a trivial task and requires substantial domain knowledge. In addition, the result is a single view on the database in the form of a specific event log. If we desire to change our view, e.g. to focus on another business process, and generate another event log, it is necessary to go back to the source of data. This paper proposes a meta model to integrate both process and data perspectives, relating one to the other and allowing to generate different views from it at any moment in a highly flexible way. This approach decouples the data extraction from the application of analysis techniques, enabling its use in different contexts.

Keywords: *Process Mining*, *Database*, *Data schema*, *Meta model*, *Event extraction*.

1 Introduction

The field of process mining offers a wide variety of techniques to analyze event data. Process discovery, conformance and compliance checking, performance analysis, process monitoring and prediction, and operational support are some of the techniques that process mining provides in order to better understand and improve business processes. However, most of these techniques rely on the existence of an event log.

Anyone who has dealt with obtaining event logs in real-life scenarios knows that this is not a trivial task. It is not common to find logs exactly in the right form. In many occasions, they simply do not exist and need to be extracted from some sort of storage, like databases. In these situations, when a database exists, several approaches are available to extract events. The most general is the classical extraction in which events are manually obtained from the tables in the database. To do so, a lot of domain knowledge is required in order to select the right data. Some work has been done in this field to assist in the extraction and log generation task [2]. Also, studies have been performed on how to extract events in very specific environments like SAP [5, 6, 13] or other ERP systems [8]. A more general solution to extract events from databases, regardless of the application under study, is presented in [10], which describes how to automatically obtain events from database systems that generate *redo logs* as a way to recover from failure. The mentioned approaches aim at, eventually, generating an *event log*, i.e. a set of traces, each of them containing a set of *events*. These events represent operations or actions performed in the system under study, and are grouped in traces following some kind of criteria. However, there are multiple ways in which events can be selected and grouped

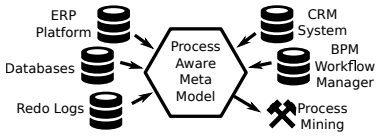

Fig. 1: Data gathering from several systems to a meta model

into traces. Depending on the perspective we want to have on the data, we need to extract event logs differently. Also, a database contains a lot more information than just events. The extraction of events and its representation as a plain event log can be seen as a "lossy" process during which valuable information can get lost. Considering the prevalence of databases as a source for event logs, it makes sense to gather as much information as possible, combining the process view with the actual data.

We see that process mining techniques grow more and more sophisticated. Yet, the most time-consuming activity, event log extraction, is hardly supported. This paper provides mature support to tackle the problem of obtaining, transforming, organizing and deriving data and process information from databases. This makes easier to connect the registration system of enterprises with analysis tools, generating different views on the data in a flexible way. Also, this work presents a comprehensive integration of process and data information in a consistent and unified format. All of this is formally supported with automated techniques. Moreover, the provided solution has the benefit of being universal, being applicable regardless of the specific system in use. Figure 1 depicts an environment in which the information is scattered over several systems from a different nature, like ERPs, CRMs, BPM managers, database systems, redo logs, etc. In such a heterogeneous environment, the goal is to extract, transform and derive data from all sources to a common representation that is able to connect all the pieces together such that analysis techniques like process mining can be readily applied.

The remainder of this paper is structured as follows: Section 2 presents a running example used through the paper. Section 3 explains the proposed meta model and the formalization. Implementation details are presented in Section 4. The approach is evaluated in Section 5, and Section 6 shows the related work. Finally, Section 7 presents the conclusions and future work.

2 Running Example

In this section we propose a running example to explain and illustrate our approach. Assume we want to analyze a setting where concerts are organized and concert tickets are sold. To do so, a database is used to store all the information related to concerts, concert halls (*hall*), seats, tickets, bands, performance of bands in concerts (*band_playing*), customers and bookings. Figure 2 shows the data schema of the database. In it we see many different elements of the involved process represented. Let us consider now a complex question that can be of interest from a business point of view: *What is the interaction with the process of customers between 18 and 25 years old who bought tickets for concerts of band X?* This question represents a challenge starting from the given database for several reasons:

1. The database does not provide an integrated view of process and data. Therefore, questions related to the execution of the underlying process cannot be answered with a query.
2. The current database schema fits the purpose of storing the information in this specific setting, but it does not have enough flexibility to extend its functionality allocating new kinds of data such as events or objects of a different nature.

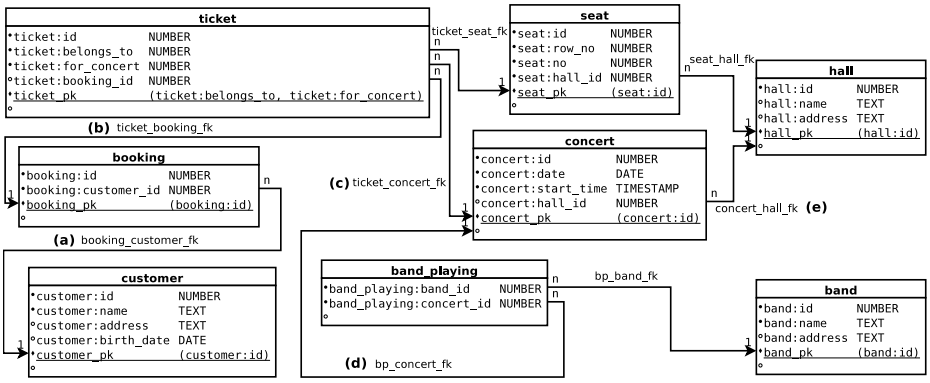

Fig. 2: Data schema of the example database

3. The setting lacks execution information in an accessible way (events, traces and logs are missing so one cannot apply process mining without a lot of extra work), and there is no assistance on how to extract or derive this information from the given data.
4. If we plan to use the data as it is, it requires to adapt to the way it is stored for every question we want to answer.

All these reasons make the analysis complex, if not impossible, in many settings. At best, such an analysis can only be carried out by either the extraction of a highly specialized event log or the creation of a complex ad hoc query.

3 Meta Model

As has been shown before, a need exists for a way to store execution information in a structured way, something that accepts data from different sources and allows to build further analysis techniques independently from the origin of this data. Efforts in this field have already been made as can be observed in [14] with the OpenXES standard. This standard defines structure to manage and manipulate logs, containing events and traces and the corresponding attributes. Therefore, XES is a good target format to represent behavior. However, a XES file is just one view on the data and, despite being an extensible format, it does not provide a predefined structure to store all the linked information we want to consider.

Because of this, it seems necessary to define a structured way to store additional information that can be linked to the classical event log. This new way to generalize and store information must provide sufficient details about the process, the data types and the relations between all the elements, making it possible to answer questions at the business level, while looking at two different perspectives: data and process.

3.1 Requirements

To be able to combine the data and process perspectives in a single structure, it is important to define a set of requirements that a meta model must fulfill. It seems reasonable to define requirements that consider backwards-compatibility with well established standards, support of additional information, its structure and the correlation between process and data views:

1. The meta model must be compatible with the current meta model of XES, i.e. any XES log can be transformed to the new meta model and back without loss of information,
2. It must be possible to store several logs in the new meta model, avoiding event duplication,
3. Logs stored in the same meta model can share events and belong to different processes,
4. It must be possible to store some notion of process in the meta model,
5. The meta model must allow to store additional information, like database objects, together with the events, traces and processes, and the correlation between all these elements,
6. The structure of additional data must be precisely modeled,
7. All information mentioned must be self contained in a single storage format, easy to share and exchange, similarly to the way that XES logs are handled.

The following section describes the proposed meta model which complies with these requirements, providing a formalization of the concepts along with explanations.

3.2 Formalization

Considering the typical environments subject to study in the process mining field, we can say that it is common to find systems backed up by some sort of database storage system. Regardless of the specific technology behind these databases, all of them have in common some kind of structure for data. We can describe our meta model as a way to integrate process and data perspectives, providing flexibility on its inspection and assistance to reconstruct the missing parts. Figure 3 shows a high level representation of the meta model. On the right hand side, the data perspective is considered, while the left models the process view. Assuming that the starting point of our approach is data, we see that the less abstract elements of the meta model, *events* and *versions*, are related, providing the connection between the process and data view. These are the basic blocks of the whole structure and, usually, the rest can be derived from them. However, in Section 5 we will see that, given enough information, we can also derive any of these two basic blocks from the other.

The data side considers three elements: **data model**, **objects** and **versions**. The data model provides a schema describing the objects of the database. The objects represent the unique entities of data that ever existed or will exist in our database, while the versions represent the specific values of the attributes of an object during a period of time. Versions represent the evolution of objects through time. The process side considers **events**, **instances** and **processes**. Processes describe the behavior of the system. Instances are traces of execution for a given process, being sets of events ordered through time. These events represent the most granular kind of execution data, denoting the occurrence of an activity or action at a certain point in time.

The remainder of this section proposes a formalization of the elements in this meta model, starting from the data and continuing with the process side. As has been mentioned before,

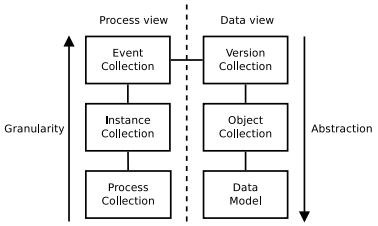

Fig. 3: Diagram of the meta model at a high level

we can assume a way to classify elements in types or *Classes* exists. Looking at our running example, we can distinguish between a *ticket* class and a *customer* class. This leads to the definition of data model as a way to describe the schema of our data.

Definition 1 (Data Model) *A data model is a tuple* $DM = (CL, AT, classOfAttribute, RS, sourceClass, targetClass)$ *such that*

- *CL is a set of class names,*
- *AT is a set of attribute names,*
- $classOfAttribute \in AT \rightarrow CL$ *is a function that maps each attribute to a class,*
- *RS is a set of relationship names,*
- $sourceClass \in RS \rightarrow CL$ *is a function that maps each relationship to its source class,*
- $targetClass \in RS \rightarrow CL$ *is a function that maps each relationship to its target class*

Each of these elements belonging to a *Class* represents a unique entity, something that can be differentiated from the other elements of the same class, e.g. *Customer A* and *Customer B*. We will call them *Objects*, being unique entities according to our meta model.

Definition 2 (Object Collection) *Assume OBJ to be the set of all possible objects. An object collection OC is a set of objects such that* $OC \subseteq OBJ$.

Something we know as well is that, during the execution of a process, the nature of these elements can change over time. Modifications can be made on the attributes of these *objects*. Each of these represents *mutations* of an object, modifying the values of some of its attributes, e.g. modifying the address of a customer. As a result, despite being the same object, we will be looking at a different *version* of it. The notion of *Object Version* is therefore introduced to show the different stages in the life-cycle of an *Object*.

During the execution of a process, operations will be performed and, many times, links between elements are established. These links allow to relate *Tickets* to *Concerts*, or *Customers* to *Bookings*, for example. These relationships are of a structured nature and usually exist at the data model level, being defined between *Classes*. Therefore, we know upfront that elements of the class *Ticket* can be related somehow to elements of the class *Concert*. *Relationships* is the name we use to call the definition of these links at the data model level. However, the actual instances of these *Relationships* appear at the *Object Version* level, connecting specific versions of objects during a specific period of time. These specific connections are called *Relations*.

Definition 3 (Version Collection) *Assume V to be some universe of values, TS a universe of timestamps and* $DM = (CL, AT, classOfAttribute, RS, sourceClass, targetClass)$ *a data model. A version collection is a tuple* $OVC = (OV, attValue, startTimestamp, endTimestamp, REL)$ *such that*

- *OV is a set of object versions,*
- $attValue \in (AT \times OV) \not\rightarrow V$ *is a function that maps a pair of object version and attribute to a value,*
- $startTimestamp \in OV \rightarrow TS$ *is a function that maps each object version to a start timestamp,*
- $endTimestamp \in OV \rightarrow TS$ *is a function that maps each object version to an end timestamp such that* $\forall ov \in OVC : endTimestamp(ov) \geq startTimestamp(ov)$,
- $REL \subseteq RS \times OV \times OV$ *is a set of triples relating pairs of object versions through a specific relationship.*

At this point, it is time to consider the *process* side of the meta model. The most basic piece of information we can find in a process event log is an event. These are defined by some attributes, among which we find a few typical ones like *timestamp*, *resource* or *lifecycle*.

Definition 4 (Event Collection) *Assume V to be some universe of values and TS a universe of timestamps. An event collection is a tuple* $EC = (EV, EVAT, eventAttributeValue, eventTimestamp, eventLifecycle, eventResource)$ *such that*

- EV *is a set of events,*
- $EVAT$ *is a set of event attribute names,*
- $eventAttributeValue \in EV \times EVAT \not\to V$ *is a function that maps a pair of an event and event attribute name to a value,*
- $eventTimestamp \in EV \to TS$ *is a function that maps each event to a timestamp,*
- $eventLifecycle \in EV \to \{start, complete, ...\}$ *is a function that maps each event to a value for its life-cycle attribute,*
- $eventResource \in EV \to V$ *is a function that maps each event to a value for its resource attribute.*

When we consider events of the same activity but relating to a different lifecycle, we gather them under the same *activity instance*. For example, two events that belong to the activity *make booking* could have different lifecycle values, being *start* the one denoting the beginning of the operation (first event) and *complete* the one denoting the finalization of the operation (second event). Therefore, both events belong to the same *activity instance*. Each of these activity instances can belong to different *cases* or *traces*. At the same time, *cases* can belong to different *logs*, that represent a whole set of *traces* on the behavior of a process.

Definition 5 (Instance Collection) *An instance collection is a tuple* $IC = (AI, CS, LG, aisOfCase, casesOfLog)$ *such that*

- AI *is a set of activity instances,*
- CS *is a set of cases,*
- LG *is a set of logs,*
- $aisOfCase \in CS \to \mathcal{P}(AI)$ *is a function that maps each case to a set of activity instances,*
- $casesOfLog \in LG \to \mathcal{P}(CS)$ *is a function that maps each log to a set of cases.*

The last piece of the our meta model is the *process model collection*. This part stores *process models* on an abstract level, i.e. as sets of *activities*. An *activity* can belong to different *processes* at the same time.

Definition 6 (Process Model Collection) *A process model collection is a tuple* $PMC = (PM, AC, actOfProc)$ *such that*

- PM *is a set of processes,*
- AC *is a set of activities,*
- $actOfProc \in PM \to \mathcal{P}(AC)$ *is a function that maps each process to a set of activities.*

Now we have all the pieces of our meta model, but it is still necessary to wire them together. A *connected meta model* defines the connections between these blocks. Therefore, we see that *versions* belong to *objects* (*objectOfVersion*) and *objects* belong to a class (*classOfObject*). In the same way, *events* belong to *activity instances* (*eventAI*), *activity instances* belong to *activities* (*activityOfAI*) and can belong to different *cases* (*aisOfCase*), *cases* to different *logs* (*casesOfLog*) and *logs* to process (*processOfLog*). Connecting both data and process views, we

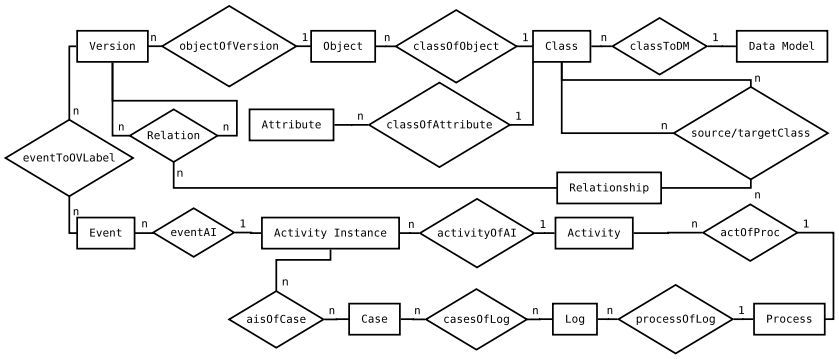

Fig. 4: ER diagram of the meta model

find *events* and *versions*. They are related (*eventToOVLabel*) in a way that can be interpreted as a causal relation between *events* and *versions*, i.e. when *events* happen they trigger the creation of *versions* as a result of modifications on data (the update of an attribute for instance). Another possibility is that the event represents a read access or query of the values of a *version*.

Definition 7 (Connected Meta Model) *Assume V to be some universe of values,* $DM = (CL, AT, classOfAttribute, RS, sourceClass, targetClass)$ *a data model, OC an object collection,* $OVC = (OV, attValue, startTimestamp, endTimestamp, REL)$ *a version collection,* $EC = (EV, EVAT, eventAttributeValue, eventTimestamp, eventLifecycle, eventResource)$ *an event collection,* $IC = (AI, CS, LG, aisOfCase, casesOfLog)$ *an instance collection and* $PMC = (PM, AC, actOfProc)$ *a process model collection. A connected meta model is a tuple* $CMM = (DM, OC, classOfObject, objectOfVersion, OVC, EC, eventToOVLabel, IC, eventAI, PMC, activityOfAI, processOfLog)$ *such that*

- $classOfObject \in OC \rightarrow CL$ *is a function that maps each object to a class,*
- $objectOfVersion \in OV \rightarrow OC$ *is a function that maps each object version to an object,*
- $eventToOVLabel \in EV \times OV \nrightarrow V$ *is a function mapping pairs of an event and an object version to a label. If a pair* $(ev,ov) \in domain(eventToOVLabel)$*, this means that both event and object version are linked. The label itself defines the nature of such link, e.g "insert", "update", "read", "delete", etc,*
- $eventAI \in EV \rightarrow AI$ *is a function that maps each event to an activity instance,*
- $activityOfAI \in AI \rightarrow AC$ *is a function that maps each activity instance to an activity,*
- $processOfLog \in LG \rightarrow PM$ *is a function that maps each log to a process.*

An instantiation of this meta model fulfills the requirements set in Section 3.1 in terms of storage of data and process view. Some characteristics of this meta model that enable full compatibility with the XES standard have been omitted in this formalization for the sake of brevity. In addition to this formalization, an implementation has been made. This was required in order to provide tools that assist in the exploration of the information contained within the meta model. More details on this implementation are explained in the following section.

4 Implementation

The library OpenSLEX, based on the meta model proposed in this work, has been implemented in Java[1]. This library provides an interface to insert data in the meta model, and to

[1] `http://www.win.tue.nl/~egonzale/projects/openslex/`

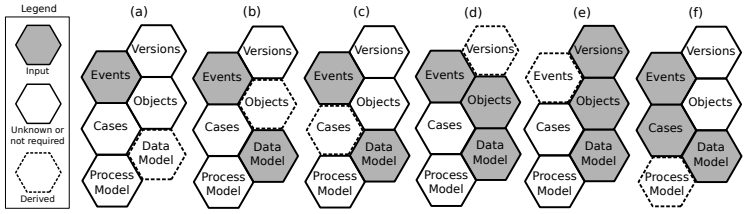

Fig. 5: Input scenarios to complete meta model elements

access it in a similar way to how XES Logs are managed by OpenXES [14]. However, under the hood it relies on SQL technology. Specifically, the meta model is stored in an SQLite[2] file. This provides some advantages like an SQL query engine, a standardized format as well as storage in self contained single data files that benefits its exchange and portability. Figure 4 shows an ER diagram of the internal structure of the meta model. However, it represents a simplified version to make it more understandable and easy to visualize. The complete class diagram of the meta model can be accessed in the tool's website[3]. In addition to the library mentioned earlier, an inspection tool[3] has been developed. This tool allows to explore the content of OpenSLEX files by means of a GUI in an exploratory fashion, which lets the user dig into the data and apply some basic filters on each element of the structure. The tool presents a series of blocks that contain the *activities*, *cases*, *activity instances*, *events*, *event attribute values*, *data model*, *objects*, *object versions*, *object version attribute values* and *relations* in the meta model. Some of the lists in the inspector (*cases*, *activity instances*, *events* and *objects*) have tabs that allow to filter the content they show. For instance, if the tab *"Per Activity"* in the *cases* list is clicked, only cases that contain events of such activity will be shown. In the same way, if the tab *"Per Case"* in the *events* list is clicked, only events contained in the selected case will be displayed. An additional block in the tool displays the attributes of the selected event.

5 Evaluation

The development of the meta model presented in this paper has been partly motivated by the need of a general way to capture the information contained in different systems combining the data and process view. These systems, usually backed by a database, use very different ways to internally store their data. Therefore, in order to extract this data, it is necessary to define a translation mechanism tailored to the wide variety of such environments. Because of this, *the evaluation aims at demonstrating the possibility of transforming information from different environments to the proposed meta model*. Specifically, three source environments are analyzed:

1. Database Redo Logs: files generated by the DBMS in order to maintain the consistency of the database in case of failure or rollback operations.
2. In-table version storage: Application-specific schema to store new versions of objects as a new row in each table.
3. SAP-style change table: changes on tables are recorded in a "redo log" style as a separate table, the way it is done in SAP systems.

The benefit of transforming the data to a common representation is that it allows for decoupling the application of techniques for the analysis from the sources of data. In addition,

[2] `http://www.sqlite.org/`
[3] `http://www.win.tue.nl/~egonzale/projects/meta-model/`

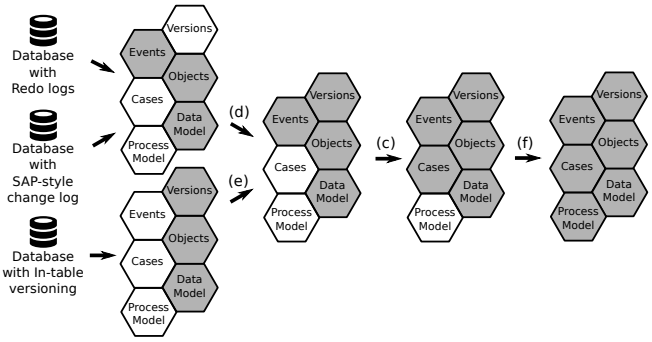

Fig. 6: Meta model completion in the three evaluated environments

a centralized representation allows to link data from different sources. However, the source of data may be incomplete. In this case, when transforming the data to fit in our meta model is not enough, we need to apply some inference techniques. This allows to derive the missing information and create a complete and fully integrated view.

The first part of this evaluation (Section 5.1) presents the different scenarios that we can find when transforming data. Each of these scenarios start from data that corresponds to different parts of the meta model. Then, it shows how to derive the missing elements from the given starting point. Sections 5.2, 5.3 and 5.4 analyze the three realistic environments mentioned before. We will demonstrate that data extraction is possible and that the meta model can be used to apply process mining instantly. Section 5.5 shows an example of the corresponding output meta model for the three environments.

5.1 Meta Model Completion Scenarios

It is rare to find an environment that explicitly provides the information to fill every *cell* of our meta model. This means that additional steps need to be taken to evolve from an incomplete meta model to a complete one. To do so, Figure 5 presents several scenarios in which, starting from a certain input, it is possible to infer other elements. Applying these steps consecutively will lead us, in the end, to a completely filled meta model:

a One of the most basic elements we require in our meta model to be able to infer other elements is the *event collection*. Starting from this input and applying schema, primary key and foreign key discovery techniques [12, 15], it is possible to obtain a data model describing the structure of the original data.
b The events, when combined with a data model, constitute one of the basic components and allows to infer objects. To do so, it is necessary to know the attributes of each class that identify the objects (primary keys). Finding the unique values for such attributes in the events corresponding to each class results in the list of unique objects of the meta model.
c Also, we can derive cases from the combination of events and a data model. The event splitting technique described in [10], which uses the transitive relations between events defined by the data model, allows to generate different sets of cases, or event logs.
d The events of each object can be processed to infer the *object versions* as results of the execution of each event. To do so, events must contain the values of the attributes of the object they relate to at a certain point in time or, at least, the values of the attributes that were affected (modified) by the event. Then, ordering the events by (ascending) timestamp allows to reconstruct the versions of each object.

Table 1: Fragment of a redo log: each line corresponds to the occurrence of an event

#	Time + Op + Table	Redo	Undo
1	2014-11-27 15:57:08.0 + INSERT+CUSTOMER	`insert into "SAMPLEDB". "CUSTOMER" ("ID", "NAME", "ADDRESS", "BIRTH_DATE") values ('17299', 'Name1', 'Address1', TO_DATE( '01-AUG-06', 'DD-MON-RR'));`	`delete from "SAMPLEDB". "CUSTOMER" where "ID" = '17299' and "NAME" = 'Name1' and "ADDRESS" = 'Address1' and "BIRTH_DATE" = TO_DATE( '01-AUG-06', 'DD-MON-RR') and ROWID = '1';`
2	2014-11-27 16:07:02.0 + UPDATE+CUSTOMER	`update "SAMPLEDB". "CUSTOMER" set "NAME" = 'Name2' where "NAME" = 'Name1' and ROWID = '1';`	`update "SAMPLEDB". "CUSTOMER" set "NAME" = 'Name1' where "NAME" = 'Name2' and ROWID = '1';`
3	2014-11-27 16:07:16.0 + INSERT+BOOKING	`insert into "SAMPLEDB". "BOOKING" ("ID", "CUSTOMER_ID") values ('36846', '17299');`	`delete from "SAMPLEDB". "BOOKING" where "ID" = '36846' and "CUSTOMER_ID" = '17299' and ROWID = '2';`
4	2014-11-27 16:07:16.0 + UPDATE+TICKET	`update "SAMPLEDB". "TICKET" set "BOOKING_ID" = '36846' where "BOOKING_ID" IS NULL and ROWID = '3';`	`update "SAMPLEDB". "TICKET" set "BOOKING_ID" = NULL where "BOOKING_ID" = '36846' and ROWID = '3';`
5	2014-11-27 16:07:17.0 + INSERT+BOOKING	`insert into "SAMPLEDB". "BOOKING" ("ID", "CUSTOMER_ID") values ('36876', '17299');`	`delete from "SAMPLEDB". "BOOKING" where "ID" = '36876' and "CUSTOMER_ID" = '17299' and ROWID = '4';`
6	2014-11-27 16:07:17.0 + TICKET+UPDATE	`update "SAMPLEDB". "TICKET" set "ID" = '36876' where "BOOKING_ID" IS NULL and ROWID = '5';`	`update "SAMPLEDB". "TICKET" set "ID" = NULL where "BOOKING_ID" = '36876' and ROWID = '5';`

e The inverse of scenario *d* is the one in which events are inferred from object versions. Looking at the attributes that differ between consecutive versions it is possible to create the corresponding event for the modification.

f Finally, a set of cases is required to discover a process model using any of the multiple miners available in the process mining field.

The following three sections analyze realistic environments and relate them to these scenarios to demonstrate that the complete meta model structure can be derived in each of them. The goal is to create an integrated view of data and process, specially when event logs are not directly available.

5.2 Database Redo Logs

The first environment focuses on database *redo logs*, a mechanism present in many DBMSs to guarantee consistency, as well as providing additional features such as rollback, point-in-time recovery, etc. Redo logs have already been considered in previous works [1, 10] as a source of event data for process mining. Table 1 shows an example of a redo log obtained from an Oracle DBMS. After its processing, explained in [10], these records are transformed into events.

Figure 6 shows a general overview of how the meta model elements are completed according to the starting input data and the steps taken to derive the missing ones. In this case, the analysis of database redo logs allows to obtain a set of events, together with the objects they belong to and the data model of the database. These elements alone are not sufficient to do process mining without the existence of an event log (a set of traces). In addition, the versions of the objects of the database need to be inferred from the events as well.

Fortunately, a technique to build logs using different perspectives (Trace ID Patterns) is presented in [10]. The existence or definition of a data model is required for this technique to work. Figure 6 shows a diagram of the data transformation performed by the technique, and how it fits in the proposed meta model structure. The data model is automatically extracted from the database schema and is the one included in the meta model. This data model, together

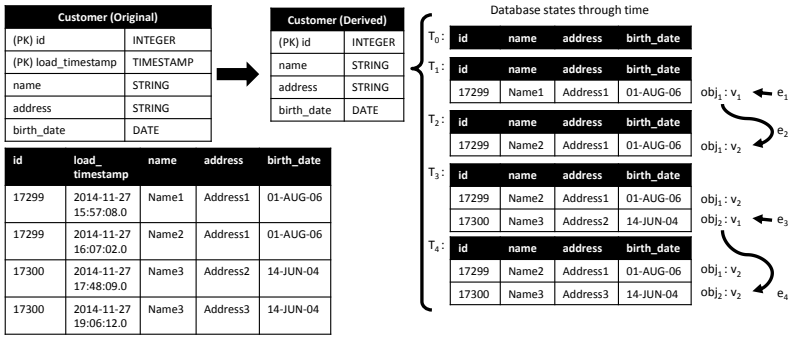

Fig. 7: Example of in-table versioning and its transformation into objects and versions

with the extracted events, allows to generate both cases (c) and object versions (d). Then, process discovery completes the meta model with a process (f). Once the meta model structure is filled with data, we can make queries on it taking advantage of the established connections between all the elements and apply process mining to do the analysis.

5.3 In-table Versioning

It is not always possible to get redo logs from databases. Sometimes they are disabled or not supported by the DBMS. Also, we simply may not be able to obtain credentials to access them. Whatever the reason, we often face a situation in which events are not explicitly stored. This enormously limits the analysis that can be performed on the data. The challenge in this environment is to obtain, somehow, an event log to complete our data.

It can be the case that we encounter an environment such that, despite of lacking events, versioning of objects is kept in the database, i.e. it is possible to retrieve the old value for any attribute of an object at a certain point in time. This is achieved by means of duplication of the modified versions of rows. The table at the bottom left corner of Figure 7 shows an example of an in-table versioning of objects. We see that the primary key is formed by the fields *id* and *load_timestamp*. Each row represents a version of an object and every new reference to the same *id* at a later *load_timestamp* represents an update. Therefore, if we order rows (ascending) by *id* and *load_timestamp*, we get sets of versions for each object. The first one (with older *load_timestamp*) represents an insertion, and the rest updates on the values.

Looking at Figure 7 it is clear that, ordering by timestamp the versions in the original set (bottom left), we can reconstruct the different states of the database (right). Each new row in the original table represents a change in the state of the database. Performing this process for all the tables, allows to infer the events in a setting where they were not explicitly stored. Figure 6 shows that, thanks to the meta model proposed it is possible to derive events starting from a data model, a set of objects, and their versions as input (Figure 5.e). The next step is to obtain cases from the events and data model applying the technique from [10] to split event collections into cases selecting an appropriate *Trace ID Pattern* (scenario *c*). Finally, process discovery will allow us to obtain a process model to complete the meta model structure (scenario *f*).

As a result of the whole procedure, we have a meta model completely filled with data (original and derived) that enables any kind of analysis available nowadays in the process mining field. Moreover, it allows for extended analysis combining data and process perspectives.

CDHDR	
OBJECTCLAS	STRING
CHANGENR	INTEGER
OBJECTID	INTEGER
USERNAME	STRING
UDATE	DATE
UTIME	TIME
TCODE	INTEGER

CDPOS	
OBJECTCLAS	STRING
CHANGENR	INTEGER
TABNAME	STRING
TABKEY	STRING
FNAME	STRING
VALUE_NEW	STRING
VALUE_OLD	STRING

OBJECTCLAS	CHANGENR	OBJECTID	USERNAME	UDATE	UTIME	TCODE
CUST	0001	0000001	USER1	2014-11-27	15:57:08.0	0001
CUST	0002	0000001	USER2	2014-11-27	16:07:02.0	0002
CUST	0003	0000002	USER2	2014-11-27	17:48:09.0	0003
CUST	0004	0000002	USER1	2014-11-27	19:06:12.0	0004
...	...	...	...	...	...	...

OBJECTCLAS	CHANGENR	TABNAME	TABKEY	FNAME	VALUE_NEW	VALUE_OLD
CUST	0001	CUSTOMER	17299	name	Name1	
CUST	0001	CUSTOMER	17299	address	Address1	
CUST	0001	CUSTOMER	17299	birth_date	01-AUG-06	
CUST	0002	CUSTOMER	17299	name	Name2	Name1
CUST	0003	CUSTOMER	17300	name	Name3	
CUST	0003	CUSTOMER	17300	address	Address2	
CUST	0003	CUSTOMER	17300	birth_date	14-JUN-04	
CUST	0004	CUSTOMER	17300	address	Address3	Address2
...	...	...	...	...	...	...

Customer	
(PK) id	INTEGER
name	STRING
address	STRING
birth_date	DATE

Fig. 8: Example of SAP change tables CDHDR and CDPOS

5.4 SAP-style Change Table

The last environment we will consider is related to very widespread ERP systems such as SAP. These systems provide a huge amount of functionalities to companies by means of configurable modules. They can run on various platforms and rely on databases to store all their information. However, in order to make them as flexible as possible, the implementation tries to be independent of the specific storage technology running underneath. We see SAP systems running on MSSQL, Oracle or other technologies but they do not make intensive use of the features that the database vendor provides. Therefore, data relations are often not defined in the database schema, but managed at the application level. This makes the life of the analyst who would be interested in obtaining event logs a bit more complicated. Fortunately, SAP implements its own redo log like mechanism to store changes in data, and it represents a valid source of data for our purposes.In this setting we lack event logs, object versions, a complete data model and processes. Without some of these elements, performing any kind of process mining analysis becomes very complicated, if not impossible. For instance, the lack of an event log does not allow for the discovery of a process and, without it, performance or conformance analysis are not possible. To overcome this problem, we need to infer the lacking elements from the available information in the SAP database.

First, it must be noted that, despite the absence of an explicitly defined data model, SAP uses a consistent naming system for their tables and columns, and there is lots of documentation available that describes the data model of the whole SAP table landscape. On the other hand, to extract the events we need to process the change log. This SAP-style change log, as can be observed in Figure 8, is based on two change tables: *CDHDR* and *CDPOS*. The first table (*CDHDR*) stores one entry per change performed on the data with a unique change id (*CHANGENR*) and other additional details. The second table (*CDPOS*) stores one entry per field changed. Several fields in a data object can be changed at the same time and will share the same *CHANGENR*. For each field changed, the table name is recorded (*TABNAME*) together with the field name (*FNAME*), the key of the row affected by the change (*TABKEY*) and the old and new values of the field (*VALUE_OLD*, *VALUE_NEW*).

As can be seen in Figure 6, after processing the change log and providing an SAP data model, we are in a situation in which the events, objects and data model are known. Then, we can infer the versions of each object (d), split the events in cases (c) and finally, discover a process model (f). With all these ingredients it becomes possible to perform any process mining analysis and answer complex questions combining process and data perspectives.

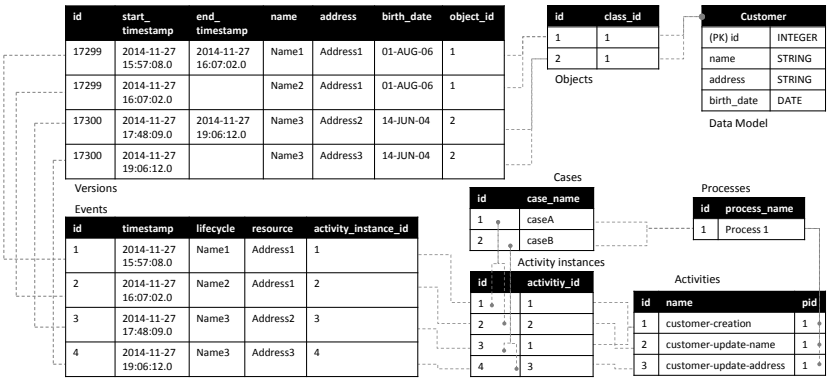

Fig. 9: Sumarized example of resulting meta model

5.5 Resulting Meta Model

In the three previous sections we have explored different environments in which, from a given starting input, we can derive the missing blocks to completely fill our meta model. The three environments are different, but based on the running example presented in Section 2. Therefore, we assume that the same information can be accessed in all three of them. As a result, the resulting meta model will contain the same information, but different parts have been derived through a different procedure depending on the starting input data. Figure 9 provides a simplified view on the final content of the meta model for any of the three environments. However, a full meta model considering all the tables presented in the running example is available online[4] and can be explored using our tool shown in Section 4. Now we have all the information we need in a centralized storage. However, it cannot be used yet to do process mining analysis. Most of these techniques require an event log in XES format. Fortunately, our meta model allows full compatibility with this target format. In addition, our meta model has the advantage of allowing to generate many different event logs depending on the view we want to obtain of our data. Let us reconsider the question proposed in Section 2: *What is the interaction with the process of customers between 18 and 25 years old who bought tickets for concerts of band X?*

In order to answer this question, we need to make a proper selection on the data and transform it to a XES event log. The OpenSLEX library provides the functionality required to programmatically select the desired view of the data in our meta model and instantly export it to XES. The great advantage is that, regardless of the source environment from which we obtained the data, the structure in which to make this selection is always the same. Figure 10 shows the result of mining the selected event log. We see that customers make two kind of operations: ticket booking and customer updates. In order to buy a ticket, customers make a booking (*BOOKING+INSERT*) and, afterward, the corresponding ticket is updated (*TICKET+UPDATE*). This means that the ticket is assigned to the booking to avoid it from being sold twice. The other operation, customer updates (*CUSTOMER+UPDATE*), is performed quite often and it corresponds with changes in the details of the customer, e.g. a change of address. The dashed lines denote deviations of the log respect to the discovered model. In this image we see that, in 5 cases, the *TICKET+UPDATE* operation was skipped after executing *BOOKING+INSERT*. This could mean that the booking process does not finish or it is empty. Therefore, no ticket is ever booked. Also, we see some deviations represented by a dashed

[4] http://www.win.tue.nl/~egonzale/projects/meta-model/mm-01.zip

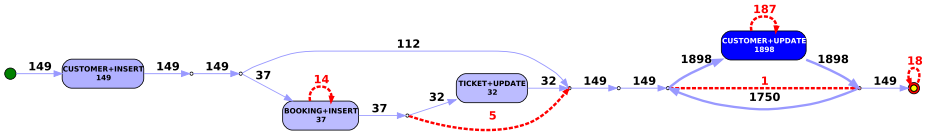

Fig. 10: Process model of the selected events

loop line on top of an activity or place, for example the one in activity *BOOKING+INSERT* that reads a counter with the value 14. This means that a certain amount of "move on log" operations took place, i.e. an event happened in the log that cannot be aligned with the model.

At this point we have shown that, from the proposed meta model, it is possible to generate views on the data. Then, exporting them to the target XES format allows to perform process mining analysis to get more insights on the behavior of the system under study.

6 Related Work

Several efforts have been made trying to provide structure to the representation of execution data in the information systems field. Workflow management systems are an example of environment in which the community has focused on providing models to describe their functioning and allow for the analysis of their behavior. Papers like [9, 11] provide meta models to give structure to audit trails on workflows. However, they focus mainly on the workflow or process perspective. Process mining has different needs and the desire to store event data in a unified way is obvious. In [3], the authors provide a meta model to define event logs, which would evolve later in the OpenXES format [14]. This format represents a great achievement from the point of view of standardization, and allows to exchange logs and develop mining techniques assuming a common representation of the data. The XES format is, in fact, a target format from our perspective and not a source of information, i.e. we aim at, from a richer source, generate different views on data in XES format to enable process mining.

The main flaw of these approaches resides in the way they force the representation of complex systems by means of a flat event log. The data perspective is missing, only allowing to add attributes at the event, trace or log level. More recent works try to improve the situation, analyzing data dependencies [7] in business models with the purpose of improving them, or even observing changes on object states to improve their analysis [4]. However, none of the existing approaches provides a generic and standard way of gathering, classifying, storing and connecting process and data perspectives on information systems, specially when dealing with databases where the concept of structured process can be fuzzy or nonexistent.

7 Conclusion

In this paper, a meta model has been proposed that provides a bigger picture of the reality of business information systems. This meta model aligns the data and process perspectives and enables the application of existing process mining techniques (at the same time that unleashes a new way to query data and historical information). This is possible thanks to the combination of data and process perspectives on the analysis of information systems. The applicability of the technique has been demonstrated by means of the analysis of several real-life environments. Also, an implementation of the proposed solution has been developed and tested. However, from the authors' point of view, the main contribution of this work is not only on

what new features it enables, but on the universality of the proposed solution. Its applicability to so many different environments provides a common ground to separate data extraction and analysis as different problems, creating an interface that is much richer and powerful than the current existing standards. Several challenges remain open. For instance, a system that enhances the query building experience, allowing for a more natural and user friendly way is desirable. Also, mechanisms to exploit the benefits of the process part of the structure when combined to the data will make the solution really beneficial in comparison to the regular queries that can be performed in the source database systems. In addition, the development of techniques to incorporate data from more varied sources and systems will certainly make the proposed meta model a real candidate for extended use and standardization.

References

1. van der Aalst, W.M.P.: Extracting event data from databases to unleash process mining. In: vom Brocke, J., Schmiedel, T. (eds.) BPM - Driving Innovation in a Digital World, pp. 105–128. Management for Professionals, Springer International Publishing (2015)
2. Buijs, J.: Mapping Data Sources to XES in a Generic Way. Master's thesis, Technische Universiteit Eindhoven, The Netherlands (2010)
3. van Dongen, B.F., van der Aalst, W.M.P.: A meta model for process mining data. EMOI-INTEROP 160, 30 (2005)
4. Herzberg, N., Meyer, A., Weske, M.: Improving business process intelligence by observing object state transitions. Data & Knowledge Engineering 98, 144–164 (2015)
5. Ingvaldsen, J.E., Gulla, J.A.: Preprocessing support for large scale process mining of SAP transactions. In: Business Process Management Workshops. pp. 30–41. Springer (2008)
6. Mahendrawathi, E., Astuti, H.M., Wardhani, I.R.K.: Material movement analysis for warehouse business process improvement with process mining: A case study. In: Asia Pacific Business Process Management, pp. 115–127. Springer (2015)
7. Meyer, A., Pufahl, L., Fahland, D., Weske, M.: Modeling and enacting complex data dependencies in business processes. In: Proceedings of the 11th international conference on Business Process Management. pp. 171–186. Springer-Verlag (2013)
8. Mueller-Wickop, N., Schultz, M.: ERP event log preprocessing: Timestamps vs. accounting logic. In: Design Science at the Intersection of Physical and Virtual Design, Lecture Notes in Computer Science, vol. 7939, pp. 105–119. Springer Berlin Heidelberg (2013)
9. zur Muhlen, M.: Evaluation of workflow management systems using meta models. In: Systems Sciences. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on (1999)
10. González-López de Murillas, E., van der Aalst, W.M.P., Reijers, H.A.: Process mining on databases: Unearthing historical data from redo logs. In: Business Process Management. Springer (2015)
11. Rosemann, M., Zur Muehlen, M.: Evaluation of workflow management systems-a meta model approach. Australian Journal of Information Systems 6(1) (1998)
12. Sismanis, Y., Brown, P., Haas, P.J., Reinwald, B.: Gordian: efficient and scalable discovery of composite keys. In: Proceedings of the 32nd international conference on Very large data bases. pp. 691–702. VLDB Endowment (2006)
13. Štolfa, J., Kopka, M., Štolfa, S., Koběrskỳ, O., Snášel, V.: An application of process mining to invoice verification process in sap. In: Innovations in Bio-inspired Computing and Applications, pp. 61–74. Springer (2014)
14. Verbeek, H., Buijs, J.C., Van Dongen, B.F., van der Aalst, W.M.P.: XES, XESame, and ProM 6. In: Information Systems Evolution, pp. 60–75. Springer (2011)
15. Zhang, M., Hadjieleftheriou, M., Ooi, B.C., Procopiuc, C.M., Srivastava, D.: On multi-column foreign key discovery. Proceedings of the VLDB Endowment 3(1-2), 805–814 (2010)