Automatic Discovery of Data-Centric and Artifact-Centric Processes

Erik H.J. Nooijen, Boudewijn F. van Dongen, and Dirk Fahland

Eindhoven University of Technology
Eindhoven, The Netherlands
enooijen@gmail.com, {b.f.v.dongen,d.fahland}@tue.nl

Abstract. Process discovery is a technique that allows for automatically discovering a process model from recorded executions of a process as it happens in reality. This technique has successfully been applied for classical processes where one process execution is constituted by a single case with a unique case identifier. Data-centric and artifact-centric systems such as ERP systems violate this assumption. Here a process execution is driven by process data having various notions of interrelated identifiers that distinguish the various interrelated data objects of the process. Classical process mining techniques fail in this setting. This paper presents an automatic technique for discovering for each notion of data object in the process a separate process model that describes the evolution of this object, also known as artifact life-cycle model. Given a relational database that stores process execution information of a data-centric system, the technique extracts event information, case identifiers and their interrelations, discovers the central process data objects and their associated events, and decomposes the data source into multiple logs, each describing the cases of a separate data object. Then classical process discovery techniques can be applied to obtain a process model for each object. The technique is implemented and has been evaluated on the production ERP system of a large retailer.

Keywords: artifact, process discovery, ERP system, event log.

1 Introduction

Process discovery is a technique for automatically discovering a process model from recorded executions of the process. The technique is successfully applied for classical processes where each process execution is recorded as a *case* (the sequence of its events) in an *event log*. Each event of the process is related to exactly one *case* by a *case id*. [1]

However, when looking at the data models of ERP products such as SAP Business Suite, Microsoft Dynamics AX, Oracle E-Business Suite, Exact Globe, Infor ERP, and Oracle JD Edwards EnterpriseOne, one can easily see that this assumption is *not* valid for real-life processes, which are *data-centric*. There are one-to-many and many-to-many relationships between data *objects*, such as customers, orderlines, orders, deliveries, payments, etc., and a single event can relate to and update several objects. Such systems do not have a unique notion of a *process instance* by which we can trace and isolate its executions, and process discovery fails.

M. La Rosa and P. Soffer (Eds.): BPM 2012 Workshops, LNBIP 132, pp. 316–327, 2013.
© Springer-Verlag Berlin Heidelberg 2013

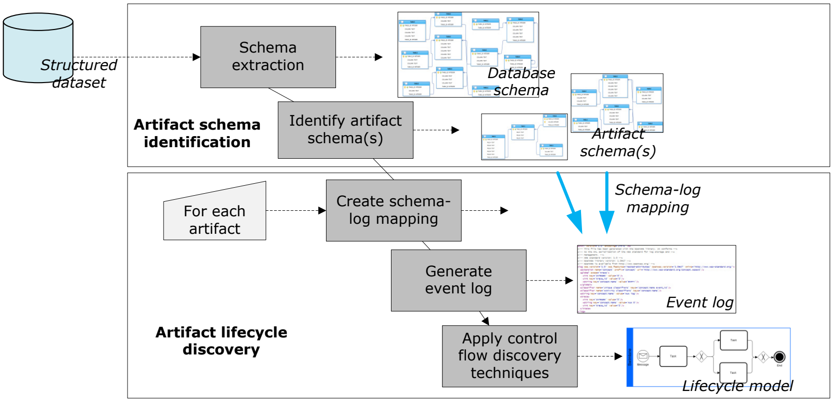
Structured dataset
Schema extraction
Database schema
Artifact schema identification
Identify artifact schema(s)
Artifact schema(s)
Schema-log mapping
For each artifact
Create schema-log mapping
Generate event log
Event log
Artifact lifecycle discovery
Apply control flow discovery techniques
Lifecycle model

Fig. 1. Approach to Artifact Discovery

Previous approaches to use this data for process discovery particularly failed to separate events related to various objects; in particular analyzing what was part of the process was hard and time consuming [11, 15, 16]. The *artifact-centric approach* [7] provides an appropriate conceptual lense: the entire process is seen as a set of interacting business entities called *artifacts*. Each of these artifacts can be described by an information schema (called an artifact schema) and a non-trivial lifecycle describing how the artifact evolves through a process execution [9, 10].

Process discovery for artifact-centric processes is an unsolved problem. The problem reads as follow. We assume a structured data source $\boldsymbol{R}$ to be given that contains information about the events that have occurred in past process executions, usually in the form of *timestamps* written in the records of $\boldsymbol{R}$. For example, we found production databases of ERP systems to satisfy this assumption. From this data source, we want to discover (1) the artifacts (i.e., business entities) of the process, (2) their information model (i.e., artifact schema), and (3) the life-cycle model of each artifact.

In this paper we present a first automatic technique for discovering artifact-centric processes from a structured data source $\boldsymbol{R}$. Our technique, illustrated in Fig. 1 reuses a number of existing techniques and fills in a crucial missing gap to solve the problem. For the given structured data source $\boldsymbol{R}$, first the schema $S_{\boldsymbol{R}}$ including column types and primary and foreign keys is rediscovered (this may be necessary as the documented schema of $\boldsymbol{R}$ can be incomplete regarding its actual contents, e.g., non-documented foreign key relations). Then the schema $S_{\boldsymbol{R}}$ is partitioned into artifact schemas $A_1, \ldots, A_k$ using weighted k-means clustering, where k is a parameter chosen by the user. Each artifact schema A_i describes the information model of one artifact and consists of all tables that contain relevant information about A_i. We then extract from $\boldsymbol{R}$ a log that describes all instances of A_i and their evolution over time. For this, we extract from artifact schema A_i automatically an event specification (called *schema-log mapping*) in terms of attributes of A_i. This event type specification is then used to construct database queries which extract from $\boldsymbol{R}$ all events with their attributes, group them into cases, order them by time stamp, and write the result into a classical log L_i of A_i. Each case of L_i is a sequence of events related to the same case id of A_i, which satisfies the assumptions for classical processes. Then a classical process discovery algorithm can be used to obtain a life-cycle model for artifact A_i.

In the following, we first discuss related work to extracting event information from data sources in Sect. 2. Sect. 3 presents the techniques to discover artifact schemas from a relational data source. Sect. 4 presents the main technical contribution of this paper: to extract event specifications from artifact schemas which is then used for actual log extraction and life-cycle discovery. We report on experimental results in Sect. 5, and conclude and discuss open problems in Sect. 6.

2 Related Work

In principle, an artifact-centric model of a running system could be obtained through interviews of process stakeholders [5], by first looking at what data is important and then investigating how the process operates on the data [12]. Besides being time-consuming, this approach suffers from the fact that interviews will reveal how people think the process should run rather than how it is actually run [1]. Thus, an automated approach to discovering artifact models from process data sources is preferable.

Over 40 algorithms were developed for control flow discovery given an event log [17]. The authors of [17] provide an overview of these algorithms and describe when each algorithm can be applied successfully. The process analysis approach described in [6] explains how these techniques should be applied and which points of attentions should be taken into account to improve the quality of the results.

All process discovery techniques assume an *event log* to be given as input which consists of a set of *cases* being sequences of *events*; events of one case relate to the same *case identifiers* [1].

Related work on event log extraction can be separated into support for event log extraction in general and case studies on event log extraction from specific ERP systems. The most recent generic approach to event log extraction is XESame [18]. In this, as in all known generic approaches, first a mapping between source data and event log needs to be defined *manually*. Then an algorithm extracts events, sorts them into traces, and writes traces into a log. In case of XESame the mapping is translated to SQL queries on a database which returns the events of the log.

The only ERP systems for which event log extraction was studied were SAP [11,15] and PeopleSoft [16]. A variety of approaches were tested to extract event logs from these systems. The underlying assumption of these approaches was the existence of a unique case identifier. As ERP systems in general provide multiple case identifiers, the majority of these approaches failed; success could only be reported when database tables were carefully selected by hand. The artifact-centric approach of multiple interrelated artifacts [7] sheds a better view on data in ERP systems which we exploit in the following.

3 Discovering Artifact Schemas

In this paper we want to solve the problem of discovering an artifact-centric process model from a structured data source, i.e., a relational database $\boldsymbol{R}$. In light of existing work discussed in Sect. 2, we need (1) a technique to *automatically* identify the relevant case identifiers in $\boldsymbol{R}$, each case identifier then gives rise to an artifact (a business object with a life-cycle); (2) a technique to *automatically* extract from $\boldsymbol{R}$ an event log for each artifact, preferably by leveraging a generic approach to event log extraction; (3) discover artifact life-cycle models for each log.

We solve the first problem in this section by applying a number of existing data mining techniques. We solve log extraction by leveraging the generic approach of [18] in Sect. 4. Finally, artifact life-cycle discovery is solved by applying any classical process discovery algorithm [1, 17] on the extracted log. Fig. 1 shows the overall approach that was already outlined in the introduction; in the following we present the details.

3.1 Assumed Input and Schema Extraction

We assume a relational database $\boldsymbol{R}$ (e.g., an ERP system's database) to be given as input for the discovery. We assume that $\boldsymbol{R}$ recorded its state evolved over time in *timestamp attributes*, for instance, important updates of a record were logged in a separate attribute. We found this to be a feasible assumption for many ERP systems in practice. If $\boldsymbol{R}$ has no historic information, then process discovery is infeasible, however, one could use trigger mechanisms of active databases to log updates of $\boldsymbol{R}$ in a generic way.

To identify the relevant case identifers in $\boldsymbol{R}$ and corresponding artifact schemas, the schema information of $\boldsymbol{R}$ needs to be *complete*: each column has to have *type*, each table needs a *primary key* and functional dependencies between tables need to be documented as *foreign key* relations. However in reality, schema information in ERP system databases is often incomplete [15, 16]; typically due to data dependencies created at the application layer that are not documented in $\boldsymbol{R}$. Thus, *schema extraction* techniques may be required to reconstruct the database schema. Various techniques exist to group columns into a number of attributes with the same domain [3, 22], to rediscover primary keys of a table [2] and to rediscover foreign keys between tables [4, 13, 21]. A particular focus has to be put on identifying *timestamp* attributes of $\boldsymbol{R}$ as these document events of the process. A detailed comparison is given in [14].

The result of schema extraction is a relational schema $S = (\boldsymbol{T}, \boldsymbol{F}, \boldsymbol{D}, dom)$ of the database with a set $\boldsymbol{T}$ of table schemas and a set $\boldsymbol{F}$ of foreign keys. Each table schema $T = (\boldsymbol{C}_T, \boldsymbol{C}_p) \in \boldsymbol{T}$ contains a set $\boldsymbol{C}_T$ of columns and a primary key $\boldsymbol{C}_p \subseteq \boldsymbol{C}_T$; let $\boldsymbol{C}$ denote all columns of S. Each $F = (T_p, \boldsymbol{C}_p, T_c, \boldsymbol{C}_c) \in \boldsymbol{F}$ is a foreign key from parent table T_p with primary key $\boldsymbol{C}_p$ to child table T_c with referencing columns $\boldsymbol{C}_c$. Function $dom : \boldsymbol{C} \to D$ assigns each column a domain from the set $\boldsymbol{D}$ of domains.

Note that schema extraction techniques discover data dependencies of the application that are not documented in the database [15, 16]. For instance when a column C in $\boldsymbol{R}$ is used as a unique index by the application, but not declared as primary key in $\boldsymbol{R}$, schema extraction will identify C as primary key. Correspondingly, undocumented foreign key relations (used in the application, but not declared as such) are identified.

3.2 Discovering Artifact Schemas

Database $\boldsymbol{R}$ and its schema S contain all process data as a whole. The idea of artifacts is to decompose this data into the business objects, or *artifacts*, of the process. Each artifact instance has a unique identifier and follows a *life-cycle* describing how attributes of the artifact change as the process evolves. An *artifact schema* describes the data model of an artifact in terms of $\boldsymbol{R}$. Technically, an artifact schema $A = (\boldsymbol{T}, \boldsymbol{F}, \boldsymbol{D}, dom, T^m)$ is a relational schema $(\boldsymbol{T}, \boldsymbol{F}, \boldsymbol{D}, dom)$ that distinguishes a main table $T^m \in \boldsymbol{T}$ of the artifact, the primary key of the main table is the identifier of the artifact. All other tables in A define additional attributes of the artifact.

In principle, one could identify artifact schemas from S through interviews [12]. In the following, we identify artifact schemas $A_1, \ldots, A_k$ from S *automatically* through *clustering*. The idea is that business objects (and in particular their identifiers) materialize in $\boldsymbol{R}$ as somehow "important" tables. Attributes of these artifacts are materialized in "auxiliary" tables related to the "important" tables, thus, the tables that constitute an artifact form a cluster of corresponding tables. Such clusters of tables can be identified using standard *schema summarization techniques* [19,20].

Schema summarization first defines a *distance* between any two tables in S (based on the actual records in the tables of $\boldsymbol{R}$). The distance function between tables incorporate two factors: importance and foreign key relations.

1. Importance of a table is defined by its entropy (the more unique records a table has, the more important it is). The higher the importance of two tables is, the farther they are away (each important table defines a business object, two business objects should be represented separately).
2. Foreign key relations between tables associate auxiliary tables to important tables. Here, a child table is closer to a parent table if there are more records in the parent table relating to the child table. Various definitions are possible [14].

The concrete definition of the distance function between two tables based on their records is omitted here for space limitations, see [14] for details. Based on this distance function the tables of S are clustered into k clusters (for a user-chosen number k) using *weighted k-means clustering*. In the clusters, all tables of one cluster are closer to each other than to any table in another cluster. Experience has shown that in each cluster, a unique *main table* T^m with the least distance to all other tables in the cluster exists [19,20]. Thus, the clustering returns a set $\{A_1, ..., A_k\}$ of artifact schemas which solves the first problem: to automatically discover artifact schemas from a structured datasource.

The parameter k determines how many artifacts shall be returned. If k is chosen wrongly, say $k = 1$, the artifacts will have an unnatural shape. We found an iterative approach of gradually increasing k until the rightly shaped artifacts appear, to be feasible. Finding the right number k based on $\boldsymbol{R}$ alone might require domain knowledge or more sophisticated technique, which we consider as further work.

4 Extracting Logs and Discovering Life-Cycles

Having discovered artifact schemas $A_1, ..., A_k$ from $\boldsymbol{R}$, the next step is to extract for each artifact A_i a log L_i describing the life-cycle of A_i. The artifact schema A_i contains all structural information of this artifact, including timestamp attributes that record when an instance of A_i changed its state. The actual information is stored in $\boldsymbol{R}$ and has to be extracted.

For this we define a *schema-log mapping* that defines (1) a set of *event types* identified in A_i, and (2) a mapping from the attributes and tables of A_i to these event types. This mapping can then be used to construct database queries which extract the actual events from $\boldsymbol{R}$. In the following, we first present a *automatic* approach to *discover* a schema-log mapping from A_i, and then discuss the log extraction based on this mapping.

4.1 Automatically Discovering a Schema-Log Mapping

Schema-to-log mappings can be identified automatically by a four-step approach based on timestamps and foreign key relations, which are available by the techniques of Sect. 3. The first step of the approach is to identify event (type) columns based on their domain: Exactly one event should be created for each value in one of these columns. The remaining columns are then assigned as attributes to either the artifact instances or to one or more event types. The event type and attribute information is used to create event mappings. Finally these event mappings and artifact instance attribute information are combined to create a trace mapping. The result of the algorithm is an event log trace mapping $TM = (\boldsymbol{C}_{\mathrm{TID}}, T_{\mathrm{from}}, \boldsymbol{F}_{link}, \boldsymbol{EM}, \boldsymbol{AM}_{\mathrm{T}}, \boldsymbol{LA}_{\mathrm{T}})$ with:

- columns $\boldsymbol{C}_{\mathrm{TID}}$ identify the different traces,
- main table T_{from}, links to other tables of the artifact in the form of foreign keys $\boldsymbol{F}_{\mathrm{link}}$,
- a set of event mappings $\boldsymbol{EM}$, a set of attribute mappings $\boldsymbol{AM}_{\mathrm{T}}$ of the trace, and set of list attributes $\boldsymbol{LA}_{\mathrm{T}}$ of the trace.

Each event mapping $\boldsymbol{EM} = (event\_comlumn\_name, \boldsymbol{C}_{\mathrm{EID}}, C_e, T_{\mathrm{from}}, \boldsymbol{F}_{\mathrm{link}}, \boldsymbol{AM}_{\mathrm{E}}, \boldsymbol{LA}_{\mathrm{E}}) \in \boldsymbol{EM}$ defines one event type of the artifact with:

- event $event\_column\_name$,
- columns $\boldsymbol{C}_{\mathrm{EID}}$ defining the eventID,
- event column C_e defining the time stamp of the event,
- main table T_{from}, links to other tables in the form of foreign keys $\boldsymbol{F}_{link}$, and
- attribute mappings $\boldsymbol{AM}_{\mathrm{E}}$ and list attributes $\boldsymbol{LA}_{\mathrm{E}}$ of the event type.

A list attribute (of a trace or of an event) is an attribute with multiple values and defined by the list attribute mapping $LA = (key, \boldsymbol{C}_{\mathrm{AID}}, T_{\mathrm{from}}, \boldsymbol{F}_{fink}, \boldsymbol{AM}_{\mathrm{L}}, \boldsymbol{LA}_{\mathrm{L}})$ with:

- the given key,
- attributeID columns $\boldsymbol{C}_{\mathrm{AID}}$,
- main table T_{from}, links to other tables in the form of foreign keys $\boldsymbol{F}_{link}$, and
- attribute mappings $\boldsymbol{AM}_{\mathrm{L}}$ and list attributes $\boldsymbol{LA}_{\mathrm{L}}$.

Finally each attribute mapping defines an attribute with a single value, i.e., $AM = (name, type, C_a) \in \boldsymbol{AM}$ has a given $name$, $type$ and attribute column C_a. Note that a list attribute can recursively contain further list attributes.

Algorithm 1 shows the CreateTraceMapping algorithm which creates a mapping from a schema to an event log. First all event types in the artifact schema are identified automatically (line 1). Then columns are assigned as attributes to traces (lines 2 to 4) and events (lines 7 and 8). Next mappings are created for event types (lines 9 and 10) and the trace (lines 13 and 14) using the CreateMapping algorithm shown in

Algorithm 1. CreateTraceMapping($\boldsymbol{S}$)

```
Require: An artifact schema $\boldsymbol{S} = (\boldsymbol{T}, \boldsymbol{F}, \boldsymbol{D}, dom, T^m)$
$\boldsymbol{ET} \leftarrow IdentifyEventTypes(\boldsymbol{S})$
$\boldsymbol{T}_{\text{instance}} \leftarrow \{T^m\} \cup AllParents(\{T^m\}, \boldsymbol{S})$
$\boldsymbol{T}_{\text{instance}} \leftarrow \boldsymbol{T}_{\text{instance}} \cup SelectInstanceChildTables(\boldsymbol{T}^m, \boldsymbol{T}_{\text{instance}}, \boldsymbol{S})$
$\boldsymbol{C}_{\text{A}} \leftarrow GetNonEventColumns(\boldsymbol{T}_{\text{instance}})$
$\boldsymbol{EM} \leftarrow \emptyset$
for all $ET = (T_{\text{ET}}, C_e) \in \boldsymbol{ET}$ do
    $\boldsymbol{T}_{\text{event}} \leftarrow \{T_{\text{ET}}\} \cup SelectEventAttributeTables(T_{\text{ET}}, \{T_{\text{ET}}\} \cup \boldsymbol{T}_{\text{instance}}, \boldsymbol{S})$
    $\boldsymbol{C}_a \leftarrow GetNonEventColumns(\boldsymbol{T}_{\text{event}} \setminus \boldsymbol{T}_{\text{instance}})$
    $(\boldsymbol{C}_{\text{EID}}, T_{\text{from}}, \boldsymbol{F}_{\text{link}}, \boldsymbol{AM}_{\text{E}}, \boldsymbol{LA}_{\text{E}}) \leftarrow CreateMapping(T^m, T_{ET}, \boldsymbol{T}_{\text{event}}, \boldsymbol{C}_{\text{a}}, \boldsymbol{S})$
    $EM \leftarrow (event\_column\_name, \boldsymbol{C}_{\text{EID}}, C_e, T_{\text{from}}, \boldsymbol{F}_{\text{link}}, \boldsymbol{AM}_E, \boldsymbol{LA}_E)$
    $\boldsymbol{EM} \leftarrow \boldsymbol{EM} \cup \{EM\}$
end for
$(\boldsymbol{C}_{\text{TID}}, T_{\text{from}}, \boldsymbol{F}_{\text{link}}, \boldsymbol{AM}_{\text{T}}, \boldsymbol{LA}_{\text{T}}) \leftarrow CreateMapping(T^m, T^m, \boldsymbol{T}_{\text{instance}}, \boldsymbol{C}_{\text{A}}, \boldsymbol{S})$
$TM \leftarrow (\boldsymbol{C}_{\text{TID}}, T_{\text{from}}, \boldsymbol{F}_{\text{link}}, \boldsymbol{EM}, \boldsymbol{AM}_{\text{T}}, \boldsymbol{LA}_{\text{T}})$
return A TraceMapping $TM$ for the artifact
```

Algorithm 2. The steps in the algorithms are briefly described below. Further details of the algorithms can be found in [14].

Event types are identified by selecting all columns with a timestamp domain as event columns, except for columns that are part of a parent table of the main table. The latter columns are excluded since they are identical for several instances and therefore less likely to be events. For each event column an event type ET is constructed with event

Algorithm 2. CreateMapping($T^m, T_0, \boldsymbol{T}_{\text{attr}}, \boldsymbol{C}_{\text{attr}}, \boldsymbol{S}$)

```
Require: A main artifact table $T^m$, base table $T_0$ (with primary key $\boldsymbol{C}_{\text{ID}}$), a set of attribute
    tables $\boldsymbol{T}_{\text{attr}}$, a set of attribute columns $\boldsymbol{C}_{\text{attr}}$ and an artifact schema $\boldsymbol{S}$
$T_{\text{from}} \leftarrow T^m$
$\boldsymbol{F}_{\text{link}} \leftarrow Path(T^m, T_0)$
$\boldsymbol{AM} \leftarrow \emptyset$
$\boldsymbol{LA} \leftarrow \emptyset$
$(\boldsymbol{T}_{\text{one2one}}, \boldsymbol{T}_{\text{one2many}}) \leftarrow SplitOneAndMany(T_0, \boldsymbol{S})$
for all $T \in (\boldsymbol{T}_{\text{one2one}} \cap \boldsymbol{T}_{\text{attr}}) \cup \{T_0\}$ do
    $\boldsymbol{F}_{\text{link}} \leftarrow \boldsymbol{F}_{\text{link}} \cup Path(T_0, T)$
    for all $C \in \boldsymbol{C}_{\text{T}} \cap \boldsymbol{C}_{\text{attr}}$ do
        $AM \leftarrow CreateAttributeMapping(C)$
        $\boldsymbol{AM} \leftarrow \boldsymbol{AM} \cup \{AM\}$
    end for
end for
$\boldsymbol{T}_{\text{attr}} \leftarrow \boldsymbol{T}_{\text{attr}} \setminus \boldsymbol{T}_{\text{one2one}}$
for all $T \in \boldsymbol{T}_{\text{one2many}}$ do
    if $T \in \boldsymbol{T}_{\text{attr}} \vee (\boldsymbol{T}_{\text{attr}} \cap AllChildren(T, \boldsymbol{S}) \neq \emptyset$ then
        $LA \leftarrow CreateMapping(T^m, T, \boldsymbol{T}_{\text{attr}}, \boldsymbol{C}_{\text{attr}}, \boldsymbol{S})$
        $\boldsymbol{LA} \leftarrow \boldsymbol{LA} \cup \{LA\}$
    end if
end for
return general mapping item $(\boldsymbol{C}_{\text{ID}}, T_{\text{from}}, \boldsymbol{F}_{\text{link}}, \boldsymbol{AM}, \boldsymbol{LA})$
```

table T_{ET}, event column C_e and an initially empty set of event attribute columns $\boldsymbol{C}_a$. T_{ET} is the table that contains C_e.

All columns that are not considered to be events are considered to be attributes. These attribute columns are assigned to the most specific event possible or as instance attributes if it is not possible to assign them to a specific event. For example: If an attribute column is part of a table without event columns, then it will be assigned to event columns in the parent table (assuming they exist). If there are event columns in the same table, the attribute columns will be assigned to those event columns. The assignment is done based on the table that contains the column as following.

Columns in the set of artifact instance tables $\boldsymbol{T}_{\mathrm{instance}}$ are assigned as instance attributes. The set of artifact instance tables consists of the main artifact table, all of its parents and all children that do not have another parent table with event columns (the instance child tables).

For each event type, all columns in the corresponding set of event attribute tables $\boldsymbol{T}_{\mathrm{event}}$ are assigned as event attributes. For event columns in the main artifact table there are no separate event attributes, thus then the set is empty. Otherwise the set consists of (1) the event table T_{ET}, (2) all child tables for which there is a foreign key path from the event table to the child table that does not contain another event table and (3) all parent tables of the child tables that are not part of the set of instance tables $\boldsymbol{T}_{\mathrm{instance}}$ and do not have another event table as one of their parents. Note that the second subset may contain tables that are also assigned to other event types.

The *CreateMapping* algorithm creates a tuple $(\boldsymbol{C}_{\mathrm{ID}}, T_{\mathrm{from}}, \boldsymbol{F}_{\mathrm{link}}, \boldsymbol{AM}, \boldsymbol{LA})$ called a "general mapping item" that serves as the basis for a trace mapping, event mapping or list attribute mapping. The basic idea is that each created mapping consists of a set of tables for which only one record exists for each record in the chosen base table T_0, thus ensuring that multiple values do not occur. The algorithm starts by splitting the given attribute tables $\boldsymbol{T}_{\mathrm{attr}}$ into a set for which this condition holds $\boldsymbol{T}_{\mathrm{one2one}}$ and a set of attribute tables for which this condition does not hold $\boldsymbol{T}_{\mathrm{one2many}}$ as explained below. One mapping is then created for the base table and all tables in $\boldsymbol{T}_{\mathrm{one2one}}$. This mapping contains a number of submappings (the list attributes $\boldsymbol{LA}$) as required for the tables in $\boldsymbol{T}_{\mathrm{one2many}}$. Note that to create an event mapping the event column C_e and an event name are added to the resulting general mapping item (as shown on line 10 of Algorithm 1), and to create a trace mapping the set of event mappings $\boldsymbol{EM}$ is added (as shown on line 14 of Algorithm 1).

The split of tables into the $\boldsymbol{T}_{\mathrm{one2one}}$ and $\boldsymbol{T}_{\mathrm{one2many}}$ sets is done by recursively verifying foreign keys in the child direction and the parent direction. In the parent direction there will always be only one record for each record in the base table. In the child direction it has to be checked if more records exist in the child table for each record in the base table. A repeated part of the algorithm is the path between two tables which consists of the sequence of foreign keys connecting those tables; it can be calculated using e.g. Dijkstra's algorithm [8].

4.2 Extracting Logs and Discovering Life-Cycles

Extracting logs. The extracted event log-trace mapping TM defines for a given artifact schema S how to shape the event information contained in the database $\boldsymbol{R}$ into events (with attributes), and how to group events into different traces (distinguished by their trace ids and having further attributes). This information is sufficient to automatically extract a classical log (sequences of events) from $\boldsymbol{R}$. For the extraction, we employ (and slightly adapt) the log extraction technique of XESame [18]. XESame is a technique and tool that extracts classical logs in XES format from a database in 4 steps: (1) specify an event-log to trace mapping, (2) construct database queries to extract data, (3) execute the queries to populate a cache database and (4) create a XES event log from the cache database.

The first step in XESame is manual: the user manually specifies an event log-trace mapping based on the given database tables, columns and keys. Algorithm 1 does the same, but fully automatically. Thus, by handing the event log-trace mapping to the second step of XESame, XESame automatically generates the database queries needed to extract the log. Technically, XESame then extracts for each given event mapping an event and finally groups events to traces based on the trace mapping.

The events of an event mapping $(name, \boldsymbol{C}_{\mathrm{EID}}, C_e, T_{\mathrm{from}}, \boldsymbol{F}_{\mathrm{link}}, \boldsymbol{AM}_{\mathrm{E}}, \boldsymbol{LA}_{\mathrm{E}}) \in \boldsymbol{EM}$ are extracted by first joining the tables containing the time stamp attribute C_e and the event id attribute $\boldsymbol{C}_{\mathrm{EID}}$ with the main table T_{from} (this may require to include further tables in the join based on the foreign key relations in $\boldsymbol{F}_{\mathrm{link}}$). Each record in the joined table defines an event with the given name, that occurred at the time-stamp written in column C_e. Note that the joined table also contains the trace id columns $\boldsymbol{C}_{\mathrm{TID}}$ of T_{from}, thus associating each event with exactly one trace. Attributes of this event are obtained from $\boldsymbol{AM}_{\mathrm{E}}$ and $\boldsymbol{LA}_{\mathrm{E}}$ in a similar way by joining the table containing $\boldsymbol{C}_{\mathrm{EID}}$ with the tables of the id columns of the respective attribute.

All events of all traces are extracted in this way, then grouped by the values on the trace id columns $\boldsymbol{C}_{\mathrm{TID}}$ of T_{from}, and finally ordered by their time-stamp attribute values. Each group defines a trace which gets additional attributes; again by joining the main table T_{from} with the attribute identifying tables as specified in the attribute and list attribute mappings. The resulting traces of events are written in XES format. We slightly adopted the approach of [18] for our purposes by defining explicit event ids and attribute ids in the schema-log mapping; details can be found in [14].

Discovering artifact life-cycles. This technique allows to extract logs $L_1, \ldots, L_k$ for artifacts $A_1, \ldots, A_k$ from $\boldsymbol{R}$. This effectively reduces the problem of discovering artifact life-cycles in $\boldsymbol{R}$ to the problem of discovering a process model from each log $L_1, \ldots, L_k$. For this problem a large number of existing process discovery algorithms can be applied [1, 17].

5 Empirical Evaluation

The approach described in the previous sections was evaluated using a prototype implementation. We evaluated the technique on an artificial data-set $\boldsymbol{R}_A$ of an order-to-cash process, and on a real-life dataset $\boldsymbol{R}_R$ obtained from the production ERP system of a large food wholesale and retail company. $\boldsymbol{R}_R$ comprised $>$ 300 tables containing $>$ 40GiB of data. Details on the datasets and the prototype implementation can be found in [14].

The reallife dataset showed that different steps in our approach are differently hard to solve. During schema extraction, finding attribute types required >15hrs to discover all

Table 1. Results on schema-logmapping

	$\lvert T\rvert$	$\lvert C\rvert$	$\lvert ET\rvert$	$\lvert LA\rvert$	$\lvert AM\rvert$	time
A1	3	10	0	0	5	<0.5 s
A2	6	23	9	2	8	<0.5 s
A3	10	35	11	3	13	<0.5 s
R1	1	195	23	0	171	<0.1 s
R2	47	869	127	0	841	1.4 s

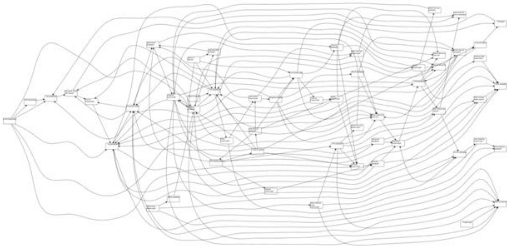

Fig. 2. Discovered life-cycle of R2

timestamped attribute types. Key discovery is an $\mathcal{NP}$-complete problem; we observed runtimes of 4.5hrs to find all primary keys in the reallife dataset. Foreign key discovery took 5hrs to find all single column foreign keys and 6days to find all double column foreign keys. Finally, artifact schema discovery required approx. 17hrs to compute table entropies and approx. 5hrs to compute table distances for clustering; clustering itself succeeded in less than a second. This allows to try various numbers k of clusters to identify without computational penalty. See [14] for details.

For $\boldsymbol{R}_A$ we could identify 3 reasonable artifacts (A1-A3). For $\boldsymbol{R}_R$ analysis of the right number of artifacts was more involved. In an iterative approach, we could identify around 20 different artifacts. The largest one comprised 47 tables over 869 columns see Tab. 1 which shows the numbers for A1-A3 and two artifacts R1 and R2 of $\boldsymbol{R}_R$. We then discovered schema-log mappings using the technique of Sect. 4 requiring less then 2 seconds in all cases. Artifacts A2 and A3 follow a life-cycle whereas A1 has no event associated; a closer analysis revealed that A1 is a static data object that relates instances of A2 to instances of A3. For R1 and R2 we identified 23 and 127 event types and 171 and 841 attributes, respectively.

Log extraction with XESame took more time as the entire dataset has to be processed. For artifacts A1-A3 logs of 100-200 traces and approx. 10 events per trace could be extracted within a few seconds; for R1 and R2 extraction required several hours where serializing logs files takes the lion share of the time. For validation, we sampled the data

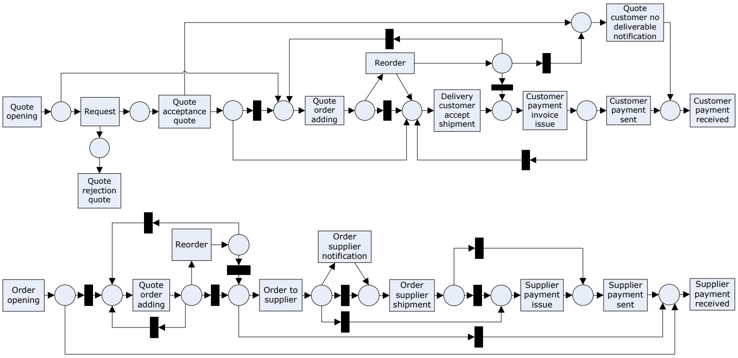

Fig. 3. Discovered life-cycle models of A2 and A3 of an order-to-cash process

source of R2 to 1000 traces of >246,000 events which required approx. 1hr to execute the query and approx. 32hrs to write the log file.

Using the Heuristics Miner [1] we obtained the life-cycle model of R2 shown in Fig. 2. Despite its complex structure it was validated as correct by the process owner. For A2 and A3 we had expected lifecycle models available; to validate precision of our technique we checked fitness of the expected models of A2 and A3 to the extracted logs (i.e., whether the models can replay the log) [1], and obtained high fitness values of 0.99 and 0.95. The lifecycle models discovered from these logs using a genetic miner [1] are shown in Fig. 3. Note that both life-cycle models share some activities, for instance *reorder*, indicating that instances of these artifacts synchronize occasionally through a process execution on the same event.

6 Conclusion

This paper addressed the problem of automatically discovering a process model of a data-centric process. Such processes lack a unique notion of a case, but rather provide multiple notions of cases related to the business objects of the process. Following the artifact-centric approach [7], we provided a technique to automatically extract artifact schemas $A_1, \ldots, A_k$ and corresponding artifact life-cycle logs $L_1, \ldots, L_k$ which describe how each of the artifacts evolved during past process executions. From these logs life-cycle models can be discovered with classical techniques.

Our approach is conceptually similar to the manual interviewing approach of [12] by first identifying objects and then processes. However, our approach is fully automatic up to picking the number k of artifacts to be discovered. The complete approach combines a number of non-trivial, existing techniques for schema discovery, schema summarization, log extraction, and life-cycle discovery. Technically, we contributed a new and first automatic discovery of schema-log mapping needed for log extraction. Our technique is general as it has no restrictions on the input apart from it being a relational database in which an event's timestamp is recorded in a separate database column. This assumption is backed by practice: most ERP systems such as SAP and PeopleSoft record events in that form. The approach is implemented in a prototype tool and was validated on actual data from an ERP production system of a large retailer.

However, some open problems remain. Currently, the user has to pick the number k of artifacts to be discovered. While an iterative approach to find the right k yielding the "right" artifacts has proven feasible, a more automatic approach to identify the relevant artifacts of the process is needed. Generally, it could be desirable to include domain information when discovering artifact schemas. The same remark applies for discovering the schema-log mapping. The identified definition of event types and their attributes could improve in quality if domain knowledge is included or the mapping is manually refined afterwards. Finally, our technique currently focuses on discovering artifact life-cycles, but ignores artifact interactions as they are documented by object relations in the original data source; more research is required here.

Acknowledgements. The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement no. 257593 (ACSI).

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Abedjan, Z., Naumann, F.: Advancing the Discovery of Unique Column Combinations. In: CIKM 2011, pp. 1565–1570. ACM (2011)
3. Ahmadi, B., Hadjieleftheriou, M., Seidl, T., Srivastava, D., Venkatasubramanian, S.: Type-Based Categorization of Relational Attributes. In: EDBT 2009, pp. 84–95. ACM (2009)
4. Bauckmann, J., Leser, U., Naumann, F., Tietz, V.: Efficiently Detecting Inclusion Dependencies, pp. 1448–1450. IEEE (April 2007)
5. Bhattacharya, K., Guttman, R., Lyman, K., Heath, I.I.I., Kumaran, S., Nandi, P., Wu, F., Athma, P., Freiberg, C., Johannsen, L., et al.: A model-driven approach to industrializing discovery processes in pharmaceutical research. IBM Systems Journal 44(1), 145–162 (2005)
6. Bozkaya, M., Gabriels, J., Werf, J.: Process Diagnostics: A Method Based on Process Mining, pp. 22–27. IEEE (February 2009)
7. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull. 32(3), 3–9 (2009)
8. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1(1), 269–271 (1959)
9. Dumas, M.: On the Convergence of Data and Process Engineering. In: Eder, J., Bielikova, M., Tjoa, A.M. (eds.) ADBIS 2011. LNCS, vol. 6909, pp. 19–26. Springer, Heidelberg (2011)
10. Heath, T.: Siena: a tool for modeling and executing artifact-centric business processes (December 2009)
11. Ingvaldsen, J.E., Gulla, J.A.: Preprocessing Support for Large Scale Process Mining of SAP Transactions. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM 2007 Workshops. LNCS, vol. 4928, pp. 30–41. Springer, Heidelberg (2008)
12. Liu, R., Bhattacharya, K., Wu, F.Y.: Modeling Business Contexture and Behavior Using Business Artifacts. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007. LNCS, vol. 4495, pp. 324–339. Springer, Heidelberg (2007)
13. Marchi, F.D., Lopes, S., Petit, J.M.: Unary and n-ary inclusion dependency discovery in relational databases. J. Intell. Inf. Syst. 32(1), 53–73 (2009)
14. Nooijen, E.: Artifact-Centric Process Analysis: Process Discovery in ERP Systems (April 2012)
15. Piessens, D.: Event Log Extraction from SAP ECC 6.0 (April 2011)
16. Ramesh, A.: Process mining in PeopleSoft (2006)
17. Tiwari, A., Turner, C., Majeed, B.: A review of business process mining: State-of-the-art and future trends. Business Process Management Journal 14(1), 5–22 (2008)
18. 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)
19. Wu, W., Reinwald, B., Sismanis, Y., Manjrekar, R.: Discovering Topical Structures of Databases. In: SIGMOD 2008, pp. 1019–1030. ACM (2008)
20. Yang, X., Procopiuc, C.M., Srivastava, D.: Summarizing relational databases. Proc. VLDB Endow. 2, 634–645 (2009)
21. Zhang, M., Hadjieleftheriou, M., Ooi, B.C., Procopiuc, C.M., Srivastava, D.: On multi-column foreign key discovery. Proc. VLDB Endow. 3, 805–814 (2010)
22. Zhang, M., Hadjieleftheriou, M., Ooi, B.C., Procopiuc, C.M., Srivastava, D.: Automatic discovery of attributes in relational databases. In: SIGMOD 2011, pp. 109–120. ACM (2011)