Discovering Interacting Artifacts from ERP Systems

Xixi Lu, Marijn Nagelkerke, Dennis van de Wiel, and Dirk Fahland

Abstract—*Enterprise Resource Planning* (ERP) systems are widely used to manage business documents along a business processes and allow very detailed recording of event data of past process executions and involved documents. This recorded event data is the basis for auditing and detecting *unusual flows*. *Process mining* techniques can analyze event data of processes stored in linear event logs to discover a process model that reveals unusual executions. Existing approaches to obtain linear event logs from ERP data require a single case identifier to which all behavior can be related. However, in ERP systems processes such as *Order to Cash* operate on multiple interrelated business objects, each having their own case identifier, their own behavior, and interact with each other. Forcing these into a single case creates ambiguous dependencies caused by *data convergence* and *divergence* which obscures unusual flows in the resulting process model. In this paper, we present a new semi-automatic, end-to-end approach for analyzing event data in a plain database of an ERP system for unusual executions. More precisely, we identify an *artifact-centric process model* describing the business objects, their life-cycles, and how the various objects *interact* along their life-cycles. This way, we prevent data divergence and convergence. We report on two case studies where our approach allowed to successfully analyze processes of ERP systems and reliably revealed unusual flows later confirmed by domain experts.

Index Terms—Process Discovery, Artifact-Centric Processes, Outlier Detection, Relational Data, Log Conversion, ERP Systems

1 Introduction

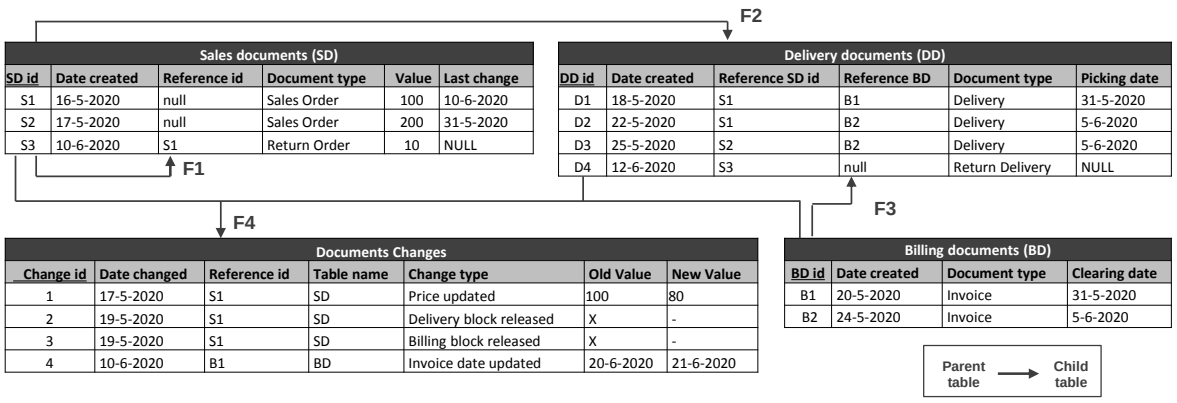
Information systems (IS) not only store and process data in an organization but also record *event data* about how and when information changed. This "historical event data" can be used to analyze, for instance, whether information processing in the past conformed to the prescribed processes or to compliance requirements. For example, has each order by a gold customer been delivered with priority shipping, or have all delivery documents been created before creating the invoice? Manual analysis of historic event data is time consuming and error-prone as often hundreds of thousands of records need to be checked.

Process mining [1] offers automated techniques for this task. The most prominent technique is to discover from historical event data a graphical process model describing historic behavior; the discovered model can be visually explored to identify the main flows and the unusual flows of the process. Process analysts and domain experts can then for instance identify the historic events that correspond to unusual flows, investigate circumstances and possible causes for this behavior, and devise concrete measures to improve the process [2], [3]. The success of the analysis often depends on whether unusual behavior is easy to distinguish visually from normal behavior. Prerequisite to this analysis is a *process event log* that describes how all information changes occurred from the perspective of a particular process; its underlying assumption is that each event can unambiguously be mapped to a particular execution of the process.

Problem Description. In general, information access is not tied to a particular process execution; rather the same information can be accessed and changed from various processes and applications. A typical example is *Enterprise Resource Planning* (ERP) systems (such as SAP and Oracle Enterprise) built using service-oriented architecture. They separate the information and the business processes (which uses the information) in different layers to increase the re-usability and flexibility of the system [4], [5]. The information are stored as *business objects* (also called *documents*) related to each other through one-to-many and many-to-many relations, typically in the relational database. Accesses to these objects are encapsulated in services. Information changes occur when users proceed with high-level end-to-end business processes and invoke services to update business objects, known as *transactions*, and the completion of a transaction is logged as an event also called *transactional data* (see Sect. A for a detailed discussion).

Fig. 1 shows a simplified example of the transactional data of an *Order to Cash* (OTC) process supported by SAP systems; Fig. 2 visualizes the events of Fig. 1 that are related to document creation. There are

- *X. Lu and D. Fahland are with the Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands, 5600 MB. E-mail: x.lu@tue.nl and d.fahland@tue.nl*
- *M. Nagelkerke and D. van de Wiel are with KPMG IT Advisory N.V., Eindhoven, The Netherlands, 6513 AM. E-mail: Nagelkerke.marijn@kpmg.nl and vandewiel.dennis@kpmg.nl*



Sales documents (SD)

SD id	Date created	Reference id	Document type	Value	Last change
S1	16-5-2020	null	Sales Order	100	10-6-2020
S2	17-5-2020	null	Sales Order	200	31-5-2020
S3	10-6-2020	S1	Return Order	10	NULL

Delivery documents (DD)

DD id	Date created	Reference SD id	Reference BD	Document type	Picking date
D1	18-5-2020	S1	B1	Delivery	31-5-2020
D2	22-5-2020	S1	B2	Delivery	5-6-2020
D3	25-5-2020	S2	B2	Delivery	5-6-2020
D4	12-6-2020	S3	null	Return Delivery	NULL

Documents Changes

Change id	Date changed	Reference id	Table name	Change type	Old Value	New Value
1	17-5-2020	S1	SD	Price updated	100	80
2	19-5-2020	S1	SD	Delivery block released	X	-
3	19-5-2020	S1	SD	Billing block released	X	-
4	10-6-2020	B1	BD	Invoice date updated	20-6-2020	21-6-2020

Billing documents (BD)

BD id	Date created	Document type	Clearing date
B1	20-5-2020	Invoice	31-5-2020
B2	24-5-2020	Invoice	5-6-2020

Fig. 1. The tables of the simplified OTC example

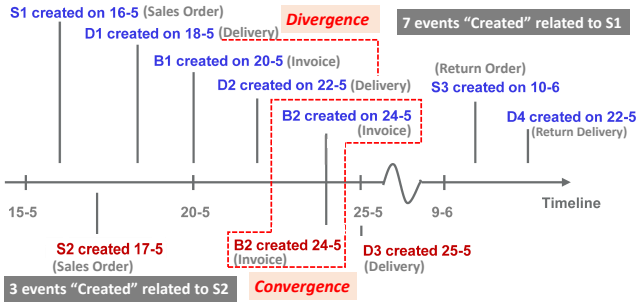


Fig. 2. A time-line regarding the creation of documents of the OTC example.

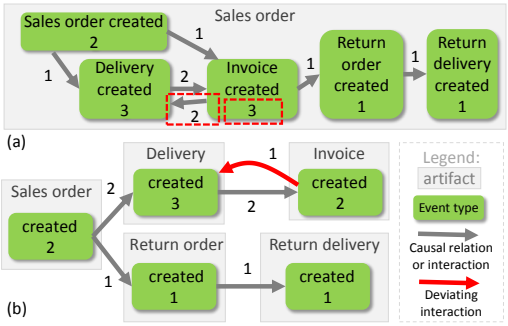


Fig. 3. Artifact-centric model of the behavior in Fig. 2

two sales orders S1 and S2; creation of S1 is followed by creation of a delivery document D1, an invoice B1, another delivery document D2, and another invoice B2 which also contains billing information about S2. Creation of S2 is also followed by creation of another delivery document D3. Further, there is a return order S3 related to S1 with its own return delivery document D4. The many-to-many relations between documents surface in the transactional data of Fig. 1: a sales document can be related to multiple billing documents (S1 is related to B1 and B2) and a billing document can be related to multiple sales document (B2 is related to S1 and S2). This behavior already contains an *unusual flow*: delivery documents were created twice *before* the billing document (main flow), but once the order was reversed (B2 before D3).

The main research problem addressed in this paper is to provide (semi-)automated techniques to

1) to reconstruct *accurate* graphical models which describe the high level end-to-end business processes that were executed in reality, from transactional data recorded during the execution and
2) to identify main flows and unusual flows to help users analyze their business processes and the used business objects.

Classical process mining techniques cannot be applied directly. Many previous studies have shown that an attempt to cast transactional data over objects with many-to-many relations into a single process event log and to discover a single process model describing all transactional data is bound to fail. This step leads to false dependencies between events and duplicate events which obscures the main flow and hinders the detection of unusual flows [6], [7], [8], [9], [10]. Casting the events of Fig. 2 into a single log of the *Sales order* yields the model of Fig. 3(a) which is inaccurate: two invoices are created before their deliveries instead of one, and three invoices are created instead of two (known as *divergence* and *convergence*, respectively) [9].

Contribution. We propose to approach the problem under the "conceptual lens" of *artifact-centric models* [11], [12]. An *artifact* is a data object over an information model; each artifact instance exposes *services* that allow changing its informational contents; a *life-cycle model* governs when which service of the artifact can be invoked; the invocation of a service in one artifact may trigger the invocation of another service in another artifact. Information models of different artifacts can be in one-to-many and many-to-many relations, which allows to describe behavior over complex data in terms of multiple objects interacting via service invocations. We apply the artifact-centric view to our problem as follows: each document of an ERP system can be seen as an artifact; transactions on the document are service calls on the artifacts; behavioral dependencies between transactions of documents can

be seen as life-cycle behavior and dependencies of service calls. With these concepts, the transactional data of Fig. 1 can be described as the artifact-centric model of Fig. 3(b). The model visualizes the order in which objects are created and also highlights the one unusual flow of invoice B2 being created before delivery D2.

The problem of discovering an artifact-centric process model from relational ERP data decomposes into two sub-problems:

1) Given a relational data source, identify a set of artifacts, extract for each artifact an event log, and discover a model of its life-cycle.
2) Given a set of artifacts and their data source, identify interactions between the artifacts, between their instances, between their event types and between their events. As a result, obtain a complete artifact-centric process model.

Fig. 4 shows the overview of our approach: the steps for discovering individual artifacts (problem 1) are shown by filled arcs, the steps for discovering interactions between artifacts (problem 2) are shown by dashed arcs. In a nutshell, (1.1-1.2) we use the data schema to discover *artifact schemas* and then *artifact types* which detail all timestamped columns related to a particular business object. (1.3) For each artifact we then extract a classical *event log* [1], each *case* describes all events related to one *instance* of the artifact. (1.4) Existing process discovery algorithms allow discovering a life-cycle model of the artifact. In parallel, (2.1) we discover interactions between artifacts from foreign key relations in the data source; (2.2) during log extraction, each case of an artifact is annotated with references to cases of other artifacts this case interacts with. (2.3) The case references are refined into interactions between events of different artifacts, which we (2.4) generalize to interactions between artifact life-cycles.

We implemented our approach and conducted two case studies. In both case studies the discovered process models were assessed as accurate graphical representations of the source data by domain experts; accurate insights about real process executions and unusual flows could be obtained exploratively and much faster than with existing best practices. In particular, by treating any one-to-many or many-to-many relation as an *interaction* between two artifacts, we could eliminate divergence and convergence, the interactions discovered in (2.1-2.4) were meaningful to business users, and unusual flows were detected accurately.

The remainder of this paper is structured as follows. Sect. 2 discusses related work. Sect. 3 illustrates our extended approach to identify artifacts and their life-cycles from a given relational data source. In Sect. 4, we discuss interactions between artifacts on different levels and show how to identify these interactions to obtain a complete artifact-centric model. We implemented our technique and report on two case studies in Sect. 5. Sect. 6 concludes the paper.

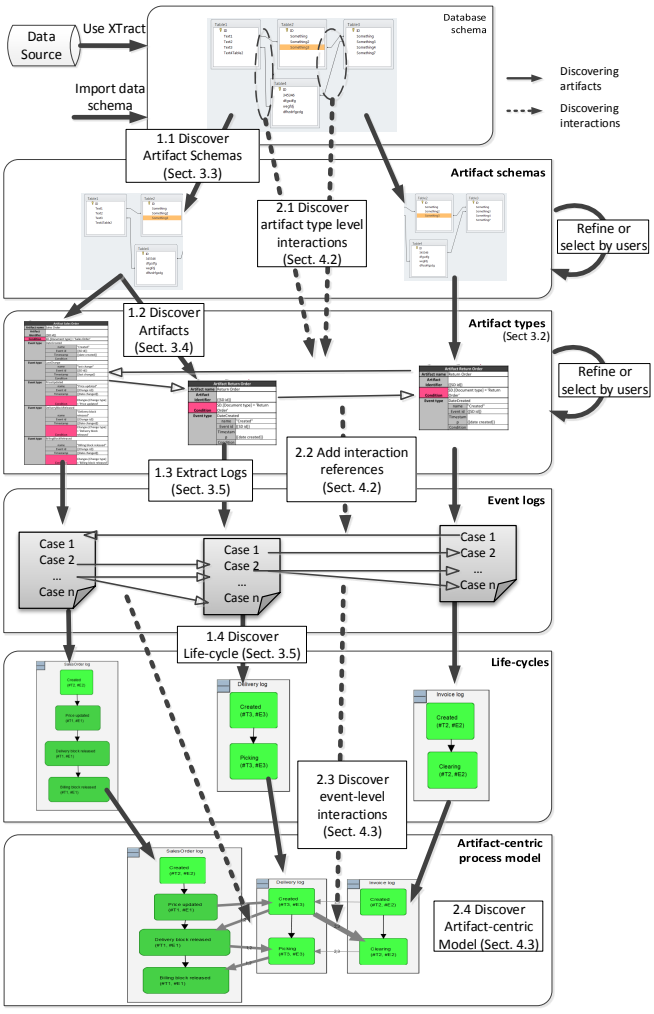


Fig. 4. An overview of our approach.

2 Related Work

We discuss existing work along the main problems addresses in this paper: (1) discovering conceptual entities and their relations from a relational data structure, (2) extracting event logs from relational data structures, (3) discovering models or specifications of a single entity/process from an event log, and (4) discovering/analyzing relations and interactions between multiple objects and processes.

Entity discovery. The relational schema used in a database may differ significantly from the conceptual entities which it represents, mostly to improve system performance. Various existing works solve different steps along the way. After discovering the actual relational schema from the data source [13], [14], [15], an (extended) ER model can be retrieved that turns foreign keys between tables into proper relations between entities [16], [17], [18]. The artifact discovery

problem faced in this paper (Sect. 3) goes one step further: *one artifact type may comprise multiple entities* as long as they are considered to be following a joint life-cycle; see [19, Chap.2] for a discussion. This problem has been partly addressed in [20] through schema *schema summarization* techniques [21], but *convergence* and *divergence* [9] may still arise.

It is also possible to discover entities and artifact types from a *raw event stream* (instead of a relational structure) where each event carries "enough" attributes and identifiers. The approach in [22] first reconstructs a simple relational schema from all events and their attributes, two related entities can be grouped into the same artifact if one entity is always created before the other (in the event stream); this extraction dismisses interactions between different artifacts which is crucial to our approach (step 2.1 in Fig. 4). This work extends ideas of [20] and presents a first complete solution to discovering entities, artifacts, and their interactions from relational data in Sect. 3 (steps 1.1 and 1.2 in Fig. 4) and Sect. 4 (step 2.1).

Log Extraction. Existing work on extracting event logs from relational data sources (step 1.3 in Fig. 4) mainly focus on identifying a monolithic process definition and extracting one event log where each trace describes the (isolated) execution of one process instance. Manually approaches to extracting data from relational databases of SAP systems particularly failed to separate events related to various processes; analyzing what was part of the process was error-prone and time consuming [23], [24]. The generic log extraction approach of [8] lets the user define a mapping from from tables and columns to log concepts such as traces, events, and attributes (assuming the existence of a single case identifier to which all events can be related); various works exist to improve finding optimal case identifiers and relations between the identifiers and events[9], [10], [25]. If the event data is structured along multiple case identifiers as in ERP systems, all these approaches suffer from data convergence and divergence [9]. In this work, we identify multiple artifact types (each having their own case identifier) and separate events into artifact types such that convergence and divergence do not arise; having identified proper case identifiers and related events, we then reuse the approach of [8] to extract an event log for each artifact type. No existing work extracts attributes that describe the interaction between different artifact instances; we present a first solution in Sect. 4 (step 2.2 of Fig. 4).

Model discovery. Much research has been conducted on the problem of discovering a (single) process model from other information artifacts. *Process mining* [1] takes as input an event log where each trace describes the execution of one process instance. An event in the log is a *high-level* event corresponding to a complex user action or system action, potentially involving dozens or thousands of method calls, service invocations, and data updates. The log describes behavior that *actually* happened allowing to discover unusual and exceptional flows not intended by the original process design. Some well known process discovery techniques are Alpha algorithm [26], (Flexible) Heuristic miner [27], Genetic process mining [28], ILP mining [29], Fuzzy mining [30], and Inductive Mining [31], [32]. De Weerdt et al. [33] compared various discovery algorithms using real-life event logs. Existing discovery techniques mainly focus on a single process and assume the model operates in an isolated environment. We will reuse existing process discovery techniques when discovering artifact life-cycle models (step 1.4) and artifact interactions (step 2.3 of Fig. 4).

One can also use low-level event logs where one event corresponds to an atomic operation (method invocation, data read/write, message exchange). Low-level event logs are usually considered when discovering models and specifications of particular software artifacts (the object-oriented source code of a module, the GUI, etc.). Various techniques are available to discover formal behavioral specifications such as automata [34], [35], scenario-based specifications [36], or object-usage models [37] from low-level event logs; see [38], [39] for overviews. Like artifacts, object-usage models describe how an object is being used in a context. These techniques rely on the assumption of sequential execution (on a single machine) and strict patterns (following code execution), while our problem features a high degree of concurrency and user-driven behavior. Concurrent use and user influence is considered in [40] being essentially a variant of process mining discussed above.

Other works use event data generated by users in the application interface to discover models of how a user operates an application. These events can be used to analyze styles of process modeling [41] or problem solving strategies in program development environments [42]; these works cannot analyze events beyond the user interface which is the scope of this paper. In [43] it is shown how to generate application interface test models by generating user interface on a web interface; that work *synthesises* user behavior whereas we *analyze actual* user behavior.

Interactions and deviations. The notion of *artifacts* [11], [12] where a (complex) process emerges from the interplay of multiple related objects has proven to be a useful conceptual lens to describe behavioral data of ERP systems. The feasibility of the artifact idea in process mining was demonstrated in [44], [45] by checking the conformance of a given artifact-centric model to event data with multiple case identifiers. In [46], [20], the XTract approach was introduced which allows for fully automatic discovery of an artifact-centric model (multiple artifacts and their life-cycles) from a given relational data source. It is also possible to discover artifact-centric process mod-

els from event streams where events contain enough attributes to discover entities and relations [22]; this work also shows how to produce life-cycle models in GSM notation [47], a declarative language for describing artifact-centric processes. Both approaches are limited to identifying individual artifacts, extracting logs, and discovering life-cycles, but cannot identify interactions between artifacts and may suffer from convergence and divergence. In this paper, we extend this approach to avoid these problems and also discover interactions between artifacts.

With respect to the second problem of discovering interactions between artifacts, much less literature has been found. Petermann et al. [48] proposed to represent relational data as graphs in which nodes are objects or instances and edges are relations, which is comparable to (2.1) in Fig. 4. However, the scope of their approach is limited to instances and direct relations between objects, while neglecting the dynamic life-cycles of instances and the interrelations between them. Conforti et al. [49] address data divergence and convergence by contextualizing one-to-many relations as subprocesses of a process instead of interactions between artifacts; this approach is unable to handle many-to-many relations as encountered in this paper.

Also object-usage models and scenario-based specifications have been used to study object interactions. In [50] it is shown how to discover from source code how an (object-oriented) object is being used in a caller context; such models can also be discovered from low-level execution traces [37]. Also scenario-based specifications discovered from low-level event logs [36] describe interactions between multiple objects. However, all these works either focus on a single object or do not distinguish multiple instances of several interacting objects in many-to-many relations, i.e., two orders being processed in three deliveries, which is a crucial property of our problem. Using event logs from two different versions of an object, it is possible to detect changes in object usage [51]. In this paper, we want to detect deviations of usage of a single version of an object to identify outlier behavior.

To summarize, our approach addresses a more general problem than all preceding approaches: (1) discover multiple artifacts (comprising multiple entities) that are in many-to-many relations to each other such that data divergence and convergence do not arise, and (2) discover interactions between artifacts and identify outliers in these interactions. Sect. 3 and Sect. 4 address the first and second problem, respectively, and explain our approach more in detail.

3 Artifact Discovery

Our first goal is to identify the high-level conceptual business objects stored in the data source and discover for each such object a model of its life-cycle. However, the relational schema of the data source may differ significantly from the conceptual model it represents, usually due to performance optimizations. After structuring the problem (Sect. 3.1) we show how to identify all conceptual objects and their event data in a relational data source in terms of artifacts (Sect. 3.2-3.4). Then existing log extraction and process discovery techniques can be applied to obtain a life-cycle model for each artifact (Sect. 3.5).

3.1 Relational Schema vs. Conceptual Model

One can describe the difference between conceptual high-level models and relational schemata in terms of four basic operations. (1) *Horizontal partitioning* specializes a general entity (or artifact) into multiple, more specific tables. For example, "Documents" are distinguished into "Sales Documents" and "Delivery Documents" stored in different tables, see Fig. 1. (2) *Vertical partitioning* distributes properties of one entity into multiple different tables. For example, the "Changes" to a "Delivery Document" are stored in the separate "Document Changes" table. (3) *Horizontal Anti-Partitioning* generalizes data from multiple entities into one table. For example, changes of different document types are all stored in the same "Document Changes" table rather than in separate tables. (4) *Vertical Anti-Partitioning* aggregates attributes of multiple entities into the same table. For example, "Sales Documents" aggregates attributes for "Sales Order" and "Return Order" (even though "Reference id" is only required by "Return Order"). The examples also show that one table may be the result of multiple such operations.

Event-based analysis of conceptual artifacts requires to undo these operations: (D.1) recover conceptual entities from the relation schema, (D.2) group entities that together describe one real-life business object into an *artifact type*, (D.3) such that an event log of the artifact can be extracted, and (D.4) convergence and divergence do not arise. As previous works do not solve (D.2) and (D.4), see Sect. 2, we propose the following semi-automatic approach.

3.2 Artifact Types based on Relational Schemas

To address (D.1)-(D.3), we adopt ideas of [20] and ground the definition of a conceptual artifact directly in the relational data source itself. An *artifact type* defines all attributes of the artifact and the tables where these attributes are stored:

- the primary key of one of these tables is chosen as the *artifact identifier*, each value of the artifact identifier defines a new artifact instance;
- each time-stamped attribute (together with the artifact identifier) becomes an *event type* of the artifact life-cycle, each time-stamp value defines an event in the corresponding artifact instance.

Artifact types and event types can carry further attributes: any attribute in a table holding a timestamp

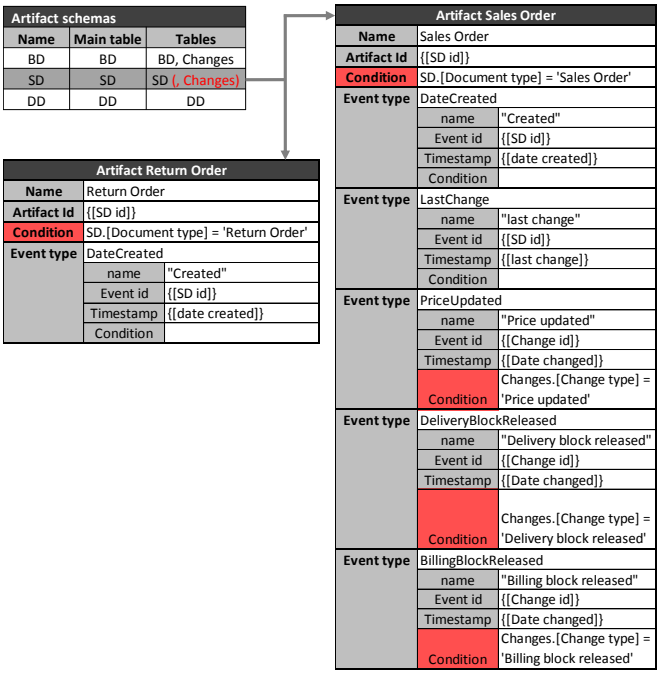


Fig. 5. The artifact schemas of Fig. 1 (top left) and two artifact types derived from schema SD.

becomes an *event-level attribute*; its value provides more information about the event. Any other attributed related to the artifact identifer becomes an *artifact-level attribute*. For example, *Old Value* and *New Value* are event-level attributes of the events stored in table *Document Changes* of Fig. 1. Attributes of a single artifact may be are stored in *different* tables (to allow reversing vertical partitioning).

In contrast to [20], our notion of an artifact type is defined on the attribute level (rather than table level). This allows omitting attributes of a table in an artifact type definition and mapping attributes of the same table to different artifacts (reverses vertical anti-partitioning). Moreover, the same attribute may be *shared* by different artifact types (reverses horizontal anti-partitioning); in this case a *discriminating condition* has to be provided to relate records in the source to the correct artifact type/event type. Fig. 5 illustrates two artifact types Sales Order and Return Order grounded in the same tables of Fig. 1; the primary key *SD.id* is refined by the two conditions *SD.[Document type] = 'Sales Order'* and *SD.[Document type] = 'Return Order'*; timestamp attribute *Date changed* is refined into 3 different event types of Sales Order by 3 different conditions. Formal definitions of artifact types are given in Sect. B.3.

3.3 Artifact Schema Discovery

To discover artifact types from the relational source we first compute an *abstraction* of an artifact type, called *artifact schema* that only reverses vertical partitioning while also ensuring (D.4), as shown below. Refining an artifact schema into artifact types to reverse the other operations of Sect. 3.1 may require user input as discussed in Sect. 3.4.

An *artifact schema* is a set of tables that together contain all attributes of an artifact type – or multiple artifact types of the same shape. These tables obey 3 principles. (1) The tables of an artifact are related to each other via one or more references (being the result of vertical partitioning). (2) As each artifact type has an artifact identifier, there is a main table T_m to which all other tables of the artifact refer. (3) As convergence and divergence is a side-effect of denormalizing a one-to-many reference during log extraction, the tables of each artifact type *are only be related by one-to-one references*. Timestamp attributes related to each other via one-to-many references should go into different artifact types.

From these principles, one obtains a set of artifact schemas as follows: First obtain the relational schema $\mathfrak{S}$ of the data source (either from documentation or by recovering it from the tables [20]). Partition the set of all tables in $\mathfrak{S}$ into maximal sets $\mathbb{T}_1, \ldots, \mathbb{T}_k$ such that all tables in each $\mathbb{T}_i$ are connected via one-to-one references only. In each $\mathbb{T}_i$, pick the table which has no incoming reference as the main table $T_{m,i}$[1]; $(\mathbb{T}_i, T_{m,i})$ is an *artifact schema*. For example, from the relational schema of Fig. 1, we obtain the 3 artifact schemas shown in Fig. 5 (top left); note that table *Changes* is initially not part of schema *SD*. It has to be added manually as we discuss later.

Any one-to-many reference is now between two different artifact schemas. This way, event types related to each other via one-to-many references are now separated into different artifacts; and convergence and divergence within one artifact can no longer occur. Behavioral dependencies arising from event types separated by one-to-many references will be expressed as *interactions between different artifacts*; see Sect. 4 and the overview in Fig. 4.

Artifact schemas are discovered based on structural properties only and might not fit domain knowledge. Thus, in a second step, a user may add or remove tables from a schema to obtain the intended artifact. This way, also one-to-many references may be included in an artifact schema at the potential cost of data convergence and divergence; see [19, Chap.2] for a detailed discussion. Moreover, to reverse vertical anti-partitioning where one table stores information of several artifacts, we explicitly allow artifact schemas to overlap in tables. In Fig. 5, artifact schema *SD* is extended with table *Changes*.

1. While the existence of a unique main table $T_{m,i}$ cannot be formally guaranteed for all relational schemas, previous studies and our own results suggest that such a table can always be found in practice [9], [24], [52], [7], [20].

3.4 Artifact Discovery and Refinement

From an artifact schema $\mathfrak{S}_A$ (a set of related tables), a generic artifact-type definition A of $\mathfrak{S}_A$ (detailing identifier, event-types, and related attributes, but without discriminating conditions) can be obtained automatically using the algorithm *CreateTraceMapping($\mathfrak{S}_A$)* of [20]. By this algorithm, the primary key of the main table of $\mathfrak{S}_A$ becomes the artifact identifier. Each time-stamped column C in a table T in $\mathfrak{S}_A$ becomes an event type E_C, every other non-timestamped column in T defines an attribute of event type E_C. Every non-timestamped column in any table in $\mathfrak{S}_A$ that cannot be related to one specific event type defines an artifact-level attribute. This generic artifact type needs to be refined to revert all operations of Sect. 3.1, as we show next.

Refining artifacts. In case $\mathfrak{S}_A$ contains information about *multiple* similar artifacts types (due to horizontal anti-partitioning), A has to be refined: create a copy of A for each different artifact type $A_1, \ldots, A_n$ and define a condition $\varphi_1, \ldots, \varphi_n$ over the artifact-level attributes of A that allows to select only records of the respective artifact type, e.g. *SD.[Document type] = 'Sales Order'*. In principle, the conditions $\varphi_1, \ldots, \varphi_n$ have to be the given by the user. However, in the presence of a discriminating column C holding finitely many values $v_1, \ldots, v_n$, such as *Document type* in table *SD* of Fig. 1, the conditions for each artifact type can be generated automatically as $C = v_1, \ldots, C = v_n$; the user only has to specify the name of the discriminating column. This can be generalized to multiple discriminating columns.

The resulting artifact type then should be refined by the user. For instance, by removing event types or attributes she is (currently) not interested in or which are side-effects of vertical anti-partitioning. Moreover, one event type can be refined into multiple event types by defining a discriminating condition over event-level attributes detailing the kind of event. In Fig. 5, column *Date changed* is refined into three event types based on the different values of discriminatory column *Change type*. A tool supporting these operations is shown in [53, Chap.6].

Handling generalization Identification of artifact schemas reverts vertical partitioning; manual refinements of artifact schema and artifact types as described above allows to revert horizontal and vertical anti-partitioning (but requires domain knowledge.) Reverting horizontal partitioning (i.e., specialization of a general entity into multiple tables) is similar to generalizing entities and highly depends on the given relational schema [54]; see Sect. B.4 for a detailed discussion.

3.5 Log Extraction and Life-Cycle Discovery

An artifact type essentially specifies how to extract an event log from the data source (each component refers to columns and attributes). In [46], [20] it is shown how to map an artifact schema to a *log extraction specification* for which the technique in [8] produces a number of SQL queries which extract artifact instances, events, and serializes them in an XES event log. The definitions of [46], [20] can be adapted for our artifact types: instead of extracting data from all columns of a table, only extract the columns specified in the artifact type, and for any discriminating condition φ append a *WHERE* φ clause in the extracting SQL query; see [53, Chap.4.3] for details. Fig. 6 shows the event log extracted from Fig. 1 using the artifact type definition *Sales Order* of Fig. 5.

Log	Name	Sales Order		
Trace	ID = S1, Document type = "Sales Order", value = 100			
	ID	name	timestamp	event attrs
Event e1	S1	Date created	16-5-2020	-
Event e2	1	Price updated	17-5-2020	Old value = "100", New value = "80"
Event e3	2	Delivery block released	19-5-2020	Old value = "x", New value = "-"
Event e4	3	Billing block released	19-5-2020	Old value = "x", New value = "-"
Event e5	S1	Last change	10-6-2020	-
Trace	ID = S2, Document type = "Sales Order", value = 200			
	ID	name	timestamp	event attrs
Event e1	S1	Date created	17-5-2020	-
Event e2	S1	Last change	31-5-2020	-

Fig. 6. An example of event log extracted for artifact *Sales Order*

The resulting event log of the artifact type can be given to any existing process discovery technique to discover a life-cycle model of that artifact. Different discovery techniques have been compared extensively on a conceptual and on empirical level [33].

One characteristic specific to artifacts is that, unlike in classical workflow processes, concurrency may be of secondary concern (i.e., if a business object may never be accessed concurrently by two users/processes at the same time, then discovering a transition system model [55] could prevent finding false concurrency). The subsequent interaction discovery requires that each event of an artifact is translated into (exactly one) action of the life-cycle model as otherwise interactions cannot be discovered properly. This assumption excludes algorithms that may discard certain events during discovery or that may duplicate

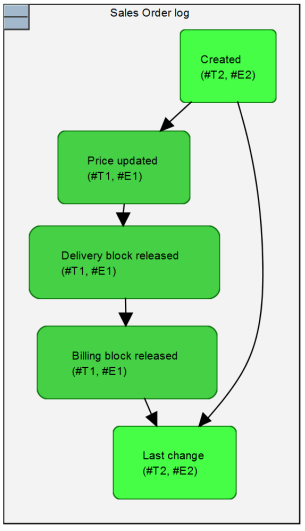


Fig. 7. The life-cycle discovered for the artifact *Sales Order*

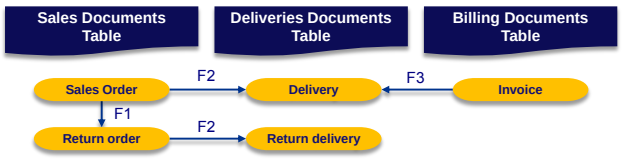


Fig. 8. The five artifact types of Fig. 1 and their type-level interactions (ARTIs).

tasks. We applied the flexible heuristic miner [27] in our evaluation (Sect. 5); applying this miner on the event log of Fig. 6 yields the model shown in Fig. 7.

Discussion. The presented artifact discovery technique heavily builds on earlier work [20]. That work only discovers abstract artifact schemas from which event logs are extracted directly, a schema may contain one-to-many relations giving rise to convergence and divergence. This work prevents one-to-many relations within an artifact schema, refines an artifact schema into artifact types (defined on attribute level) which can be refined further based on domain knowledge. Both approaches are compared in our evaluation in Sect. 5.

4 Interaction Discovery

In Sect. 3 we inferred *temporal relations* between the transactions of each individual business object (expressed as its life-cycle model). For this, we considered all timestamp values *structurally related* (via one-to-one relations) to the identifer of the business object.

Next, we refine the *structural one-to-many and many-to-many* relations between business objects into *temporal relations* between their transactions (expressed as interactions between life-cycle models). We outline the basic idea of our approach by an example (Sect. 4.1) and provide definitions and algorithms afterwards (Sect. 4.2-4.3).

4.1 Basic Idea

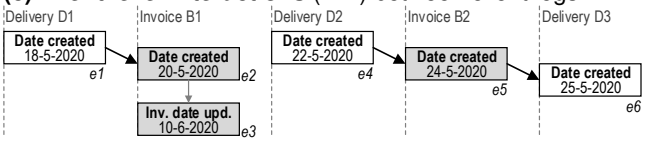
Consider the five artifact types shown in Fig. 8 identified from the tables of Fig. 1 (also using the *Document Type* attribute for refinement). Reference F_3 indicates that *Invoices* are related to *Deliveries*. Specifically, invoice B_1 is related to delivery D_1, and invoice B_2 is related to two deliveries D_2 and D_3. We call a structural reference between two artifact types an *artifact type level interaction* (ARTI) and each pair in the reference an *artifact instance level interaction* (ARI). Figure 9(a,b) summarizes both; we discuss how to detect ARTIs and ARIs in Sect. 4.2.

Our assumption is that a structural reference between (tables of) two different artifact types implies a *behavioral relation* between their instances. Thus, from the order of events in related artifact instances, we can infer temporal relations between event types of

(a) Artifact type level interactions (ARTI):
F_3 (Invoice $\rightarrow$ Delivery)

b) Artifact instance level interactions (ARI):
$B_1 \mapsto D_1$, $B_2 \mapsto D_2$, $B_2 \mapsto D_3$

(c) Event level interactions (EVI) between event logs:



(d) Event type level interactions (EVTI) between life-cycle models :

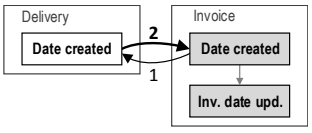


Fig. 9. Interaction discovery for the *Invoice* and *Delivery* artifacts of Fig. 1.

related artifact types. For example, Fig. 9(c) visualizes the extracted traces for invoices B_1, B_2 and for delivers D_1, D_2, D_3. Based on the ARIs, we consider trace B_1 together with trace D_1 and trace B_2 together with traces D_2 and D_3. Looking at the order of events *in different related traces*, we observe that $e2$ *directly follows* $e1$, $e5$ follows $e4$, and $e6$ follows $e5$. We call such ordering information *event level interactions* (EVI). By generalizing the ordering to event types, we obtain *event type level interactions* (EVTI) as shown in Fig. 9(d): the *Create* transaction for *Delivery* objects leads to a *Create* transaction for *Invoice* objects in two cases, but in one case the order is reversed. We discuss various ways to discover EVIs, EVTIs, and unusual flows in Sect. 4.3.

4.2 Interactions between Artifacts

Conceptually, any non-empty relation between the *main tables* of two artifact types is an *artifact type level interaction* (ARTI). We call the source of an ARTI the parent (e.g. *Invoices* of $F3$), and the target the child (e.g. *Delivery* of $F3$), though this structural ordering gives no indication of temporal ordering of events. Any pair of artifacts instances of an ARTI is an *instance level interaction* (ARI).

In practice, not all discovered artifact types are relevant in an analysis. For example, when analyzing the delivery and invoice documents of an SAP OTC process (see Fig. 10 for its ER-model) artifacts derived from *delivery lines* and *invoice lines* tables are irrelevant and shall be omitted. However, now the relation between delivery and invoice documents can no longer be analyzed as the connecting artifact interactions are omitted as well.

Indirect ARTIs. To allow omitting artifacts and yet study interactions of the remaining artifacts, we introduce *indirect* ARTIs. An *indirect* ARTI is a sequence

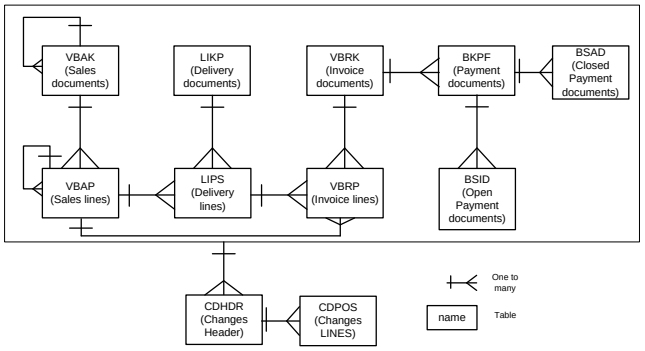


Fig. 10. ER-model of the SAP OTC process

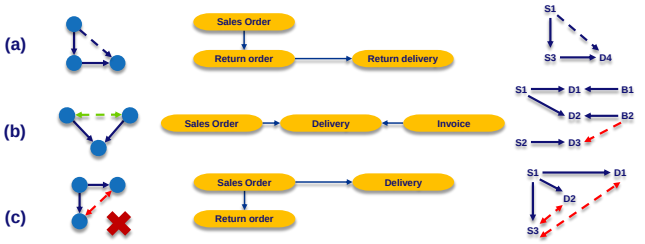


Fig. 11. Examples of strong (a), weak (b), and invalid (c) indirect ARTIs based on data of Fig. 1.

of direct ARTIs; the first (last) artifact in the sequence is the parent (child). There are three types of ARTIs as illustrated in Fig. 11. (a) In a *strong* ARTI, all references have the same direction, thus, each child instance has exactly one parent instance, e.g., $D4$ refers to only $S1$ via $S3$ and to no other instances. (b) In a *weak* ARTI, *one* intermediate artifact is the child of two direct ARTIs (i.e., reference direction changes), which allows that a child instance of the indirect ARTI has two or more parent instances (leading to over-approximation). For example, sales order $S1$ is linked to two invoices $B1$ and $B2$ while $B2$ is also linked to $S2$. (c) An ARTI is *invalid* if an intermediate artifact is the parent of two direct ARTIs. In this case the ARIs are arbitrary and unreliable, e.g., in Fig. 11 one cannot infer whether return order S3 is linked to D1 or D2 or both. Formal definitions of ARTIs and ARIs are given in Sect. C.1.

Discovering ARTIs. We discover direct and valid indirect ARTIs as follows. In the graph having the main tables of *all* artifact types as nodes and non-empty references between main tables as directed edge, each edge is a direct ARTI. We identify all strong indirect ARTIs by a depth-first-search on the graph along the directed edges; the user can prune search at depth m. Finally, all weak indirect ARTIs are identified by joining any two (direct or strong indirect) ARTIs that share the same child table; the user can limit the over-approximation in indirect ARTIs by restricting the second ARTI in the join to length $\leq k$. Further any found ARTI can be omitted if it contains $< r$ ARIs (to focus on frequent interactions only).

Enriching logs with ARIs. For each artifact type A, and each ARTI $I_{A,B}$ where A is the source, we can generate an SQL query joining the tables in the ARTI to obtain all ARIs. Then for each instance a of A we add to the trace of a (in the extracted event log) with an attribute listing the identifiers of all child instances of a. For example, the trace of *Invoice* $B2$ gets the attribute $interact : Delivery = \{D2, D3\}$. The algorithms are given in Sect. C.2.

4.3 Interactions between Event Types

Next, we refine the extracted ARI between two artifacts into behavioral relations between their event types. From a log perspective, if a trace t_a of an artifact instance a refers to an instance b of another artifact with trace t_b, we call the pair (t_a, t_b) *interacting traces*. Two interacting traces indicate that some events between them may interact. We discuss several techniques to classify that "two events interact." and to derive EVTIs. In particular, we distinguish frequent interactions and infrequent interactions and consider the infrequent ones as outliers; see Sect. C.3 for formal definitions.

Merge interacting traces. The first step for identifying whether events "interact" is to merge any two interacting traces: simply order the union of all their events by timestamp. Doing this for all interacting traces gives a merged log in which we can study the temporal order of events of two artifacts together.

Classifying interactions. We propose 5 different classification techniques. (1) Adapt the *directly-follows relation* (DF) [1]: an event e_A *DF-interacts* with e_B iff e_B directly follows e_A in a merged trace and e_B and e_A originate in different artifacts; the pair (e_A, e_B) is a *DF*-EVI. By projecting all DF-interactions to their event types, we obtain the corresponding *DF*-EVTI as illustrated in Fig. 9. (2) Alternatively, one could only consider those EVTI pairs with the maximum number of DF-EVIs in the merged log (*DF-max*-EVTI); other thresholds are possible as well. (3) Apply an existing process discovery algorithm $\mathcal{D}$ on the merged log L: in the resulting model $D(L)$ any direct causal relation between event types from different artifacts defines a $\mathcal{D}$-EVTI. (4) Absolute precedence (*AP*-EVTI): event type A *AP-interacts* with event type B iff in every trace of the merged event log, every event of type A occurs before any event of type B. (5) Shortest time between events (*ST*-EVTI): event type A *ST-interacts* with event type B iff events of type B occurs after events of type A, are from different artifacts, and the average time delay between events of A and events of B in the same trace is minimal among all pairs of event types. Although each merged log has only one *ST*-EVTI pair, we found this classifier useful for identifying the main EVTI and for hiding complexities when there are many interactions between two artifact

types (see Sect. 5). For each EvTI discovered, we simply add an edge between the two event types from the life-cycle models, which results in a complete artifact-centric process model showing artifacts' life-cycles and interactions between their transactions.

Classifying unusual flows. Classifiers (2)-(5) are designed to identify main flows of artifact interactions while classifier (1) identifies all interactions. Thus, by first computing all interactions using DF-EvTI and then removing main EvTIs using any of the other techniques one obtains the set of infrequent interactions (we found $\mathcal{D}$-EvTI to be most effective). In high-volume systems such as ERP systems, the infrequent flows are typically unusual flows that warrant further investigation.

5 Case Studies and Evaluation

The techniques are implemented as (1) a standalone tool based on [20] for artifact discovery and log extraction, and (2) a plugin to the Process Mining toolkit ProM (www.promtools.org) for discovering artifact life-cycles, their interactions, and unusual flows. The ProM plugin in particular allows to interactively change the view on an artifact-centric model by hiding certain event types or changing the interaction classifier; see [53, Chap.6] for details.

For our evaluation, we aimed at the following research questions. **(RQ1)** Do the returned models correctly describe the business objects in the source system? **(RQ2)** Do the returned models correctly describe main flow and unusual flows of transactions recorded in the system? **(RQ3)** Do the resulting models aid non-technical domain experts in understanding their data and draw conclusions about the data? We conducted two case studies using two real-life, production data sets from two different ERP systems. The first case study considered (RQ1) and (RQ2) and compared our approach to earlier work, the second case study considered (RQ2) and (RQ3).

5.1 Case I - Order To Cash in SAP

As this paper presents the first end-to-end approach to analyze *all* event data in an ERP system, we evaluated (RQ1) and (RQ2) wrt. different sets of techniques. For (RQ1), we compared the two artifact-centric approach of this paper ("this" in the following) and of [20]; for (RQ2), we compared this approach to classical log extraction [10], [8] and discovery which is the current standard in automated analysis of ERP event data ("classic").

Context and data. The first case study was performed for the Order to Cash (OTC) process supported by SAP systems; that process organizes orders, payments, and deliveries similar to our OTC running example, but has many variations supported by complex data structure [53, p.69]. The source data has been provided by KPMG and comprised 2 months of data in 11 tables of a production SAP OTC implementation; see Fig. 10 for the ER-model. In total, we considered 134,826 records of 5-49 columns (33 avg.).

	OTC process in SAP		PA process in Oracle	
Steps	Manual input	Results	Manual input	Results
Import	scope	11 tables&PKFK	scope	7 tables&PkFK
1.1		8 art. schemas		6 art. schemas
1.2	3 columns as input	35 artifacts		6 artifacts
2.1	k=2, m=1, r=1	? Interactions	k=2, m=1, r=1	7 interactions
1.2Refine	Only documents	18 artifacts	Scope selection	3 artifacts
2.1Refine	Only preceding relations	29 interactions		3 interactions
1.3&2.2		18 logs		3 logs
1.4-2.4	use HM	see Figure.	use HM	see Figure.

Fig. 12. Steps followed for both case studies

Setup. For both (RQ1) and (RQ2), all approaches took the entire original data set as input. We then compared the resulting models as follows. For (RQ1), a returned artifact is *correct* if it has a meaningful interpretation as a business object of the OTC process. We checked precision and recall of the resulting model wrt. correct artifacts based on expert knowledge. For (RQ2), a flow (an arrow from an event type A to an event type B) in the model is *correct* if any events of type A and B occurring in that order in the data source belong to (transitively) related object instances in the data source; this was checked by querying the data source. For the ability to distinguish main flow from unusual flow, we tested whether an unusual flow in the model contained false positives; this again required expert assessment. The authors with an affiliation to KPMG, acted as the experts, provided the requirements for the target model, and evaluated the results based on their expertise in ERP systems and data analytics.

Execution. We took the following steps. (This) we followed our approach of Fig. 4; Fig. 12 shows the parameters of each step; only document-level artifacts were considered; DF-EvTI and $\mathcal{D}$-EvTI with [27] were used to find unusual flows. (Classic) We chose the *sales order* identifiers as case identifiers (as creation of a sales orders is the starting point of the OTC process); all timestamp attributes of document-level tables with a (transitive) relation to the case identifier were included in the log extraction; during log extraction, any event which was indirectly related to a sales order instance was added exactly once to the trace of this sales order. ([20]) We imported the relational schema also used in our approached and identified artifact schemas with k-means clustering starting at $k = 2$ and incremented until no new artifact schemas were found (at $k = 10$); all results were considered in the analysis. See [19, Chap.7.3] for details.

Results for (RQ1). (This) We obtained 18 artifact types connected by 29 ArTIs; these corresponded precisely to the 18 document level business objects classified by the experts. ([20]) As the approach returns clusters of tables but cannot separate records within a table, the

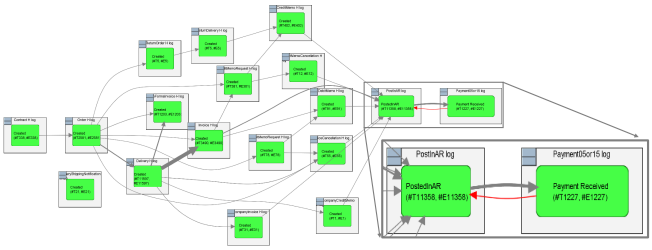


Fig. 13. SAP OTC process - artifact-centric model with outliers

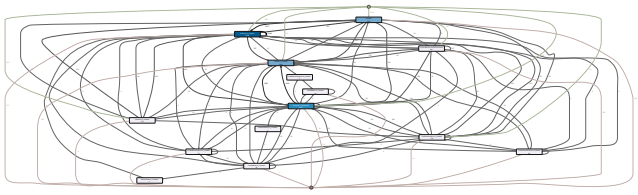


Fig. 14. SAP OTC process case study - a life-cycle of sales orders obtained using a classical log conversion approach.

resulting model essentially describes artifact schemas instead of artifact types. For $k = 9$, we correctly identified 7 of the 8 artifact schemas but also obtained two incorrect artifact schemas which did not contain business objects but the change records of the actual objects. Further, one artifact schema contained two different business objects connected by a one-to-many reference. For $k < 9$ more business objects were grouped into the same artifact schema; for $k > 9$, empty artifact schemas were returned instead of refining existing clusters with one-to-many relations. Sect. E provides details.

Results for (RQ2). (This) We obtained the model shown in Fig. 13 which classifies one unusual flow highlighted by the red arc. By construction of the technique, all flows in the model are correct (as flows are only identified from events in related tables). The unusual flow from *Payments Received* to *Invoice Created* was unanticipated and indicates that there were payments received before the corresponding invoices were created in the account receivable, which arouse our interest. We validated the unusual flow by looking into the database (the SQL query used can be found in [53, p.76]) and found that for the cases that caused this flow, the database indeed contained a *Payment Received* date earlier than the *Posted In AR* date. Further investigation revealed that the cases was (indeed) manually changed by someone using the transaction code FB05, which is used to apply for cash (automated or manually) incurring risks. Thus, we automatically detected an unexpected but true, unusual action. No other unusual flows were found in the data source. The flows described by this model were assessed as accurate by the experts.

(Classic) Fig. 14 shows the process model obtained from the classically extracted event log obtained with Fluxicon Disco (www.fluxicon.com/disco) which is based on the directly-follows relations in the log. The model is rather convoluted and unreadable due to data convergence and divergence and contains many incorrect flows. For instance, 9 out of the 14 event types have a self-loop, which are false positives, as no event in the original data set is directly related to another event of the same type. To quantify the false positives, we compared the directly-follows relations of the extracted event log to the directly-follows relations (arcs between event types) of the artifact-centric model in Fig. 13 (assuming ground truth for the latter). We found that 6696 out of 13644 directly-follows pairs to be incorrect, i.e., about 50%. This resulted in 36 out of the 79 arcs in Fig. 14 to be incorrect.

We then located events of directly-follows pairs of the classic log that are incorrect wrt. model of Fig. 13 in the original data source; we checked whether the business object instances to which these events belong are structurally related in the data model; in all cases, the business objects had no relation. Hence, none of the additional flows identified by the classic approach was correct. This also implies that the artifact-centric model did not miss any correct flow. We also observed wrong statistics: while the original data source contains 338 *contract* objects, the extracted log contains *741 contract created* events due to event duplication. Sect. E provides details.

Discussion. Our approach allows to correctly identify all business objects of the process in terms of artifacts; the existing approach failed both due to a conceptual limitation (cannot separate records in a table) and in precision and recall. As a limitation, our technique requires expert knowledge to correctly define all artifacts in the first place. However, once structural artifact types are defined, the resulting artifact-centric model correctly classifies main flows and unusual flows. No false positives or false negatives were found. Sect. E.3 reports on an explorative analysis where we investigated the impact of complex life-cycle models with many event types on the interactions between artifacts.

5.2 Case II - Project Administration in Oracle

The second case study aimed at (RQ2) and (RQ3) had been performed for the project administration (PA) process supported by the Oracle ERP system of an educational organization (called "client" in the following).

Context and data. The Project Administration (PA) process supported by Oracle Enterprise ERP system facilitates thousands of projects running by the educational organization, e.g. research projects. In short, the

PA process starts with creating *projects* in the system. During the execution of the project, *tasks* are created for the project to declare different *expenditures* related to a task, e.g. personnel, materials. For assessing financial risks, *expenditures* should end before *tasks* are completed and before corresponding *projects* are closed. The source data comprised 7 tables over in total 16805 records and 134 columns.

Setup. We applied the artifact-centric approach to discover a fixed set of artifacts, life-cycle models, and artifact interactions. Using our interactive ProM plugin [53, Chap.6.2], we then explored the model and produced different views highlighting different flows. Unusual flows were identified both with the tool and by visual inspection on the model; the correctness of these flows was then discussed with the client in an interactive session of one hour. As the client was unfamiliar with the notation of the model, the client was first trained to read the model before we validated findings and gathered feedback.

Execution. The steps followed in our approach are shown in Fig. 12. Documentation about the data schema (esp. primary keys and foreign keys) was available. For the log extraction step, we considered only the artifacts *Projects*, *Tasks* and *Expenditures* (as these are the primary objects in the process), each containing 1132, 1236 and 3100 instances, respectively. The three event logs were imported into ProM. The heuristic miner was used for the artifact-centric process discovery; see Sect. E.4 for details.

Results. In total 5 different views on the artifact-centric model were generated; Fig. 15 shows the view highlighting the main interactions obtained using the *DF-max*-EvTI classifier; see [53, Chap.7.2] for all other views. On the 5 views, we identified in total 10 unusual flows. 7 flows were classified as unusual by our technique. 2 unusual flows were identified by the experts from KPMG by visual inspection of the model. 1 unusual flow was identified by the client themselves during the discussion by visual inspection of the model.

All identified unusual flows were confirmed as *correct* wrt. the source data, i.e., no false positives were identified. Four of the unusual flows were explained by the client as rare cases which can happen in the process; for two unusual flows the client indicated that the intended process may have not been followed and further investigation was required; one unusual flow could be traced to an implementation detail of the Oracle system; one unusual flow, though correct wrt. the source data, could neither be explained by the expert nor by the client; finally, two unusual flows were identified due to inaccuracies in the source data (i.e., some timestamps had only granularity of entire days). Fig. 15 shows an unusual flow identified by experts and confirmed as a rare case in the process

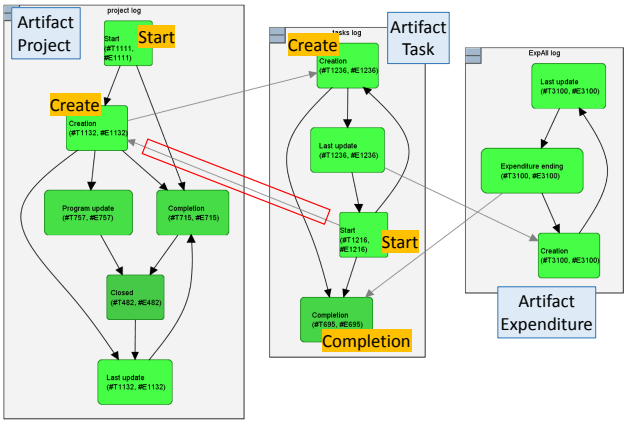


Fig. 15. A proclet system discovered by using the *DF-max*-EvTI classifier

by the client. Two of the unusual flows are discussed in more detail in Sect. E.5; a detailed discussion on all flows is available in [53, Chap.7.2].

The discussion of unusual flows led to a further analysis question which required to explore the data further and refine the *Expenditure* artifact based on different expenditure types; findings are reported in Sect. E.6.

Discussion. We could show that the approach is generic and can be applied on a different process supported by a different ERP system. We were able to use the discovered artifact-centric models to help experts and clients communicate and identify deviations in their real processes. All identified unusual flows were correct (though two unusual flows occurred due to inaccurate data). Moreover, a relatively short training of less than one hour sufficed to enable a domain expert unfamiliar with the notation to identify unusual flows on their own, even in the case where an unusual flow is not identified automatically but has to be recognized by the user. The lessons learned for conducting such an artifact-centric analysis have been summarized as a methodology in [19, Chap.6].

6 Conclusion

In this paper, we addressed the problem of discovering a process model from event data of stored in a relational data source, in particular event data of ERP systems. We proposed to discover a model that describes the process as a set of interacting data objects (of a process), each following its own life-cycle, also called *artifacts*. We presented a semi-automatic, end-to-end approach to identify artifacts in a relational data source and extract a life-cycle event log for each identified artifact. From each log, a life-cycle model of this artifact can be identified using existing process discovery techniques. Second, we provide, for the first time, a family of techniques to discover causal dependencies, between artifacts at the

type level and at the event level. This information can be visualized as interactions between the extracted artifact life-cycle models. We validated our approach in two case studies using real-life data from ERP systems. In the case studies, the discovered models accurately describe the real executions of the recorded business processes. The case studies also show that the discovered models provide useful insights into the processes and allow users to identify unusual flows of executions.

Future Research. This paper made a first step towards a fully discovery of artifact-centric process models from a relational data source. Currently, our approach for discovering artifacts still needs manual steps such as indicating a column for splitting the artifacts or splitting the event types. More advanced algorithms can be developed to identify the "perfect" artifact automatically by using, for example, metrics and heuristics. Furthermore, we considered the line level artifacts (e.g. sales order lines) as separate artifacts in our case study and omitted them from the log extraction. It would be interesting to investigate the hierarchy of artifacts; for example, supporting the discovery of sub-artifacts within artifacts. A limitation of the current interaction discovery is that it is limited to two artifacts. We would like to discover the interaction flow between multiple artifacts by merging multiple artifacts for example.

ACKNOWLEDGMENTS

We thank B.F. van Dongen and H.M.W. Verbeek for their substantial support in writing this paper. We also thank W. van Kessel (KPMG) for his substantial support in analyzing the Oracle case study.

REFERENCES

[1] W. v. d. Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[2] M. van Eck, X. Lu, S. Leemans, and W. van der Aalst, "Pm^2: a process mining project methodology," in *CAiSE 2015 (accepted)*, 2015.

[3] S. Suriadi, M. T. Wynn, C. Ouyang, A. H. ter Hofstede, and N. J. van Dijk, "Understanding Process Behaviours in a Large Insurance Company in Australia: A Case Study," in *Advanced Information Systems Engineering*. Springer, 2013, pp. 449–464.

[4] M. Al-Mashari and M. Zairi, "Supply-chain re-engineering using enterprise resource planning (erp) systems: an analysis of a sap r/3 implementation case," *International Journal of Physical Distribution & Logistics Management*, vol. 30, no. 3/4, pp. 296–313, 2000.

[5] A. D. Brucker and I. Hang, "Secure and compliant implementation of business process-driven systems," in *Business Process Management Workshops*. Springer, 2013, pp. 662–674.

[6] M. v. Giessel, "Process Mining in SAP R/3: A method for applying process mining to SAP R/3," Master's thesis, Eindhoven University of Technology, 2004.

[7] I. Segers, "Investigating the application of process mining for auditing purposes," Master's thesis, Eindhoven University of Technology, 2007.

[8] J. Buijs, "Mapping data sources to xes in a generic way," Master's thesis, Eindhoven University of Technology, 2010.

[9] D. Piessens, "Event Log Extraction from SAP ECC 6.0," Master's thesis, Eindhoven University of Technology, 2011.

[10] A. Roest, "A Practitioners Guide Towards Process Mining on ERP Systems - Implemented and Tested for SAP Order to Cash," Master's thesis, Eindhoven University of Technology, 2012.

[11] A. Nigam and N. Caswell, "Business artifacts: An approach to operational specification," *IBM Systems Journal*, vol. 42, no. 3, pp. 428–445, 2003.

[12] D. Cohn and R. Hull, "Business artifacts: A data-centric approach to modeling business operations and processes," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 32, no. 3, pp. 3–9, 2009.

[13] R. J. Miller and P. Andritsos, "Schema discovery," *IEEE Data Eng. Bull.*, vol. 26, no. 3, pp. 40–45, 2003.

[14] J. Turmo, A. Ageno, and N. Català, "Adaptive information extraction," *ACM Comput. Surv.*, vol. 38, no. 2, 2006.

[15] S. Sarawagi, "Information extraction," *Foundations and Trends in Databases*, vol. 1, no. 3, pp. 261–377, 2008.

[16] V. M. Markowitz and J. A. Makowsky, "Identifying extended entity-relationship object structures in relational schemas," *IEEE Trans. Software Eng.*, vol. 16, no. 8, pp. 777–790, 1990.

[17] R. H. L. Chiang, T. M. Barron, and V. C. Storey, "Reverse engineering of relational databases: Extraction of an EER model from a relational database," *Data Knowl. Eng.*, vol. 12, no. 2, pp. 107–142, 1994.

[18] R. Alhajj, "Extracting the extended entity-relationship model from a legacy relational database," *Inf. Syst.*, vol. 28, no. 6, pp. 597–618, 2003.

[19] X. Lu, M. Nagelkerke, D. van de Wiel, and D. Fahland, "Discovering Interacting Artifacts from ERP Systems (Ext. Version)," bpm-center.org, BPM Center Report BPM-15-08, 2015.

[20] E. Nooijen, B. v. Dongen, and D. Fahland, "Automatic Discovery of Data-Centric and Artifact-Centric Processes," in *Business Process Management Workshops*. Springer, 2013, pp. 316–327.

[21] C. Yu and H. V. Jagadish, "Schema summarization," in *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*. ACM, 2006, pp. 319–330.

[22] V. Popova, D. Fahland, and M. Dumas, "Artifact lifecycle discovery," *International Journal of Cooperative Information Systems, World Scientific.*, 2014 (to appear).

[23] J. E. Ingvaldsen and J. A. Gulla, "Preprocessing support for large scale process mining of sap transactions," in *Business Process Management Workshops*. Springer, 2008, pp. 30–41.

[24] A. Ramesh, "Process mining in peoplesoft," Master's thesis, Eindhoven University of Technology, 2006.

[25] K. Yano, Y. Nomura, and T. Kanai, "A practical approach to automated business process discovery," in *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2013 17th IEEE International*. IEEE, 2013, pp. 53–62.

[26] W. v. d. Aalst, A. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 16, no. 9, pp. 1128–1142, 2004.

[27] A. Weijters and J. Ribeiro, "Flexible heuristics miner (fhm)," in *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*. IEEE, 2011, pp. 310–317.

[28] A. d. Medeiros, A. Weijters, and W. v. d. Aalst, "Genetic process mining: an experimental evaluation," *Data Mining and Knowledge Discovery*, vol. 14, no. 2, pp. 245–304, 2007.

[29] J. v. d. Werf, B. v. Dongen, C. Hurkens, and A. Serebrenik, "Process discovery using integer linear programming," in *Applications and Theory of Petri Nets*. Springer, 2008, pp. 368–387.

[30] C. Günther and W. v. d. Aalst, "Fuzzy mining–adaptive process simplification based on multi-perspective metrics," in *Business Process Management*. Springer, 2007, pp. 328–343.

[31] S. J. J. Leemans, D. Fahland, and W. M. P. v. d. Aalst, "Discovering block-structured process models from event logs-a constructive approach," in *Application and Theory of Petri Nets and Concurrency*. Springer, 2013, pp. 311–329.

[32] ——, "Discovering block-structured process models from non-conforming event logs," in *In 9th International Workshop on Business Process Intelligence 2013 (BPI), Beijing, China*, 2013.

[33] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens, "A multi-dimensional quality assessment of state-of-the-art

process discovery algorithms using real-life event logs," *Information Systems*, vol. 37, no. 7, pp. 654–676, 2012.
[34] D. Lo and S. Khoo, "Smartic: towards building an accurate, robust and scalable specification miner," in *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2006, Portland, Oregon, USA, November 5-11, 2006*, 2006, pp. 265–275.
[35] M. Gabel and Z. Su, "Javert: Fully automatic mining of general temporal properties from dynamic traces," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. SIGSOFT '08/FSE-16, 2008, pp. 339–349.
[36] D. Fahland, D. Lo, and S. Maoz, "Mining branching-time scenarios," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, 2013, pp. 443–453.
[37] M. Pradel and T. Gross, "Automatic generation of object usage specifications from large method traces," in *Automated Software Engineering, 2009. ASE '09. 24th IEEE/ACM International Conference on*, Nov 2009, pp. 371–382.
[38] A. Zeller, "Specifications for free," in *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*, 2011, pp. 2–12.
[39] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A systematic survey of program comprehension through dynamic analysis," *Software Engineering, IEEE Transactions on*, vol. 35, no. 5, pp. 684–702, Sept 2009.
[40] J. Lou, Q. Fu, S. Yang, J. Li, and B. Wu, "Mining program workflow from interleaved traces," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, 2010, pp. 613–622.
[41] J. Pinggera, P. Soffer, D. Fahland, M. Weidlich, S. Zugal, B. Weber, H. Reijers, and J. Mendling, "Styles in business process modeling: an exploration and a model," *Software & Systems Modeling*, pp. 1–26, 2013.
[42] R. Minelli, A. Mocci, M. Lanza, and L. Baracchi, "Visualizing developer interactions," in *Software Visualization (VISSOFT), 2014 Second IEEE Working Conference on*, Sept 2014, pp. 147–156.
[43] M. Schur, A. Roth, and A. Zeller, "Mining behavior models from enterprise web applications," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013, 2013, pp. 422–432.
[44] D. Fahland, M. d. Leoni, B. v. Dongen, and W. v. d. Aalst, "Behavioral conformance of artifact-centric process models," in *Business Information Systems*. Springer, 2011, pp. 37–49.
[45] ——, "Conformance checking of interacting processes with overlapping instances," in *Business Process Management*. Springer, 2011, pp. 345–361.
[46] E. Nooijen, "Artifact-Centric Process Analysis–Process discovery in ERP systems." Master's thesis, Eindhoven University of Technology, 2012.
[47] R. Hull, E. Damaggio, R. D. Masellis, F. Fournier, M. Gupta, F. T. Heath, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculn, "Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events." in *DEBS*, *ACM*, 2011, pp. 51–62.
[48] A. Petermann, M. Junghanns, R. Muller, and E. Rahm, "Biiig: enabling business intelligence with integrated instance graphs," in *Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on*. IEEE, 2014, pp. 4–11.
[49] R. Conforti, M. Dumas, L. García-Bañuelos, and M. La Rosa, "Beyond tasks and gateways: Discovering bpmn models with subprocesses, boundary events and activity markers," in *Business Process Management*. Springer, 2014, pp. 101–117.
[50] A. Wasylkowski, A. Zeller, and C. Lindig, "Detecting object usage anomalies," in *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC-FSE '07. ACM, 2007, pp. 35–44.
[51] A. Mocci and M. Sangiorgio, "Detecting component changes at run time with behavior models," *Computing*, vol. 95, no. 3, pp. 191–221, 2013.
[52] J. E. Ingvaldsen and J. A. Gulla, "Preprocessing support for large scale process mining of SAP transactions," in *Business Process Management Workshops, BPM 2007 International Workshops, BPI, BPD, CBP, ProHealth, RefMod, semantics4ws, Brisbane, Australia, September 24, 2007, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 4928. Springer, 2007, pp. 30–41.
[53] X. Lu, "Artifact-Centric Log Extraction and Process Discovery," Master's thesis, Eindhoven University of Technology, 2013.
[54] D. Embley and B. Thalheim, *Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges*. Springer, 2012.
[55] W. M. P. van der Aalst, V. Rubin, H. M. W. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther, "Process mining: a two-step approach to balance between underfitting and overfitting," *Software and System Modeling*, vol. 9, no. 1, pp. 87–111, 2010.

Xixi Lu M.Sc. is a PhD candidate at the Eindhoven University of Technology researching in the areas of architecture of business information systems and process mining. She received her Master of Science degree in Business Information Systems from the Eindhoven University of Technology. She also studied at Tsinghua university as an exchange student. Her research interests include business process management, conformance checking (using Petri nets and recorded event logs), deviation analysis, and process mining in general.

Marijn Nagelkerke M.Sc. RE is advisor in the KPMG IT Advisory practice in The Netherlands. Marijn has a Master of Science degree in Business Information Systems from the Eindhoven University of Technology. As an IT auditor & advisor he is involved in supporting the financial statement audit which has a main focus on the detection of deviations in client processes and compliance with financial regulations. Marijn is a member of KPMG's Center of Excellence on Data Analytics in which he applies data analytics on financial and IT systems. His main expertise is on data analytics and process mining and he specializes on processes supported by Oracle e-Business Suite/Financials.

Dennis van de Wiel M.Sc. RE is manager in the KPMG IT Advisory practice in The Netherlands. Dennis has a Master of Science degree in Econometrics from Maastricht University. As an IT audit manager he is involved in supporting the financial statement audit and on advising clients on how to further improve their processes and leverage their IT systems & investments. Dennis is lead of KPMG's Center of Excellence on Data Analytics which services over 6 KPMG countries in data analytics on financial and IT systems. His main expertise is on processes supported by the SAP ERP system, and Dennis specializes in large complex implementations at mainly corporate clients of KPMG.

Dirk Fahland Dr. is assistant professor at the Eindhoven University of Technology researching in the area of distributed systems, he received his Ph.D. in Computer Science from the Humboldt-Univeristät zu Berlin, Germany, and the Eindhoven University of Technology, the Netherlands, in 2010. His research interests include distributed processes and systems built from distributed components for which he investigates modeling systems (using process modeling languages, Petri nets, or scenario-based techniques), analyzing systems for errors or misconformances (through verification or simulation), and process mining/specification mining techniques for discovering system models from event logs. He particularly focuses on distributed system with multi-instance characteristics and their synchronizing and interacting behaviors. His results appeared in journals such as Software & Systems Modeling, The Computer Journal, Data and Knowledge Engineering, and Information Systems.

APPENDIX A
PROBLEM CONTEXT

As today's business are becoming more process-driven, service-oriented infrastructures supporting business executions frequently are organized several layers:

- business object layer modularizes functionalities and increases their re-usability;
- business process layer orchestrates functionalities to deliver end-to-end business services and values; and
- user interface layer handles interactions between user and system.

ERP systems such SAP and Oracle Enterprise are examples of such a service-oriented infrastructure. SAP has many modules where each module handle particular business objects; access to these objects is encapsulated in services. The high-level end-to-end processes in an SAP system invoke various services to update business objects which trigger state changes in the life-cycles of a business object and interactions (or service invocations) between these objects. For example, Order to Cash process of SAP can invoke services from modules like Sales & Distribution (SD), Production Planning (PP), Material management (MM) and Finance & Controlling (FICO) and uses business objects provided by these modules to execute the end-to-end process that starts from (1) receiving orders from customers to (2) manufacturing the products to (3) delivering them to customers, and finally (4) receiving payments from customers.

These modules and the business object layer remain the same but depending on the company, the business process that is executed may be configured differently. For example, most webshop companies do not manufacture and therefore do not use Production Planning modules or related objects. Moreover, even in a configured process of a company, not all executions invoke the same services of the SAP system.

In this paper, we are reconstructing these business processes and the service calls that were executed in reality, from transactional data recorded during the execution, by discovering the used business objects, their life-cycles and their interdependencies.

The discovered end-to-end processes can be used to analyze for example how the business objects are used and whether the current way of using are desirable. This helps business users to understand how their processes are running in the reality and improve their processes.

APPENDIX B
ARTIFACT DISCOVERY – TECHNICAL DETAILS

B.1 Preliminaries - Relational Data

The relational concepts used, i.e. table, column, reference and data schema, are listed in this section.

Definition 1 (Table, Column). $\mathbb{T} = \{T_1, \cdots, T_n\}$ *is a set of tables of a data source, where each* table $T_i = \langle \mathbb{C}, \mathbb{C}_p \rangle$ *is a tuple of its* columns $\mathbb{C}$ *and its* primary keys $\mathbb{C}_p$.

In the OTC example, there are four tables, each of which has one column as primary key, i.e. $\mathbb{T}$ = {SD, DD , BD, Changes} and e.g. table SD = ⟨{SD id, Date Created, Reference id, Document Type, Value, Last change }, {SD id}⟩.

Definition 2 (Reference). $F = \langle T_p, \mathbb{C}_p, T_c, \mathbb{C}_c, F_{condition} \rangle$ *is a reference if and only if*

- T_p *is the* parent table ,
- $\mathbb{C}_p$ *is an ordered subset of columns denoting the primary key of the parent table,*
- T_c *is the* child table,
- $\mathbb{C}_c$ *is an ordered subset of columns denoting the foreign key, and*
- $F_{condition}$ *is the extra condition for the reference (which can be appended in the FROM part or the WHERE part of an SQL query).*

The condition $F_{condition}$ reflects the as-is situation in various ERP systems such as SAP where $\mathbb{C}_c$ only is a proper reference to an entry in T_p if that entry has a particular value in particular column of T_p. For example, the foreign key F_4 can be defined by three references, and one of these references is ⟨ [SD], {SD id}, [Changes], {Reference Id}, " [Changes].[Table name] = "SD " "⟩. The condition $F_{condition}$ could be empty indicating $F_{condition}$ is true.

Definition 3 (Data schema). $\mathfrak{S} = \langle \mathbb{T}, \mathbb{F}, \mathbb{D}, column\_domain \rangle$ *is a data schema with:*

- $\mathbb{T}$ *is a set of the tables with the primary keys of each table filled in;*
- $\mathbb{F}$ *is a set of references between the tables;*
- $\mathbb{D}$ *is a set of domains; and*
- $column\_domain$ *that assigns each column a domain.*

B.2 Artifact Schema Identification

The formal definition of artifact schema and the algorithm for computing artifact schemata are listed.

Definition 4 (Artifact Schema). $\mathfrak{S}_A = \langle \mathbb{T}_A, \mathbb{F}_A, \mathbb{D}_A, column\_domain, T_m \rangle$ *is an artifact-schema if and only if* $\mathfrak{S}_A$ *a subset of the schema* $\mathfrak{S} = \langle \mathbb{T}, \mathbb{F}, \mathbb{D}, column\_domain \rangle$*, i.e.,*

- $\mathbb{T}_A \subseteq \mathbb{T}$ *is a subset of tables;*
- $\mathbb{F}_A \subseteq \mathbb{F}$ *is a subset of reference;*
- $\mathbb{D}_A \subseteq \mathbb{D}$ *is a subset of domains;*
- $column\_domain$ *is the assignment function of the schema; and*
- $T_m \in \mathbb{T}_A$ *is the main table in which the trace identifiers can be found.*

Algorithm *ComputeArtifactSchemas*($\mathfrak{S}$)
1. Let a graph $G = (\mathbb{T}_G, \mathbb{F}_G) \leftarrow (\mathfrak{S}.\mathbb{T}, \mathfrak{S}.\mathbb{F})$

2. **for** $F \in \mathbb{F}_G$
3. **do if** (F is not one-to-one)
4. **then** remove F from $\mathbb{F}_G$
5. **for** a connected sub graph $g = (\mathbb{T}_g, \mathbb{F}_g) \subseteq G$
6. **do** $\mathbb{T}_A \leftarrow \mathbb{T}_g$, $\mathbb{F}_A \leftarrow \mathbb{F}_g$,
7. select a table $T_m \in \mathbb{T}_A$ which has no parent table in $\mathbb{T}_A$
8. $\mathbb{D}_A \leftarrow$ the union set of domains of columns of the tables in $\mathbb{T}_A$
9. create a new artifact schema $\mathfrak{S}_A = \langle \mathbb{T}_A, \mathbb{F}_A, \mathbb{D}_A, \mathfrak{S}.column\_domain, T_m \rangle$
10. add artifact schema $\mathfrak{S}_A$ to $\mathbb{S}$
11. **return** $\mathbb{S}$

The starting point for finding artifact schemas in the relational data source is its schema $\mathfrak{S}$. From this graph, the references which are not one-to-one are removed, thus resulting in a graph only connected by one-to-one references. Each of the resulting connected sub-graphs can be considered as valid artifact schemas as it only contains tables which are linked by one-to-one references. The main table T_m can be selected as a table which has no parent in the set $\mathbb{T}_A$ of the selected tables. The set $\mathbb{D}_A$ of domains is the union set of all domains of columns of the tables in $\mathbb{T}_A$. We can obtain an artifact schema $\mathfrak{S}_A$ and add it to the set $\mathbb{S}$ to be returned.

B.3 Artifact Identification

The formal definitions of *event type* and *artifact type* are presented in this section. In addition, an example of an artifact type is shown in Tab. 1

Definition 5 (Event types). $E_i = \langle E_{name}, \mathbb{C}_{Eid}, C_{time}, \mathbb{C}_{Eattrs}, E_{condition} \rangle \in \mathbb{E}$ *is an event type if and only if:*

- E_{name} *is the name of the event type;*
- $\mathbb{C}_{Eid}$ *is a set of columns defining the event identifier;*
- C_{time} *is the column indicating the ordering (or the timestamps) of events of this event type;*
- $\mathbb{C}_{Eattrs}$ *is a set of columns denoting the attributes of the event type; and*
- $E_{condition}$ *is a condition (which can be appended in the FROM part or the WHERE part of an SQL query) to distinguish various event types stored in the same column* C_{time} *of the data source.*

Definition 6 (Artifact types). $A = \langle A_{name}, \mathbb{C}_{Aid}, \mathbb{E}, \mathbb{C}_{attrs}, \mathbb{I}, \mathfrak{S}_A, A_{condition} \rangle$ *is an artifact if and only if:*

- A_{name} *is and artifact name;*
- $\mathbb{C}_{Aid}$ *is a set of columns denoting the case identifier of the artifact;*
- $\mathbb{E}$ *is a set of event types;*
- $\mathbb{C}_{attrs}$ *is a set of columns denoting the case attributes,*
- $\mathbb{I}$ *is a set of interactions between this artifact* A *and other artifacts (which remains as an empty set in this section);*
- $\mathfrak{S}_A$ *is the corresponding artifact schema; and*
- $A_{condition}$ *is the an artifact condition which is an extra condition (which can be appended in the FROM part or the WHERE part of an SQL query) that is used to distinguish various artifacts having the same main table* T_m *(or having the same artifact schema).*

TABLE 1
An example of the Sales Order artifact

Artifact's component	Value
A_{name}	Sales Order
$\mathbb{C}_{Aid}$	{ SD id }
$E_1 \in \mathbb{E}$	$\langle E_{name} =$ date created, $\mathbb{C}_{Eid} =$ {SD id}, $C_{time} =$ [date created], $\mathbb{C}_{Eattrs} = \{\}$, $E_{condition} = \emptyset \rangle$
$E_2 \in \mathbb{E}$	$\langle E_{name} =$ last change, $\mathbb{C}_{Eid} =$ {SD id}, $C_{time} =$ [latest change], $\mathbb{C}_{Eattrs} = \{\}$, $E_{condition} = \emptyset \rangle$
$\mathbb{C}_{attrs}$	{ [Document type], [Value]}
$\mathbb{I}$	$\emptyset$
$A_{condition}$	T_m.[Document type] = 'Sales Order'

B.4 Handling Generalization

In this section, we discuss how to handle horizontal partitioning in the data source, that is, when information about a conceptual general artifact is not stored as such, but has been distributed over many different tables. For example in Fig. 1, one could be interested in extracting a general "Documents" artifact rather than one artifact for different document types.

Generalizing different artifact types into one general artifact is similar to generalizing entities and highly depends on the given relational schema [54].

1) The specialization is materialized in the relational schema by a discriminating attribute. In this case all artifact types are found in the same tables, and hence will be contained the same artifact schema. When defining the artifact type, one simply specifies are more general discriminating condition $A_{condition}$ in Def. 6. The resulting general artifact type then contains more or even all event types and attributes in the artifact schema.
2) The specialization is materialized as an "IS-A" relationship with a "general table" and foreign keys from its specializations. In this case, the general table and all specializations of interest become part of the artifact schema, and the general table is chosen as main table. Artifact type definition proceeds as described above.
3) The specialization is materialized as separate tables without an "IS-A" relationship. In this case no generalizing main table for the different specializations can be defined. Two solutions are possible. (1) One can first extract the life-cycle event log for each specialized artifact, and then

merge the resulting event logs into one generalized event log. Prefixing values of identifier attributes prevents collisions of different specializations. (2) For the purpose of the analysis, one could transform a copy of the original relational source, for example by introducing an "IS-A" relationship with appropriate foreign keys.

APPENDIX C
INTERACTION DISCOVERY – TECHNICAL DETAILS

In this section, the formal definitions and algorithms for discovering interactions between artifacts are given.

C.1 Interaction Types and Definitions

We provide formal definitions of ARTIs and ARIs.

Definition 7 (Direct artifact type level interaction, Parent artifact, Child artifact). *Let $\mathbb{A}$ be a set of artifacts and $\mathbb{F}$ a set of references. $(A_S, F, A_T) \in \mathbb{A} \times \mathbb{F} \times \mathbb{A}$ is a direct, artifact type level interaction between the two artifacts A_S and A_T, if and only if, (1) $F = \langle T^S_m, \mathbb{C}^S_{Aid}, T^T_m, \mathbb{C}^T_c, S_{condition}\rangle$ is a reference from some attributes $\mathbb{C}^T_c$ of the main table of A_T to the artifact identifier $\mathbb{C}^S_{Aid}$ of A_S, and (2) there has to be an entry s in T^S_m referring to an entry t in T^T_m that s satisfies the condition $A_S.A_{condition}$, and t satisfy the condition $A_T.A_{condition}$.*

We denote the artifact A_S as the parent artifact *and the artifact A_T as the* child artifact.

For example, $(A_{SalesOrder}, F_2, A_{ReturnOrder})$ is a direct, artifact type level interaction.

The *direct* ARIs are instantiated from the interactions on the type level by joining the main tables based on direct type level interactions. Below, an SQL query is shown which select the direct instance level interactions of the direct type level interaction $(A_{SalesOrder}, F_2, A_{ReturnOrder})$. The resulting instance level interaction is $(S1, S3)$. Note that the query can be generated automatically using the artifacts and references. For technical details, we refer to [53, Chap.4].

```
SELECT DISTINCT
 SalesOrder.[SD ID] AS [SO id],
 ReturnOrder.[SD ID] AS [RO id]
FROM  SD AS SalesOrder
INNER JOIN  SD AS ReturnOrder
ON SalesOrder.[SD ID] = ReturnOrder.[REFERENCE ID]
 AND SalesOrder.[DocumentType] = 'Sales Order'
 AND ReturnOrder.[DocumentType] = 'Return Order'
```

Lemma 1 (Reference property). *Given a direct, type level interaction (A_S, F, A_T), an instance in the parent artifact A_S can be linked to zero or multiple instances of the child artifact A_T, whereas an instance in the child artifact A_T can only be linked to zero or one instance of the parent artifact A_S.*

Proof: According to the definition of a direct type level interaction, the reference F is a direct foreign key between the two main tables. Therefore, the property of a foreign key (or the structure of a relational database) indicates that an entry in the parent table (in this case the main table $A_S.T_m$ of the parent artifact) is linked to zero or multiple entries in the child table (in this case the main table $A_T.T_m$ of child artifact), whereas an entry in the child table is only linked to zero or one entry in the parent table, which automatically implies the Reference property since each entry in the main table refers to an artifact instance. □

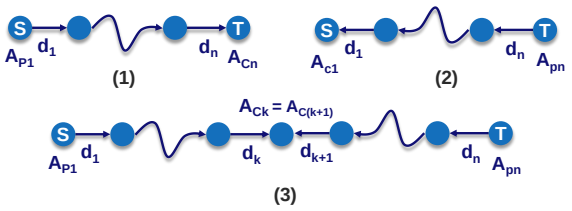


Fig. 16. Type level interactions

Definition 8 (Valid artifact type level interactions). *Let $\mathbb{A}$ be a set of artifacts and $\mathbb{F}$ be a set of references. An* artifact type level interaction *$I_{S,T} = \langle d_1, \cdots, d_n\rangle \in (\mathbb{A} \times \mathbb{F} \times \mathbb{A})^*$ between artifacts A_S and A_T with $n \geq 1$ is a sequence of direct interactions $d_i = (A_{P_i}, F_i, A_{C_i}) \in \mathbb{A} \times \mathbb{F} \times \mathbb{A}$ and $1 \leq i \leq n$, which satisfies one of the following properties (also shown in Figure 16):*

(1) *the parent artifact A_{P_1} of the first direct interaction d_1 is the artifact A_S, and the child artifact A_{P_n} of the last direct interaction d_n is the artifact A_T, and for $1 \leq i < n$, $A_{C_i} = A_{P_{i+1}}$; or*

(2) *the child artifact A_{C_1} of first direct interaction d_1 is the artifact A_S, and the parent artifact A_{P_n} of the last direct interaction d_n is the artifact A_T, and for $1 \leq i < n$, $A_{P_i} = A_{C_{i+1}}$; or*

(3) *there is a number k and $1 \leq k \leq n$, for $1 \leq i < k$, $A_{C_i} = A_{P_{i+1}}$, and $A_{C_k} = A_{C_{k+1}}$, and for $k < i < n$, $A_{P_i} = A_{C_{i+1}}$, and the parent artifact A_{P_1} of first direct interaction d_1 is the artifact A_S, and the parent artifact A_{P_n} of the last direct interaction d_n is the artifact A_T.*

Additionally, in all cases the number of instance level interactions is greater than zero.

Definition 9 (Strong interaction). *An artifact type level interaction $I_{S,T} = \langle d_1, \cdots, d_n\rangle$ between artifacts A_S and A_T that satisfies Definition 8(1) or (2) is called* strong. *The* length of strong joins *of this interaction is n.*

Definition 10 (Weak interaction). *An artifact type level interaction $I_{S,T} = \langle d_1, \cdots, d_n\rangle$ between artifacts A_S and A_T that satisfies Definition 8(3) is called* weak. *The* length of strong joins *of this interaction is k, and the* length of weak joins *is $m = n - k$.*

If $m > 0$, then the interaction relation may be describe an over-approximation of actual interactions.

Let $lengthSJ(I_{S,T}) = k$ and $lengthWJ(I_{S,T}) = m$ denote the length of strong and weak joins of $I_{S,T}$.

Invalid Interactions. Compared to weak interactions, a sequence that has two consecutive direct interactions (A_{S_i}, F_i, A_{T_i}) and (A_{S_i}, F_j, A_{T_j}) with the same parent artifact A_{S_i} and different target artifacts is invalid. Figure 11(c) shows an example: the interaction composed of $(A_{SalesOrder}, F_i, A_{Delivery})$ and $(A_{SalesOrder}, F_j, A_{ReturnOrder})$. Since an instance of the artifact *Sales Order* can be linked to multiple instances of the artifact *Delivery* and multiple instances of the *Return Order*, it is impossible to determine the exact relations between the instances only based on the references. For instance, D1 and D2 refer to S1, but also S3 refers to S1, but there is no explicit relation that indicates whether D1 is then related to S3 or D2 to S3. Therefore, this type of interaction is considered to be invalid.

C.2 Artifact Type Level Interaction Discovery

In this section, the algorithms for computing artifact type level interactions are presented. First, given a set of artifacts $\mathbb{A} = \{A_1, \cdots, A_n\}$, the algorithm *ConstructInteractionGraph*$(\mathbb{A}, \mathbb{F})$ computes an interaction graph $G = (\mathbb{A}, D)$. From the interaction graph G, the algorithm *CalculatesInteractions*$(\mathbb{A}, G, r, k, m)$, with $r > 0$, $k \geq 1$ and $0 \leq m \leq k$ compute a set of ARTIs $\mathbb{I}$, direct and indirect, for each artifact.

Definition 11 (Interaction graphs). *Given a set $\mathbb{A}$ of artifacts and a set $\mathbb{F}$ of references, $G = (\mathbb{A}, D)$ with $D \subseteq \mathbb{A} \times \mathbb{F} \times \mathbb{A}$ is an* interaction graph*, if and only if, for each $d = (A_s, F, A_t) \in D$, d is a direct, artifact type level interaction between artifacts A_s and A_t.*

The set of outgoing edges of artifact $A_S \in \mathbb{A}$ is denoted by $outEdges(A_S) = \{(A_S, F_i, x) \in D \mid x \in \mathbb{A}\}$. Similarly, the incoming edges of artifact A_S is defined as $inEdges(A_S) = \{(x, F_i, A_S) \in D \mid x \in \mathbb{A}\}$.

For example, Figure 8 shows the interaction graph of the OTC example including the direct interactions (denoted by the arcs) between the artifacts (denoted by the elliptic nodes).

Algorithm *ConstructInteractionGraph*$(\mathbb{A}, \mathbb{F})$
1. $D \leftarrow \emptyset$
2. **for** $F = \langle T_p, \mathbb{C}_p, T_c, \mathbb{C}_c, S_{condition} \rangle \in \mathbb{F}$
3. **do for** $A_p \in Artifacts(T_P)$, $A_c \in Artifacts(T_c)$
4. **do if** $countSelect(A_p, F, A_c) \geq 0$
5. **then** $D \leftarrow D \cup (A_p, F, A_c)$
6. **return** $G = (\mathbb{A}, D)$

Algorithm *CalculatesInteractions*$(\mathbb{A}, G, r, k, m)$
1. **for** $A_S \in \mathbb{A}$
2. **do** $\mathbb{I} \leftarrow \emptyset$
3. **for** $(A_S, F, A_t) \in outEdges(A_S)$
4. **do** $I_{current} \leftarrow \langle (A_S, F, A_t) \rangle$
5. **if** $countSelect(I_{current}) \geq r$
6. **then** $\mathbb{I} \leftarrow \mathbb{I} \cup I_{current}$
7. calculateJoins$(A_S, A_t, I_{current}, \mathbb{I}, r, k-1, m)$
8. $A.\mathbb{I} \leftarrow \mathbb{I}$
9. **return** $\mathbb{A}$

Algorithm *calculateJoins*$(A_S, A_{current}, I_{current}, \mathbb{I}, r, k, m)$
1. **if** $k > 0$
2. **then for** $(A_{current}, F, A_{next}) \in outEdges(A_{current})$
3. **do** $I_{next} \leftarrow I_{current} + \langle (A_{current}, F, A_{next}) \rangle$
4. $count \leftarrow selectCount(I_{next})$
5. **if** $count \geq r$
6. **then** $\mathbb{I} \leftarrow \mathbb{I} \cup I_{next}$
7. **if** $count > 0$
8. **then** calculateJoins$(A_S, A_{next}, I_{next}, \mathbb{I}, r, k-1, m)$
9. **if** $m > 0 \wedge lengthSJ(I_{current}) > lengthWJ(I_{current})$
10. **then for** $(A_{next}, F, A_{current}) \in inEdges(A_{current}) \backslash set(I_{current})$
11. **do** $I_{next} \leftarrow I_{current} + \langle (A_{next}, F, A_{current}) \rangle$
12. $count \leftarrow selectCount(I_{next})$
13. **if** $count \geq r$
14. **then** $\mathbb{I} \leftarrow \mathbb{I} \cup I_{next}$
15. **if** $count > 0$
16. **then** calculateJoins$(A_S, A_{next}, I_{next}, \mathbb{I}, r, 0, m-1)$
17. **return**

Single-Sided Discovery. Note that a design decision is made to restrict the identification of symmetric interactions on the parent artifact by using the condition that the length of strong joins is at least the length of weak joins, in addition to that the number of strong joins greater than zero (i.e. $k \geq 1$).

C.3 Event Type Level Interaction

The formal definition of event logs, traces and events [1] are listed.

Definition 12 (Event universe $\mathcal{E}$). *Let $\mathcal{E}$ be the* event universe*, i.e. the set of all possible event identifiers. Events may be characterized by various* attributes*. Let AN be a set of attribute names. For any event $e \in \mathcal{E}$ and name $n \in AN$: $\#_n(e)$ is the value of attribute n for event e. If event e does not have an attribute named n, then $\#_n(e) = \perp$ (null value).*

For example, $\#_{type}(e)$ is the event type associated to event e describing the activity that has been executed; $\#_{time}(e)$ is the timestamp of event e (also denoted by $\mathcal{T}(e)$).

Definition 13 (Case universe $\mathcal{L}$). *Let $\mathcal{L}$ be the* case universe*, i.e. the set of all possible case identifiers. A case also has attributes. For any case $c \in \mathcal{L}$ and name $n \in AN$: $\#_n(c)$ is the value of attribute n for case c. If case c does not have an attribute named n, then $\#_n(c) = \perp$.*

Definition 14 (Traces). *Each case has a special mandatory attribute* trace*: $\#_{trace}(c) = \sigma_c \in \mathcal{E}^*$, which is a finite sequence of events $\sigma \in \mathcal{E}^*$ such that each event appears only once. We assume traces in a log contain at least one event, i.e. $\#_{trace}(c) \neq \langle\rangle$.*

Definition 15 (Logs). *An* event log *is a set of cases $L \subseteq \mathcal{L}$ such that each event appears at most once in the entire log. We use $\mathcal{A}$ to denote the set of all event types appearing in log L, i.e. $\mathcal{A} = \{\#_{type}(e) \mid c \in L \wedge e \in \#_{trace}(c)\}$.*

Definition 16 (Projection $\mathcal{F}$). *Given a log L and an event type $E \in \mathcal{A}$, $\mathcal{F}_E(L)$ filters L and retain, for each of its traces, the event of event type E only.*

For example, the trace sales order σ_{S1} consists of two events, i.e. $\langle(S1, created, \text{16-5-2020}), (S1, latestchange, \text{10-6-2020})\rangle$. If we apply the project function on the trace σ_{S1} with $E = created$, we obtain $\mathcal{F}_{created}(\sigma_{S1}) = \langle(S1, created, \text{16-5-2020})\rangle$.

Definition 17 (Trace precedence $<_\mathcal{T}$). *Given two traces σ_s and σ_t, $\sigma_s <_\mathcal{T} \sigma_t$, if and only if, for each event $e_s \in \sigma_s$ and $e_t \in \sigma_t$, the timestamp $\#_{time}(e_s)$ of the event e_s is before the timestamps $\#_{time}(e_t)$.*

For example, $\mathcal{F}_E(\sigma_s) <_\mathcal{T} \mathcal{F}_E(\sigma_t)$ means that each event $e_s \in \sigma_s$ which has the event type E is executed earlier than each event $e_t \in \sigma_t$ which has the event type E. $\sigma_s =_\mathcal{T} \sigma_t$, $\sigma_s >_\mathcal{T} \sigma_t$, $\sigma_s \leq_\mathcal{T} \sigma_t$ and $\sigma_s \geq_\mathcal{T} \sigma_t$ are also similarly defined.

Definition 18 (Function $\#_{\mathcal{I}\_T}(c)$). *Given a log L, of which each case is enriched with instance level interactions, a case c, and artifact T, we use the notation $\#_{\mathcal{I}\_T}(c)$ to retrieve the instance level interactions between the case c and the set of cases of artifact T.*

Definition 19 (Function $\mathcal{I}$). *Given two event logs L_S and L_T, we define $\mathcal{I}(L_S, L_T) = \{(c_s, c_t) \mid c_s \in L_S \wedge c_t \in L_T \wedge c_t \in \#_{\mathcal{I}\_T}(c_s)\} \subseteq L_S \times L_T$ as the* set of trace level interactions *between the logs L_S and L_T, where each $(c_s, c_t) \in \mathcal{I}(L_S, L_T)$ indicates that there is an instance level interaction between the two traces c_s and c_t seen from the* parent artifact A_S *to the* child artifact A_T.

Definition 20 (Event type level interactions (EvTI)). *Given two artifact types A_S and A_T, and their set of event types $\mathcal{A}_S$ and $\mathcal{A}_T$, respectively, an event type level interaction (a_p, a_c) between A_S and A_T satisfies, at least, the requirement that $a_p \in \mathcal{A}_S$ and $a_c \in \mathcal{A}_T$, or vice versa.*

In the following, we present two methods to identify event type level interactions between the events of two interacting traces L_S and L_T. More specifically, we identify the relation $X \subseteq (\mathcal{A}_S \times \mathcal{A}_T) \cup (\mathcal{A}_T \times \mathcal{A}_S)$ where $\mathcal{A}_S$ and $\mathcal{A}_T$ denote the event types of L_S and L_T, respectively.

C.4 EvTI Discovery by Merging Logs

We present the *CalcETLInteractionsByMergingLogs* algorithm below. In general, for each $(\sigma_s, \sigma_t) \in \mathcal{I}(L_S, L_T)$, we merge the two traces to a new trace using the merge function $\mathcal{M}$ (see Line 5) that simply builds the union of the events of two traces σ_s and σ_t and orders all events by their timestamp (see [53, Chap.5] for a formal definition). All merged traces are added to the log L_{new} containing events originally from two artifacts.

Now, a process discovery algorithm $Miner()$ can be applied on the merged log L_{new} to discover dependencies between the two sets of event types, i.e. $\mathcal{A}_S$ and $\mathcal{A}_T$. (see Line 7). Each direct succession (E_i, E_j) between the two event types E_i and E_j discovered where $E_i \in \mathcal{A}_S$ and $E_j \in \mathcal{A}_T$ (or vice versa) is considered an *event type level interaction* between artifacts A_S and A_T, and added to the result X (see Lines 8-9).

Algorithm *CalcETLInteractionsByMergingLogs*(L_S, L_T)
1. **if** $\mathcal{I}(L_S, L_T) = \emptyset$
2. **then return** $\emptyset$
3. **else** $L_{new} \leftarrow []$, $X \leftarrow \emptyset$
4. **for** $(\sigma_s, \sigma_t) \in \mathcal{I}(L_S, L_T)$
5. **do** $\sigma_{new} \leftarrow \mathcal{M}(\sigma_s, \sigma_t)$
6. if $\sigma_{new} \neq \langle\rangle$ then add σ_{new} to L_{new}
7. $M_{new} \leftarrow Miner(L_{new})$
8. **for** (E_i, E_j) where E_j is a direct successor of E_i in the model M_{new}, and $(E_i, E_j) \in \mathcal{A}_S \times \mathcal{A}_T$ or $(E_i, E_j) \in \mathcal{A}_T \times \mathcal{A}_S$
9. **do** add (E_i, E_j) to X.
10. **return** (L_{new}, X)

For example, if we use the heuristic miner as the $Miner(L_{new})$, a simple causality net $DG = (\mathcal{A}_{new}, D)$, in which $D \subseteq \mathcal{A}_{new} \times \mathcal{A}_{new}$ indicates the direct succession between event types, is obtained from the log L_{new}. Here, a dependency $(E_i, E_j) \in D$ with $E_i \in A_{L_S}$ and $E_j \in A_{L_T}$ (or vice versa) is returned as an event type level interaction. When using a miner that returns a Petri net, we first compute the direct succession between transitions by omitting the places and then identify event type level interactions.

An important remark is that different process discovery techniques return a different set of event type level interactions. The meaning of an event type level interactions identified also varies depending on the discovery miner chosen as $Miner(L)$. For example, the interactions (dependencies) between two event types identified by the alpha miner are absolute precedence, thus event type A always before event type B, and no B is found before A in the log. In contrast, the event type level interactions returned by the flexible heuristic miner have a different meaning, i.e. event type A is mainly before event type B, and B before A is much less frequent (or lower than the threshold) in the log.

C.5 EvTI Discovery by Using Defined Criteria

Definition 21 (Criterion *absolute precedence*). *For each $(E_p, E_c) \in \mathcal{A}_{subS} \times \mathcal{A}_{subT} \cup \mathcal{A}_{subT} \times \mathcal{A}_{subS}$, $(E_p, E_c) \in X$ is an ETLI according to* absolute precedence, *iff, for all $(\sigma_s, \sigma_t) \in \mathcal{I}(L_S, L_T) : \mathcal{F}_{E_p}(\sigma_s) \leq_\mathcal{T} \mathcal{F}_{E_c}(\sigma_t)$*

Definition 22 (Criterion *existing precedence*). *For each $(E_p, E_c) \in \mathcal{A}_{subS} \times \mathcal{A}_{subT} \cup \mathcal{A}_{subT} \times \mathcal{A}_{subS}$, $(E_p, E_c) \in X$ is an ETLI according to* existing precedence, *iff, there is $(\sigma_s, \sigma_t) \in \mathcal{I}(L_S, L_T) : \mathcal{F}_{E_p}(\sigma_s) \leq_\mathcal{T} \mathcal{F}_{E_c}(\sigma_t)$*

Definition 23 (Criterion *shortest time*). *$(E_p, E_c) \in X$ is an ETLI according to* shortest time, *iff,*

1) $\min_{(E_p,E_c) \in \mathcal{A}_{subS} \times \mathcal{A}_{subT}} \left(\frac{\sum_{(\sigma_s,\sigma_t) \in \mathcal{I}(L_S,L_T)} AvgTimeDur(\mathcal{F}_{E_p}(\sigma_s), \mathcal{F}_{E_c}(\sigma_t))}{|\mathcal{I}(L_S,L_T)|} \right)$, *or,*

2) $\min_{(E_p,E_c)\in\mathcal{A}_{subT}\times\mathcal{A}_{subS}}($
$\frac{\sum_{(\sigma_s,\sigma_t)\in\mathcal{I}(L_S,L_T)} AvgTimeDur(\mathcal{F}_{E_p}(\sigma_s),\mathcal{F}_{E_c}(\sigma_t))}{|\mathcal{I}(L_S,L_T)|}$)
)

Definition 24 (Event level interactions). *Let* $\sigma = \langle e_1, e_2, \cdots, e_n\rangle$ *be a trace. For all* $i = [1, ..., n-1]$, *we call event* e_{i+1} *the direct successor of* e_i*; we write* $(e_i, e_{i+1}) \in succ(\sigma)$. *Let* $succ(L)$ *denote the direct successors of all traces in L. Now, if L is merged from two logs* L_S *and* L_T, *an event* $e_i \in L$ *indicates* $e_i \in \mathcal{E}_S \cup \mathcal{E}_T$ *which is the union set of events of* L_S *and* L_T. *We define an* event level interaction *as follows. A succession* $(e_i, e_{i+1}) \in succ(L)$ *with L merged from* L_S *and* L_T *is an* event level interaction *between the two events if and only if* $e_i \in \mathcal{E}_S$ *and* $e_j \in \mathcal{E}_T$ *or vice versa.*

Definition 25 (Criterion *existence of an event level interaction*). $(E_p, E_c) \in X$ *is an ETLI according to* existence of an event level interaction, *iff, there exists* $(e_i, e_{i+1}) \in succ(L)$ *such that (1)* $e_i \in \mathcal{E}_S \wedge e_{i+1} \in \mathcal{E}_T$ *(or vice versa) and (2)* $\#_{type}(e_i) = E_p \wedge \#_{type}(e_{i+1}) = E_c$

Definition 26 (Criterion *max number of event level interactions*). $(E_p, E_c) \in X$ *is an ETLI according to* max number of event level interactions, *iff, , iff,*

$$\max_{(E_p,E_c)\in\mathcal{A}_{subS}\times\mathcal{A}_{subT}} \bigcup_{(\sigma_s,\sigma_t)\in\mathcal{I}(L_S,L_T)} ($$
$$\{(e_i,e_{i+1}) \mid (e_i,e_{i+1}) \in \sigma \wedge \sigma \in \mathcal{M}(\sigma_s,\sigma_t)$$
$$\wedge \#_{type}(e_i) = E_p \wedge \#_{type}(e_{i+1}) = E_c\})$$
$$\vee$$
$$\max_{(E_p,E_c)\in\mathcal{A}_{subT}\times\mathcal{A}_{subS}} \bigcup_{(\sigma_s,\sigma_t)\in\mathcal{I}(L_S,L_T)} ($$
$$\{(e_i,e_{i+1}) \mid (e_i,e_{i+1}) \in \sigma \wedge \sigma \in \mathcal{M}(\sigma_s,\sigma_t)$$
$$\wedge \#_{type}(e_i) = E_p \wedge \#_{type}(e_{i+1}) = E_c\})$$

APPENDIX D
COMPLEXITY ANALYSIS

In this section, we provide a complexity analysis of our approach. The running time of algorithms are summarized in Table 2. We use the same number to refer to the same step in the overview shown in Figure 4.

In the following analysis, we use $|\mathbb{T}|$ to denote the number of tables, $|\mathbb{C}|$ to denote the number of columns, $|\mathbb{F}|$ the number of references, $|\mathbb{A}|$ the number of artifact types, $|\mathbb{E}|$ the number of event types, and $|L|$ to denote the size of log in term of number of events.

The database schema identification (1.0), including primary key and foreign key extraction, is an NP-hard problem. Our approach and tools support both importing the existing data schemas, which takes $O(|\mathbb{T}| + |\mathbb{F}|)$, as well as using the original XTract approach to discover data schemas, which is exponential in number of columns.

TABLE 2
Running time analysis

Step	Running time
1.0	NP-hard or $O(\|\mathbb{T}\| + \|\mathbb{F}\|)$
1.1	$O(\|\mathbb{T}\| + \|\mathbb{F}\|)$
1.2	$O(\|\mathbb{A}\| \times \|\mathbb{C}\|)$
1.3	$O(\|Entries\|^2)$
1.4	running time on the discovery algorithm selected
2.1	$O(\|\mathbb{A}\| \times \|\mathbb{F}\|^{k+m})$
2.2	$O(\|Entries\|^2)$
2.3	$O(\|L\|^2$+ running time of discovery algorithm)
2.4	$O(\|\mathbb{E}\|^2)$

The artifact schema identification (1.1) runs in linear with respect to the number of tables or the number of references. The artifact identification (1.2), which follows, runs in worst case $O(|\mathbb{A}| \times |\mathbb{C}|)$ because for each artifact type the algorithm has to include all columns (as identifiers, event types, or attributes of the artifact type). The extraction of a log of a defined artifact type (1.3) currently takes quadratic in terms of the number of entries in the data set, in worst case, since each entry in the main table is joined with all other entries to obtain its events and attributes. In theory, this running time can be improved to be linear in terms of the number of entries. Finally, the complexity of the discovery of a life-cycle of an artifact (1.4) depends on the discovery algorithm selected, which might be linear or exponential in terms of the number of events of the log for the artifact.

For discovering interactions between artifact types (2.1), the algorithm basically follows a depth-first-search to select all paths composed of references of which the number of strong joins is at most k and the number of weak joins is at most m, and therefore, grows exponentially in $(k + m)$. The interactions between artifact types are then used to extract interactions between artifact instances for logs (2.2), which takes quadratic in terms of the number of entries in the data set and is executed during step 1.3. Discovering interactions between event types (2.3) of two interacting artifacts requires to merge their logs, which takes $O(|L|^2)$, to run a discovery algorithm. To discovery an artifact-centric model, step 2.3 is re-run for every two interacting artifacts, and between each two event types, thus $O(|\mathbb{E}|^2)$, if they interact, we add an event type level interaction.

During the case studies, which is discussed in Section 5, steps (1.3) combined with (2.2) are the most time-consuming part; to extract about 30000 traces, in total circa 30000 events, it takes almost a hour. Other steps executed during the two case studies take less than ten minutes[2].

2. We import the data schema, and for step (2.1) we use $k = 2$ and $m = 1$

TABLE 3
Artifact schemas obtained using the original XTract approach.

k	2	3	4	5	6	7	8	9	10
CDPOS	C1	C1	C1	C1	C1	C1	C1	C1	C1
CDHDR			C2	C2	C2	C2	C2	C2	C2
VBAK							C8	C8	C8
VBAP		C2		C5	C5	C5	C5	C5	C5
LIPS									
LIKP						C7	C7	C7	C7
VBRK					C6	C6	C6	C6	C6
VBRP	C2		C3	C3	C3	C3	C3	C3	C3
BKPF		C3	C4	C4	C4	C4	C4	C4	C4
BSAD								C9	C9
no table									C10

APPENDIX E
CASE STUDIES AND EVALUATION – DETAILS

In this section, more details on the case studies are reported.

E.1 SAP Order To Cash Process: Data and Discovery

Fig. 17 shows the details of tables used for discovering artifacts and their interactions in the SAP - OTC process case study. Fig. 18 shows a part of the artifact type definition obtained; see [19] for all artifact type definitions and further details on the extraction. Fig. 19 shows the interaction graph discovered for the process.

Table Name	Constraint and Time scope	Row Count used	Column Count
BKPF	where '2012-09-01' <= cpudt and cpudt < '2012-11-01' and awtyp = 'vbrk'	11358	32
BSID	where '2012-09-01' <= cpudt and cpudt < '2012-11-01'	4428	49
BSAD	where '2012-09-01' <= cpudt and cpudt < '2012-11-01'	911	49
CDHDR	where '2012-09-01' <= [UDATE] and [UDATE] < '2012-11-01' and (cdhdr.objectclas = 'VERKBELEG' or cdhdr.objectclas = 'faktBELEG')	13903	9
CDPOS	inner join cdhdr on cdhdr.changenr = cdpos.changenr where '2012-09-01' <= [UDATE] and [UDATE] < '2012-11-01' and (cdhdr.objectclas = 'VERKBELEG' or cdhdr.objectclas = 'faktBELEG')	56018	10
DDFTX	where tabname = 'vbak' or tabname = 'vbrk'	237	5
VBAK	where '2012-09-01' <= erdat and erdat < '2012-11-01'	3383	40
VBAP	where '2012-09-01' <= erdat and erdat < '2012-11-01'	4317	35
VBRK	where '2012-09-01' <= erdat and erdat < '2012-11-01'	5285	37
VBRP	where '2012-09-01' <= erdat and erdat < '2012-11-01'	10206	31
LIKP	where '2012-09-01' <= erdat and erdat < '2012-11-01'	11623	51
LIPS	where '2012-09-01' <= erdat and erdat < '2012-11-01'	13157	15

Fig. 17. SAP OTC process - tables and record counts

E.2 SAP Order To Cash Process: Result Details

The artifact schemas returned by [20] are listed in Table 3 when given k as the number of artifact types. Each artifact schema is converted into an artifact. For instance, when k is 2, tables from *VBAP* to BSAD are returned as one artifact type (C2) with main table *VBRP*; when k is 5, tables *VBAK*, *VBAP*, *LIPS* and *LIKP* are returned as one artifact type (C5) with the main table *VBAP*.

Art.Schema	Maintable	Artifact	Artifact condition	Extracted
BKPF	BKPF	PostInAR	<maintable>.awtyp = 'VBRK'	Yes
BSAD	BSAD	Payment05or15	(<maintable>.bschl = '05' or <maintable>.bschl = '15')	Yes
		Payment01or11	(<maintable>.bschl = '01' or <maintable>.bschl = '11')	
VBAK	VBAK	Order H	<maintable>.vbtyp = 'C'	Yes

Fig. 18. SAP OTC process - artifacts

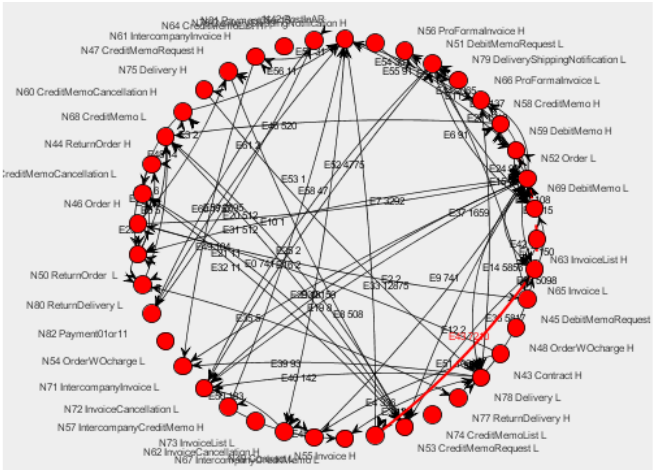


Fig. 19. SAP OTC process - interaction graph

Using classical conversion approach, we obtained a sales-order-oriented event log. The directly-follows-graph of the sales-order-oriented event log is shown in Figure 14. The absolute number of directly follows relations between event types is counted and summarized in Figure 20.

E.3 SAP Order To Cash Process: Exploration on Complex Life-Cycles and Interactions

In Sect. 5.1, we analyzed and compared flows between the *Create* event types of the different artifacts. To further investigate the correctness of the model, we also explored the possibility of creating complex life-cycle models and their impact on artifact interactions.

For this, we returned to the data extraction and now included in the *Sales Order* artifact type all event types found in the change tables *CDHDR* and *CDPOS*; see [53, Chap.7.1.3] for details. Since the trace identifiers have not changed, and interactions have not changed, there is no need to re-extract other artifacts. Using the *DF-max*-EvTI classifier, we obtained the artifact-centric model shown in Fig. 21.

Complexity of Interactions. The refined model of Fig. 21 shows that the event-types of the *Sales Order* artifact have clearly differ in their interactions with other artifacts. The red circles indicate interesting differences. For instance, most artifacts, e.g. *Delivery*, *Credit Memo Request*, *Debit Memo Request* are generally created directly after the creation of *Sales Order*, whereas the invoices (that are directly related to a

n: violating n: non-violating n: aligning		C	O	D	RO	RD	I	DR	DM	IC	CR	CM	PI	AR	P	Total
Contract H_Created	C	0	371	316	0	0	15	0	0	0	0	0	9	22	0	
(Order) Created	O	53	0	1472	0	0	346	0	0	0	0	0	59	341	6	
Delivery H_Created	D	46	907	2439	0	0	597	0	0	1	1	1	212	620	23	
ReturnOrder H_Created	RO	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
ReturnDelivery H_Created	RD	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
Invoice H_Created	I	0	65	47	0	0	142	1	0	17	3	1	90	1360	278	
DebitMemoRequest H_Created	DR	0	0	0	0	0	0	0	2	0	0	0	0	0	0	
DebitMemo H_Created	DM	0	0	0	0	0	0	0	3	0	0	0	0	5	0	
InvoiceCancellation H_Created	IC	0	0	1	0	0	10	0	0	2	0	0	0	22	1	
CreditMemoRequest H_Created	CR	0	0	1	0	0	1	0	0	0	6	12	0	8	0	
CreditMemo H_Created	CM	0	0	0	1	0	1	0	0	0	7	2	0	15	1	
ProFormaInvoice H_Created	PI	0	18	141	0	0	138	0	0	2	2	2	257	108	13	
PostInAR_PostedInAR	AR	1	49	42	0	1	1298	1	9	18	14	15	89	854	354	
Payment05or15_Payment Received	P	0	31	12	0	0	26	0	0	0	0	0	0	78	107	
Sum total number of edges		100	1441	4472	1	1	2574	2	14	40	33	34	716	3433	783	13644
Sum the number of "False" edge		100	1070	2683	1	1	1468	1	12	20	27	17	257	932	107	6696
Relative False Edges w.r.t. Total in %		100%	74%	60%	100%	100%	57%	50%	86%	50%	82%	50%	36%	27%	14%	49%

Fig. 20. Statistics wrt. directly-follows relations discovered in the sales-order-oriented log

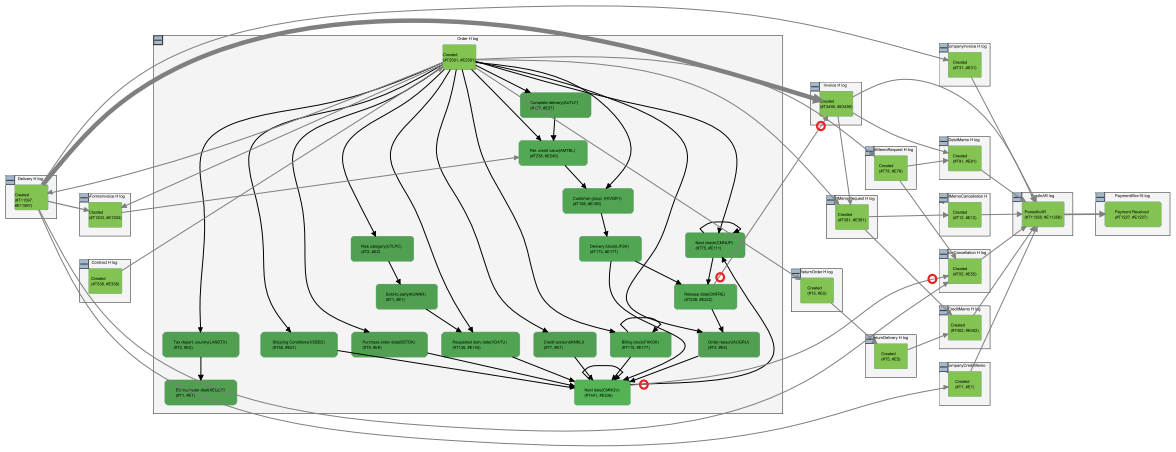


Fig. 21. SAP OTC process - the same artifact-centric model as Figure 14 except the life-cycle of artifact *Sales order* is extended

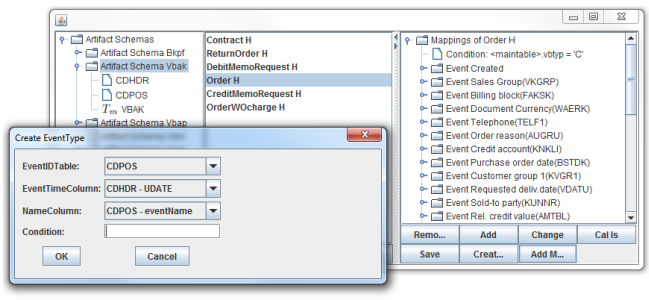


Fig. 22. SAP OTC process - the Orders artifact with change event types

sales order via *vgbel* and *vgpos*) are created after the *Release Date* change event type of this sales order. Similar for the invoice cancellation artifact, of which the creation generally takes place after the *Next Date* of the sales order changes.

Discussion. At this point, while the results show that while artifact-centric models and interactions between artifacts are able to distinguish main flow and unusual flows, we also noted a limitation of our technique. Showing all interactions discovered on the merged log makes the artifact-centric model very complex and almost impossible to analyze, further simplification or filtering techniques have to be developed.

E.4 Oracle Project Administration Process: Data and Discovery

The tables used in the Oracle PA process analysis is shown in Fig. 23. Documentation about the data schema (esp. primary keys and foreign keys) was available.

The six artifacts are shown in Figure 24. Seven direct type level interactions are found between the artifacts which are shown in Figure 25.

E.5 Oracle Project Administration Process: Results Details

We report on two unusual flows identified in the case study of Sect. 5.2, in particular on the confirmation of unusual flows in the underlying data, and the

Table Name	Row Count downloaded	Used	Row Count used	Column Count
FND_USER	905		905	27
HR_ALL_ORGANIZATION_UNITS	1053		1053	43
MTL_SYSTEM_ITEMS_B	0			
OE_ORDER_HEADERS_ALL	0			
PA_COST_DISTRIBUTION_LINES_ALL	95943	x	5543	15
PA_EXPENDITURE_COMMENTS	55149		30511	7
PA_EXPENDITURE_ITEMS_ALL	96978	x	5620	32
PA_EXPENDITURE_TYPES	94	x	94	10
PA_EXPENDITURES_ALL	682590	x	3100	24
PA_PROJECT_CUSTOMERS_V	16238		16238	17
PA_PROJECT_STATUSES	80	x	80	23
PA_PROJECTS_ALL	5364	x	1132	29
PA_TASKS	2416	x	1236	31
PA_TRANSACTION_SOURCES	48		48	4
PAY_COST_ALLOCATION_KEYFLEX	1694		1694	11
PO_HEADERS_ALL	5186		5186	139
PO_LINE_LOCATIONS_ALL	7700		7700	148
PO_LINES_ALL	7700		7700	135

Fig. 23. Oracle PA process table record counts

Artifacts	Maintable	Extracted
Project	PA_PROJECTS_ALL	x
ExpAll	PA_EXPENDITURES_ALL	x
Tasks	PA_TASKS	x
CostDistr	PA_COST_DISTRIBUTION_LINES_ALL	
ExpItem	PA_EXPENDITURE_ITEMS_ALL	
ExpTypes	PA_EXPENDITURE_TYPES	

Interaction From	To	via
Project	Tasks	
Tasks	ExpAll	ExpItem

Fig. 24. The artifacts created for Oracle PA process

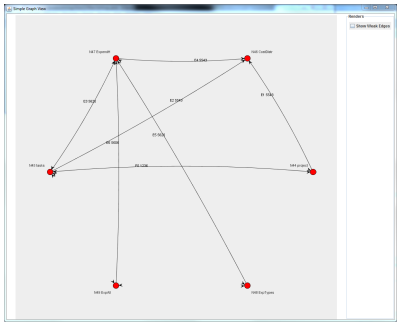


Fig. 25. Interaction graph of PA artifacts

corresponding explanations from the client. Further unusual flows are discussed in [19, Chap.7.4] and [53, Chap.7.2].

Q1: Tasks started before their projects created, are these double administration work? Fig. 15 shows a main EvTI of the process from *Task Started* to *Project Created*. This observation indicates that tasks had been started before their corresponding project had been created. Although the flow is not automatically revealed as a deviating flow, the experts from KPMG considered this as an unusual flow. In the interview, the client validated this observation and indicated that the flow is allowed if the time duration between *Project Created* and *Task Created* is short. The client stated that when creating a project in the system, the specification of all tasks related to this project are known, thus the creation time of tasks should happen shortly after the creation of the project in the system. If not, it may indicate that double administrative work have been performed. The average time between the creation of

PROJECT_ID	CREATION_DATE	CLOSED_DATE	COMPLETION_DATE	EXPENDITURE_ID	CREATION_DATE
6[redacted]9	2012-06-18 09:52:18.000	2012-08-30 00:00:00.000	2012-08-31 00:00:00.000	7[redacted]8	2012-08-31 18:11:56.000
6[redacted]9	2012-06-18 09:52:18.000	2012-08-30 00:00:00.000	2012-08-31 00:00:00.000	7[redacted]2	2012-08-31 18:11:58.000
6[redacted]5	2012-07-16 15:17:21.000	2012-09-19 00:00:00.000	2012-09-30 00:00:00.000	7[redacted]2	2012-09-19 18:15:24.000
6[redacted]5	2012-07-16 15:17:21.000	2012-09-19 00:00:00.000	2012-09-30 00:00:00.000	7[redacted]9	2012-09-19 18:15:16.000
6[redacted]5	2012-07-16 15:17:21.000	2012-09-19 00:00:00.000	2012-09-30 00:00:00.000	7[redacted]9	2012-09-19 18:15:24.000

Fig. 27. Expenditures created after projects closed in database

project and tasks is 1.088 day. Of the 1236 tasks, we found 1197 tasks that had been created less than a day after the project was created; 8 tasks that had been created between a day and 14 days after the project was created; and 31 tasks had been created later than 14 days after the project had started. We found no task had been created before the creation of its project, as the model had shown.

Q2: Do projects that closed before its tasks completed incur any financial risks? Another unexpected unusual observation that was made on the view shown in Fig. 26 which shows that many projects were closed before the related tasks completed. Open tasks allow expenditures to be booked while the project may already be closed. Triggered by this observation, the client asked us to further investigate whether there are expenditures created after the projects are closed or completed. We found two projects had been created in parallel with the creation of the expenditures indicating that there are expenditures which are created after the two projects are closed. Verifying this observation in the database, we retrieved that 5 out of the 28 expenditures related to these two projects were indeed created after the two projects were closed, shown in Fig. 27. Two of the five expenditures are created more than 24 hours after the closure of the projects, others on the same day. This result again shows that our approach allows to identify true unusual flows.

E.6 Oracle Project Administration Process: Exploration

During the discussion, the client and the experts suggested to explore whether different expenditure objects are used differently in the process. For instance, personnel costs are constant during the execution of projects, thus less interesting, but all other categories may show variations and may thus be more interesting to analyze. For this, we refined the artifact type *Expenditures* based on the expenditure *EXPENDITURE CATEGORY* column in the *PA EXPENDITURE TYPES* table [53, p.87]. The resulting model is shown in Fig. 28 and confirms the assumptions that each type of expenditure has different event type level interactions with the project life cycle. For example, the staff expenditures are created towards the beginning of the project (9), whereas the others expenditures are created after the project is definitive (i.e., created in the system) or after the Update Program event type(10) of the project.

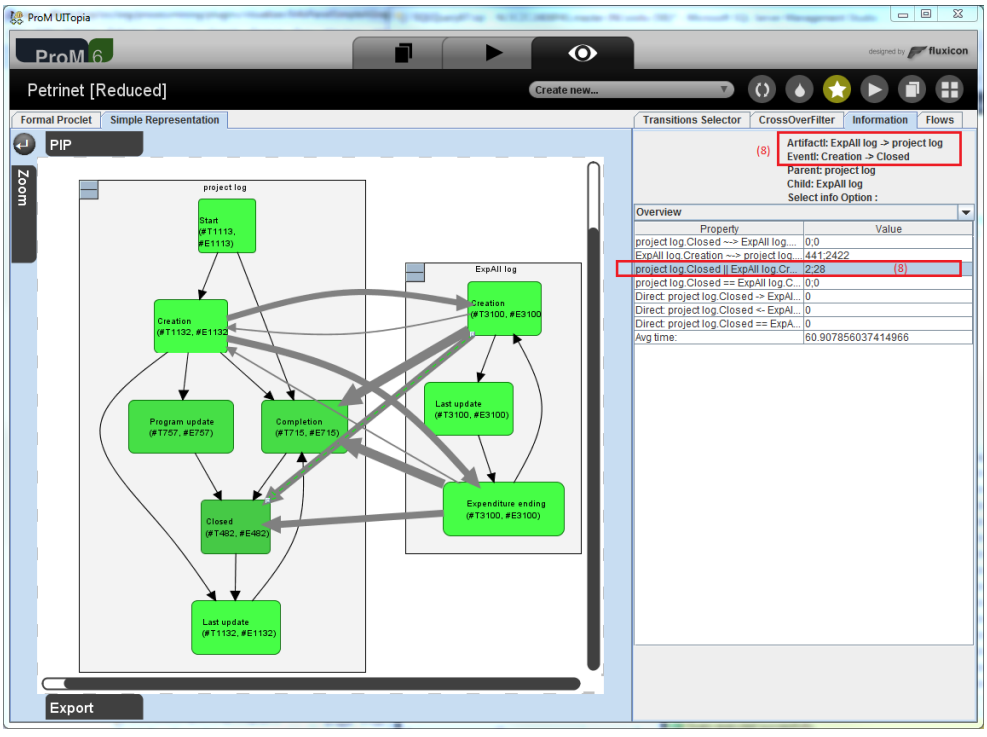


Fig. 26. Expenditures created after projects closed

Fig. 28. Interactions between the eight expenditure artifacts and the project life cycle