

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.

1.1 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. These systems usually follow service-oriented architectures which separate (1) the high level business processes that invoke information accesses and (2) the information itself into different layers [4], [5]. The information is encapsulated in *business objects* or *documents* and is typically stored in relational database. Moreover, these objects are related to each other through one-to-many and many-to-many relations and reused in various processes. 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*. The completion of a transaction is logged as an event also called *transactional data*.

The idea is to use transactional data to discover end-to-end business processes that are executed in reality (see Section 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 two sales orders S1 followed by creation of a delivery B1, another delivery document

SAP
EXHIBIT 1015

• X. Lu and D. Fahland are with the Department of Mathematics and Computer Science, Eindhoven University of Technology, 5600 MB, Eindhoven, The Netherlands. E-mail: {x.lu, d.fahland}@tue.nl.

• M. Nagelkerke and D. van de Wiel are with the KPMG IT Advisory N.V., 6513 AM, Eindhoven, The Netherlands. E-mail: {Nagelkerke.marijn, vandewiel.dennis}@kpmg.nl.

Manuscript received 21 Feb. 2014; revised 31 July 2015; accepted 14 Aug. 2015. Date of publication 27 Aug. 2015; date of current version 11 Dec. 2015. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.

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.

1.1 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. These systems usually follow service-oriented architectures which separate (1) the high level business processes that invoke information accesses and (2) the information itself into different layers [4], [5]. The information is encapsulated in *business objects* or *documents* and is typically stored in relational database. Moreover, these objects are related to each other through one-to-many and many-to-many relations and reused in various processes. 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*. The completion of a transaction is logged as an event also called *transactional data*.

The idea is to use transactional data to discover end-to-end business processes that are executed in reality (see Section 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 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

• X. Lu and D. Fahland are with the Department of Mathematics and Computer Science, Eindhoven University of Technology, 5600 MB, Eindhoven, The Netherlands. E-mail: {x.lu, d.fahland}@tue.nl.

• M. Nagelkerke and D. van de Wiel are with the KPMG IT Advisory N.V., 6513 AM, Eindhoven, The Netherlands. E-mail: {Nagelkerke.marijn, vandewiel.dennis}@kpmg.nl.

Manuscript received 21 Feb. 2014; revised 31 July 2015; accepted 14 Aug. 2015. Date of publication 27 Aug. 2015; date of current version 11 Dec. 2015. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TSC.2015.2474358

Authorized licensed use limited to: Klarquist Sparkman LLP. Downloaded on September 14, 2026 at 20:26:34 UTC from IEEE Xplore. Restrictions apply.

1939-1374 © 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

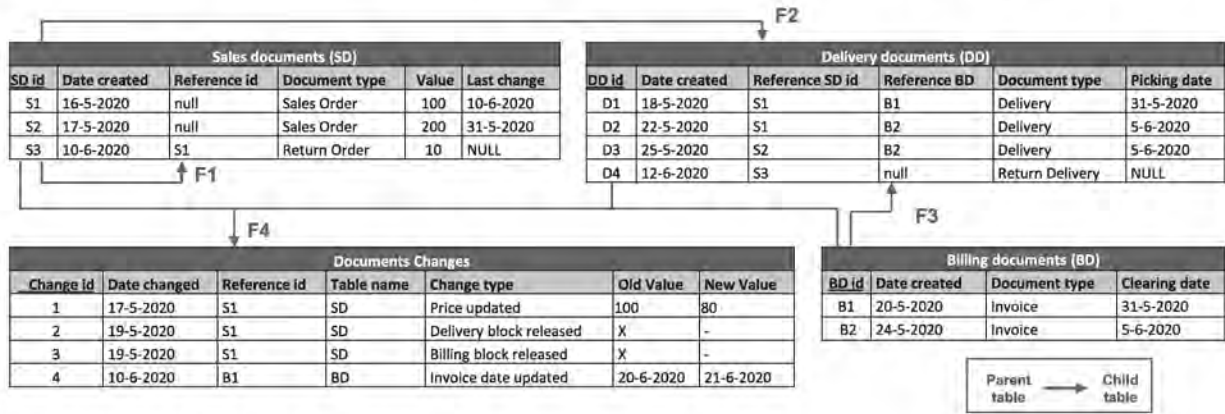


Fig. 1. The tables of the simplified OTC example.

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) 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) 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. 3a which is inaccurate: two invoices are created before their deliveries instead of one,

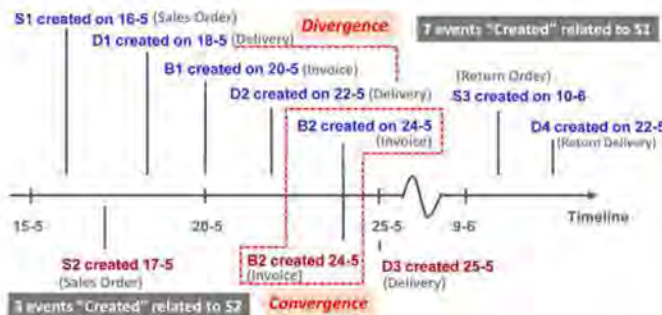


Fig. 2. A time-line regarding the creation of documents of the OTC example.

and three invoices are created instead of two (known as *divergence* and *convergence*, respectively) [9].

1.2 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. 3b. 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.



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

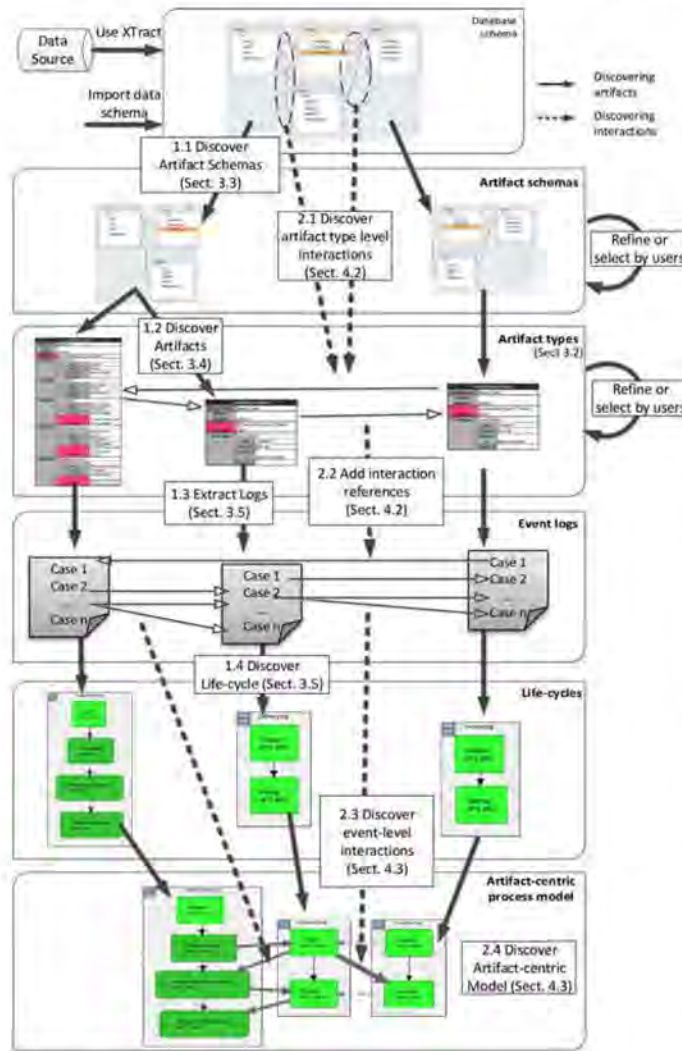


Fig. 4. An overview of our approach.

- 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. Section 2 discusses related work. Section 3 illustrates our extended approach to identify artifacts and their life-cycles from a given relational data source. In Section 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 Section 5. Section 6 concludes the paper.

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.

2.1 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 (Section 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, Chapter 2] for a discussion. This problem has been partly addressed in [20] through *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 Section 3 (steps 1.1 and 1.2 in Fig. 4) and Section 4 (step 2.1).

2.2 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 Section 4 (step 2.2 of Fig. 4).

2.3 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 Weerd 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 *synthesizes* user behavior whereas we *analyze actual* user behavior.

2.4 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 [20], [46], 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 models 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

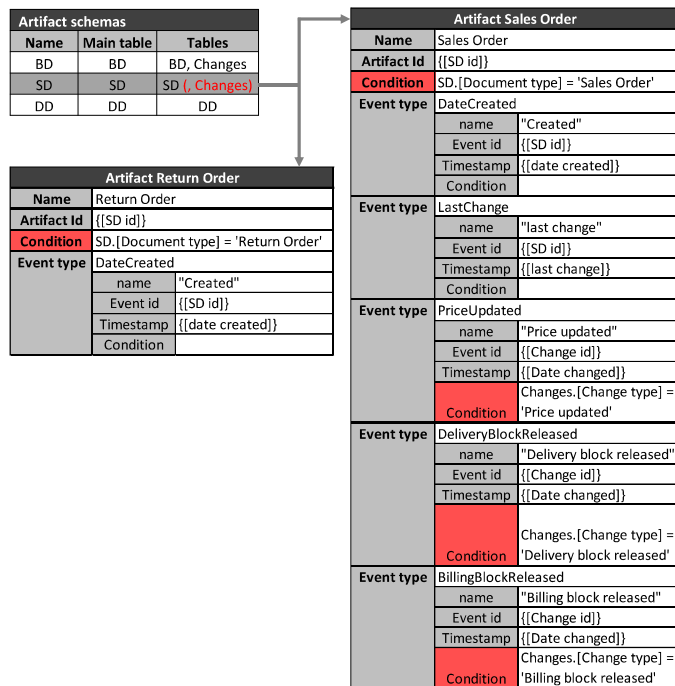


Fig. 5. The artifact schemas of Fig. 1(top left) and two artifact types derived from schema SD.

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. Sections 3 and 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 (Section 3.1) we show how to identify all conceptual objects and their event data in a relational data source in terms of artifacts (Sections 3.2, 3.3, and 3.4). Then existing log extraction and process discovery techniques can be applied to obtain a life-cycle model for each artifact (Section 3.5).

3.1 Relational Schema versus 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 Section 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 becomes an *event-level attribute*; its value provides more information about the event. Any other attributed related to the artifact identifier 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 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 three different event types of Sales Order by three different conditions. Formal definitions of artifact types are given in Section B.3, which can be found on the Computer Society Digital Library at <http://doi.ieeecomputersociety.org/10.1109/TSC.2015.2474358>.

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 Section 3.1 may require user input as discussed in Section 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 three 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 $\mathcal{S}$ of the data source (either from documentation or by recovering it from the tables [20]). Partition the set of all tables in $\mathcal{S}$ into maximal sets $\mathbb{T}_1, \dots, \mathbb{T}_k$ 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 three 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 Section 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, Chapter 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*.

3.4 Artifact Discovery and Refinement

From an artifact schema $\mathcal{S}_A$ (a set of related tables), a generic artifact-type definition A of $\mathcal{S}_A$ (detailing identifier, event-types, and related attributes, but without discriminating conditions) can be obtained automatically using the

algorithm *CreateTraceMapping*($\mathcal{S}_A$) of [20]. By this algorithm, the primary key of the main table of $\mathcal{S}_A$ becomes the artifact identifier. Each time-stamped column C in a table T in $\mathcal{S}_A$ becomes an event type E_C , every other non-time-stamped column in T defines an attribute of event type E_C . Every non-timestamped column in any table in $\mathcal{S}_A$ 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 Section 3.1, as we show next.

3.4.1 Refining Artifacts

In case $\mathcal{S}_A$ 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 $A_1, \dots, A_n$ and define a condition $\varphi_1, \dots, \varphi_n$ 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 $\varphi_1, \dots, \varphi_n$ have to be the given by the user. However, in the presence of a discriminating column C holding finitely many values $v_1, \dots, v_n$, such as *Document type* in table *SD* of Fig. 1, the conditions for each artifact type can be generated automatically as $C = v_1, \dots, C = v_n$; 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, Chapter 6].

3.4.2 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 Section B.4, available in the online supplemental material, 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 [20], [46] 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 [20], [46] 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, Chapter 4.3] for details. Fig. 6 shows the event log extracted

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 [7], [9], [20], [24], [52].

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 = "x"
Event e4	3	Billing block released	19-5-2020	Old value = "x", New value = "x"
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	-

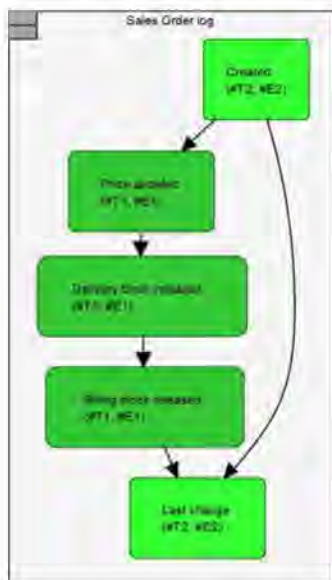
Fig. 6. An example of event log extracted for artifact *Sales Order*.

from Fig. 1 using the artifact type definition *Sales Order* of Fig. 5.

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

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

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

Authorized licensed use limited to: Klarquist Sparkman LLP. Downloaded on September 14, 2026 at 20:26:34 UTC from IEEE Xplore. Restrictions apply.



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

can be refined further based on domain knowledge. Both approaches are compared in our evaluation in Section 5.

4 INTERACTION DISCOVERY

In Section 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 identifier 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 (Section 4.1) and provide definitions and algorithms afterwards (Sections 4.2 and 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). Figs. 9a and 9b summarizes both; we discuss how to detect ARTIs and ARIs in Section 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 related artifact types. For

(a) Artifact type level interactions (ARTI):

F_3 (Invoice $\rightarrow$ Delivery)

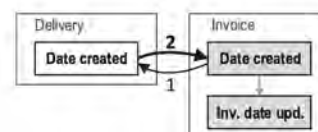
(b) Artifact instance level interactions (ARI):

$B_1 \mapsto D_1, B_2 \mapsto D_2, B_2 \mapsto D_3$

(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.

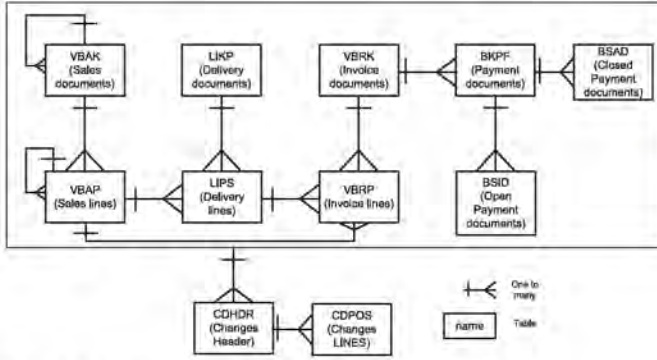


Fig. 10. ER-model of the SAP OTC process.

example, Fig. 9c visualizes the extracted traces for invoices B_1, B_2 and for deliveries 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 e_2 directly follows e_1 , e_5 follows e_4 , and e_6 follows e_5 . 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. 9d: 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 Section 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*. We call the source of an ARTI the parent (e.g., *Invoices* of F_3), and the target the child (e.g., *Delivery* of F_3), 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.

4.2.1 Indirect ARTIs

To allow omitting artifacts and yet study interactions of the remaining artifacts, we introduce *indirect ARTIs*. An *indirect ARTI* is a sequence 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., D_4 refers to only S_1 via S_3 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 S_1 is linked to two invoices B_1 and B_2 while B_2 is also linked to S_2 . (c) An ARTI is *invalid* if an intermediate artifact is the parent



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

of two direct ARTIs. In this case the ARIs are arbitrary and unreliable, e.g., in Fig. 11 one cannot infer whether return order S_3 is linked to D_1 or D_2 or both. Formal definitions of ARTIs and ARIs are given in Section C.1, available in the online supplemental material.

4.2.2 Discovering ARTIs and Extracting ARIs

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 $< \tau$ ARIs (to focus on frequent interactions only).

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* B_2 gets the attribute *interact : Delivery* = $\{D_2, D_3\}$. The algorithms are given in Section C.2, available in the online supplemental material.

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 Section C.3, available in the online supplemental material, for formal definitions.

4.3.1 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.

4.3.2 Classifying Interactions

We propose five 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 *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 Section 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.

4.3.3 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 *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, Chapter 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 [8], [10] and discovery which is the current standard in automated analysis of ERP event data (“classic”).

5.1.1 Context, Data and Setup

The first case study was performed for the Order to Cash 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 two 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.).

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.

5.1.2 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 *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 approach and identified artifact schemas with k -means clustering starting at $k = 2$ and incremented until no new artifact schemas were

Steps	OTC process in SAP		PA process in Oracle	
	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	$k=2, m=1, r=1$	7 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.

found (at $k = 10$); all results were considered in the analysis. See [19, Chapter 7.3] for details.

5.1.3 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 resulting model essentially describes artifact schemas instead of artifact types. For $k = 9$, we correctly identified seven of the eight 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. Section E, available in the online supplemental material, provides details.

5.1.4 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, nine out of the 14 event types have a self-loop, which are false positives, as no event in the original data set is directly



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

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 6,696 out of 13,644 directly-follows pairs to be incorrect, i.e., about 50 percent. 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. Section E, available in the online supplemental material, provides details.

5.1.5 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. Section E.3, available in the online supplemental material, 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

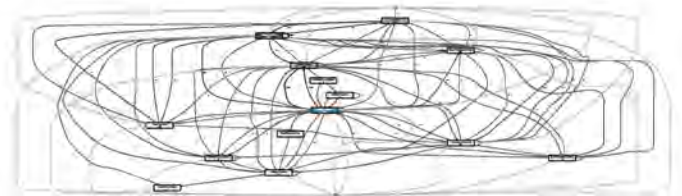


Fig. 14. SAP OTC process case study—a life-cycle of sales orders obtained using a classical log conversion approach.

supported by the Oracle ERP system of an educational organization (called “client” in the following).

5.2.1 Context, Data and Setup

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 seven tables over in total 16,805 records and 134 columns.

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, Chapter 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.

5.2.2 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 Section E.4, available in the online supplemental material, for details.

5.2.3 Results

In total five 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, Chapter 7.2] for all other views. On the five views, we identified in total 10 unusual flows. Seven flows were classified as unusual by our technique. Two unusual flows were identified by the experts from KPMG by visual inspection of the model. One 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

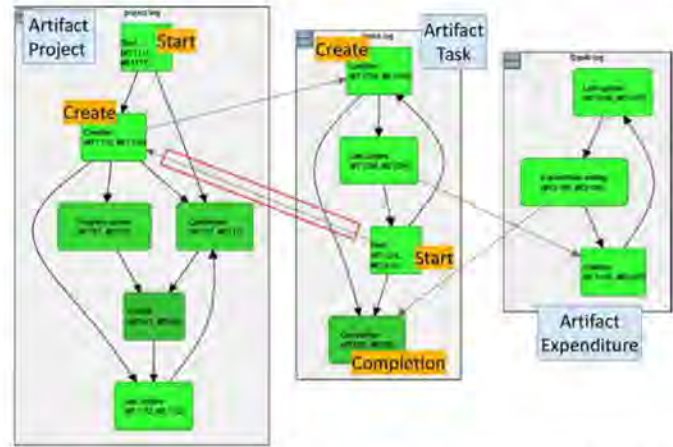


Fig. 15. A proclet system discovered by using the *DF-max-EvTI* classifier.

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 by the client. Two of the unusual flows are discussed in more detail in Section E.5, available in the online supplemental material; a detailed discussion on all flows is available in [53, Chapter 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 Section E.6, available in the online supplemental material.

5.2.4 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 1 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, Chapter 6].

6 CONCLUSION AND FUTURE WORK

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.

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

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

REFERENCES

- [1] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. New York, NY, USA: Springer, 2011.
- [2] M. L. van Eck, X. Lu, S. J. J. Leemans, and W. M. P. van der Aalst, “Pm²: A process mining project methodology,” in *Proc. 27th Int. Conf. Advanced Info. Syst. Eng. (CAiSE)*, 2015, pp. 297–313, Doi: 10.1007/978-3-319-19069-3_19.
- [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 *Proc. 27th Int. Conf. Adv. Inf. Syst. Eng.*, 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,” *Int. J. Phys. Distrib. Logistics Manage.*, vol. 30, nos. 3/4, pp. 296–313, 2000.
- [5] A. D. Brucker and I. Hang, “Secure and compliant implementation of business process-driven systems,” in *Proc. Bus. Process Manage. Workshops*, 2013, pp. 662–674.
- [6] M. van Giessel, “Process mining in SAP R/3: A method for applying process mining to SAP R/3,” master’s thesis, Eindhoven Univ. Technol., Eindhoven, Netherlands, 2004.
- [7] I. Segers, “Investigating the application of process mining for auditing purposes,” master’s thesis, Eindhoven Univ. Technol., Eindhoven, Netherlands, 2007.
- [8] J. Buijs, “Mapping data sources to XES in a generic way,” master’s thesis, Eindhoven Univ. Technol., Eindhoven, Netherlands, 2010.
- [9] D. Piessens, “Event log extraction from SAP ECC 6.0,” master’s thesis, Eindhoven Univ. Technol., Eindhoven, Netherlands, 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 Univ. Technol., Eindhoven, Netherlands, 2012.
- [11] A. Nigam and N. Caswell, “Business artifacts: An approach to operational specification,” *IBM Syst. J.*, 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,” *Bull. IEEE Comput. Soc. Techn. Committee Data Eng.*, vol. 32, no. 3, pp. 3–9, Sep. 2009.
- [13] R. J. Miller and P. Andritsos, “Schema discovery,” *IEEE Data Eng. Bull.*, vol. 26, no. 3, pp. 40–45, Sep. 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,” *Found. Trends 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. Softw. Eng.*, vol. 16, no. 8, pp. 777–790, Aug. 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. Alhaji, “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. F. van Dongen, and D. Fahland, “Automatic discovery of data-centric and artifact-centric processes,” in *Proc. Bus. Process Manage. Workshops*, 2013, pp. 316–327.
- [21] C. Yu and H. V. Jagadish, “Schema summarization,” in *Proc. 32nd Int. Conf. Very Large Data Bases*, Seoul, Korea, Sep. 12–15, 2006, 2006, pp. 319–330.
- [22] V. Popova, D. Fahland, and M. Dumas, “Artifact lifecycle discovery,” *Int. J. Cooperative Inform. Syst.*, World Scientific, vol. 24, no. 1, 2015.
- [23] J. E. Ingvaldsen and J. A. Gulla, “Preprocessing support for large scale process mining of sap transactions,” in *Proc. Bus. Process Manage. Workshops*, 2008, pp. 30–41.
- [24] A. Ramesh, “Process mining in peoplesoft,” master’s thesis, Eindhoven Univ. Technol., Eindhoven, Netherlands, 2006.
- [25] K. Yano, Y. Nomura, and T. Kanai, “A practical approach to automated business process discovery,” in *Proc. 17th IEEE Int. Enterprise Distrib. Object Comput. Conf. Workshops*, 2013, pp. 53–62.
- [26] W. M. P. van der Aalst, A. Weijters, and L. Maruster, “Workflow mining: Discovering process models from event logs,” *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, Sep. 2004.
- [27] A. Weijters and J. Ribeiro, “Flexible heuristics miner (FHM)” in *Proc. IEEE Symp. Comput. Intell. Data Mining.*, 2011, pp. 310–317.
- [28] A. d. Medeiros, A. Weijters, and W. M. P. van der Aalst, “Genetic process mining: An experimental evaluation,” *Data Mining Knowl. Discovery*, vol. 14, no. 2, pp. 245–304, 2007.
- [29] J. van der Werf, B. F. van Dongen, C. Hurkens, and A. Serebrenik, “Process discovery using integer linear programming,” in *Proc. 29th Int. Conf. Appl. Theory Petri Nets*, 2008, pp. 368–387.
- [30] C. Günther and W. M. P. van der Aalst, “Fuzzy mining—adaptive process simplification based on multi-perspective metrics,” in *Proc. 5th Int. Conf. Bus. Process Manage.*, 2007, pp. 328–343.
- [31] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, “Discovering block-structured process models from event logs—za constructive approach,” in *Proc. 34th Int. Conf. Appl. Theory Petri Nets Concurrency*, 2013, pp. 311–329.
- [32] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, “Discovering block-structured process models from non-conforming event logs,” presented at the 9th Int. Workshop Business Process Intelligence, Beijing, China, 2013.
- [33] J. De Weerd, 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,” *Inf. Syst.*, 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 *Proc. 14th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, Portland, Oregon, USA, Nov. 5–11, 2006, pp. 265–275.
- [35] M. Gabel and Z. Su, “Javert: Fully automatic mining of general temporal properties from dynamic traces,” in *Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2008, pp. 339–349.
- [36] D. Fahland, D. Lo, and S. Maoz, “Mining branching-time scenarios,” in *Proc. 28th IEEE/ACM Int. Conf. Automated Softw. Eng.*, Silicon Valley, CA, USA, Nov. 11–15, 2013, 2013, pp. 443–453.

- [37] M. Pradel and T. Gross, "Automatic generation of object usage specifications from large method traces," in *Proc. 24th IEEE/ACM Int. Conf. Automated Softw. Eng.*, Nov. 2009, pp. 371–382.
- [38] A. Zeller, "Specifications for free," in *Proc. 3rd Int. Symp. NASA Formal Methods*, Pasadena, CA, USA, Apr. 18–20, 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," *IEEE Trans. Softw. Eng.*, vol. 35, no. 5, pp. 684–702, Sep. 2009.
- [40] J. Lou, Q. Fu, S. Yang, J. Li, and B. Wu, "Mining program workflow from interleaved traces," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Washington, DC, USA, Jul. 25–28, 2010, pp. 613–622.
- [41] J. Pinggera, P. Soffer, D. Fahland, M. Weidlich, S. Zugel, B. Weber, H. Reijers, and J. Mendling, "Styles in business process modeling: An exploration and a model," *Softw. Syst. Model.*, vol. 14, no. 3, pp. 1–26, 2013.
- [42] R. Minelli, A. Mocci, M. Lanza, and L. Baracchi, "Visualizing developer interactions," in *Proc. 2nd IEEE Working Conf. Softw. Vis.*, Sep. 2014, pp. 147–156.
- [43] M. Schur, A. Roth, and A. Zeller, "Mining behavior models from enterprise web applications," in *Proc. 9th Joint Meeting Found. Softw. Eng.*, 2013, pp. 422–432.
- [44] D. Fahland, M. de Leoni, B. F. van Dongen, and W. M. P. van der Aalst, "Behavioral conformance of artifact-centric process models," in *Proc. 14th Int. Conf. Bus. Inf. Syst.*, 2011, pp. 37–49.
- [45] D. Fahland, M. de Leoni, B. F. van Dongen, and W. M. P. van der Aalst, "Conformance checking of interacting processes with overlapping instances," in *Proc. 9th Int. Conf. Bus. Process Manage.*, 2011, pp. 345–361.
- [46] E. Nooijen, "Artifact-centric process analysis—Process discovery in ERP systems," master's thesis, Eindhoven Univ. Technol., Eindhoven, Netherlands, 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 *Proc. 5th ACM Int. Conf. Distrib. Event-Based Syst.*, 2011, pp. 51–62.
- [48] A. Petermann, M. Junghanns, R. Muller, and E. Rahm, "BIIIG: enabling business intelligence with integrated instance graphs," in *Proc. IEEE 30th Int. Conf. Data Eng. Workshops*, 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 *Proc. 12th Int. Conf. Bus. Process Manage.*, 2014, pp. 101–117.
- [50] A. Wasylkowski, A. Zeller, and C. Lindig, "Detecting object usage anomalies," in *Proc. 6th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, 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 *Proc. Int. Workshops Bus. Process Manage., BPI, BPD, CBP, ProHealth, RefMod*, Brisbane, Qld., Australia, Sep. 24, 2007, Revised Selected Papers, 2007, pp. 30–41.
- [53] X. Lu, "Artifact-centric log extraction and process discovery," master's thesis, Eindhoven Univ. Technol., Eindhoven, Netherlands, 2013.
- [54] D. Embley and B. Thalheim, *Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges*. New York, NY, USA: 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," *Softw. Syst. Model.*, vol. 9, no. 1, pp. 87–111, 2010.



Xixi Lu received the bachelor's degree in computer science and the master's of science degree in business information systems from the Eindhoven University of Technology. She is currently working toward the PhD degree at the Eindhoven University of Technology researching in the areas of architecture of business information systems and process mining. She also studied economics and management at Tsinghua university as an exchange student. Her research interests include business process management, conformance checking between process models and recorded logs, deviation detection and analysis, and process mining in general.



Marijn Nagelkerke received the master's of science degree in business information systems from the Eindhoven University of Technology. RE is advisor in the KPMG IT Advisory practice in The Netherlands. 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. He 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 received the master's of science degree in econometrics from Maastricht University degree. RE is manager in the KPMG IT Advisory practice in The Netherlands. 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. He is a lead in 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 he specializes in large complex implementations at mainly corporate clients of KPMG.



Dirk Fahland received the PhD degree in computer science from the Humboldt-Universität zu Berlin, Germany, and the Eindhoven University of Technology, the Netherlands, in 2010. He is an assistant professor at the Eindhoven University of Technology researching in the area of distributed systems. 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*.