Extracting Multiple Viewpoint Models from Relational Databases

Alessandro Berti[1] [0000−0003−1830−4013] and Wil van der Aalst[1] [0000−0002−0955−6940]

Process and Data Science department, Lehrstuhl fur Informatik 9 52074 Aachen, RWTH Aachen University, Germany

Abstract. Much time in process mining projects is spent on finding and understanding data sources and extracting the event data needed. As a result, only a fraction of time is spent actually applying techniques to discover, control and predict the business process. Moreover, current process mining techniques assume a *single* case notion. However, in real-life processes often different case notions are intertwined. For example, events of the same order handling process may refer to customers, orders, order lines, deliveries, and payments. Therefore, we propose to use *Multiple Viewpoint (MVP) models* that relate events through objects and that relate activities through classes. The required event data are much closer to existing relational databases. MVP models provide a holistic view on the process, but also allow for the extraction of classical event logs using different viewpoints. This way existing process mining techniques can be used for each viewpoint without the need for new data extractions and transformations. We provide a toolchain allowing for the discovery of MVP models (annotated with performance and frequency information) from relational databases. Moreover, we demonstrate that classical process mining techniques can be applied to any selected viewpoint.

Keywords: Process Mining · Process Discovery · Artifact-Centric Process Models · Relational Databases.

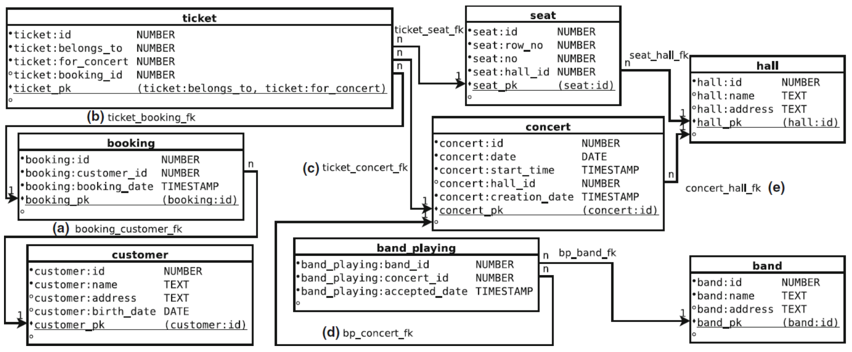
1 Introduction

Process mining is a growing branch of data science that aims to extract insights from event data recorded in information systems. Examples of process mining techniques include process discovery algorithms that are able to find descriptive process models, conformance checking algorithms that compare event data with a given process model to find deviations, and predictive algorithms that use the discovered process model to anticipate bottlenecks or compliance problems. Gathering high-quality event data is a prerequisite for the successful application of process mining projects. However, event data are often hidden in existing information systems (e.g., the ERP systems of SAP, Microsoft, Oracle). Most systems are built on top of relational databases, to ensure data integrity and normalization. Relational databases contain entities (tables) and relations between entities. Events correspond to updates of the database, i.e., changes of the "state" of the information system. These updates may have been stored in tables of the databases (through *in-table versioning* like the change tables in SAP or any table that contains dates or timestamps) or can be retrieved using some database log (like *redo logs* [37, 28] explicitly storing all database updates).

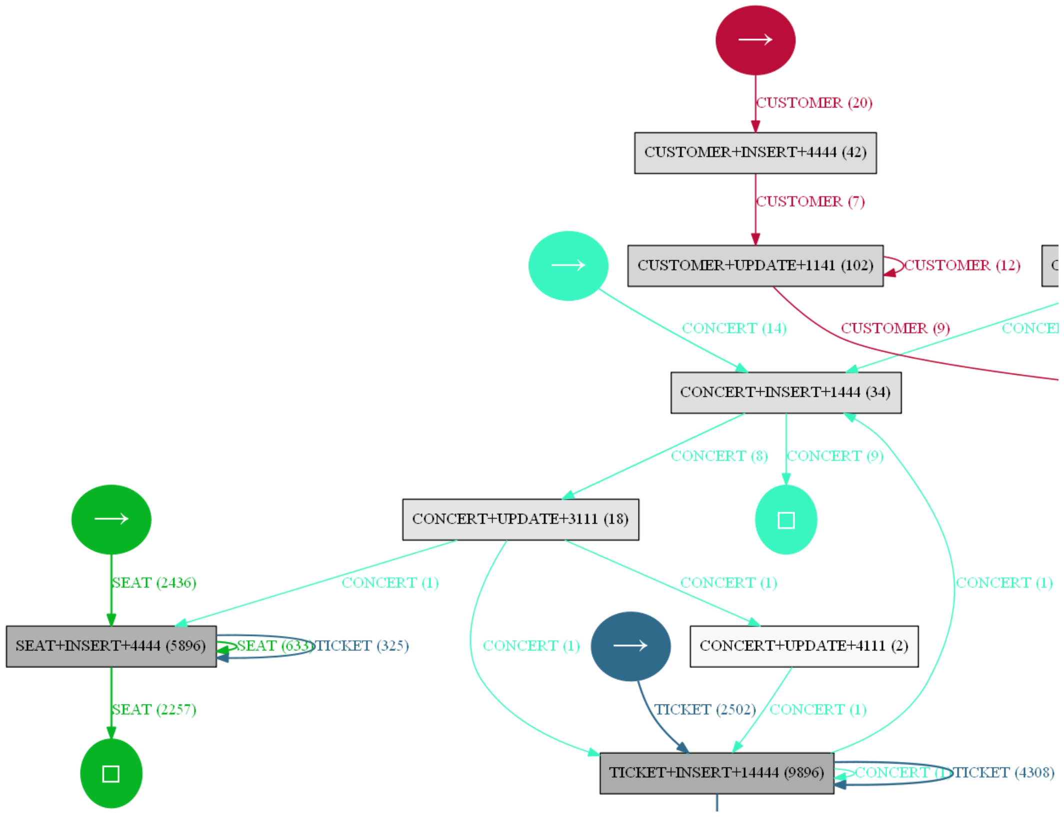
Extracting events from a database requires domain knowledge and may be very time-consuming. One of the reasons is that process mining techniques require a classical event log where each event needs to refer to a case. The case notion is used to "correlate" events and the corresponding process model shows the life-cycle model for the selected case notion. However, in the same process there may be suppliers, customers, orders, order lines, deliveries, and payments. These correspond to objects of the class model describing the database. One event may refer to a subset of such objects. A payment may refer to the payment itself, a customer, and an order. A delivery may refer to the delivery itself, but also to a subset of order lines or even multiple orders. Several views on the database could be retrieved, this means that, for the same database, several event logs and process models need to be extracted. When the process involves many different entities, the construction of the view is not easy. While there are some methods to automatically infer a/some case notion(s) from unstructured data [2, 15, 6], in most cases the specification happens manually. Moreover, the data extractions and transformations may be time-consuming and one quickly loses the overview. This often leads to divergence and convergence errors: events are forgotten or inadvertently duplicated leading to incorrect conclusions.

This paper introduces a new modeling technique that is able to calculate a graph where relationships between activities are shown without forcing the user to specify a case notion, since different case notions are combined in one succint diagram. The resulting models are called MVP models. Such models belong to the class of artifact-centric models [11, 31] that combine data and process in a holistic manner to discover patterns and check compliance [22]. MVP models are annotated with frequency and performance information (e.g., delays) and provide a holistic view on the whole process. The colors of the relationships refer to the original classes. Using frequency based filtering and selections the MVP model can be seamlessly simplified. Any non-empty subset of classes provides a *viewpoint*. Given a viewpoint and an MVP model, we can automatically generate a classical event log and apply existing process mining techniques. Hence, the holistic view of the MVP model is complemented by detailed viewpoint models using conventional notations like Petri nets, BPMN models, process trees, or simple directly-follows graphs.

The techniques have been implemented using PM4Py (pm4py.org). In-memory computation is used to handle large data sets quickly. The approach has been evaluated using a log extracted from a real-life information system, and has proven to scale linearly with the number of events, classes and asymptotically linearly with the number of objects per class, while the execution time grows quadratically with the number of activities. This is a stark contrast with existing techniques (like OCBC models) with significantly worse complexity.



(a) Database model of a concert database.



(b) MVP model annotated with frequencies.

Fig. 1. Illustration of the approach using an example database taken from [29]. The classes are highlighted using different colors and labels on arcs. Based on a viewpoint (i.e., a set of classes) a classical event log can be generated and analyzed.

In Section 7.4, an assessment on real-life database event logs is done. Moreover, a comparison with some existing techniques (OpenSLEX [29, 27] and OCBC models [1]) is performed, considering the execution time and the usability of these approaches.

The remainder of the paper is organized as follows. Section 2 presents related work. In Section 3 classical and database event logs are introduced. Section 4 presents our approach to discover Multiple Viewpoint (MVP) models including Event-to-Object (E2O) graphs, Event-to-Event (E2E) multigraphs, and Activity-to-Activity (A2A) multigraphs. Section 5 introduces the notion of viewpoints and the automatic creation of classical event logs. Section 6 presents our implementation using PM4Py. In Section 7, we evaluate MVP models and our implementation by comparing results for real-life data sets with competing techniques.

2 Related Work

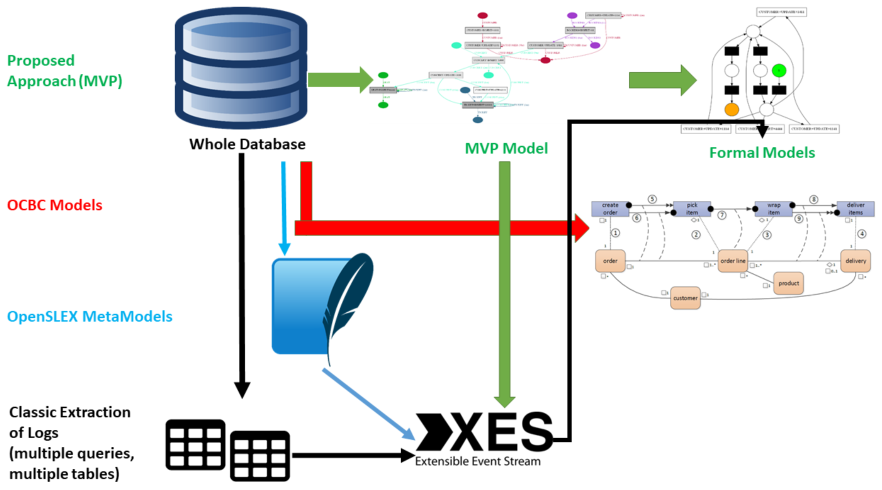


Fig. 2. Comparison of four different process mining ETL scenarios. Classical log extraction consider different queries, possibly targeting different tables, to prepare event logs for classic process mining techniques. OpenSLEX provides an easier access point to retrieve event data, but still requires the manual specification of database queries. OCBC technique extracts a single process model from the database schema, that could be useful for the purpose of understanding and checking the schema, but provide no way to retrieve a classical event log. MVP technique, on the other hand, provides an easy visualization of the relationships between activities on top of a database, moreover it provides the possibility to get formal models (directly from the MVP models) and event logs to use with classic process mining techniques.

Related work may be divided into different categories:

– Approaches to extract event data from databases and to make queries easier.
– Representation of Artifact-centric models.
– Discovery of process models combining several case notions.

A few example approaches are shown in Fig. 2 and related to MVP models.

2.1 Related Work: Extracting Event Data From Databases

There has been earlier work on making SQL queries easier [8, 5]. The basic idea is to provide the business analyst a way to express queries in a simpler language (SPARQL query). Some other papers related to database querying in the context of process management are [26, 21, 33, 25, 35].

In [32], an automated technique to discover, for each notion of data object in the process, a separate process model that describes the evolution of this object, is presented. The technique is based on relational databases, and decomposes the data source into multiple logs, each describing the cases of a separate data object.

The OpenSLEX [29, 27] is an high-level meta-model, that permits easier queries, obtained from the raw database data scattered through tables (for example, the case identifier, the activity and the timestamp may be columns of different tables that need to be joined together). The aim of OpenSLEX is to let user focus on the analysis, dealing only with elements such as events, activities, cases, logs, objects, objects versions, object classes and attributes that are introduced in [29]. The meta-model could be seen as a schema that captures all the pieces of information necessary to apply process mining to database environments. To obtain classical event logs, a case notion (connecting events to each other) needs to be used. The OpenSLEX implementation provides indeed some connectors for database logs (redo logs, in-table versioning, or specific database formats [27, 18]). In the implementation described in [29], OpenSLEX is supported by an SQLite database.

2.2 Related Work: Representation of Artifact-Centric Models

Business artifacts (cf. [11, 30]) combine data and process in an holistic manner as the basic building block. These correspond to key business entities which evolve as they pass through the business's operation.

In [4] a formal artifact-based business model, and declarative semantics based on the use of business rules, are introduced along with a preliminary set of technical results on the static analysis of the semantics of an artifact-based business process model.

The Guard-Stage-Milestone (GSM) meta-model [17, 16] is a formalism for designing business artifacts in which the intended behavior is described in a declarative way, without requiring an explicit specification of the control flow.

Some approaches which focus on compliance checking are introduced in [19, 24]. In [19], support for data-aware compliance rules is proposed in a scalable way thanks to an abstraction approach that can serve as preprocessing step to the actual compliance checking. In [24], compliance rule graphs are introduced, that are a framework with support for root cause analysis, and that can provide assistance in proactively enforcing compliance by deriving measures to render the rule activation satisfied. In [14], conformance checking on artifact-centric processes is approached by partitioning the problem into behavioral conformance of single artifacts and interaction conformance between artifacts, solving behavioral conformance by a reduction to existing techniques.

In [34], the concept of relational process structure is introduced, aiming to overcome some limitations of small processes such as artifacts, object lifecycles, or Proclets. In these paradigms, a business process arises from the interactions between small processes. However, many-to-many relationships support is lacking. The relational process structure provides full support for many-to-many relationships and cardinality constraints at both design- and run-time.

2.3 Related Work: Discovery of Process Models Using Several Case Notions

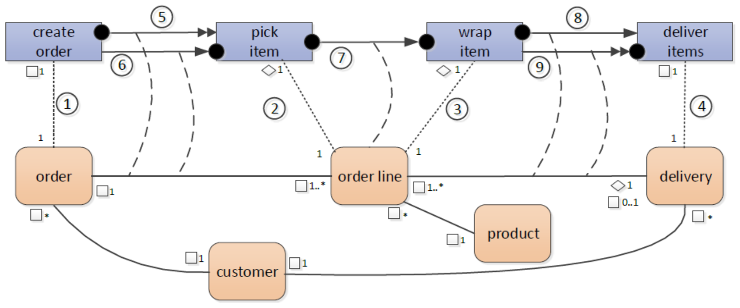


Fig. 3. Representation of a small Object-centric Behavioral Constraint (OCBC) model (taken from [1]).

In this category, we aim to describe some models that could be discovered by combining several case notions, or studying the interactions between the different case notions: interacting artifacts [23], multi-perspective models [13], object-centric models [1].

In [23], a semi-automatic approach is presented to discover the various objects supporting the system from the plain database of an ERP (Enterprise Resources Planning) system. An artifact-centric process model is identified describing the system's objects, their life-cycles, and some detailed information about interactions between objects.

Multi-instance mining [13] was introduced to discover models where different perspectives of the process can be identified. Instead of focusing on the events or activities that are executed in the context of a particular process, the focus is on the states of the different perspectives and on the discovery of the relationships between them. The *Composite State Machine Miner* [12] supports the approach [13]. It quantifies and visualises the interactions between perspectives to provide additional process insights.

Object-centric models [1] are process models, involving entities and relations in the ER model, where multiple case notions may coexist. A small OCBC model is represented in Fig. 3. In the representation, the lower part (yellow) represents the class model, and the upper part represents the activities and the constraints between them. The activities and the classes are connected by arcs if they are in relationship. OCBC models can be discovered, and can be used for compliance checking, from XOC logs. The XOC format is important because it extends the XES format with support to database-related information (related objects, relationships, state of the object model), and is one of the few choices to store database event logs along with instances of the OpenSLEX meta-model. XOC logs are using XML and contain a list of events. Each event is referring some objects, and contains the status of the database at the moment the event happened. In [20], the algorithm to infer OCBC models is described, that takes an XOC log as well as a set of possible behavioral constraint types as input, that means users can specify the constraint type set based on their needs. ProM 6 plug-ins have been realized to import XOC logs, for the discovery of a process model, and for conformance checking on top of OCBC models. The discovery algorithm can discover constraints of 9 different types.

3 Database Event Logs

Relational databases are organized in entities (classes of *objects* sharing some properties), relationships (connections between entities [10]), attributes (properties of entities and relationships). Events can be viewed as updates of a database (e.g. insertion of new objects, changes to existing objects, removal of existing objects). Some ways to retrieve events from databases are:

- Using redo logs (see [29]). These are logs where each operation in the database is saved with a timestamp; this helps to guarantee consistency, and possibility to rollback and recovery.
- Using in-table versioning. In this case, the primary key is extended with a timestamp column. For each phase of the lifecycle of an object, a new entry is added to the in-table versioning, sharing the base primary key values but with different values for the timestamp column.

An event may be linked to several objects (for example, the event that starts a marketing campaign in a CRM system may be linked to several customers), and an object may be linked to several events (for example, each customer can be related to all the tickets it opens). For the following definition, let $\mathcal{U}_E$ be the universe of events (all the events happening in a database context), $\mathcal{U}_C$ be the universe of case identifiers, $\mathcal{U}_A$ be the universe of activities (names referring to a particular step of a process), $\mathcal{U}_{\text{attr}}$ be the universe of attribute names (all the names of the attributes that can be related to an event), $\mathcal{U}_{\text{val}}$ be the universe of attribute values (all the possible values for attributes).

In this paper we consider event data closer to real-life information systems. Before providing a definition for database event logs, we define the classical event log concept.

Definition 1 (Classical Event Log) *A log is a tuple* $L = (C_I, E, A, \mathit{case\_ev}, \mathit{act}, \mathit{attr}, \leq)$ *where:*

- $C_I \subseteq \mathcal{U}_C$ *is a set of case identifiers.*
- $E \subseteq \mathcal{U}_E$ *is a set of events.*
- $A \subseteq \mathcal{U}_A$ *is the set of activities.*
- $\mathit{case\_ev} \in C_I \to \mathcal{P}(E) \backslash \{\emptyset\}$ *maps case identifiers onto set of events (belonging to the case).*
- $\mathit{act} \in E \to \mathcal{U}_A$ *maps events onto activities.*
- $\mathit{attr} \in E \to (\mathcal{U}_{attr} \not\rightarrow \mathcal{U}_{val})$ *maps events onto a partial function assigning values to some attributes.*
- $\leq \;\subseteq E \times E$ *defines a total order on events.*

This classical event log notion matches the XES storage format [36], that is the common source of information for process mining tools like Disco, ProcessGold, Celonis, QPR, Minit, . . . An example attribute of an event e is the timestamp $\text{attr}(e)(\mathit{time})$ which refer to the time the event happened. While, in general, an event belongs to a single case, in Def. 1 the function case_ev might be such that cases share events.

For events extracted from a database, the function case_ev is not given, since an event may be related to different objects, and different case notions may exist. In the following Def. 2, database event logs are introduced.

Definition 2 (Database Event Log) *Let* $\mathcal{U}_O$ *be the universe of objects (all the objects that are instantiated in the database context) and* $\mathcal{U}_{OC}$ *be the universe of object classes (a class defines the structure and the behavior of a set of objects). A database event log is a tuple* $L_D = (E, O, C, A, \mathit{class}, \mathit{act}, \mathit{attr}, \mathit{EO}, \leq)$ *where:*

- $E \subseteq \mathcal{U}_E$ *is the set of events.*
- $O \subseteq \mathcal{U}_O$ *is the set of objects.*
- $C \subseteq \mathcal{U}_{OC}$ *is the set of object classes.*
- $A \subseteq \mathcal{U}_A$ *is the set of activities.*
- $\mathit{class} : O \to C$ *is a function that associates each object to the corresponding object class.*
- $\mathit{act} \in E \to A$ *maps events onto activities.*
- $\mathit{attr} \in E \to (\mathcal{U}_{attr} \not\rightarrow \mathcal{U}_{val})$ *maps events onto a partial function assigning values to some attributes.*
- $\mathit{EO} \subseteq E \times O$ *relates events to sets of object references.*
- $\leq \;\subseteq E \times E$ *defines a total order on events.*

This definition differs from Def. 1 because a case notion is missing (no *case_ev* function) and events are related to objects in a possible many-to-many relation. A function *EO* is introduced that relates events to sets of object references. Moreover, the sets of objects and classes in the event log are specified, and a function *class* that associates each object to its class is introduced.

4 Approach to Obtain MVP models

In this section, the different ingredients of MVP models will be introduced. The E2O graph will be obtained directly from the database logs; the E2E multigraph will be obtained in linear complexity by calculating directly-follows relationships between events in the perspective of some object; the A2A multigraph will be obtained in linear complexity by calculating directly-follows relationships between activities in the perspective of some object class, using the information stored in the E2E multigraph. The A2A multigraph is the main contributor to the visualization of the MVP model. A projection function will be given in Section 5 to obtain a classical event log when a so-called viewpoint is chosen.

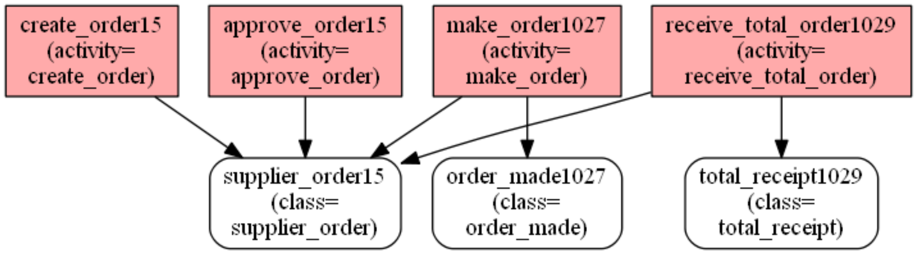

create_order15 (activity= create_order)
approve_order15 (activity= approve_order)
make_order1027 (activity= make_order)
receive_total_order1029 (activity= receive_total_order)
supplier_order15 (class= supplier_order)
order_made1027 (class= order_made)
total_receipt1029 (class= total_receipt)

Fig. 4. Visualization of part of the E2O graph of an example database event log (found in the erp.xoc test file). Events (red nodes) are connected to objects (white nodes).

MVP models are composed of several graphs (E2O, E2E, A2A) and auxiliary functions (a complete definition will be presented at the end of this section), and are constructed by reading a representation of event data retrieved from a database (importing from intermediate structures like OpenSLEX or XOC logs).

Definition 3 (E2O Graph) *Let* $L_D = (E, O, C, A, class, act, attr, EO, \leq)$ *be a database event log. The Event-to-Object graph (E2O) corresponding to the database event log* L_D *can be defined as:*

$$E2O(L_D) = (E \cup O, EO)$$

Here, the nodes are the events (E) and the objects (O), and EO (as retrieved from the log) is a subset of $E \times O$.

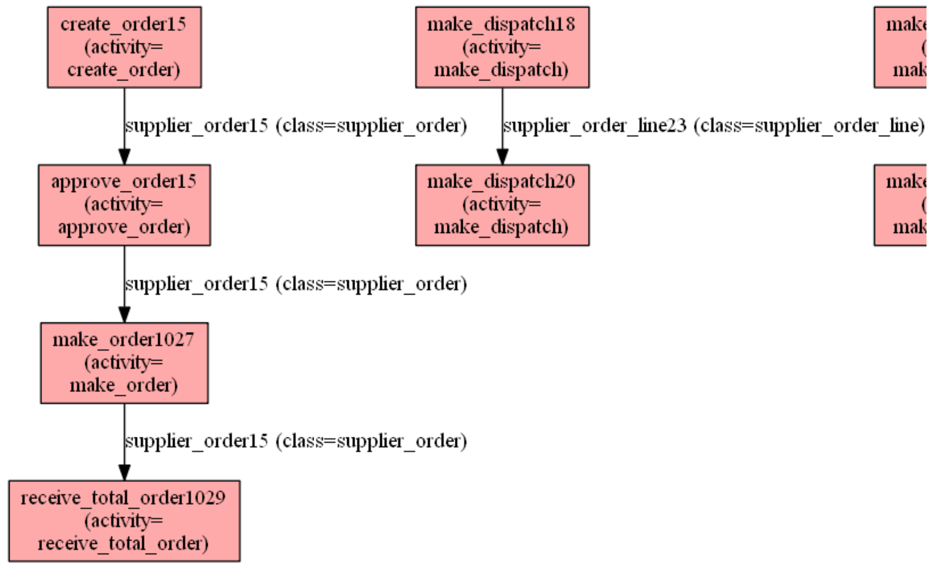

create_order15 (activity= create_order)
supplier_order15 (class=supplier_order)
approve_order15 (activity= approve_order)
supplier_order15 (class=supplier_order)
make_order1027 (activity= make_order)
supplier_order15 (class=supplier_order)
receive_total_order1029 (activity= receive_total_order)
make_dispatch18 (activity= make_dispatch)
supplier_order_line23 (class=supplier_order_line)
make_dispatch20 (activity= make_dispatch)

Fig. 5. Visualization of part of the E2E multigraph of an example database event log (erp.xoc test file). Events are connected to events; in the edge label, the (object) perspective has been reported.

The E2O graph is obtained directly from the data without any transformation. The remaining steps in the construction of an MVP model are the construction of the E2E multigraph and of the A2A multigraph.

Definition 4 (Sequence of related events) *Let $L_D = (E, O, C, A, class, act, attr, EO, \leq)$ be a database event log. For $o \in O$, the following sequence of related events is defined:*

$$\widetilde{O}(o) = \{e_1, \ldots, e_n\}$$

such that $\{e_1, e_2, \ldots, e_n\} = \{e \mid (e, o) \in EO\}$ and $\forall_{1 \leq i < j \leq n} \;\; e_i < e_j$.

Definition 5 (E2E Multigraph) *Let $L_D = (E, O, C, A, class, act, attr, EO, \leq)$ be a database event log. The Event-to-Event multigraph (E2E) on the database event log L_D can be defined as:*

$$E2E(L_D) = (E, F_E, \Pi^E_{perf})$$

Where the nodes are the events (E) and the set of edges F_E is defined as:

$$F_E = \{(e_1, e_2, o) \in E \times E \times O \mid \exists_{2 \leq i \leq |\widetilde{O}(o)|} \; \widetilde{O}_{i-1}(o) = e_1, \widetilde{O}_i(o) = e_2\}$$

and $\Pi^E_{perf} : F_E \rightarrow \mathbb{R}^+ \cup \{0\}$, $\Pi^E_{perf}(e_1, e_2, o) = attr(e_2)(time) - attr(e_1)(time)$ associates each edge to a non-negative real number expressing its duration (performance).

The introduction of the set F_E is useful for the definition of the A2A multigraph. Although it does not make sense to represent the overall E2E multigraph, involving relationships between all events, it may be useful to display directly-follows relationships involving a small subgroup of events.

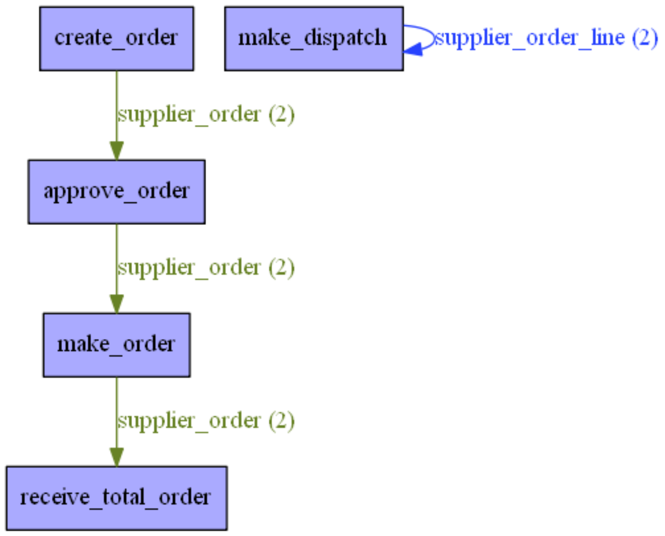


Fig. 6. Visualization of the A2A multigraph of an example database event log (erp.xoc test file). Activities are connected to activities; in the edge label, the (class) perspective along with the count of the occurrences has been reported.

Definition 6 (A2A Multigraph) *Let $L_D = (E, O, C, A, class, act, attr, EO, \leq)$ be a database event log. Let $AE : A \times A \times C \rightarrow \mathcal{P}(E \times E \times O)$ such that for $a_1, a_2 \in A$ and $c \in C$:*

$$AE(a_1, a_2, c) = \{(e_1, e_2, o) \in F_E \mid act(e_1) = a_1 \wedge act(e_2) = a_2 \wedge class(o) = c\}$$

The AE function associates to each triple (a_1, a_2, c) the set of all the events of the corresponding activities and classes. The Activity-to-Activity multigraph (A2A) on the database event log L_D can be defined as:

$$A2A(L_D) = (A, F_A, \Pi^A_{count}, \Pi^A_{perf})$$

Where the nodes are the activities (A), the set of edges F_A is defined as:

$$F_A = \{(a_1, a_2, c) \in A \times A \times C \mid AE(a_1, a_2, c) \neq \emptyset\}$$

and:

- $\Pi^A_{count}(a_1, a_2, c) = |AE(a_1, a_2, c)|$ *is the number of occurrences associated to the edge* $(a_1, a_2, c) \in F_A$*, that is the number of corresponding edges contained in* $AE(a_1, a_2, c)$*.*
- $\Pi^A_{perf}(a_1, a_2, c) = \frac{\sum_{f_E \in AE(a_1,a_2,c)} \Pi^E_{perf}(f_E)}{\Pi^A_{count}(a_1,a_2,c)}$ *is the performance associated to the edge* $(a_1, a_2, c) \in F_A$*, that is the average of the duration of the corresponding edges contained in* $AE(a_1, a_2, c)$*. An high average duration may correspond to a bottleneck in the process.*

The following definitions are useful for the representation of an MVP model, introducing clear start and end points for each class and contributing to the possibility to filter out edges.

Definition 7 (Start and End Activities of a Class)
Let $L_D = (E, O, C, A, class, act, attr, EO, \leq)$ *be a database event log. Let* $c \in C$ *be a class. The following functions are defined:*

- $START_A(L_D) : C \to \mathcal{P}(A)$, $START_A(L_D)(c) = \{act(\widetilde{O}_1(o)) \mid o \in O \wedge |\widetilde{O}(o)| \geq 1 \wedge class(o) = c\}$ *is the set of start activities of class c.*
- $END_A(L_D) : C \to \mathcal{P}(A)$, $END_A(L_D)(c) = \{act(\widetilde{O}_{|\widetilde{O}(o)|}(o)) \mid o \in O \wedge |\widetilde{O}(o)| \geq 1 \wedge class(o) = c\}$ *is the set of end activities of class c.*

Definition 8 (Dependency Threshold between Activities given a Class)

Let $L_D = (E, O, C, A, class, act, attr, EO, \leq)$ *be a database event log. Let* F_A *be the set of edges in* $A2A(L_D)$*. For* $(a_1, a_2, c) \in F_A$ *it is possible to define a dependency measure in the following way:*

$$dep_A(L_D) : F_A \to [0, 1]$$

$$dep_A(L_D)(a_1, a_2, c) = \begin{cases} \frac{\Pi^A_{count}(a_1,a_2,c)}{\Pi^A_{count}(a_1,a_2,c)+1} & \text{if } a_1 = a_2 \vee (a_2, a_1, c) \notin F_A \\ \frac{\Pi^A_{count}(a_1,a_2,c)-\Pi^A_{count}(a_2,a_1,c)}{\Pi^A_{count}(a_1,a_2,c)+\Pi^A_{count}(a_2,a_1,c)+1} & \text{if } a_1 \neq a_2 \wedge (a_2, a_1, c) \in F_A \end{cases}$$

Definition 9 (MVP Discovery) *Let* $L_D = (E, O, C, A, class, act, attr, EO, \leq)$ *be a database event log. We define as MVP model discovered from the log* L_D*, and we refer to it as* $MVP(L_D)$*, the following object:*

$$MVP(L_D) = (L_D, E2O(L_D), E2E(L_D), A2A(L_D), START_A(L_D), END_A(L_D), dep_A(L_D))$$

Given a dependency threshold $d \in [-1, 1]$, a representation of an MVP model draws as many edges between a couple of activities $(a_1, a_2) \in A \times A$ as the number of classes $c \in C$ such that $dep_A(L_D)(a_1, a_2, c)$ is defined and $\text{dep}_A(L_D)(a_1, a_2, c) \geq d$.

The visualization of a MVP model is valuable:

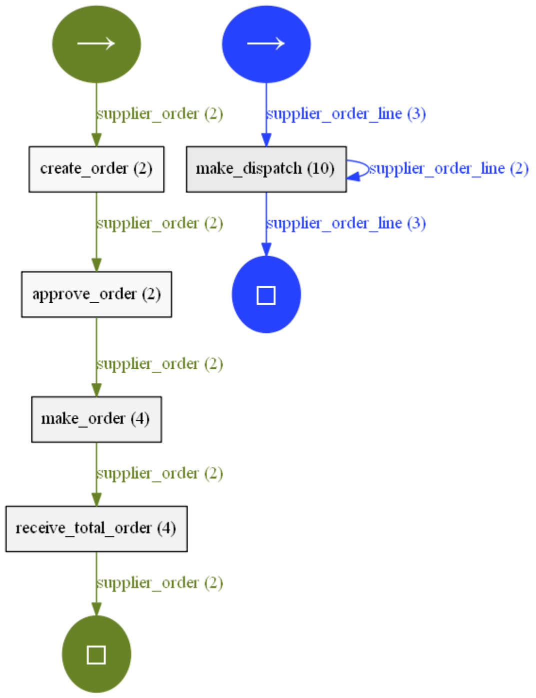


Fig. 7. Visualization of the MVP Model of an example database event log (erp.xoc test file). Activities are connected to activities; in the edge label, the (class) perspective along with the count of the occurrences has been reported. In addition to the A2A multigraph, start and end nodes are associated to each class; moreover, the edges that are reported in the A2A multigraph are filtered according to the dependency threshold (in this simple example, no arc is filtered).

- An holistic view on the classes and the activities of the database, and on the order in which they happen, is provided.
- Arcs are decorated with frequency/performance information. Frequency information helps to understand the most frequent paths of a process, and performance information helps to discover the bottlenecks of a process.

5 Viewpoints: Retrieval of DFGs and Logs

The goal of this section is to provide some ways to retrieve, from an MVP model MVP(L_D), built upon the database event log L_D, a particular viewpoint on the model.

Definition 10 (Viewpoint)
Let $L_D = (E, O, C, A, class, act, attr, EO, \leq)$ be a database event log. Let

$$MVP(L_D) = (L_D, E2O(L_D), E2E(L_D), A2A(L_D), START_A(L_D), END_A(L_D), dep_A(L_D))$$

be an MVP model. A viewpoint is a set of classes $V(L_D) \subseteq C$. A viewpoint is corresponding to a subset of edges in the E2E graph (that is F_E):

$$V_E(L_D) = \{(e_1, e_2, o) \in F_E \mid o \in V(L_D)\}$$

From a view, we can obtain two different final outputs.

– A Directly-Follows Graph (DFG), that includes the edges related to the classes contained in $V(L_D)$.
– A classical event log, that includes all the events that are related to the objects having a class contained in $V(L_D)$.

An example of projection could be found in Fig. 8, where a singleton viewpoint containing a single class is chosen, a Directly-Follows Graph is obtained and a Petri net is obtained through the application of Inductive Miner Directly-Follows.

5.1 Projection on a Directly-Follows Graph

The concept of a Directly-Follows Graph is introduced in the following definition:

Definition 11 (DFG) *A Directly-Follows Graph is a weighted directed graph:*

$$DFG = (N_{DFG}, E_{DFG}, c_{DFG})$$

Where N_{DFG} (the nodes) are the activities, and $E_{DFG} \subseteq N_{DFG} \times N_{DFG}$ is the set of all the edges between activities that happened in direct succession, and $c_{DFG} : E_{DFG} \to \mathbb{R}^+$ is the count function that aims to represent how many times two different activities happened in direct succession.

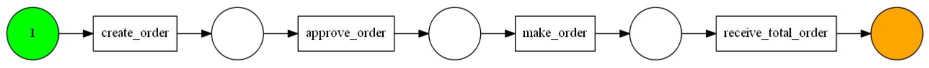


Fig. 8. Petri net extracted using Inductive Miner Directly-Follows from the projection of the MVP model extracted from the erp.xoc test file on the viewpoint containing the *supplier_order* class.

From a viewpoint $V(L_D)$ on MVP(L_D), it is possible to obtain a Directly-Follows Graph as explained in the following definition:

Definition 12 (DFG given a viewpoint) *Given a viewpoint $V(L_D)$ on $MVP(L_D)$, a DFG could be obtained taking:*

- $N_{DFG} = \bigcup_{(e_1,e_2,o) \in V_E(L_D)} \{act(e_1), act(e_2)\}$
- $E_{DFG} = \bigcup_{(e_1,e_2,o) \in V_E(L_D)} \{(act(e_1), act(e_2))\}$
- $c_{DFG}(a_1, a_2) = |\{(e_1, e_2, o) \in V_E(L_D) \mid act(e_1) = a_1 \wedge act(e_2) = a_2\}|$

where $V_E(L_D)$ is obtained as in Def. 10.

An example could be provided. Let $V_E(L_D) = \{(e_1, e_2, o_1), (e_2, e_3, o_1), (e_4, e_5, o_2), (e_5, e_6, o_2)\}$ be the set of edges associated to a viewpoint, such that:

- $\text{act}(e_1) = \text{act}(e_4) = A$
- $\text{act}(e_2) = \text{act}(e_5) = B$
- $\text{act}(e_3) = C$
- $\text{act}(e_6) = D$

Then the DFG is such that $N_{\text{DFG}} = \{A, B, C, D\}$, $E_{\text{DFG}} = \{(A, B), (B, C), (B, D)\}$, $c_{\text{DFG}}(B, C) = c_{\text{DFG}}(B, D) = 1$, $c_{\text{DFG}}(A, B) = 2$.

5.2 Projection on a Log

The projection of an MVP model MVP(L_D) obtained from a database event log L_D to a classical event log (see Section 3) could be introduced when a viewpoint $V(L_D) \subseteq C$ is chosen.

Indeed, the information contained in an MVP model could be used to determine a case notion C_D that is used to transform the database event log L_D into a classical event log, in a way that events belonging to the same process execution can be grouped.

The definition of case notion on database event logs could be introduced:

Definition 13 (Case Notion) *Let $L_D = (E, O, C, A, class, act, attr, EO, \leq)$ be a database event log. A case notion is a set of sets of events $C_D \subseteq \mathcal{P}(E) \setminus \{\emptyset\}$.*

The case notion does not need to cover all the events contained in E, moreover the intersection between sets of events contained in the case notion may also not be empty.

An example of case notion could be provided. Let $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ be a set of events. Then a case notion might be $C_D = \{\{e_1, e_2\}, \{e_3, e_4\}, \{e_1, e_4, e_5\}\}$. Let's note that the union of all these sets is not E, and the intersection between $\{e_1, e_2\}$ and $\{e_1, e_4, e_5\}$ is not empty.

When a case notion is defined, it is possible to define the projection function from the database event log L_D to the classical event log. An assumption is that the case notion C_D is contained in the universe of case identifiers $\mathcal{U}_C$. This helps to define the function case_ev in a simpler way.

Definition 14 (Projection function) *Let $L_D = (E, O, C, A, class, act, attr, EO, \leq)$ be an event log in a database context. Let $C_D \subseteq \mathcal{P}(E) \setminus \{\emptyset\}$ be a case notion. Then it is possible to define a projection function from a database event log to a classical event log as:*

$$proj(L_D, C_D) = (C_D, E, A, case\_ev, act, attr, \leq)$$

where $case\_ev \in C_D \to \mathcal{P}(E) \setminus \{\emptyset\}$ is such that for all $c \in C_D$, $case\_ev(c) = c$.

Given an MVP model MVP(L_D) and a viewpoint $V(L_D)$ on that defines a set of edges $V_E(L_D)$ in the E2E multigraph, a case notion is defined as:

$$\mathrm{C}_D = \left\{ \bigcup_{o' \in O, \mathrm{class}(o') \in V(L_D), \widetilde{O}(o) \cap \widetilde{O}(o') \neq \emptyset} \widetilde{O}(o') \;\mid o \in O, \mathrm{class}(o) \in V(L_D) \right\}$$

A classical event log is obtained as $L = \mathrm{proj}(L_D, C_D)$.

6 Tool

MVP Models have been implemented in a feature branch of the PM4Py Process Mining library[1] [3]. The architecture of the tool provides a clear separation between the management of the log (object), the MVP discovery algorithm and the MVP visualization. Moreover, utilities have been provided to generate a database event log and to visualize the E2O and the E2E multigraphs. A reference technical manual with a description of the features provided in tool is contained in http://www.alessandroberti.it/technical.pdf.

The provided features are:

- **Log management**: log importing (XOC, OpenSLEX, Parquet), log exporting (XOC, Parquet).
- **Model discovery**: discovery of a MVP model with frequency or performance decoration.
- **Visualization**: E2O graphs, E2E and A2A multigraphs, MVP models (with possibility to filter out edges with a dependency measure that is below the threshold).
- **Database log generation**: the option to generate an MVP model specifying the number of events, activities, classes and a number of objects for each class is provided.
- **Projection on a Viewpoint**: projection on a DFG and on a log.
- **Storage of MVP models**: importing/exporting of MVP models into a dump file.

Example logs are provided in the tests folder of the repository, in particular:

[1] The repository can be accessed at the URL https://github.com/Javert899/pm4py-source

event_id	event_activity	event_timestamp	supplier_order	supplier_order_line
create_order15	create_order	2016-10-21 11:38:26	supplier_order15	NaN
approve_order15	approve_order	2016-10-21 11:38:53	supplier_order15	NaN
make_order1027	make_order	2016-10-21 11:40:00	NaN	NaN
make_order1027	make_order	2016-10-21 11:40:00	supplier_order15	NaN
make_dispatch19	make_dispatch	2016-10-21 11:42:31	NaN	NaN
make_dispatch19	make_dispatch	2016-10-21 11:42:31	NaN	supplier_order_line22
make_dispatch18	make_dispatch	2016-10-21 11:42:31	NaN	NaN
make_dispatch18	make_dispatch	2016-10-21 11:42:31	NaN	supplier_order_line23
receive_partial_order1028	receive_partial_order	2016-10-21 11:43:00	NaN	NaN
make_dispatch21	make_dispatch	2016-10-21 11:44:25	NaN	NaN
make_dispatch21	make_dispatch	2016-10-21 11:44:25	NaN	supplier_order_line22
make_dispatch20	make_dispatch	2016-10-21 11:44:25	NaN	NaN
make_dispatch20	make_dispatch	2016-10-21 11:44:25	NaN	supplier_order_line23
receive_total_order1029	receive_total_order	2016-10-21 11:45:00	NaN	NaN
receive_total_order1029	receive_total_order	2016-10-21 11:45:00	supplier_order15	NaN
create_invoice17	create_invoice	2016-10-21 11:45:46	NaN	NaN
create_invoice17	create_invoice	2016-10-21 11:45:46	NaN	NaN
create_payment11	create_payment	2016-10-21 11:46:14	NaN	NaN
create_payment11	create_payment	2016-10-21 11:46:14	NaN	NaN
create_payment12	create_payment	2016-10-21 11:46:29	NaN	NaN
create_payment12	create_payment	2016-10-21 11:46:29	NaN	NaN
create_order16	create_order	2016-10-21 11:56:35	supplier_order16	NaN
approve_order16	approve_order	2016-10-21 11:56:50	supplier_order16	NaN
make_order1033	make_order	2016-10-21 11:57:00	NaN	NaN
make_order1033	make_order	2016-10-21 11:57:00	supplier_order16	NaN
make_dispatch22	make_dispatch	2016-10-21 11:57:28	NaN	NaN
make_dispatch22	make_dispatch	2016-10-21 11:57:28	NaN	supplier_order_line24
receive_total_order1034	receive_total_order	2016-10-21 11:58:00	NaN	NaN
receive_total_order1034	receive_total_order	2016-10-21 11:58:00	supplier_order16	NaN
create_invoice18	create_invoice	2016-10-21 11:58:36	NaN	NaN
create_invoice18	create_invoice	2016-10-21 11:58:36	NaN	NaN

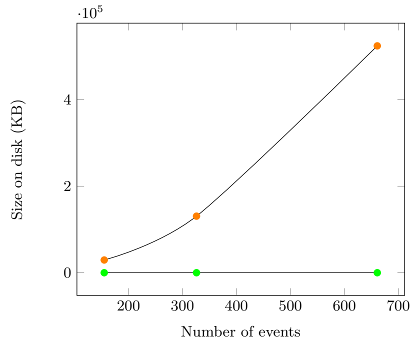
Fig. 9. Representation of a database log in the Parquet columnar format. This is different from classic CSV event logs since the same event is repeated in multiple rows, one for each related object. Both information related to events and related objects are columns of a table. This structure reflects the way information is stored inside the in-memory Pandas dataframe that supports the filtering and the discovery operations.

- *logOpportunities.parquet* is the database log used in the assessment and extracted from the Dynamics CRM system.
- *metamodel.slexmm* provides the OpenSLEX metamodel of the Concert database.
- *erp.xoc* provides an example XOC log extracted from a Dollibar ERP system.

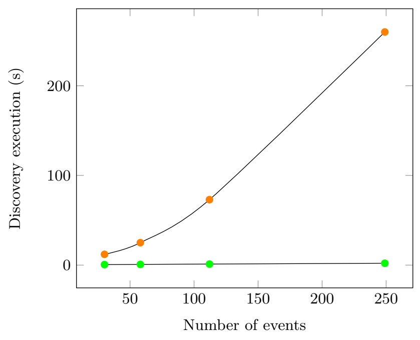
The storage used in the tool for the database event logs is represented in Fig. 9. Several rows are associated to an event, and contain an ID, an activity, a timestamp, and a single object identifier in the column corresponding to its class. This permits to store events in a tabular format using basic types for the columns (strings, integers, dates) to maximize the query performance.

7 Assessment

The assessment of MVP models will show how they perform against two competing approaches (OCBC models [20] and OpenSLEX [29]), both considering the execution time and the simplicity/readability/amount of information contained in the models. These approaches have been chosen because are very recent. The scalability of the MVP models implementation contained in PM4Py will be analyzed on some synthetic logs. Moreover, an assessment using a real information system (Microsoft Dynamics CRM) will be shown.



a) Difference in storage required for storing MVP logs (the line with green dots) and XOC logs (the line with orange dots) for different data sets.



b) Difference in discovery speed between MVP models (the line with green dots) and OCBC models (the line with orange dots) for different data sets.

Fig. 10. Comparison between MVP models and OCBC models, in (a) size on disk of the proposed log storage b) discovery speed. The XOC format that is tested is the one described in [1].

7.1 Comparison with Related Approaches

OCBC models are powerful descriptions of the relationships between activities and classes at the database level. Having said that, OCBC models have scalability issues: XOC logs as proposed in [1] store a snapshot of the object model per event, this becomes quickly unfeasible also for few hundred database events (as it will shown in the assessment). In [9], an updated version of XOC, storing in each event the updates to the object model, has been proposed, although a XOC event log in such format is not public available. Moreover, the final visualization (even if it can be filtered on some types of constraints) lacks understandability. A serious issue is the lack of support for frequency/performance decoration (i.e. the number of occurrences of the arc, the time passed between activity).

OpenSLEX provides a way to ingest a database event log into a meta-model instance that is easier to query. An issue is that they do not offer any visual clue on underlying relationships between activities, making the life difficult to the user in first instance. Moreover, although the queries on OpenSLEX are easier than the queries done directly on the database, there is an effort required by the user to understand the concepts described in [29].

7.2 Evaluation of the Execution Time

OCBC models (tests have been performed on the original version of XOC, proposed in [1], for which several logs are public available) have scalability problems with regards to the log format (XOC), that requires the storage, for each event, of the status of the entire object model. In Fig. 10 (a), the size on disk (in KB)

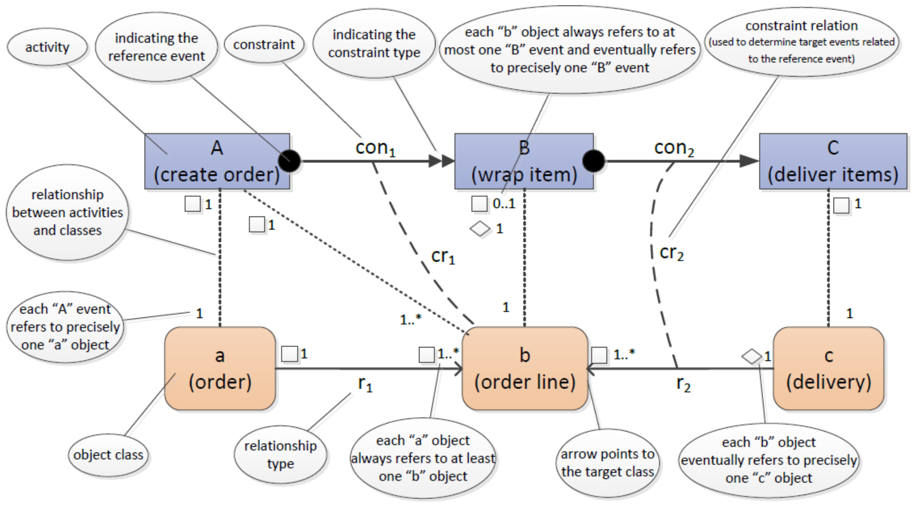

activity
indicating the reference event
constraint
indicating the constraint type
each "b" object always refers to at most one "B" event and eventually refers to precisely one "B" event
constraint relation (used to determine target events related to the reference event)
A (create order)
con1
B (wrap item)
con2
C (deliver items)
relationship between activities and classes
cr1
cr2
each "A" event refers to precisely one "a" object
a (order)
b (order line)
c (delivery)
r1
r2
object class
relationship type
each "a" object always refers to at least one "b" object
arrow points to the target class
each "b" object eventually refers to precisely one "c" object

Fig. 11. Usability assessment: example model illustrating the main ingredients of OCBC models.

of a log containing the specified number of events has been considered. To obtain an OCBC model, for a log with just 661 events, a XOC log of 512 MB is required, that is 17476 greater than the amount of disk space that permits to discover an MVP model from the same database.

In Fig. 10 (b), the execution speed of the discovery procedure has been compared between MVP models and OCBC models. To obtain an OCBC model, for a log with just 249 events, a time of 4 minutes and 20 seconds is required, while for discovering an MVP model from the same database 2 seconds are required, that is 127 times faster. This result holds also for the new version of the XOC format [9], since in the later version the object model is built in-memory by the discovery algorithm starting from the updates described in the event log.

With OpenSLEX, the comparison is more tight, although the final goal is different: OpenSLEX require a query to get a classical event log, while MVP models do not require this effort. There is only a meta-model instance available in public, that has been extracted from a synthetic database on concert management. The amount of storage required to store the OpenSLEX instance is 11 MB, while the amount of storage that permits to discover an MVP model is 222 KB (storing using Parquet format). To compare the (time) performance of OpenSLEX and MVP models, a query on the OpenSLEX instance needs to be performed. Starting in both cases from the concert database instance of the OpenSLEX meta-model, if an example query[2] is chosen, the execution speed of the query on the OpenSLEX instance is 1.73 seconds. The time needed for MVP

[2] The example query is provided in the technical report available at the address http://www.alessandroberti.it/technical.pdf

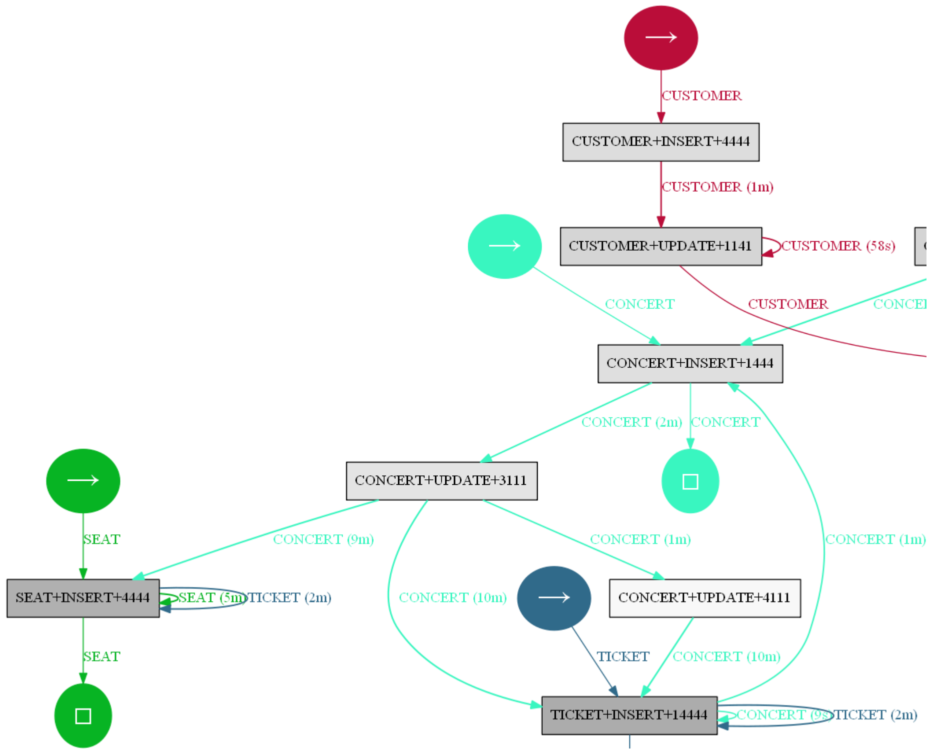


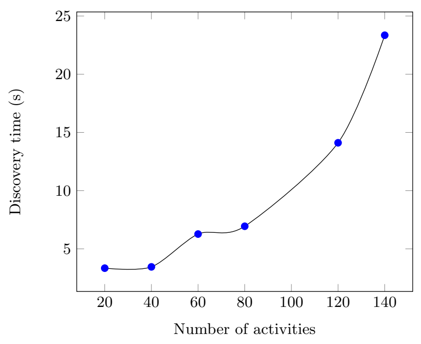
Fig. 12. Usability assessment: performance/bottleneck visualization provided by MVP models.

models to obtain a complete model with frequency and performance information is 1.96 seconds, this without requiring the specification of any query by the user.

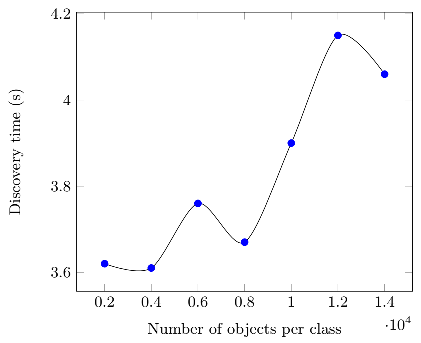
7.3 Usability of the Approaches

OCBC and MVP models both provide ways to discover a model on top of database event logs. The amount of information and constraints extracted by OCBC models is very high, and although filtering is provided to keep only some constraints, the resulting process model is complex to understand. An explanation of some ingredients of OCBC models is provided by [20] and represented in Fig. 11. This amount of information is insane, but it is very difficult also for a process analyst to be able to understand it without proper training.

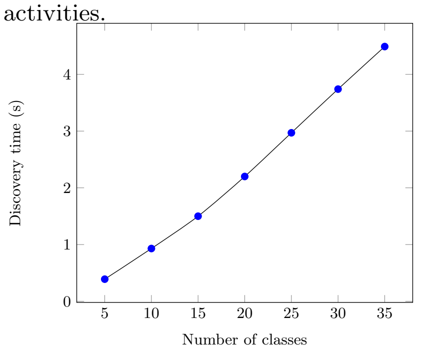
Moreover, this class of models does not provide frequency/performance information, that can be useful for the process analyst in order to detect the bottlenecks. MVP models, as represented in Fig. 12, can provide a graph in which edges are decorated by frequency/performance information. With MVP models, the following features are also available, that are not available on OCBC models:



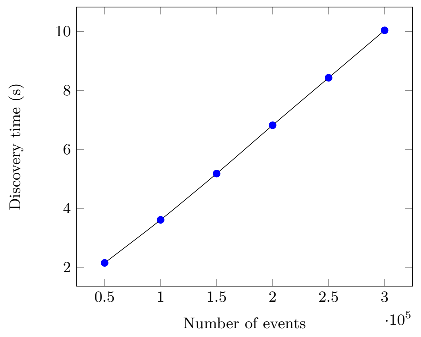
a) Performance of MVP discovery with the increase of the number of activities, when the number of classes, the number of objects per class and the number of events in the log is kept fixed: n_events = 100000, n_classes = 10 and n_objects_per_class = 1000. The logs have been obtained through the generator included in PM4Py. The execution time (in seconds) grows quadratically with the number of activities.



b) Performance of MVP discovery with the increase of the number of objects per class, when the number of classes, the number of activities and the number of events in the log is kept fixed: n_events = 100000, n_classes = 10 and n_activities = 40. The logs have been obtained through the generator included in PM4Py.



c) Performance of MVP discovery with the increase of the number of classes, when the number of objects per class, the number of activities and the number of events in the log is kept fixed: n_events = 10000, n_activities = 40 and n_objects_per_class = 6000. The logs have been obtained through the generator included in PM4Py. The execution time (in seconds) grows linearly with the number of classes.



d) Performance of MVP discovery with the increase of the number of events in the log, when the number of classes, the number of objects per class and the number of activities in the log is kept fixed: n_activities = 40, n_classes = 10 and n_objects_per_class = 6000. The logs have been obtained through the generator included in PM4Py. The execution time (in seconds) grows linearly with the number of events.

Fig. 13. Scalability assessment of MVP models, with regards to a) the number of activities b) the number of objects per class c) the number of classes d) the number of events in the log.

– Projection to a classic directly-follows graph to be used with techniques like Inductive Miner Directly-Follows and the Heuristics Miner.
– Projection to a classical event log to be used with mainstream process mining techniques.

OpenSLEX do not provide a visualization of a process model on top of the database, but require to the user the specification of a query to retrieve an event log. This is an unavoidable step, and requires time and expertise by the user. So, the retrieval of a process model using MVP requires less time and less knowledge than the retrieval from OpenSLEX.

7.4 Scalability of the Approach

In the previous section, MVP models have shown greater scalability and usability in comparison to some competing approaches. In this section, the goal is to understand more clearly the performance of the current implementation.

An assessment of the approach on simulated logs (through the log generator included in PM4Py) has been done (see Fig. 13) to see which variables influence the execution time in a quadratic way, and which variables influence it in a linear way:

– a) assesses the performance of MVP discovery with the increase of the number of activities, when the number of classes, the number of objects per class and the number of events in the log is kept fixed. The execution time (in seconds) grows quadratically with the number of activities.
– b) assesses the performance of MVP discovery with the increase of the number of objects per class, when the number of classes, the number of activities and the number of events in the log is kept fixed. Although the behavior in the figure looks erratic, the execution time (in seconds) grows asymptotically linearly with the number of objects per class.
– c) assesses the performance of MVP discovery with the increase of the number of classes, when the number of objects per class, the number of activities and the number of events in the log is kept fixed. The execution time (in seconds) grows linearly with the number of classes.
– d) assesses the performance of MVP discovery with the increase of the number of events in the log, when the number of classes, the number of objects per class and the number of activities in the log is kept fixed. The execution time (in seconds) grows linearly with the number of events.

7.5 Evaluation using CRM Data

This section presents a study of data extracted from a Microsoft Dynamics CRM demo and analyzed using MVP discovery. A Customer Relationship Management system (CRM) [7] is an information system used to manage the commercial lifecycle of an organization, including management of customers, opportunities and marketing campaigns.

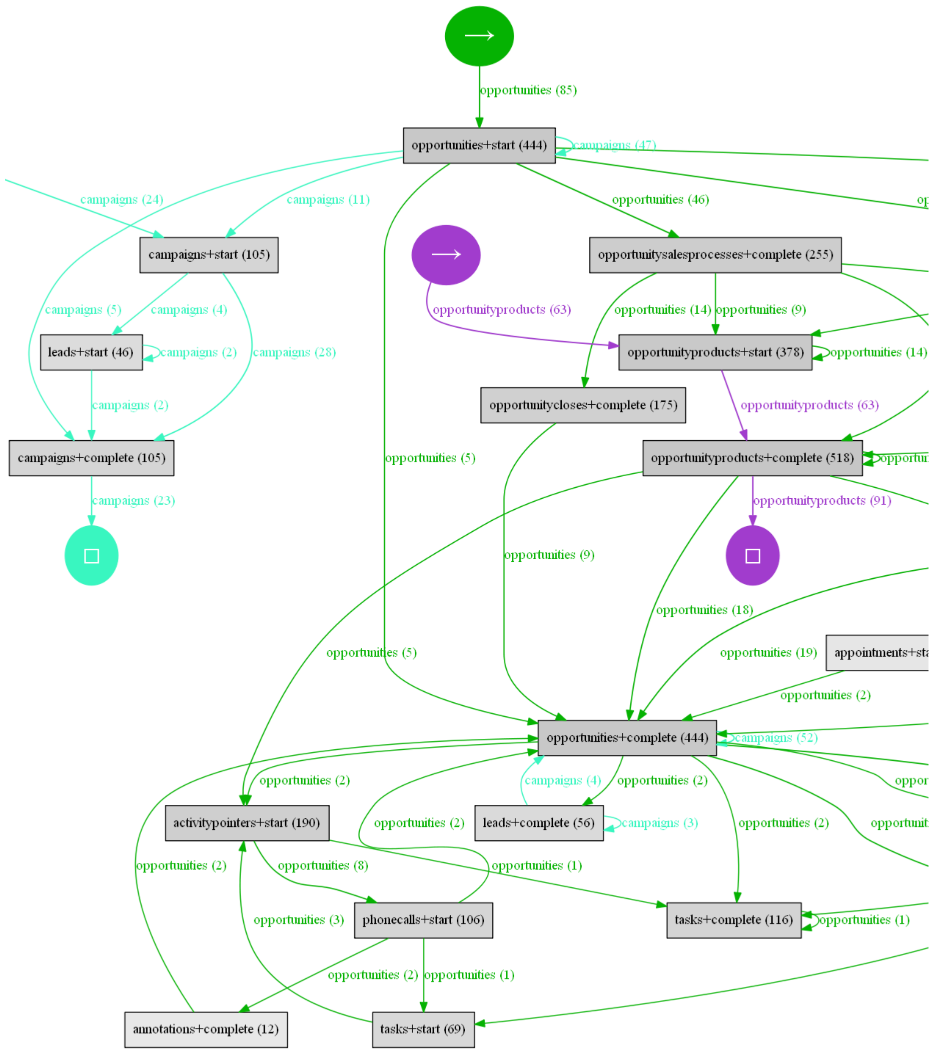


Fig. 14. Representation in one model of four perspectives (opportunities, opportunityproducts, campaigns, connections) of the Dynamics CRM database (only part of the diagram is reported).

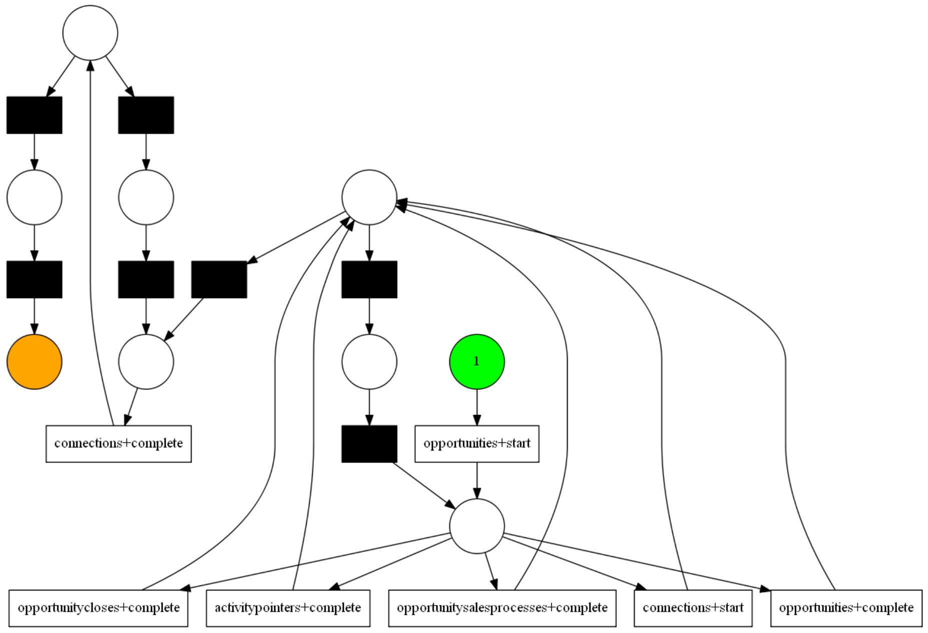

1
connections+complete
opportunities+start
opportunitycloses+complete
activitypointers+complete
opportunitysalesprocesses+complete
connections+start
opportunities+complete

Fig. 15. Petri net obtained by applying Inductive Miner Directly-Follows on the projection of the Dynamics CRM MVP model on a viewpoint containing only the 'opportunities' class. Black boxes represent invisible transitions (transitions that can be executed without correspondence with the activities of the log).

Many companies actually involve a CRM system for helping business and sales people to coordinate, share information and goals. Data extracted from Microsoft Dynamics CRM is particularly interesting since this product manages several processes of the business side, providing the possibility to define workflows and to measure KPI also through connection to the Microsoft Power BI business intelligence tool. For evaluation purposes, a database log has been generated containing data extracted from a Dynamics CRM demo. The database supporting the system contains several entities, and each entity contains several entries related to activities happening in the CRM. Each entry could be described by a unique identifier (UUID), the timestamp of creation/change, the resource that created/modified the entry, and some UUIDs of other entries belonging to the same or to different entities. Moreover, each entry is uniquely associated with the entity it belongs to.

The following strategy has been pursued in order to generate a log:

- For each entry belonging to an entity, two events have been associated: creation event (with the timestamp of creation and lifecycle *start*) and modify event (with the timestamp of modification and lifecycle *complete*).
- Each entry belonging to an entity has also been associated with an object.

– Relationships between events and objects are created accordingly to the relationships expressed by the entries (an entry may cite several UUIDs of other entries stored in the database).

The previous construction means that for the same entry there are two events (start+complete) and one object in the log.

The database log contains 5863 events, 4413 objects, 120 activities and 80 object classes, and could be stored in a 386 KB Parquet file. The complete MVP model (525 edges) can be calculated and represented in 5 seconds. Taking a viewpoint containing only classes related to opportunities management (e.g. *opportunities*, *opportunityproducts*, *campaigns*, *connections*), and choosing the frequency metric, the model obtained is represented in Fig. 14. Projecting the MVP model on a viewpoint containing only the 'opportunities' class and applying Inductive Miner Directly-Follows, the process model represented in Fig. 15 is obtained.

Being able to handle this complex database schema and visualize a model that unifies the different classes, in a very reasonable time, is a thing that is impossible with competing approaches like OCBC models (due to severe scalability issues) and OpenSLEX instances (because the resulting query would be more complex than selecting a viewpoint from an holistic model).

8 Conclusion

This paper introduces Multiple Viewpoint models (MVP), providing an holistic view on a process supported by a database. MVP models are annotated with frequency and performance measures (e.g., delays), supporting the detection of the most frequent paths and of the bottlenecks without the specification of any case notion. At the same time, a viewpoint (a non-empty subset of classes) can be chosen on the MVP models in order to get classical objects (DFGs and logs) to use with the mainstream process mining techniques. Hence, the holistic view of the MVP model is complemented by detailed viewpoint models using conventional notations like Petri nets, BPMN models and process trees. This possibility is not provided by the competing techniques described in Section 2: the focus is either on the specification of a case notion, or on the retrieval of an artifact-centric model. However, the relationships between the classes are not calculated and represented in the process model, in contrast to the technique of OCBC models. Moreover, the lack of a clear execution semantic of MVP models makes the application of conformance checking techniques possible only after choosing a viewpoint, in contrast with the OCBC technique that provides a (theoretically) powerful conformance checking approach on top of the model.

An MVP model has been discovered from a database event log extracted from a Microsoft Dynamics Customer Relationship Management (CRM) system, showing the possibility to apply the techniques described in this paper to real-life information systems.

Moreover, a comparison considering execution time and usability has been performed against two of the most recent process mining approaches on databases

(OpenSLEX and OCBC models); MVP models are relatively well performing and easy to use, since an holistic view could be obtained without any effort from the user, and any viewpoint could be chosen on the MVP model.

Scalability testing proved that MVP models scale linearly with the number of events, classes and asymptotically linearly with the number of objects contained in the database event log, while they scale quadratically with the number of activities (due to the edges calculation in the A2A multigraph). This is a remarkable result in comparison to OCBC models that show an exponential complexity on the number of events (they become unmanageable also for a small number of events).

The techniques described in this paper could in principle be implemented starting from the logs of any relational database. Hence MVP discovery supports process mining analysis directly from real-life complex information systems.

References

1. van der Aalst, W., Li, G., Montali, M.: Object-centric behavioral constraints. arXiv preprint arXiv:1703.05740 (2017)
2. Bayomie, D., Helal, I.M., Awad, A., Ezat, E., ElBastawissi, A.: Deducing case ids for unlabeled event logs. In: International Conference on Business Process Management. pp. 242–254. Springer (2016)
3. Berti, A., van Zelst, S.J., van der Aalst, W.: Process Mining for Python (PM4Py): Bridging the Gap Between Process-and Data Science p. 1316 (2019)
4. Bhattacharya, K., Gerede, C., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: International Conference on Business Process Management. pp. 288–304. Springer (2007)
5. Bouchou, B., Niang, C.: Semantic mediator querying. In: Proceedings of the 18th International Database Engineering & Applications Symposium. pp. 29–38. ACM (2014)
6. Burattin, A., Vigo, R.: A framework for semi-automated process instance discovery from decorative attributes. In: 2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM). pp. 176–183. IEEE (2011)
7. Buttle, F.: Customer relationship management. Routledge (2004)
8. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: Answering sparql queries over relational databases. Semantic Web **8**(3), 471–487 (2017)
9. de Carvalho, R.M., van der Aalst, W.M.: Extracting object-centric event logs to support process mining on databases. In: Information Systems in the Big Data Era: CAiSE Forum 2018, Tallinn, Estonia, June 11-15, 2018, Proceedings. vol. 317, p. 182. Springer (2018)
10. Chen, P.P.S.: The entity-relationship model toward a unified view of data. ACM Transactions on Database Systems (TODS) **1**(1), 9–36 (1976)
11. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull. **32**(3), 3–9 (2009)
12. van Eck, M.L., Sidorova, N., van der Aalst, W.: Composite state machine miner: Discovering and exploring multi-perspective processes. In: BPM (Demos). pp. 73–77 (2016)

13. van Eck, M.L., Sidorova, N., van der Aalst, W.: Discovering and exploring state-based models for multi-perspective processes. In: International Conference on Business Process Management. pp. 142–157. Springer (2016)
14. Fahland, D., De Leoni, M., van Dongen, B.F., van Der Aalst, W.: Behavioral conformance of artifact-centric process models. In: International Conference on Business Information Systems. pp. 37–49. Springer (2011)
15. Helal, I.M., Awad, A., El Bastawissi, A.: Runtime deduction of case id for unlabeled business process execution events. In: 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA). pp. 1–8. IEEE (2015)
16. Hull, R., Damaggio, E., De Masellis, R., Fournier, F., Gupta, M., Heath III, F.T., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., et al.: Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In: Proceedings of the 5th ACM international conference on distributed event-based system. pp. 51–62. ACM (2011)
17. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, F.T., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., et al.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: International Workshop on Web Services and Formal Methods. pp. 1–24. Springer (2010)
18. Ingvaldsen, J.E., Gulla, J.A.: Preprocessing support for large scale process mining of SAP transactions. In: International Conference on Business process management. pp. 30–41. Springer (2007)
19. Knuplesch, D., Ly, L.T., Rinderle-Ma, S., Pfeifer, H., Dadam, P.: On enabling data-aware compliance checking of business process models. In: International Conference on Conceptual Modeling. pp. 332–346. Springer (2010)
20. Li, G., de Carvalho, R.M., van der Aalst, W.: Automatic discovery of object-centric behavioral constraint models. In: International Conference on Business Information Systems. pp. 43–58. Springer (2017)
21. Liu, D., Pedrinaci, C., Domingue, J.: Semantic enabled complex event language for business process monitoring. In: Proceedings of the 4th International Workshop on Semantic Business Process Management. pp. 31–34. ACM (2009)
22. Lohmann, N.: Compliance by design for artifact-centric business processes. In: International Conference on Business Process Management. pp. 99–115. Springer (2011)
23. Lu, X., Nagelkerke, M., van de Wiel, D., Fahland, D.: Discovering interacting artifacts from ERP systems (extended version). BPM reports **1508** (2015)
24. Ly, L.T., Rinderle-Ma, S., Knuplesch, D., Dadam, P.: Monitoring business process compliance using compliance rule graphs. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". pp. 82–99. Springer (2011)
25. Metzke, T., Rogge-Solti, A., Baumgrass, A., Mendling, J., Weske, M.: Enabling semantic complex event processing in the domain of logistics. In: International Conference on Service-Oriented Computing. pp. 419–431. Springer (2013)
26. Momotko, M., Subieta, K.: Process query language: A way to make workflow processes more flexible. In: East European Conference on Advances in Databases and Information Systems. pp. 306–321. Springer (2004)
27. de Murillas, E.G.L.: Process mining on databases: extracting event data from real life data sources (2019)
28. de Murillas, E.G.L., Hoogendoorn, G., Reijers, H.A.: Redo log process mining in real life: Data challenges & opportunities. In: International Conference on Business Process Management. pp. 573–587. Springer (2017)

29. de Murillas, E.G.L., Reijers, H.A., van der Aalst, W.: Connecting databases with process mining: a meta model and toolset. Software & Systems Modeling pp. 1–39 (2018)
30. Narendra, N.C., Badr, Y., Thiran, P., Maamar, Z.: Towards a unified approach for business process modeling using context-based artifacts and web services. In: 2009 IEEE International Conference on Services Computing. pp. 332–339. IEEE (2009)
31. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal **42**(3), 428–445 (2003)
32. Nooijen, E.H., van Dongen, B.F., Fahland, D.: Automatic discovery of data-centric and artifact-centric processes. In: International Conference on Business Process Management. pp. 316–327. Springer (2012)
33. Song, L., Wang, J., Wen, L., Wang, W., Tan, S., Kong, H.: Querying process models based on the temporal relations between tasks. In: 2011 IEEE 15th International Enterprise Distributed Object Computing Conference Workshops. pp. 213–222. IEEE (2011)
34. Steinau, S., Andrews, K., Reichert, M.: The relational process structure. In: International Conference on Advanced Information Systems Engineering. pp. 53–67. Springer (2018)
35. Tang, Y., Mackey, I., Su, J.: Querying workflow logs. Information **9**(2), 25 (2018)
36. Verbeek, H., Buijs, J.C., Van Dongen, B.F., Van Der Aalst, W.M.: Xes, xesame, and prom 6. In: International Conference on Advanced Information Systems Engineering. pp. 60–75. Springer (2010)
37. Yano, K., Nomura, Y., Kanai, T.: A practical approach to automated business process discovery. In: 2013 17th IEEE International Enterprise Distributed Object Computing Conference Workshops. pp. 53–62. IEEE (2013)