Wil M. P. van der Aalst
Josep Carmona (Eds.)

Tutorial

LNBIP 448

Process Mining Handbook

Springer

Lecture Notes in Business Information Processing 448

More information about this series at https://link.springer.com/bookseries/7911

Wil M. P. van der Aalst · Josep Carmona (Eds.)

Process Mining Handbook

Springer

Editors

Wil M. P. van der Aalst
RWTH Aachen
Aachen, Germany

Josep Carmona
Universitat Politècnica de Catalunya
Barcelona, Spain

ISSN 1865-1348 ISSN 1865-1356 (electronic)
Lecture Notes in Business Information Processing
ISBN 978-3-031-08847-6 ISBN 978-3-031-08848-3 (eBook)
https://doi.org/10.1007/978-3-031-08848-3

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Process mining emerged as a new discipline around the turn of the century. The combination of event data and process models poses interesting scientific problems. Initially, the focus was on the discovery of process models (e.g., Petri nets) from example traces. However, over time the scope of process mining broadened in several directions. Next to process discovery, topics such as conformance checking and performance analysis were added. Different perspectives were added (e.g., time, resources, roles, costs, and case types) to move beyond control-flow models. Along with directly-follows graph (DFGs) and Petri nets, a wide range of process model notations has been explored in the context of event data. Examples include declarative process models, process trees, artifact-centric and object-centric process models, UML activity models, and BPMN models. In recent years, the focus also shifted from backward-looking to forward-looking, connecting process mining to neighboring disciplines such as simulation, machine learning, and automation.

Over the past two decades, the discipline did not only expand in terms of scope but also in terms of adoption and tool support. The first commercial process mining tools emerged 15 years ago (Futura Process Intelligence, Disco, etc.). Now there are over 40 commercial products next to open-source process mining tools such as ProM, PM4Py, and bupaR. The adoption in industry has accelerated in the last five years. In several regions of the world, most of the larger companies are already using process mining, and the process mining market is expected to double every 18 months in the coming years.

Given the amazing developments in the last two decades, a comprehensive process mining summer school is long overdue. This book contains the core material of the first Summer School on Process Mining organized by the IEEE Task Force on Process Mining. The Task Force on Process Mining was established in October 2009 as part of the IEEE Computational Intelligence Society. Its activities led to the International Process Mining Conference (ICPM) series, a range of successful workshops (BPI, ATAED, PODS4H, etc.), the Process Mining Manifesto (translated into 15+ languages), the XES standard, publicly available datasets, online courses, and case studies. However, a dedicated summer school on process mining was missing. Therefore, we started the preparations for this in 2020. Due to the COVID-19 pandemic, this was delayed by one year, but this gave us more time to carefully prepare this handbook on process mining.

The summer school took place in Aachen, Germany, during July 4–8, 2022. The location of the summer school was the scenic SuperC building with nice views of the city center and close to the cathedral of Aachen, which was the first UNESCO World Heritage site in Germany.

The local organization was undertaken by the Process and Data Science (PADS) group at RWTH Aachen University. The event was financially supported by Wil M. P. van der Aalst's Alexander von Humboldt (AvH) professorship. The event was also supported by the RWTH Center for Artificial Intelligence, the Center of Excellence Internet of Production (IoP), Celonis, and Springer.

The book starts with a 360-degree overview of the field of process mining (Chapter 1). This first chapter introduces the basic concepts, the different types of process mining, process modeling notations, and storage formats for events.

Chapter 2 presents the foundations of process discovery. It starts with discovering directly-follows graphs from simple event logs and highlighting the challenges. Then basic bottom-up and top-down process discovery techniques are presented that produce Petri nets and BPMN models.

Chapter 3 presents four additional process discovery techniques: an approach based on state-based regions, an approach based on language-based regions, the split mining approach, and the log skeleton-based approach.

Techniques to discover declarative process models are presented in Chapter 4. The chapter focuses on discovering declarative specifications from event logs, monitoring declarative specifications against running process executions to promptly detect violations, and reasoning on declarative process specifications.

Chapter 5 presents techniques for conformance checking. An overview of the applications of conformance checking and a general framework are presented. The goal is to compare modeled and observed behavior.

Chapter 6 discusses event data in more detail, also describing the data-preprocessing pipeline, standards like XES, and data quality problems.

Chapter 7 takes a more applied view and discusses how process mining is used in different industries and the efforts involved in creating an event log. The chapter also lists best practices, illustrated using the order-to-cash (O2C) process in an SAP system.

Chapter 8 introduces a number of techniques for process enhancement, including process extension and process improvement. For example, it is shown how to add additional perspectives to a process model.

Chapter 9 introduces event knowledge graphs as a means to model multiple entities distributed over different perspectives. It is shown how to construct, query, and aggregate event knowledge graphs to get insights into complex behaviors.

Predictive process monitoring techniques are introduced in Chapter 10. This is the branch of process mining that aims at predicting the future of ongoing (uncompleted) process executions.

Streaming process mining refers to the set of techniques and tools which have the goal of processing a stream of data (as opposed to a fixed event log). Chapter 11 presents such techniques.

The topic of responsible process mining is addressed in Chapter 12. The chapter summarizes and discusses current approaches that aim to make process mining responsible by design, using the well-known FACT criteria (Fairness, Accuracy, Confidentiality, and Transparency).

Chapter 13 discusses the evolution of the field of process mining, i.e., the transition from process discovery to process execution management. The focus is on driving business value.

Chapter 14 makes the case that healthcare is a very promising application domain for process mining with a great societal value. An overview of healthcare processes and healthcare process data is given, followed by a discussion of common use cases.

Chapter 15 shows that process mining is a valuable tool for financial auditing. Both internal and external audits are introduced, along with the connection between the two audits and the application of process mining.

Chapter 16 introduces a family of techniques, called robotic process mining, that discover repetitive routines that can be automated using robotic process automation (RPA) technology.

Chapter 17 concludes the book with an analysis of the current state of the process mining discipline and outlook on future developments and challenges. Pointers to the lecture material will be made available via www.process-mining-summer-school.org, www.processmining.org, and www.tf-pm.org. These complement this book.

Finally, we thank all the participants, authors, speakers, and the organizations supporting this once-in-a-lifetime event. In particular, we thank the Alexander von Humboldt Foundation. Enjoy reading!

April 2022

Wil M. P. van der Aalst
Josep Carmona

Contents

Introduction

Process Discovery

Conformance Checking

Data Preprocessing

Process Enhancement and Monitoring

Assorted Process Mining Topics

Industrial Perspective and Applications

Closing

Introduction

Process Mining: A 360 Degree Overview

Wil M. P. van der Aalst(✉)

Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany
wvdaalst@pads.rwth-aachen.de
http://www.vdaalst.com/

Abstract. Process mining enables organizations to uncover their actual processes, provide insights, diagnose problems, and automatically trigger corrective actions. Process mining is an emerging scientific discipline positioned at the intersection between process science and data science. The combination of process modeling and analysis with the event data present in today's information systems provides new means to tackle compliance and performance problems. This chapter provides an overview of the field of process mining introducing the different types of process mining (e.g., process discovery and conformance checking) and the basic ingredients, i.e., process models and event data. To prepare for later chapters, event logs are introduced in detail (including pointers to standards for event data such as XES and OCEL). Moreover, a brief overview of process mining applications and software is given.

Keywords: Process mining · Event data · Process modeling · Process discovery

1 Introduction

Process mining can be defined as follows: *process mining aims to improve operational processes through the systematic use of event data* [1,2]. By using a combination of event data and process models, process mining techniques provide insights, identify bottlenecks and deviations, anticipate and diagnose performance and compliance problems, and support the automation or removal of repetitive work. Process mining techniques can be *backward-looking* (e.g., finding the root causes of a bottleneck in a production process) or *forward-looking* (e.g., predicting the remaining processing time of a running case or providing recommendations to lower the failure rate). Both backward-looking and forward-looking analyses can trigger *actions* (e.g., countermeasures to address a performance or compliance problem). The focus of process mining is on *operational processes*, i.e., processes requiring the repeated execution of activities to deliver products or services. These can be found in all organizations and industries, including production, logistics, finance, sales, procurement, education, consulting, healthcare, maintenance, and government. This chapter provides a 360° overview of process mining, introducing basic concepts and positioning process mining with respect to other technologies.

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 3–34, 2022.
https://doi.org/10.1007/978-3-031-08848-3_1

The idea of using detailed data about operational processes is not new. For example, Frederick Winslow Taylor (1856–1915) collected data on specific tasks to improve labor productivity [35]. With the increasing availability of computers, spreadsheets and other business intelligence tools were used to monitor and analyze operational processes. However, in most cases, the focus was on a single task in the process, or behavior was reduced to aggregated Key Performance Indicators (KPIs) such as flow time, utilization, and costs. Process mining aims to analyze *end-to-end processes* at the level of *events*, i.e., detailed behavior is considered in order to explain and improve performance and compliance problems.

Process mining research started in the late 1990s [23]. In 2004 the first version of the open-source platform ProM was released with 29 plug-ins. Over time the ProM platform was extended and now includes over 1500 plug-ins. The first commercial process mining tools appeared around 15 years ago. Today, there are over 40 commercial process mining tools and process mining is used by thousands of organizations all over the globe. However, only a small fraction of its potential has been realized. Process mining is generic and can be applied in any organization.

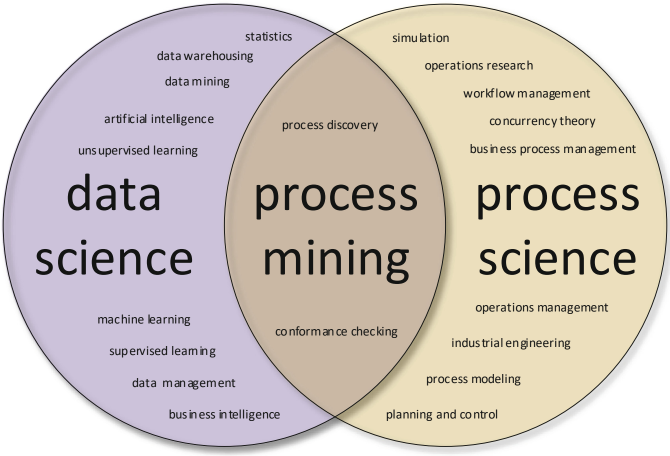


Fig. 1. Process mining = data science ∩ process science.

Figure 1 shows that process mining can be seen as the intersection of data science and process science. In [2], the following definition is proposed: “Data science is an interdisciplinary field aiming to turn data into real value. Data may be structured or unstructured, big or small, static or streaming. Value may be provided in the form of predictions, automated decisions, models learned from data, or any type of data visualization delivering insights. Data science includes data extraction, data preparation, data exploration, data transformation, storage and retrieval, computing infrastructures, various types of mining and learning, presentation of explanations and predictions, and the exploitation of results taking into account ethical, social, legal, and business aspects." In [2], process science is used as an umbrella term to refer to the broader discipline that combines knowledge from information technology and knowledge from management sciences to improve and run operational processes. In the more recent [12], the following definition is proposed: "Process science is the interdisciplinary study of continuous change. By process, we mean a coherent series of changes that unfold over time and occur at multiple levels." In [12], we emphasize the following key characteristics of process science: (1) processes are in focus, (2) processes are investigated using scientific methods, (3) an interdisciplinary lens is used, and (4) the goal of process science is to influence and change processes to realize measurable improvements. As stated in [2] and visualized in Fig. 1; process mining can be viewed as the link between data science and process science. Process mining seeks the confrontation between event data (i.e., observed behavior) and process models (hand-made models or automatically discovered models), and aims to exploit event data in a meaningful way, for example, to provide insights, identify bottlenecks, anticipate problems, record policy violations, recommend countermeasures, and streamline processes.

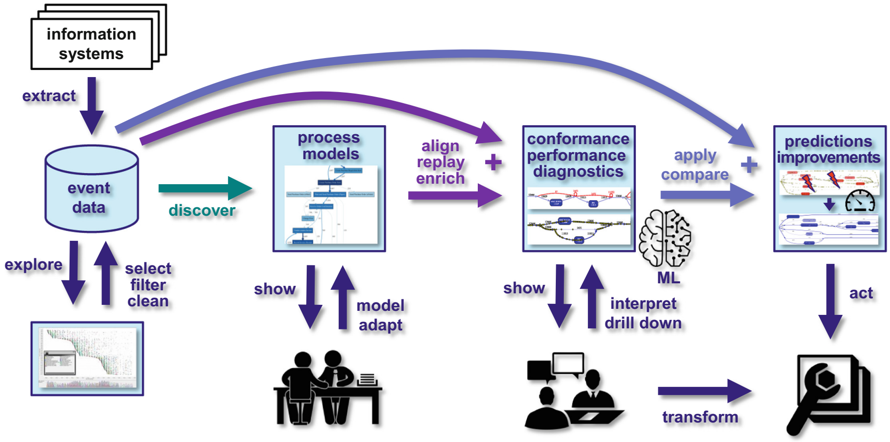


Fig. 2. 360° overview of process mining.

Figure 2 shows a high-level view of process mining. *Event data* need to be extracted from *information systems* used to support the processes that need to be analyzed. Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), and Supply Chain Management (SCM) systems store events. Examples are SAP S/4HANA, Oracle E-Business Suite, Microsoft Dynamics 365, and Salesforce CRM. Next to these sector-agnostic software systems, there are more specialized systems such as Health Information Systems (HIS). All of these systems have in common that they are loaded with event data. However, these are scattered over many database tables and need to be converted into a format that can be used for process mining. As a consequence, data extraction is an integral part of any process mining effort, and may be time-consuming. Events are often represented by a case identifier, an activity name, a timestamp, and optional attributes such as resource, location, cost, etc. Object-centric event data allow events to point to any number of objects rather than a single case (see Sect. 3).

Once extracted, event data can be *explored*, *selected*, *filtered*, and *cleaned* (see Fig. 2). Data visualization techniques such as dotted charts and sequence diagrams can be used to understand the data. Often, the data need to be scoped to the process of interest. One can use generic query languages like SQL, SPARQL, and XQuery or a dedicated Process Query Language (PQL). Data may be incomplete, duplicated, or inconsistent. For example, month and day may be swapped during manual data entry. There is a variety of techniques and approaches to address such *data quality* problems [34].

The resulting dataset is often referred to as an *event log*, i.e., a collection of events corresponding to the selected process. *Process discovery* techniques are used to automatically create process models. Commercial tools typically still resort to learning the so-called *Directly-Follows Graph* (DFG) which typically leads to underfitting process models [3]. If two activities do not occur in a fixed order, then loops are created. This leads to Spaghetti-like diagrams suggesting repetitions that are not supported by the data. However, there are numerous approaches to learning higher-level models represented using Business Process Model and Notation (BPMN), Petri nets, or Unified Modeling Language (UML) activity diagrams. In contrast to DFGs, such models are able to express concurrency. Example techniques to discover such models are the Alpha algorithm [8], region-based approaches [11,13,33,36], inductive mining techniques [28,29], and the split miner [9]. The process model returned may aim to describe all behavior observed or just the dominant behavior. Note that the event log only contains example behavior, is likely to be incomplete, and at the same time may contain infrequent behavior.

The combination of a process model and event data can be used to conduct conformance checking and performance analysis (Fig. 2). The process model may have been discovered or made by hand. Discovered process models are descriptive and hand-crafted models are often normative. *Conformance checking* relates events in the event log to activities in the process model and compares both. The goal is to find commonalities and discrepancies between the modeled behavior and the observed behavior. If the process model is normative, deviations correspond to undesired behavior (e.g., fraud or inefficiencies). If the model was discovered automatically with the goal of showing the dominant behavior, then deviations correspond to exceptional behavior (i.e., outliers). Note that most processes have a Pareto distribution, e.g., 80% of the cases can be described by only 20% of the process variants. It is often easy and desirable to create a process model describing these 80%. However, the remaining 20% cannot be discarded since these cases cover the remaining 80% of the process variants and often also the majority of performance and compliance problems. Sometimes event logs are even more unbalanced, e.g., it is not uncommon to find logs where 95% of the cases can be described by less than 5% of the process variants. In the latter case, it may be that the remaining 5% of cases (covering 95% of the process variants) consume most of the resources due to rework and exception handling.

Since events have timestamps, it is easy to overlay the process model with *performance diagnostics* (service times, waiting times, etc.). After discovering the *control-flow*, the process model can be turned into a *stochastic* model that includes probabilities and delay distributions.

After applying conformance checking and performance analysis techniques, users can see performance and compliance problems. It is possible to perform *root-cause analysis* for such problems. One may find out that critical deviations are often caused by a particular machine or supplier, or that the main bottleneck is caused by poor resource planning or excessive rework for some product types. In a procurement process, price changes by a particular supplier may explain an increase in rework. If "Receive Invoice" often occurs before "Create Purchase Requisition", then this signals a compliance problem in the same process. These are just a few examples. In principle, any process-related problem can be diagnosed as long as event data are available.

The right-hand side of Fig. 2 shows that process mining can be used to (1) transform and improve the process and (2) automatically address observed and predicted problems. The stochastic process models discovered from event data can be used to conduct "what-if" analysis using *simulation* or other techniques from *operations research* (e.g., planning). The combination of event data and process models can be used to generate *Machine Learning* (ML) problems. ML techniques can be used to predict outcomes without being explicitly programmed to do so. The uptake of ML in recent years can be attributed to progress in deep learning, where artificial neural networks having multiple layers progressively extract higher-level features from the raw input. ML techniques cannot be applied directly to event data. However, by replaying event data on discovered process models, it is possible to create a range of supervised learning problems. Examples include:

- What is the remaining processing time of a particular insurance claim?
- Are we able to handle 95% of the cases within one week?
- Is this application going to deviate from the normative process?
- Will this patient be moved to the intensive care unit?
- Will we have enough free beds in the intensive care unit tomorrow?

It is important to note that the right-hand side of Fig. 2 (i.e., extraction, discovery, conformance checking, and performance analysis) cannot be supported using mainstream Artificial Intelligence (AI) and Machine Learning (ML) technologies (e.g., neural networks). One first needs to discover an explicit process model tightly connected to the event data, to pose the right questions. However, process mining can be used to create AI/ML problems. The combination can be used to trigger corrective actions or even complete workflows addressing the problem observed. This way, event data can be turned into actions that actively address performance and compliance problems.

2 Process Models

There are many notations to describe processes, ranging from Directly-Follows Graphs (DFGs) and transition systems, to BPMN and Petri nets. We will use an example to gently introduce these notations. Consider a process involving the following activities: *buy ingredients (bi)*, *create base (cb)*, *add tomato (at)*, *add cheese (ac)*, *add salami (as)*, *bake in oven (bo)*, *eat pizza (ep)*, and *clean kitchen (ck)*. We will call this fictive process the "pizza process" and use this to illustrate the key concepts and notations.

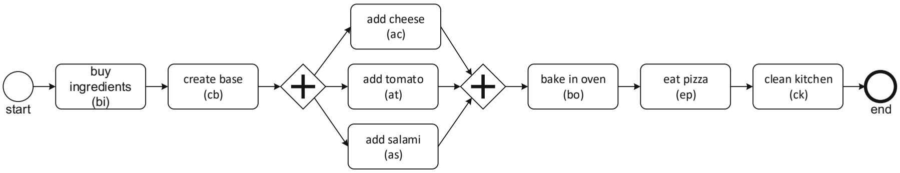


Fig. 3. BPMN model of the "pizza process". The three toppings (tomato, cheese, and salami) can be added in any order.

Figure 3 shows a process model using *Business Process Model and Notation* (BPMN) [17]. The process starts with activity *buy ingredients (bi)* followed by activity *create base (cb)*. Then three activities are executed in any order: *add tomato (at)*, *add cheese (ac)*, and *add salami (as)*. After all three toppings (tomato, cheese, and salami) have been added, the activities *bake in oven (bo)*, *eat pizza (ep)*, and *clean kitchen (ck)* are performed in sequence. Assuming that the three concurrent activities are performed in some order (i.e., interleaved), there are 3! = 6 ways to execute the "pizza process". The two diamond-shaped symbols with a + inside denote *parallel gateways*. The first one is a so-called *AND-split* starting the three concurrent branches and the second one is a so-called *AND-join*. The BPMN process starts with a *start event* (shown as a circle) and ends with an *end event* (shown as a thick circle).

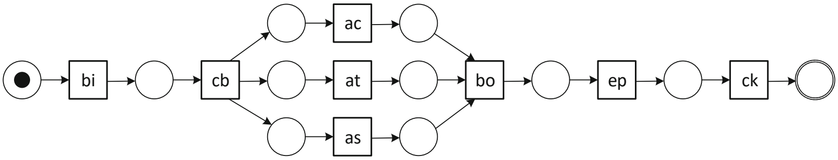


Fig. 4. Petri net modeling the "pizza process" with activities *buy ingredients (bi)*, *create base (cb)*, *add cheese (ac)*, *add tomato (at)*, *add salami (as)*, *bake in oven (bo)*, *eat pizza (ep)*, and *clean kitchen (ck)*.

Figure 4 models the same process in terms of a Petri net. This model also allows for 3! = 6 ways to execute the "pizza process". The circles correspond to *places* (to model states) and the squares correspond to *transitions* (to model activities). Places may hold tokens. A place is called marked if it contains a token. A *marking* is a distribution of tokens over places. In Fig. 4, the source place (i.e., the input place of transition *bi*) is marked, as is indicated by the token (the black dot). A transition is enabled if all input places are marked. In the initial marking shown in Fig. 4, transition *bi* (corresponding to activity *buy ingredients*) is enabled. A transition that is enabled may fire (i.e., it may occur). This means that a token is removed from each of the input places and a token is produced for each of the output places. Note that transition *cb* consumes one token and produces three tokens (one for each output place) and transition *bo* consumes three tokens (one for each input place) and produces one token. The process ends when a token is put on the sink place, i.e., the output place of *ck*. In total there are $2 + 2^3 + 3 = 13$ reachable markings. Although the behavior of the Petri net in Fig. 4 is the same as the BPMN model in Fig. 3, it is easier to refer to the states of the process model.

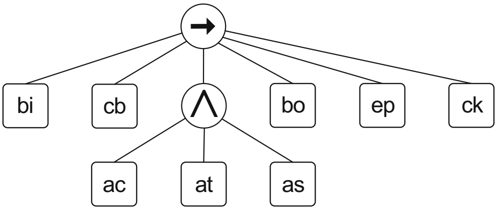


Fig. 5. Process tree of the "pizza process": $\rightarrow(bi, cb, \wedge(ac, at, as), bo, ep, ck)$.

Figure 5 models the "pizza process" using a *process tree*. This representation is rarely presented to end-users, but several mining algorithms use this internally. Process trees are closer to programming constructs, process algebras, and regular expressions. The graphical representation can be converted to a compact textual format: $\rightarrow(bi, cb, \wedge(ac, at, as), bo, ep, ck)$. A sequence operator → executes its children in sequential order. The root node in Fig. 5 denotes such a sequence, i.e., the six child nodes are executed in sequence. The third child node models the parallel execution of its three children. This subtree can be denoted by $\wedge(ac, at, as)$. Later we will see that there are four types of operators that can be used in a process tree: → (sequential composition), × (exclusive choice), ∧ (parallel composition), and ⟲ (redo loop). The semantics of a process tree can be expressed in terms of Petri nets, e.g., Fig. 5 and Fig. 4 represent the same process.

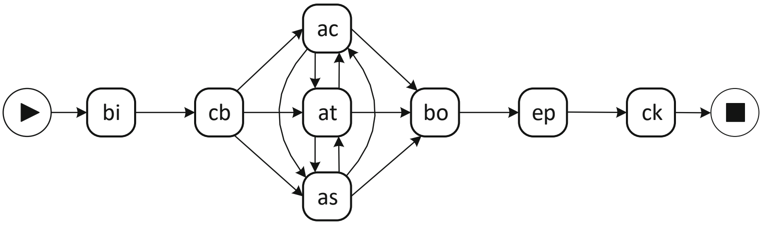


Fig. 6. DFG of the "pizza process". Note that the behavior is different, e.g., one may add 10 toppings to the pizza.

Most of the process mining tools directly show a *Directly-Follows Graph* (DFG) when loading an event log. This helps get a first impression of the behavior recorded. Figure 6 shows a DFG for our running example. There are two special nodes to model start (▶) and end (■). The other nodes represent activities. The arcs in a DFG denote the "directly-follows relation", e.g., the arc connecting *cb* to *at* shows that immediately after creating the pizza base *cb* one can add tomato paste *at*. Activity *cb* has three outgoing arcs denoting a choice, i.e., *cb* is directly followed by *at*, *ac*, or *as*. Activity *at* also has three outgoing arcs denoting that one can add another topping (*ac* or *as*) or bake the pizza (*bo*). Note that the behavior of the DFG in Fig. 6 is different from the three models shown before (i.e., the BPMN model, the Petri net, and the process tree). The DFG allows for infinitely many ways to execute the "pizza process" (instead of 3! = 6). For example, it is possible to create a pizza where each of the toppings was added 10 times. The problem is that whenever two activities can occur in any order (e.g., *at* and *ac*), there is immediately a loop in the DFG (even when both happen only once).

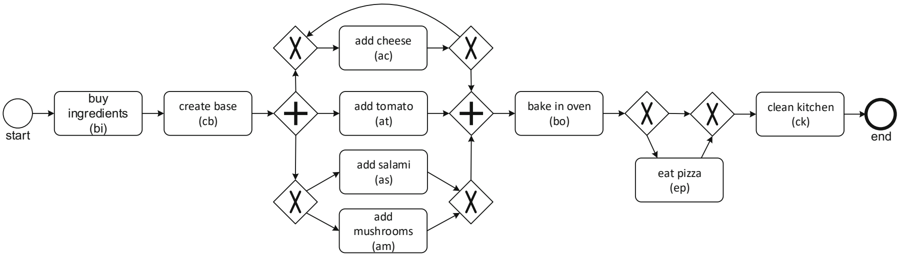


Fig. 7. BPMN model of the extended "pizza process".

To explain other process constructs such as choice, skipping, and looping we extend the "pizza process". First of all, we allow for adding multiple servings of cheese, i.e., activity *ac* can be executed multiple times after creating the pizza base and before putting the pizza in the oven. Second, instead of adding salami as a topping one can add mushrooms, i.e., there is a choice between *as* (add salami) and *am* (add mushrooms). Third, the eating of the pizza may be skipped (i.e., activity *ep* is optional).

Figure 7 shows the BPMN model with these three extensions. In total six exclusive gateways were added: three XOR-splits and three XOR-joins (see the diamond-shaped symbols with a × inside). After adding cheese, one can loop back. There is a choice between adding salami and adding mushrooms. Also the eating of the pizza can be skipped.

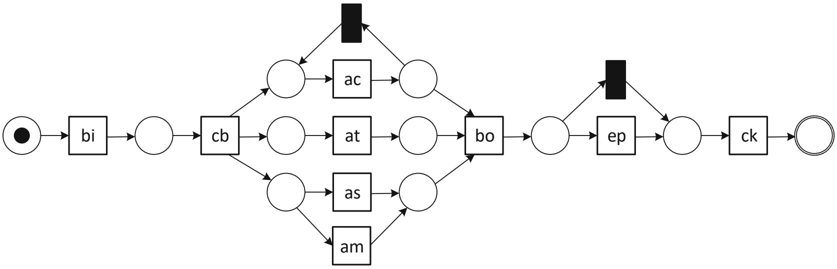


Fig. 8. Petri net modeling the extended "pizza process" with two silent transitions (to skip eating the pizza and to add more cheese), and a transition *am* corresponding to activity *add mushrooms.*

Figure 8 shows a Petri net modeling the extended process. A new transition *am* (add mushrooms) has been added. Transitions *as* and *am* share an input place. If the input place is marked, then both transitions are enabled, but only one of them can occur. If *as* consumes the token from the shared input place, then *am* gets disabled. If *am* consumes the token from the shared input place, then *as* gets disabled. This way, we model the choice between two toppings: salami and mushrooms. Figure 8 also has two new so-called *silent transitions* denoted by the two black rectangles. Sometimes such silent transitions are denoted as a normal transition with a τ label. Silent transitions do not correspond to activities and are used for routing only, e.g., skipping activities. In Fig. 8, there is one silent transition to repeatedly execute *ac* (to model adding multiple servings of cheese) and one silent transition to skip *ep*.

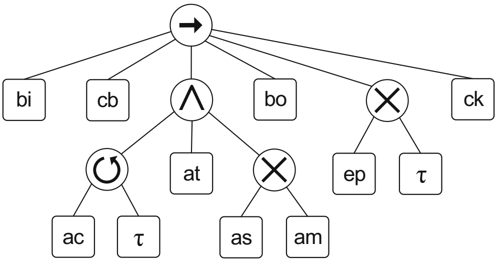


Fig. 9. Process tree of the extended "pizza process": $\rightarrow(bi, cb, \wedge(\circlearrowleft(ac, \tau), at, \times(as, am)), bo, \times(ep, \tau), ck)$.

The process tree in Fig. 9 has the same behavior as the BPMN model and Petri net just shown. The process tree uses all four operators: → (sequential composition), × (exclusive choice), ∧ (parallel composition), and ↻ (redo loop). A silent activity is denoted by τ and cannot be observed. The process tree in Fig. 9 can also be visualized in textual form: $\rightarrow(bi, cb, \wedge(\circlearrowleft(ac,\tau), at, \times(as, am)), bo, \times(ep,\tau), ck)$.

To understand the notation, we first look at a few smaller examples. Process tree $\times(a, b)$ models a choice between activities a and b. Process tree $\times(a, \tau)$ can be used to model an activity a that can be skipped. Process tree $\circlearrowleft(a, \tau)$ can be used to model the process that executes a at least once. The "redo" part is silent, so the process can loop back without executing any activity. Process tree $\circlearrowleft(\tau, a)$ models a process that executes a any number of times. The "do" part is now silent and activity a is in the "redo" part. This way it is also possible to not execute a at all.

Now let us take a look at the three modifications of our extended "pizza process": $\circlearrowleft(ac, \tau)$ models that multiple servings of cheese can be added, $\times(as, am)$ models the choice between salami and mushrooms, and $\times(ep, \tau)$ models the ability to skip eating the pizza.

The DFG shown in Fig. 10 incorporates the three extensions. Again, the behavior is different from Figs. 7, 8, and 9. Unlike the other models, the DFG allows for adding multiple servings of salami, mushrooms, and tomato paste. It is impossible to model concurrency properly, because loops are added the moment the order is not fixed. Therefore, DFGs are suitable for a quick first view of the process, but for more advanced process analytics, higher-level notations such as BPMN, Petri nets, and process trees are needed.

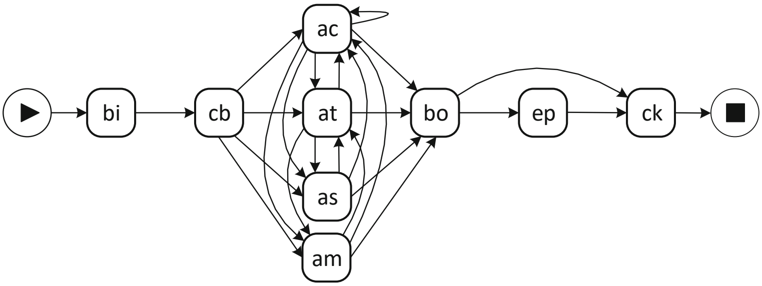


Fig. 10. DFG of the extended "pizza process". Note that the process becomes increasingly Spaghetti-like, allowing for process executions different from the BPMN model, the Petri net, and the process tree.

Note that, in this section, we focused on control-flow. However, process models can be extended with frequencies, probabilities, decision rules, roles, costs, and time delays (e.g., mean waiting times). After discovering the control-flow and replaying the event data on the model, it is easy to extend process models with data, resource, cost, and time perspectives.

3 Event Data

Using process mining, we would like to analyze and improve processes using event data. Table 1 shows a fragment of an event log in tabular form. One can think of this as a table in a relational database, a CSV (Comma Separated Value) file, or Excel spreadsheet. Each row in the table corresponds to an *event*. An event can have many different attributes. In this simple example, each event has five attributes: *case*, *activity*, *timestamp*, *resource*, and *customer*. Most process mining tools and approaches require at least three attributes: *case* (refers to a process instance), *activity* (refers to the operation, action, or task), and *timestamp* (when did the event happen). These three attributes are enough to discover and check the control-flow perspective. A case may refer to an order, a patient, an application, a student, a loan, a car, a suitcase, a speeding ticket, etc. In Table 1, each case refers to a pizza being produced and consumed. In Sect. 2 we showed process models describing this process. However, now we start from the observed behavior recorded in the event log. We can witness the same activities as before: buy ingredients (bi), create base (cb), add cheese (ac), add tomato (at), add salami (as), add mushrooms (am), bake in oven (bo), eat pizza (ep), and clean kitchen (ck). Table 1 uses a simple time format (e.g., *18:10*) to simplify the presentation (i.e., we skipped the date). Systems often use the ISO 8601 standard (or similar) to exchange date- and time-related data, e.g., *2021-09-21T18:10:00+00:00*. In the remainder, we formalize event data and provide useful notions to reason about both observed and modeled behavior. We start with some basic mathematical notations.

Table 1. Fragment of a larger event log with 6400 events, i.e., the whole table has 6400 rows. These events describe the production of 800 pizzas. Each row refers to an event having five attributes, including the three mandatory ones: case, activity, and timestamp.

Case	Activity	Timestamp	Resource	Customer
...	...	...	...	...
pizza-56	buy ingredients (bi)	18:10	Stefano	Valentina
pizza-57	buy ingredients (bi)	18:12	Stefano	Giulia
pizza-57	create base (cb)	18:16	Mario	Giulia
pizza-56	create base (cb)	18:19	Mario	Valentina
pizza-57	add tomato (at)	18:21	Mario	Giulia
pizza-57	add cheese (ac)	18:27	Mario	Giulia
pizza-56	add cheese (ac)	18:34	Mario	Valentina
pizza-56	add tomato (at)	18:44	Mario	Valentina
pizza-56	add salami (as)	18:45	Mario	Valentina
pizza-56	bake in oven (bo)	18:48	Stefano	Valentina
pizza-57	add salami (as)	18:50	Mario	Giulia

(*continued*)

Table 1. (*continued*)

Case	Activity	Timestamp	Resource	Customer
pizza-56	eat pizza (*ep*)	19:10	Valentina	Valentina
pizza-58	buy ingredients (*bi*)	19:17	Stefano	Laura
pizza-57	bake in oven (*bo*)	19:23	Stefano	Giulia
pizza-57	eat pizza (*ep*)	19:27	Giulia	Giulia
pizza-57	clean kitchen (*ck*)	19:44	Mario	Giulia
pizza-58	create base (*cb*)	19:48	Mario	Laura
pizza-58	add salami (*as*)	19:49	Mario	Laura
pizza-58	add tomato (*at*)	19:55	Mario	Laura
pizza-56	clean kitchen (*ck*)	20:08	Mario	Valentina
pizza-58	add cheese (*ac*)	20:13	Mario	Laura
pizza-58	bake in oven (*bo*)	20:29	Stefano	Laura
pizza-58	eat pizza (*ep*)	20:48	Laura	Laura
pizza-58	clean kitchen (*ck*)	20:51	Mario	Laura
...	...	...	...	...

3.1 Notations

$\mathcal{B}(A)$ is the set of all *multisets* over some set A. For some multiset $b \in \mathcal{B}(A)$, $b(a)$ denotes the number of times element $a \in A$ appears in b. Some examples: $b_1 = [\,]$, $b_2 = [x, x, y]$, $b_3 = [x, y, z]$, $b_4 = [x, x, y, x, y, z]$, and $b_5 = [x^3, y^2, z]$ are multisets over $A = \{x, y, z\}$. b_1 is the empty multiset, b_2 and b_3 both consist of three elements, and $b_4 = b_5$, i.e., the ordering of elements is irrelevant and a more compact notation may be used for repeating elements. The standard set operators can be extended to multisets, e.g., $x \in b_2$, $b_2 \uplus b_3 = b_4$, $b_5 \setminus b_2 = b_3$, $|b_5| = 6$, etc. $\{a \in b\}$ denotes the set with all elements a for which $b(a) \geq 1$. $b(X) = \sum_{a \in X} b(x)$ is the number of elements in b belonging to set X, e.g., $b_5(\{x, y\}) = 3 + 2 = 5$. $b \leq b'$ if $b(a) \leq b'(a)$ for all $a \in A$. Hence, $b_3 \leq b_4$ and $b_2 \not\leq b_3$ (because b_2 has two x's). $b < b'$ if $b \leq b'$ and $b \neq b'$. Hence, $b_3 < b_4$ and $b_4 \not< b_5$ (because $b_4 = b_5$).

$\sigma = \langle a_1, a_2, \ldots, a_n \rangle \in X^*$ denotes a *sequence* over X of length $|\sigma| = n$. $\sigma_i = a_i$ for $1 \leq i \leq |\sigma|$. $\langle\,\rangle$ is the empty sequence. $\sigma_1 \cdot \sigma_2$ is the concatenation of two sequences, e.g., $\langle x, x, y \rangle \cdot \langle x, y, z \rangle = \langle x, x, y, x, y, z \rangle$. The notation $[a \in \sigma]$ can be used to convert a sequence into a multiset. $[a \in \langle x, x, y, x, y, z \rangle] = [x^3, y^2, z]$.

$f \in X \to Y$ is a total function, i.e., $f(x) \in Y$ for any $x \in X$. $f \in X \not\to Y$ is a partial function with domain $\mathit{dom}(f) \subseteq X$. If $x \notin \mathit{dom}(f)$, then we write $f(x) = \perp$, i.e., the function is not defined for x.

3.2 Standard Event Log

An event log is a collection of events. An event e can have any number of attributes, and often we require the following three attributes to be present:

case $\#_{case}(e)$, activity $\#_{act}(e)$, and timestamp $\#_{time}(e)$. Table 1 shows example events. If e is the first visible event, then $\#_{case}(e) = \text{pizza-56}$, $\#_{act}(e) = bi$ (buy ingredients), and $\#_{time}(e) = \textit{18:10}$. For simplicity, we write *18:10*, but the full timestamp includes a date and possibly also seconds and milliseconds.

To formalize event logs, we introduce some basic notations.

Definition 1 (Universes). *$\mathcal{U}_{ev}$ is the universe of events, $\mathcal{U}_{act}$ is the universe of activities, $\mathcal{U}_{case}$ is the universe of cases, $\mathcal{U}_{time}$ is the universe of timestamps, $\mathcal{U}_{att} = \{act, case, time, \dots\}$ is the universe of attributes, $\mathcal{U}_{val}$ is the universe of values, and $\mathcal{U}_{map} = \mathcal{U}_{att} \not\rightarrow \mathcal{U}_{val}$ is the universe of attribute-value mappings. We assume that $\mathcal{U}_{act} \cup \mathcal{U}_{case} \cup \mathcal{U}_{time} \subseteq \mathcal{U}_{val}$, $\perp \notin \mathcal{U}_{val}$, and for any $f \in \mathcal{U}_{map}$: $f(act) \in \mathcal{U}_{act} \cup \{\perp\}$, $f(case) \in \mathcal{U}_{case} \cup \{\perp\}$, and $f(time) \in \mathcal{U}_{time} \cup \{\perp\}$.*

Note that standard attributes of an event (activity, case, timestamp, etc.) are treated as any other attribute. $f \in \mathcal{U}_{map}$ is a function mapping any subset of attributes onto values. For example, f could be such that $dom(f) = \{case, act, time, resource, customer, cost, size\}$, $f(case) = \text{pizza-56}$, $f(act) = bi$, $f(time) = \textit{2021-09-21T18:10:00+00:00}$, $f(resource) = \text{Stefano}$, $f(customer) = \text{Valentina}$, $f(size) = \text{33cm}$, and $f(cost) = \text{€9.99}$. Note that the last two attributes are not shown in Table 1. and that *2021-09-21T18:10:00+00:00* is abbreviated to *18:10*.

To be general, we assume that events are partially ordered. Recall that a strict partial order is *irreflexive* ($e \not\prec e$), *transitive* ($e_1 \prec e_2$ and $e_2 \prec e_3$ implies $e_1 \prec e_3$), and *asymmetric* (if $e_1 \prec e_2$, then $e_2 \not\prec e_1$).

Definition 2 (Event Log). *An event log is a tuple $L = (E, \#, \prec)$ consisting of a set of events $E \subseteq \mathcal{U}_{ev}$, a mapping $\# \in E \rightarrow \mathcal{U}_{map}$, and a strict partial ordering $\prec \subseteq E \times E$ on events.*

For any $e \in E$ and $att \in dom(\#(e))$: $\#_{att}(e) = \#(e)(att)$ is the value of attribute att for event e. For example, $\#_{act}(e)$, $\#_{case}(e)$, and $\#_{time}(e)$ are the activity, case, and timestamp of an event e.

The ordering of events respects time, i.e., if $e_1, e_2 \in E$, $\#_{time}(e_1) \neq \perp$, $\#_{time}(e_2) \neq \perp$, and $\#_{time}(e_1) < \#_{time}(e_2)$, then $e_2 \not\prec e_1$.

To be general, events can have any number of attributes and no attribute is mandatory. However, when using simplified event logs, we only consider events having a case and activity (with an order derived using timestamps).

Assume $L = (E, \#, \prec)$ is the event log in Table 1. The whole event log has 6400 events, i.e., the table has many more rows. Let $E = \{e_1, e_2, \dots, e_{6400}\}$ be the whole set of events and assume the first event shown in Table 1 is e_{433}. $\#(e_{433})$ is a mapping with $dom(\#(e_{433})) = \{case, act, time, resource, customer\}$ (the columns shown in the table). $\#_{case}(e_{433}) = \text{pizza-56}$, $\#_{act}(e_{433}) = bi$ (buy ingredients), $\#_{time}(e_{433}) = \text{18:10}$, $\#_{resource}(e_{433}) = \text{Stefano}$, and $\#_{customer}(e_{433}) = \text{Valentina}$. Assuming that the event identifiers follow the order shown in Table 1, the last event visible in the table is e_{456}, and $\#_{case}(e_{456}) = \text{pizza-58}$, $\#_{act}(e_{456}) = ck$ (clean kitchen), $\#_{time}(e_{456}) = \text{20:51}$, $\#_{resource}(e_{456}) = \text{Mario}$, and $\#_{customer}(e_{456}) = \text{Laura}$. Assuming a total order as shown in the Table, $e_{433} \prec e_{434}$, $e_{434} \prec e_{435}$, $e_{455} \prec e_{456}$, $e_{433} \prec e_{456}$, etc.

As stated in Definition 2, $\prec$ is a strict partial order and it is not allowed that timestamps (when present) and the partial order disagree. Using Table 1 and the event identifiers e_{433} and e_{456}. It cannot be that $e_{456} \prec e_{433}$, because $\#_{time}(e_{456}) > \#_{time}(e_{433})$. For two arbitrary events e_1 and e_2 it cannot be that both $\#_{time}(e_1) < \#_{time}(e_2)$ and $e_2 \prec e_1$. However, it can be that $\#_{time}(e_1) < \#_{time}(e_2)$ and $e_1 \not\prec e_2$ (the time perspective is more fine grained) or that $\#_{time}(e_1) = \#_{time}(e_2)$ and $e_1 \prec e_2$ (the partial order is more fine grained). Optionally, the partial order can be derived from the timestamps (when present): $\prec = \{(e_1, e_2) \in E \times E \mid \#_{time}(e_1) < \#_{time}(e_2)\}$. In this case, the event log is fully defined by $L = (E, \#)$ (no explicit ordering relation is needed).

It should be noted that in the often used BPI Challenge 2011 log provided by a Dutch academic hospital [16], 85% of the events have the same timestamp as the previous one. This is because, for many events, only dates are available. Many publicly available event logs have similar issues, for example, in the so-called Sepsis log [30], 30% of the events have the same timestamp as the previous one. In this event log, activities for the same case are sometimes batched, leading to events with the same timestamp. These examples illustrate that one should inspect timestamps and not take the order in the event log for granted. It may be beneficial to use partially ordered event data in case of data quality problems or when there is explicit causal information.

3.3 Simplified Event Log

For process mining techniques focusing on control-flow, it often suffices to focus only on the activity attribute and the ordering within a case. This leads to a much simpler event log notion.

Definition 3 (Simplified Event Log). *A simplified event log $L \in \mathcal{B}({\mathcal{U}_{act}}^*)$ is a multiset of traces. A trace $\sigma = \langle a_1, a_2, \ldots a_n \rangle \in {\mathcal{U}_{act}}^*$ is a sequence of activities. $L(\sigma)$ is the number of times trace σ appears in event log L.*

Consider case pizza-56 in Table 1. There are eight events having this case attribute. By ordering these events based on their timestamps we get the trace $\sigma_{\text{pizza-56}} = \langle bi, cb, ac, at, as, bo, ep, ck \rangle$. We can do the same for the other two cases shown in Table 1: $\sigma_{\text{pizza-57}} = \langle bi, cb, at, ac, as, bo, ep, ck \rangle$ and $\sigma_{\text{pizza-58}} = \langle bi, cb, as, at, ac, bo, ep, ck \rangle$. We are using the same shorthands as before, i.e., buy ingredients (bi), create base (cb), add cheese (ac), add tomato (at), add salami (as), add mushrooms (am), bake in oven (bo), eat pizza (ep), and clean kitchen (ck).

The same trace may appear multiple times in a log. For example, $L = [\langle a, b, c, e \rangle^{10}, \langle a, c, b, e \rangle^{5}, \langle a, d, e \rangle]$ is a simple event log with $10 + 5 + 1 = 16$ cases and $40 + 20 + 3 = 63$ events.

An event log with events having any number of attributes (Definition 2) can be transformed into a *simplified event log* by ignoring the additional attributes and sequentializing the events belonging to the same case. Events without a case or activity attribute are ignored in the transformation process.

Definition 4 (Conversion). *An event log $L = (E, \#, \prec)$ defines a simplified event log $\tilde{L} \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ that is constructed as follows:*

- $E' = \{e \in E \mid \#_{case}(e) \neq \perp \wedge \#_{act}(e) \neq \perp\}$ *are all events having an activity and a case attribute.*
- $C = \{\#_{case}(e) \mid e \in E'\}$ *and* $A = \{\#_{act}(e) \mid e \in E'\}$ *are the cases and activities in* L*.*
- *For any case* $c \in C$*:*
  - $E_c = \{e \in E' \mid \#_{case}(e) = c\}$ *are the events in* c*,*
  - $\sigma_c = \langle e_1, e_2, \ldots, e_n \rangle$ *is a (deterministically chosen) sequentialization of the events in* c*, i.e.,* σ_c *is such that* $\{e_1, e_2, \ldots, e_n\} = E_c$*,* $|E_c| = |\sigma_c|$*, and for any* $1 \leq i < j \leq n$*:* $e_j \not\prec e_i$*.*
  - $\tilde{\sigma}_c = \langle \#_{act}(e_1), \#_{act}(e_2), \ldots, \#_{act}(e_n) \rangle \in A^*$ *is the trace corresponding to* c *(i.e., the events in* σ_c *are replaced by the corresponding activities).*
- $\tilde{L} = [\tilde{\sigma}_c \mid c \in C] \in \mathcal{B}(A^*)$ *is the simplified event log derived from* L*.*

Let $L = (E, \#, \prec)$ be the event log corresponding to the events visible in Table 1 (assuming the order in the table). Then: $\tilde{L} = [\langle bi, cb, ac, at, as, bo, ep, ck \rangle, \langle bi, cb, at, ac, as, bo, ep, ck \rangle, \langle bi, cb, as, at, ac, bo, ep, ck \rangle]$. Table 1 only shows a fragment of the whole event log. For the whole event log $L = (E, \#, \prec)$, we have $\tilde{L} = [\langle bi, cb, ac, at, as, bo, ep, ck \rangle^{400}, \langle bi, cb, at, ac, as, bo, ep, ck \rangle^{200}, \langle bi, cb, as, at, ac, bo, ep, ck \rangle^{100}, \langle bi, cb, ac, as, at, bo, ep, ck \rangle^{50}, \langle bi, cb, at, as, ac, bo, ep, ck \rangle^{25}, \langle bi, cb, as, ac, at, bo, ep, ck \rangle^{25}]$. This event log has 800 cases and 6400 events. Using process discovery techniques we can automatically discover the models in Figs. 3, 4, 5 and 6 from such an event log. If the event log also has cases where cheese is added multiple times (e.g., $\langle bi, cb, ac, at, ac, ac, as, bo, ep, ck \rangle$), mushrooms are added instead of salami (e.g., $\langle bi, cb, ac, at, am, bo, ep, ck \rangle$), and the eating activity is skipped (e.g., $\langle bi, cb, ac, at, as, bo, ck \rangle$), then we can automatically discover the models in Figs. 7, 8, 9 and 10 using suitable process mining techniques.

3.4 Object-Centric Event Logs

Table 1 corresponds to a conventional "flat" event log where each event (i.e., row) refers to a case, activity, and timestamp. It is very natural to assume that an event has indeed a timestamp and refers to an activity. However, the assumption that it refers to precisely one case may cause problems [4]. *Object-Centric Event Logs (OCEL)* aim to overcome this limitation [22]. In OCEL, an event may refer to any number of objects (of different types) rather than a single case. Object-centric process mining techniques may produce Petri nets with different types of objects [7] or artifact-centric process models [18,19].

Table 2. Fragment of a larger Object-Centric Event Log (OCEL) with four types of objects: pizza, resource, customer, and location. One event may refer to a set of objects, e.g., three pizzas, three customer, and a location.

Activity	Timestamp	Pizza	Resource	Customer	Location
...	...	...	...	...	...
buy ingredients (*bi*)	*18:10*	{pizza-56, pizza-57, pizza-58}	{Stefano}	{Valentina, Giulia, Laura}	{supermarket}
create base (*cb*)	18.16	{pizza-57}	{Mario, Stefano}	{Giulia}	{kitchen-1}
create base (*cb*)	18.19	{pizza-56}	{Mario, Stefano}	{Valentina}	{kitchen-1}
add tomato (*at*)	18.21	{pizza-57}	{Mario}	{Giulia}	{kitchen-1}
add cheese (*ac*)	18.27	{pizza-57}	{Mario}	{Giulia}	{kitchen-1}
add cheese (*ac*)	18.34	{pizza-56}	{Mario}	{Valentina}	{kitchen-1}
add tomato (*at*)	18.44	{pizza-56}	{Mario}	{Valentina}	{kitchen-1}
add salami (*as*)	18.45	{pizza-56}	{Mario}	{Valentina}	{kitchen-1}
bake in oven (*bo*)	18.48	{pizza-56}	{Stefano}	{Valentina}	{kitchen-1}
add salami (*as*)	18.50	{pizza-57}	{Mario}	{Giulia}	{kitchen-1}
eat pizza (*ep*)	19.10	{pizza-56}	{Valentina}	{Valentina}	{restaurant}
bake in oven (*bo*)	19.23	{pizza-57}	{Stefano}	{Giulia}	{kitchen-1}
eat pizza (*ep*)	19.27	{pizza-57}	{Giulia}	{Giulia}	{restaurant}
create base (*cb*)	19.48	{pizza-58}	{Mario, Stefano}	{Laura}	{kitchen-2}
add salami (*as*)	19.49	{pizza-58}	{Mario}	{Laura}	{kitchen-2}
add tomato (*at*)	19.55	{pizza-58}	{Mario}	{Laura}	{kitchen-2}
clean kitchen (*ck*)	20.08	∅	{Mario}	∅	{kitchen-1}
add cheese (*ac*)	20.13	{pizza-58}	{Mario}	{Laura}	{kitchen-2}
bake in oven (*bo*)	20.29	{pizza-58}	{Stefano}	{Laura}	{kitchen-2}
eat pizza (*ep*)	20.48	{pizza-58}	{Laura}	{Laura}	{restaurant}
clean kitchen (*ck*)	20.51	∅	{Mario}	∅	{kitchen-2}
...	...	...	...	...	...

To understand the problem, we use Table 2, which shows OCEL data in tabular form. Compared to Table 1, we do not assume a single case notion. Instead, an event may refer to any number of objects. In this toy example, we assume four types of objects: pizza, resource, customer, and location. Assume that e is the first event listed in Table 2. $\#_{act}(e) = bi$ (buy ingredients), $\#_{time}(e) = \mathit{18{:}10}$, $\#_{pizza}(e) = \{\text{pizza-56}, \text{pizza-57}, \text{pizza-58}\}$, $\#_{resource}(e) = \{\text{Stefano}\}$, $\#_{customer}(e) = \{\text{Valentina}, \text{Giulia}, \text{Laura}\}$, and $\#_{location}(e) = \{\text{supermarket}\}$. Note that in Table 1 there were three *bi* (buy ingredients) events, one for each pizza. Hence, Table 2 is closer to reality if the ingredients were indeed bought in the same visit to the supermarket. In a classical event log with a single case identifier, we need to artificially replicate events (one *bi* event per pizza). This may lead to misleading statistics, i.e., there was just one trip to the supermarket and not three. The three pizzas were created on demand, so the *bi* event also refers to the three customers. Table 2 also shows that creating the pizza base is team work, i.e., all *cb* events are done by both Mario and Stefano. If we assume that e is the last event visible in Table 2, then $\#_{act}(e) = ck$ (clean kitchen), $\#_{time}(e) = 20.51$, $\#_{pizza}(e) = \emptyset$, $\#_{resource}(e) = \{\text{Mario}\}$, $\#_{customer}(e) = \emptyset$, and $\#_{location}(e) = \{\text{kitchen-2}\}$. This expresses that, according to this event log, cleaning the second kitchen is unrelated to the pizza prepared in it.

Definition 2 can be easily extended to allow for *Object-Centric Event Logs* (OCEL). We just need to assume that event attributes include object types and that attribute-value mappings may yield sets of values (e.g., objects) rather than individual values. Without fully formalizing this, we simply assume that $\mathcal{U}_{objtyp} \subseteq \mathcal{U}_{att}$ is the universe of *object types*, $\mathcal{U}_{objs}$ is the universe of *objects*, and $\mathcal{P}(\mathcal{U}_{objs}) \subseteq \mathcal{U}_{val}$ (i.e., values can be sets of objects). Moreover, for any $f \in \mathcal{U}_{map}$ and $ot \in \mathcal{U}_{objtyp} \cap dom(f)$: $f(ot) \subseteq \mathcal{U}_{objs}$. Hence, *attribute value mappings can be used to also map object types onto sets of objects.*

To apply classical process mining techniques, we need to convert the object-centric event data to traditional event data. For example, we need to convert Table 2 into Table 1 if we pick object type *pizza* as a case notion. This is called "flattening the event log" and always requires picking an object type as a case notion. This can be formalized in a rather straightforward manner.

Definition 5 (OCEL Conversion). *Let $L = (E, \#, \prec)$ be an event log having an object type $ot \in \mathcal{U}_{objtyp}$ such that for any $e \in E$: $\#_{ot}(e) \subseteq \mathcal{U}_{objs}$ is the set of objects of type ot involved in event e. Based on this assumption, we can create a "flattened event log" $\tilde{L}_{ot} \in \mathcal{B}({\mathcal{U}_{act}}^*)$ that is constructed as follows:*

- $E' = \{e \in E \mid \#_{ot}(e) \neq \emptyset \wedge \#_{act}(e) \neq \perp\}$ *are all events having an activity and referring to at least one object of type ot.*
- $O = \bigcup_{e \in E'} \#_{ot}(e)$ *and* $A = \{\#_{act}(e) \mid e \in E'\}$ *are the objects of type ot and activities in L.*
- *For any object $o \in O$:*
  - $E_o = \{e \in E' \mid o \in \#_{ot}(e)\}$ *are the events involving object o,*
  - $\sigma_o = \langle e_1, e_2, \ldots, e_n \rangle$ *is a (deterministically chosen) sequentialization of the events involving o, i.e., σ_o is such that $\{e_1, e_2, \ldots, e_n\} = E_o$, $|E_o| = |\sigma_o|$, and for any $1 \leq i < j \leq n$: $e_j \not\prec e_i$.*
  - $\tilde{\sigma}_o = \langle \#_{act}(e_1), \#_{act}(e_2), \ldots, \#_{act}(e_n) \rangle \in A^*$ *is the trace corresponding to o (i.e., the events in σ_o are replaced by the corresponding activities).*
- $\tilde{L} = [\tilde{\sigma}_o \mid o \in O] \in \mathcal{B}(A^*)$ *is the simplified event log derived from L.*

Definition 5 shows that any OCEL can be transformed into a simplified event log. The simplified event log is a multiset of traces where each trace refers to the "lifecycle" of an object. Consider for example $\tilde{\sigma}_{\text{pizza-56}} = \langle bi, cb, ac, at, as, bo, ep \rangle$ showing the lifecycle of pizza-56 in Table 2. $\tilde{\sigma}_{\text{Stefano}} = \langle bi, cb, cb, bo, bo, cb, bo \rangle$ is the trace corresponding to resource Stefano. $\tilde{\sigma}_{\text{Valentina}} = \langle bi, cb, ac, at, as, bo, ep \rangle$ is the trace corresponding to customer Valentina. This trace is now the same as $\sigma_{\text{pizza-56}}$, but this would not be the case if Valentina eats multiple pizzas (e.g., in subsequent visits to the restaurant). $\tilde{\sigma}_{\text{supermarket}} = \langle bi \rangle$ is the trace corresponding to the location "supermarket" (assuming there was just one visit to the supermarket). $\tilde{\sigma}_{\text{restaurant}} = \langle ep, ep, ep \rangle$ is the trace corresponding to the location "restaurant" (again considering only the events visible in Table 2). These traces are rather short because we only consider the events shown in Table 2.

By converting an OCEL to a conventional event log, we can apply all existing process mining techniques. For each object type, we can create a process model showing the "flow of objects" of that type. However, flattening the event log using *ot* as a case notion potentially leads to the following problems.

- *Deficiency*: Events in the original event log that have *no* corresponding events in the flattened event log disappear from the data set (i.e., $\#_{ot}(e) = \emptyset$). For example, when selecting object type *pizza* as a case notion, the *clean kitchen* events disappear from the event log.
- *Convergence*: Events referring to *multiple* objects of the selected type are replicated, possibly leading to unintentional duplication (i.e., $|\#_{ot}(e))| \geq 2$). For example, when selecting object type *pizza* as a case notion, the first event in Table 2 will be mapped onto three events in the flattened event log. When selecting object type *resource* as a case notion, all *create pizza base* events are duplicated in the flattened event log. The replication of events can lead to misleading diagnostics.
- *Divergence*: Events referring to *different* objects of a type *not* selected as the case notion may still be considered causally related because a more coarse-grained object is shared. For example, when selecting object type *location* as a case notion, events corresponding to different pizzas are interleaved and one can no longer see the causal dependencies.

The first two problems are easy to understand: events disappear completely (deficiency) or are replicated leading to potentially misleading management information (convergence). The problem of divergence is more subtle. To understand this better, consider $\tilde{\sigma}_{\text{kitchen-1}} = \langle cb, cb, at, ac, ac, at, as, bo, as, bo, ck \rangle$ describing the "lifecycle" of the first kitchen. In this trace one can see *cb* followed by *cb* (two subsequent create pizza base events) and *ac* followed by *ac* (two subsequent add cheese events). However, these events refer to different pizzas and are not causally related. The discovered process model is likely to show loops involving *cb* and *ac*, although these events occur precisely once per pizza.

In summary, one can create different views on the process by flattening the event data for selected object types, but one should be careful to interpret these correctly (e.g., be aware of data duplication and the blurring of causalities).

The running "pizza process" example is not very realistic, and is only used to introduce the basic concepts in a clear manner. Earlier, we mentioned CRM systems like Salesforce and ERP systems like SAP S/4HANA, Oracle E-Business Suite, and Microsoft Dynamics 365. These systems are loaded with event data scattered over many database tables. ERP and CRM systems are widely used, broad in scope, and sector-agnostic. Also, more sector-specific systems used in banking, insurance, and healthcare have event data distributed over numerous tables. These tables refer to different types of objects that are often in a one-to-many or many-to-many relation. This immediately leads to the challenges described before.

Let us consider two of the processes almost any organization has: *Purchase-to-Pay* (P2P) and *Order-to-Cash* (O2C). The P2P process is concerned with the buy-side of an organization. The O2C process is concerned with the sell-side of a company. In the P2P process the organization is dealing with purchasing documents, items, suppliers, purchase requisitions, contracts, receipts, etc. Note that there may be many purchase orders per supplier and an order may consist of multiple items. Hence, events may refer to different objects and also multiple objects of the same time. In the O2C process, we can witness similar phenomena. A customer may place three orders on the same day and each order may have several items. Items from different orders may end up in the same delivery. Moreover, items in the same order may end up in different deliveries.

P2P and O2C processes are considered simple and there is a lot of experience with extracting such data from systems such as SAP. Still, these processes are more complicated than what many people think. It is not uncommon to find thousands of process variants. This offers great opportunities for process mining, because unexpected variants provide hints on how to improve the process. However, one should not underestimate the efforts needed for data extraction. Therefore, we discussed OCEL as it sits in-between the real database tables in systems such as SAP, Oracle, and Salesforce, and the flattened event logs assumed by most systems.

3.5 XES Standard

The initial version of the XES (eXtensible Event Stream) format was defined by the *IEEE Task Force on Process Mining* in September 2010. After several iterations, XES became the official IEEE standard for storing event data in 2016 [24]. XES is supported by most of the open-source process mining tools and many of the leading commercial tools. The goal is to facilitate the seamless exchange of event data between different systems. Of course, it is also possible to do this using relational databases or simple file formats. However, XES adds semantics to the data exchanged. Therefore, we focus on the concepts and refer to [24] for the syntax.

Figure 11 shows the XES meta model expressed in terms of a UML class diagram. A XES document (e.g., an XML file) contains one log consisting of any number of traces. Each trace describes a sequential list of events corresponding to a particular case. The log, its traces, and its events may have any number of attributes. Attributes may be nested. There are five core types: *String*, *Date*, *Int*, *Float*, and *Boolean*. XES does not prescribe a fixed set of mandatory attributes for each element (log, trace, and event), e.g., an event can have any number of attributes. However, to provide semantics for such attributes, the log refers to so-called XES *extensions*. An extension gives semantics to particular attributes. For example, the *Time extension* defines a timestamp attribute of type *xs:dateTime*. This corresponds to the $\#_{time}(e)$ attribute used before. The *Organizational extension* defines a resource attribute of type *xs:string*, i.e., the

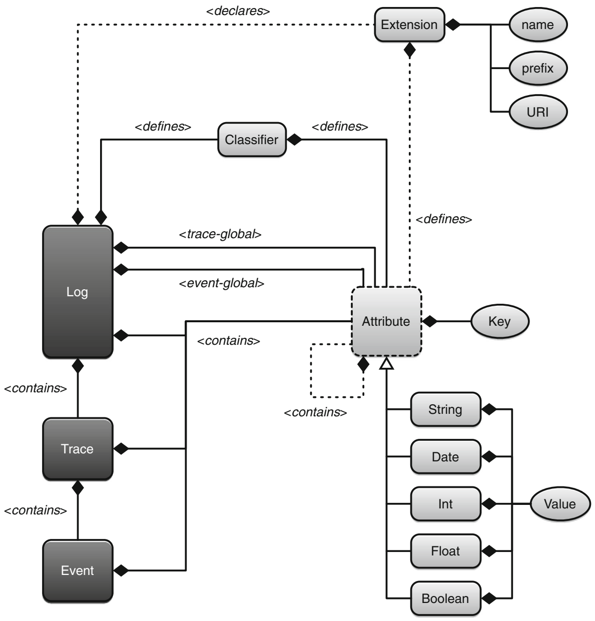


Fig. 11. Meta model of XES [24]. A log contains traces and each trace contains events [2,24]. Log, traces, and events have attributes. Extensions may define new attributes and a log should declare the extensions used in it. Global attributes are attributes that are declared to be mandatory. Such attributes reside at the trace or event level. Attributes may be nested. Event classifiers are defined for the log and assign a "label" (e.g., activity name) to each event. There may be multiple classifiers.

$\#_{resource}(e)$ attribute. Users can define their own extensions. For example, it is possible to develop domain-specific or even organization-specific extensions.

XES also supports three concepts that are of general interest and important for process mining: *classifiers*, *lifecycle* information, and *activity instances*. These concepts are interrelated as is discussed next.

Classifiers are used to attach labels to events. There is always at least one classifier and by default; this is the activity name. When turning an event log L into a simplified event log $\tilde{L} \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ in Definition 4, we are using this default classifier: each event e is mapped onto $\#_{act}(e)$. However, it is also possible to project events onto resources, locations, departments, etc., or combinations of attributes. An event classifier assigns to each event an identity, which makes it comparable to other events (via their assigned identity). Event classifiers are defined for the whole log, and there may be an arbitrary number of classifiers.

Thus far, we implicitly assumed that events are atomic. Therefore, an event has a timestamp. To handle activities that take time, XES provides the possibility to represent *lifecycle information* and to connect events through *activity instances*. An activity instance is a collection of related events that together represent the execution of an activity for a case. For example, an activity instance may be composed of a start event and a complete event. This way, we can derive information about the duration of an activity instance. The XES lifecycle model distinguishes between the following types of events: *schedule*, *assign*, *withdraw*, *reassign*, *start*, *suspend*, *resume*, *abort*, *complete*, *autoskip*, and *manualskip*. Using this XES extension, an event e has an attribute $\#_{type}(e)$. For example, assume that e_1 and e_2 are two events that belong to the same activity instance and $\#_{type}(e_1) = \text{start}$ and $\#_{type}(e_2) = \text{complete}$. $\#_{time}(e_2) - \#_{time}(e_1)$ is the duration of the activity. Similarly, we can measure waiting times, etc. Note that classifiers can also use lifecycle information, e.g., an event e is identified by the pair $(\#_{act}(e), \#_{type}(e))$. This implies that when we discover process models, there may be activities (a, start) and $(a, \text{complete})$.

Many XES logs contain lifecycle information, but few contain explicit activity instances. This implies that heuristics are needed to link events. For example, (a, start) is coupled to the first $(a, \text{complete})$ following it. However, in the trace $\langle \dots, (a, \text{start}), \dots, (a, \text{start}), \dots, (a, \text{complete}), \dots, (a, \text{complete}), \dots \rangle$, there are two possible ways to match starts and ends. Fortunately, it is often possible to extract activity instances from the original data source.

4 Different Types of Process Mining

After introducing multiple ways to represent process models (BPMN, Petri nets, process trees, and DFGs) and different types of events logs (e.g., XES and OCEL), we now briefly introduce some of the standard process mining tasks (see Fig. 12). As a starting point, we assume that high-quality event data are available. In practice, it is often time-consuming to extract event data from existing systems. As mentioned before, events may be scattered over multiple database tables or even multiple information systems using different identifiers. When starting with process mining, data extraction and data cleaning may take 80% of the time. Of course, the exact percentage depends on the type of process and information system. Also if the data pipeline is set up properly, this is a one-time effort that can be reused continuously.

4.1 Process Discovery

Event logs contain example behavior. The challenge is to discover a process model based on such example behavior. The model should not be "overfitting" (i.e., simply enumerating the observed example traces) and not "underfitting" (i.e., allow for behavior unrelated to what was observed). This is a difficult task and numerous algorithms have been proposed in literature, including the Alpha algorithm [8], region-based approaches [11,13,33,36], inductive mining techniques [28,29], and the split miner [9]. A baseline approach is the creation of a DFG, where the observed activities are added as nodes and two nodes a and b are connected through a directed arc if activity a is directly followed by activity b at least once. Obviously, such an approach is too simplistic and leads to underfitting process models. If activity a is directly followed by activity b in

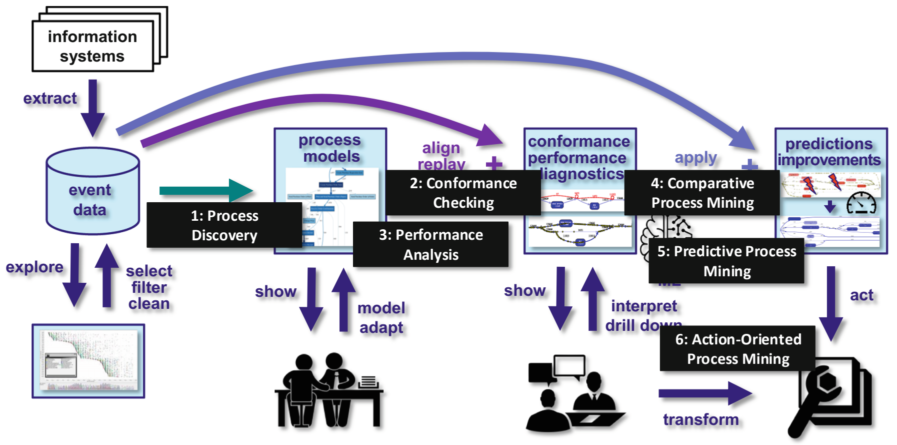


Fig. 12. Six frequently used types of process mining.

one case and activity b is directly followed by activity a in another case, then a loop is introduced. The techniques mentioned above address this problem and are able to uncover concurrency. However, there are many other challenges. The event log may contain *infrequent behavior*, i.e., traces or patterns which are less frequent compared to the mainstream behavior. Should this infrequent behavior be included or not? Hence, most approaches are parameterized to discard rare behavior. On the one hand, we often want to leave out infrequent behavior to simplify models. On the other hand, one cannot assume to have seen all behavior. Concurrency leads to an exponential number of states and a factorial number of possible traces. An unbounded loop leads to infinitely many possible traces. Process discovery is further complicated by the fact that event logs do *not* contain *negative* examples (i.e., traces that cannot happen) and are often *incomplete* (i.e., only a small fraction of all possible behavior is observed).

It is important to focus on a *particular process* or *problem*, having a *particular goal* in mind. One needs to select and filter the data based on a well-defined goal. Randomly using sliders to simplify process models may be useful for a first exploration, but will rarely lead to the desired insights.

To introduce process discovery, we focus on the *control-flow*, i.e., the ordering of activities. However, process models may include other *perspectives*, including time, data, resources, costs, etc. For example, a choice may be based on the attributes of the case or preceding event, and we may attach resource allocation rules to activities (e.g., role information and authorizations). Process discovery may add such perspectives, but we typically try to get clarity on the control-flow first. If no reasonable control-flow can be established, one should not try to add additional perspectives. Several process discovery techniques are explained in detail in [5,10].

4.2 Conformance Checking

Conformance checking requires both an event log and a process model as input. The goal is to indicate where log and model disagree. To illustrate this consider Figs. 7, 8, and 9. These three models describe exactly the same behavior of the extended "pizza process" that can be compactly described as $\rightarrow(bi, cb, \wedge(\circlearrowleft(ac, \tau), at, \times(as, am)), bo, \times(ep, \tau), ck)$. Let $M = \{\langle bi, cb, ac, at, as, bo, ep, ck, \rangle, \ldots \langle bi, cb, am, at, ac, ac, ac, bo, ep, ck, \rangle, \ldots \langle bi, cb, at, ac, am, bo, ck \rangle\}$ be the infinite set of all traces allowed by the BPMN model, Petri net, and process tree depicted in the three figures. Let $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ be an event log containing 800 traces. Assume $\sigma_1 = \langle bi, cb, ac, at, as, bo, ep, ck \rangle \in L$, $\sigma_2 = \langle bi, cb, ac, ac, at, am, ep, ck \rangle \in L$, and $\sigma_3 = \langle bi, cb, at, ac, at, as, bo, ck \rangle \in L$. Hence, $L = [\sigma_1, \sigma_2, \sigma_3, \ldots]$ and $|L| = 800$. $\sigma_1 \in M$, i.e., this is a perfectly fitting trace. $\sigma_2 \notin M$ because activity bo (bake in oven) is missing, i.e., someone was eating an uncooked pizza. $\sigma_3 \notin M$ because activity at (add tomato) occurs twice. The goal of conformance checking is to detect such deviations.

$L_{fit} = [\sigma \in L \mid \sigma \in M]$ is the multiset of fitting traces and $L_{dev} = [\sigma \in L \mid \sigma \notin M]$ is the multiset of deviating traces. Hence, fitness at the trace level can be defined as $|L_{fit}| \,/\, |L|$. The fraction is 1 if all traces are fitting and 0 if none of the traces is fitting.

There are many measures for fitness. For example, the above fraction does not take into account to what degree a trace is fitting or not. Trace $\sigma_4 = \langle bo, bo, bo, at, at, at, at, at \rangle \in L$ is obviously more deviating than σ_2 and σ_3. Moreover, it is not enough to produce a number. In practice, good diagnostics are much more important than a single quality measure.

There are many techniques for conformance checking. The two most frequently used approaches are *token-based replay* [32] and *alignments* [6,14]. For token-based replay, the process model is represented as a Petri net and traces in the event log are replayed on the model. If the trace indicates that an activity needs to take place, the corresponding transition is executed. If this is not possible because an input place is empty, a so-called *missing token* is added. Tokens that are never consumed are called *remaining tokens.* The numbers of missing and remaining tokens relative to the numbers of consumed and produced tokens indicate the severity of the conformance problem. Token-based replay can be extended to Petri nets with silent and duplicate activities using heuristics. For example, if there are two activities with the same label, pick the one that is enabled. If both are enabled, pick one of them. Similarly, silent transitions (i.e., transitions not corresponding to recorded activities) are executed when they enable a transition corresponding to the next activity in the event log. This requires an exploration of the states reachable from the current state and may lead to inconclusive results.

Compared to computing alignments, token-based replay is fairly efficient, but does not always produce valid paths through the process model. Alignments are often seen as the gold standard for conformance checking because they provide paths through the process model that are as close to the observed behavior as possible. We would like to map observed behavior onto modeled behavior to provide better diagnostics and to relate also non-fitting cases to the model. Alignments were introduced to overcome the limitations of token-based replay. The diagnostics are more detailed and more precise, because each observed trace is mapped onto a model behavior that is as close to what was observed as possible. The alignment shows common behavior, but also skipped and inserted events signaling deviations. Such skipped and inserted events are easier to interpret than missing and remaining tokens. However, for large event logs and processes, alignment computations may be intractable. Moreover, there may be many optimal alignments, making the diagnostics non-deterministic.

Several conformance checking techniques are explained in detail in [15]. When comparing observed and modeled behavior, we typically consider four main quality dimensions [1,2,6]:

- *Recall* (also called *replay fitness*): the discovered model should allow for the behavior seen in the event log. This can be quantified by the minimal number of edit operations needed to make all traces in the event log fitting into the model (or simply the fraction of perfectly fitting traces).
- *Precision*: the discovered model should not allow for behavior completely unrelated to what was seen in the event log. This can be quantified by the number of possible continuations in the model never observed in the event log.
- *Generalization*: the discovered model should generalize the example behavior seen in the event log. It is easy to create a process model that only allows for the behavior observed and nothing more. However, such a model is likely to overfit. To avoid overfitting, the model should generalize. This can only be tested on "fresh unseen" event data. To evaluate a process discovery algorithm, standard cross-validation can be used to detect overfitting problems. This is less clear when evaluating a process model rather than a discovery algorithm [6].
- *Simplicity*: the discovered model should be as simple as possible. This fourth quality criterion is related to Occam's Razor, which states that "one should not increase, beyond what is necessary, the number of entities required to explain anything".

4.3 Performance Analysis

The goal of process mining is to improve processes by uncovering problems. These may be the conformance problems just described, but (of course) also include performance problems such as untimely completion of a case, limited production, missed deadlines, tardiness, excessive rework, and recurring quality problems. Using *token-based replay* [32] and *alignments* [6,14] it is possible to relate event data to a process model. As a result, it is fairly straightforward to annotate the process model with frequency and time information. Frequencies of undesired activities and loops can be used to identify quality and efficiency problems. Since events have timestamps, it is possible to measure times in-between activities, including statistics such as mean, median, standard deviation, minimum, and maximum. This allows for analyzing performance indicators, e.g., waiting times, response times, and service times.

A *Service Level Agreement* (SLA) is an agreement between a service provider and a client. Process mining can be used to analyze SLAs, e.g., when is a particular SLA not met. Some well-known SLAs are churn/abandonment rate (number of cases lost), average speed to answer (response time seen by customer), percentage of cases handled within a predefined timeframe, first-call resolution (cases successfully handled without rerouting), percentage of duplicated cases (e.g., multiple procurement documents corresponding to the same order), mean time between failures, mean time to recovery, etc.

4.4 Comparative Process Mining

Comparative process mining uses as input *multiple* event logs, e.g., $L_1, L_2, \ldots, L_n \in \mathcal{B}(\mathcal{U}_{act}{}^*)$. These event logs may refer to different locations, periods, or categories of cases. For example, we may have the event logs L_{Aachen} and L_{Munich} referring to the same processes performed at two locations. We may have the event logs $L_{Jan}, L_{Feb}, L_{Mar}, \ldots, L_{Dec}$ referring to different periods or L_{Gold} and L_{Silver} referring to gold and silver customers.

Having multiple event logs allows for comparison and highly relevant questions. What are the striking differences and commonalities? What factors lead to these differences? Root cause analysis can be used to explain the observed differences. For example, in L_{Feb} waiting times may be much longer than in L_{Jan} due to limited resource availability. Comparative process mining may focus on frequently occurring problems, sometimes referred to as *execution gaps*. Such execution gaps include lost customers, additional work due to price changes, the merging of duplicate orders, and rework due to quality problems.

Comparative process mining is also a great tool for *inter- or intra-organizational benchmarking*. For example, an insurance company may have different regional offices. Using comparative process mining, these offices can learn from each other and increase the overall performance.

4.5 Predictive Process Mining

Process discovery, conformance checking, performance analysis, and comparative process mining are *backward-looking*. Although the value of such techniques is obvious, the actual goal is to continuously improve processes and respond to changes. Operational processes are subject to many changes, e.g., a sudden increase in the number of orders or disruptions in the supply chain. Moreover, many compliance and performance problems can be foreseen and addressed proactively. Fortunately, process models discovered and enriched using process mining can be used in a *forward-looking* manner.

Process mining can be used to create a range of *ML questions* that can be answered using standard software libraries. For example, when detecting a recurring bottleneck or deviation, it is possible to extract features from the event log and create a predictive model. This leads to a so-called *situation-feature table* with several descriptive features (e.g., people involved, path taken, and time of day) and one target feature (e.g., waiting time or decision). Then standard ML techniques ranging from regression and decision trees to neural networks can be applied to explain the target feature in terms of descriptive features. This leads to better diagnostics and explanations. Moreover, the models can be used in a predictive manner.

Predictive process mining questions also create specific ML challenges. Most ML techniques assume a fixed number of features as input (i.e., a fixed-length feature vector) and assume inputs to be independent. Artificial recurrent neural network architectures such as Long Short-Term Memory (LSTM) can be used to handle traces of variable length. Contextual features can be added to include information about the utilization of resources. However, this requires fine-tuning and domain knowledge.

A discovered process model can be viewed as a description of the *as-is* situation. Using simulation and model adaptation, it is possible to explore possible *to-be* situations. Simulation enables forward-looking forms of process mining. Comparative process mining can be used to compare the different alternatives.

4.6 Action-Oriented Process Mining

Process mining can be used to show (1) what has happened, (2) what is happening now, and (3) what will happen next in the process. Hence, it covers the full spectrum from backward-looking to forward-looking types of analysis. Backward-looking forms of process mining can lead to process redesigns and organizational changes. Forward-looking forms of process mining and diagnostics of the current state of a process can trigger improvement actions. Action-oriented process mining aims to turn diagnostics into actions. Assisted by low-code automation platforms, process mining software can trigger workflows. Some examples:

- The moment the average waiting time exceeds 2 h, additional resources are added and no new orders are accepted.
- If a supplier changes prices repeatedly for a longer period, then the supplier is blacklisted.
- If a check is repeatedly skipped by an employee, the manager is notified.

Next to triggering improvement actions, process mining can also detect repetitive work that may be automated using *Robotic Process Automation* (RPA). RPA can be used to automate repetitive tasks done by humans without changing the underlying systems. Typical examples include copying information from one system into another system. Process mining can be used to discover such repetitive tasks. The term *task mining* is often used to refer to the discovery of processes based on user-interface interactions (filling out a form, pushing a button, copying text, etc.). Task mining can be used to uncover repetitive processes that can be automated. There is also a connection to *online scheduling* and other Operations Research (OR) techniques. For example, based on historical information, it is possible to create a robust schedule with events taking place in the future. Differences between scheduled events and the actual events may trigger improvement actions.

5 Applications and Software

Process mining started as an exercise in the late 1990s trying to automatically create a Petri net from example traces [2]. According to Gartner there are now over 40 process mining vendors [26]. Some of them are listed in Table 3. Note that the list is very dynamic with new vendors emerging and large IT companies acquiring smaller process mining vendors. For an up-to-date overview, see the website www.processmining.org which lists all process mining tools.

Table 3. Some of the process mining tools available at the end of 2021. For each tool the vendor and website are listed. The last column indicates whether an academic version is available.

Vendor	Tool	Website	Acad. ver.
Abbyy	ABBYY Timeline	www.abbyy.com	No
Appian (Lana Labs)	LANA Process Mining	lanalabs.com	No
Apromore	Apromore Enterprise Edition	apromore.org	Yes
bupaR	bupaR	bupar.net	Yes
businessOptix	businessOptix	businessoptix.com	Yes
Celonis	Celonis EMS	celonis.com	Yes
Datricks	Datricks	datricks.com	Yes
DCR	DCR Portal	www.dcrsolutions.net	Yes
Deloitte	Process X-ray	processxray.deloitte.com	No
EverFlow	EverFlow	everflow.ai	No
Fluxicon	Disco	fluxicon.com	Yes
FortressIQ	FortressIQ	fortressiq.com	No
Fraunhofer FIT	PM4Py	pm4py.fit.fraunhofer.de	Yes
Hyland	Onbase	www.hyland.com	No
IBM (myInvenio)	myInvenio	my-invenio.com	No
Integris	Explora Process	integris.it	No
Kofax	Kofax Insight	www.kofax.com	No
livejourney	livejourney	www.livejourney.com	No
Logpickr	Logpickr Process Explorer 360	www.logpickr.com	No
Mavim	Mavim	www.mavim.co	No
Mehrwerk GmbH	MPM	mpm-processmining.com	No
Mindzie	mindzie	mindzie.com	Yes
Minit (Microsoft)	Minit	www.minit.io	Yes
Nintex UK ltd	Nintex	www.nintex.com	No
Oniq	IQ/A	www.oniq.com	No
PAFnow (Celonis)	PAFnow	pafnow.com	No
Process.science	process.science	www.process.science	No
ProcessDiamond	ProcessDiamond	processdiamond.com	Yes
ProcessM	PmBI	processm.com	Yes
Puzzle Data	ProDiscovery	www.puzzledata.com	No

(*continued*)

Table 3. (*continued*)

Vendor	Tool	Website	Acad. ver.
QPR Software	QPR ProcessAnalyzer	www.qpr.com	No
SAP (Signavio)	SAP Signavio	www.signavio.com	Yes
Skan AI	Skan	www.skan.ai	No
Software AG	Aris	aris-process-mining.com	Yes
Soroco	Scout Platform	soroco.com	No
StereoLogic	StereoLogic Process Mining	www.stereologic.com	No
TU/e	ProM	www.promtools.org	Yes
TU/e	RapidProM	www.rapidprom.org	Yes
UI Path	UI Path Process Mining	www.uipath.com	Yes
UltimateSuite	UltimateSuite TM/RPA	www.ultimatesuite.com	No
Upflux	Upflux	upflux.net	No
Worksoft	Worksoft	www.worksoft.com	No

All of the tools in Table 3 support the discovery of Directly-Follows Graphs (DFGs) with frequencies and times. Most of them (but not all) support some form of conformance checking and BPMN visualization. Some of the tools target process or data analysts rather than people managing or executing processes. These tools are typically lightweight and can be deployed quickly. *Enterprise-level process mining tools* are more difficult to deploy, but aim to be used by many stakeholders within an organization. For example, within Siemens, over 6000 employees are using the Celonis software to improve a range of processes. Enterprise-level process mining tools have automated connections to existing information systems (e.g., SAP, Salesforce, Oracle, ServiceNow, and Workday) to allow for the continuous ingestion of data. These tools also allow for customized dashboards to lower the threshold to use process mining. In 2020, Gartner estimated the process mining software market revenue to be $550 million, which was over 70% market size growth from the previous year [26]. The process mining market is forecast to keep growing 50% per year (Compound Annual Growth Rate) in the coming years. Note that this does not include consultancy based on process mining. The Big Four (i.e., Deloitte, Ernst & Young, KPMG, and PwC) all have process mining competence centers providing process mining services all over the globe.

The technology is generic and can be used in any domain. For example, process mining is used in

- finance and insurance (Rabobank, Wells Fargo, Hypovereinsbank, Caixa General, ADAC, APG, Suncorp, VTB, etc.),
- logistics and transport (Uber, Deutsche Bahn, Lufthansa, Airbus, Schukat, Vanderlande, etc.),
- production (ABB, Siemens, BMW, Fiat, Bosch, AkzoNobel, Bayer, Neste, etc.),
- healthcare, biomedicine, and pharmacy (Uniklinik RWTH Aachen, Charite University Hospital, GE Healthcare, Philips, Medtronic, Pfizer, Bayer, AstraZeneca, etc.),

- telecom (Deutsche Telekom, Vodafone, A1 Telekom Austria, Telekom Italia, etc.),
- food and retail (Edeka, MediaMarkt, Globus, Zalando, AB InBev, etc.),
- energy (Uniper, Chevron, Shell, BP, E.ON, etc.), and
- IT services (Dell, Xerox, IBM, Nokia, ServiceNow, etc.).

In [31], several use cases are described in detail. In [26,27], typical applications are described, and in [21] the results of a global process mining survey are presented. These show that the adoption is increasing, e.g., according to the global survey, 83% of companies already using process mining on a global scale plan to expand their initiatives [21]. *Process mining helps organizations to improve processes, provide transparency, reduce costs, ensure compliance, avoid risks, eliminate waste, and redesign problematic processes* [21]. To get a glimpse of the possible applications, the reader can take a look at the use cases collected by the IEEE Task Force on Process Mining [25] and HSPI Management Consulting [20]. Note that these cover just a fraction of the actual applications of process mining. It has become fairly standard to apply process mining to standard processes such as Purchase-to-Pay (P2P) and Order-to-Cash (O2C).

6 Summary and Outlook

This chapter aimed to provide a 360° overview of the field of process mining. We showed that process mining connects data science and process science leading to data-driven process-centric techniques and approaches. Event data and process models were introduced. Events can be grouped in event logs, but also stored in databases. In the standard setting an event has a few mandatory attributes such as case, activity, and timestamp. This can be further reduced to representing an event log by a multiset of traces where each trace is a sequence of activities. This format is often used for control-flow discovery. However, in real-life settings it is not so easy to find a single case notion. Often events may refer to multiple objects of different types. There may also be data quality problems and data may be scattered over multiple source systems. Moreover, additional attributes such as costs, time, and resources need to be incorporated in models. We introduced Directly-Follows Graphs (DFG), Petri nets, BPMN models, and process trees as basic control-flow representations. These will be used in the remainder.

We informally described six common types of process mining: (1) process discovery, (2) conformance checking, (3) performance analysis, (4) comparative process mining, (5) predictive process mining, and (6) action-oriented process mining. These characterize the scope of process mining and challenges. The chapter also provided pointers to the over 40 process mining tools and case studies.

Although process mining is already used by many of the larger organizations, it is a relatively new technology and only a fraction of its potential is realized today. Three important trends can be witnessed that together lead to a wider adoption.

- Supporting data extraction and analysis through *process-specific and domain-specific adapters and applications* ("process mining apps"). This reduces the

effort to get started with process mining and leverages past experiences in other organizations.
- Initially, process mining software aimed at experts involved in process improvement projects. However, process mining should be done *continuously* and at a *large scale*. It is a generic technology that should be accessible for many users every day. By scaling (both in terms of processes and users) and continuous use, the return on investment is the highest.
- Increasingly, *process mining and automation are combined*. Process mining diagnostics trigger corrective actions through low-code automation platforms. This is the only way to ensure that improvements are realized. Without some form of automation, workers may slip back into the old ineffective ways of working that were exposed using process mining.

Process mining can also play a role in realizing sustainability goals and help to address environmental, social and economic challenges. Process mining can help to quantify and steer sustainability efforts, e.g., by removing waste and quantifying emissions. Process mining can easily handle multiple dimensions, such as time, cash flow, resource usage, and CO_2 emissions, during analysis. Sustainability is just one of many topics where process mining can play a role. Moreover, these applications also pose interesting research questions leading to new concepts and techniques.

Acknowledgment. Funded by the Alexander von Humboldt (AvH) Stiftung and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2023 Internet of Production – 390621612.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19345-3
2. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
3. van der Aalst, W.M.P.: A practitioner's guide to process mining: limitations of the directly-follows graph. Procedia Comput. Sci. **164**, 321–328 (2019)
4. Aalst, W.M.P.: Object-centric process mining: dealing with divergence and convergence in event data. In: Ölveczky, P.C., Salaün, G. (eds.) SEFM 2019. LNCS, vol. 11724, pp. 3–25. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30446-1_1
5. van der Aalst, W.M.P.: Foundations of process discovery. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
6. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Min. Knowl. Discovery **2**(2), 182–192 (2012)
7. van der Aalst, W.M.P., Berti, A.: Discovering object-centric Petri nets. Fund. Inform. **175**(1–4), 1–40 (2020)

8. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
9. Augusto, A., Conforti, R., Marlon, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowl. Inf. Syst. **59**(2), 251–284 (2019)
10. Augusto, A., Carmona, J., Verbeek, E.: Advanced process discovery techniques. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
11. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 375–383. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_27
12. vom Brocke, J., et al.: Process Science: The Interdisciplinary Study of Continuous Change. SSRN (2021). http://ssrn.com/abstract=3916817
13. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering petri nets from event logs. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 358–373. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85758-7_26
14. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking: Relating Processes and Models. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99414-7
15. Carmona, J., van Dongen, B., Weidlich, M.: Conformance checking: foundations, milestones and challenges. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
16. van Dongen, B.F.: Real-Life Event Logs: Hospital Log (4TU.ResearchData) (2011). https://doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54
17. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.: Fundamentals of Business Process Management. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-662-56509-4
18. van Eck, M.L., Sidorova, N., van der Aalst, W.M.P.: Guided interaction exploration and performance analysis in artifact-centric process models. Bus. Inf. Syst. Eng. **61**(6), 649–663 (2018). https://doi.org/10.1007/s12599-018-0546-0
19. Fahland, D.: Describing behavior of processes with many-to-many interactions. In: Donatelli, S., Haar, S. (eds.) PETRI NETS 2019. LNCS, vol. 11522, pp. 3–24. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21571-2_1
20. Cotroneo, G., Carbone, R., Boggini, S., Cerini, M.: Process Mining: A Database of Applications (2021). HSPI Management Consulting 2021. http://www.hspi.it/
21. Galic, G., Wolf, M.: Global Process Mining Survey 2021: Delivering Value with Process Analytics - Adoption and Success Factors of Process Mining. Deloitte (2021). https://www2.deloitte.com/de/de/pages/finance/articles/global-process-mining-survey-2021.html
22. Ghahfarokhi, A.F., Park, G., Berti, A., van der Aalst, W.M.P.: OCEL Standard (2021). http://www.ocel-standard.org/
23. van der Aalst, W., et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28108-2_19
24. IEEE Task Force on Process Mining. XES Standard Definition (2016). http://www.xes-standard.org/
25. IEEE Task Force on Process Mining. Case Studies (2022). http://www.tf-pm.org/

26. Kerremans, M., Srivastava, T., Choudhary, F.: Gartner Market Guide for Process Mining, Research Note G00737056 (2021). www.gartner.com
27. Koplowitz, R., Mines, C., Vizgaitis, A., Reese, A.: Process Mining: Your Compass For Digital Transformation: The Customer Journey Is The Destination (2019). www.forrester.com
28. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann, N., Song, M., Wohed, P. (eds.) BPM 2013. LNBIP, vol. 171, pp. 66–78. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06257-0_6
29. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery with guarantees. In: Gaaloul, K., Schmidt, R., Nurcan, S., Guerreiro, S., Ma, Q. (eds.) CAISE 2015. LNBIP, vol. 214, pp. 85–101. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19237-6_6
30. Mannhardt, F.: Road Traffic Fine Management Process (4TU.ResearchData) (2016). https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460
31. Reinkemeyer, L.: Process Mining in Action: Principles, Use Cases and Outlook. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-40172-6
32. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)
33. Solé, M., Carmona, J.: Process mining from a basis of state regions. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 226–245. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13675-7_14
34. Suriadi, S., Andrews, R., ter Hofstede, A.H.M., Wynn, M.T.: Event log imperfection patterns for process mining: towards a systematic approach to cleaning event logs. Inf. Syst. **64**, 132–150 (2017)
35. Taylor, F.W.: The Principles of Scientific Management. Harper and Brothers Publishers, New York (1919)
36. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P., Verbeek, H.M.W.: Discovering workflow nets using integer linear programming. Computing **100**(5), 529–556 (2018)

Process Discovery

Foundations of Process Discovery

Wil M. P. van der Aalst(✉)

Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany
wvdaalst@pads.rwth-aachen.de
http://www.vdaalst.com/

Abstract. Process discovery is probably the most interesting, but also most challenging, process mining task. The goal is to take an event log containing example behaviors and create a process model that adequately describes the underlying process. This chapter introduces the baseline approach used in most commercial process mining tools. A simplified event log is used to create a so-called *Directly-Follows Graph* (DFG). This baseline is used to explain the challenges one faces when trying to discover a process model. After introducing DFG discovery, we focus on techniques that are able to discover models allowing for concurrency (e.g., Petri nets, process trees, and BPMN models). The chapter distinguishes two types of approaches able to discover such models: (1) *bottom-up process discovery* and (2) *top-down process discovery*. The *Alpha algorithm* is presented as an example of a bottom-up technique. The approach has many limitations, but nicely introduces the idea of discovering local constraints. The basic *inductive mining* algorithm is presented as an example of a top-down technique. This approach, combined with frequency-based filtering, works well on most event logs. These example algorithms are used to illustrate the foundations of process discovery.

Keywords: Process discovery · Process models · Petri nets · BPMN

1 Introduction

Process discovery is typically the first step after extracting event data from source systems. Based on the selected event data, process discovery algorithms automatically construct a process model describing the observed behavior. This may be challenging because, in most cases, the event data cannot be assumed to be complete, i.e., we only witnessed example behaviors. There may also be conflicting requirements (e.g., recall, precision, generalization, and simplicity) [1,3]. This makes process discovery both interesting and challenging.

Figure 1 positions this chapter. The input for process discovery is a collection of events and the output is a process model. Such a process model can be used to uncover unexpected deviations and bottlenecks. In the later stages of the process mining pipeline shown in Fig. 1, process models are used to check compliance, compare processes, detect concept drift, and predict performance and compliance problems.

Events may have many attributes and refer to multiple objects of different types [3]. However, in this chapter, we start from very basic event data. We assume that each *event*

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 37–75, 2022.
https://doi.org/10.1007/978-3-031-08848-3_2

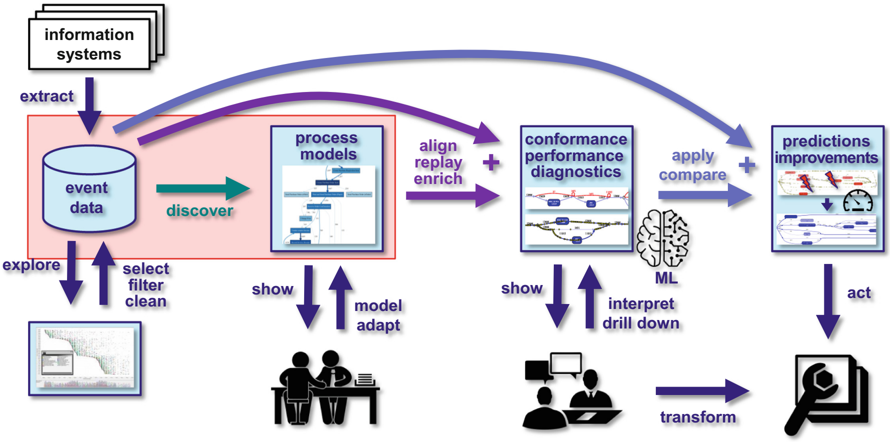


Fig. 1. This chapter focuses on process discovery. This is the first step after extracting event data from the source system(s). To set the scene, we consider only control-flow information, i.e., the ordering of activities.

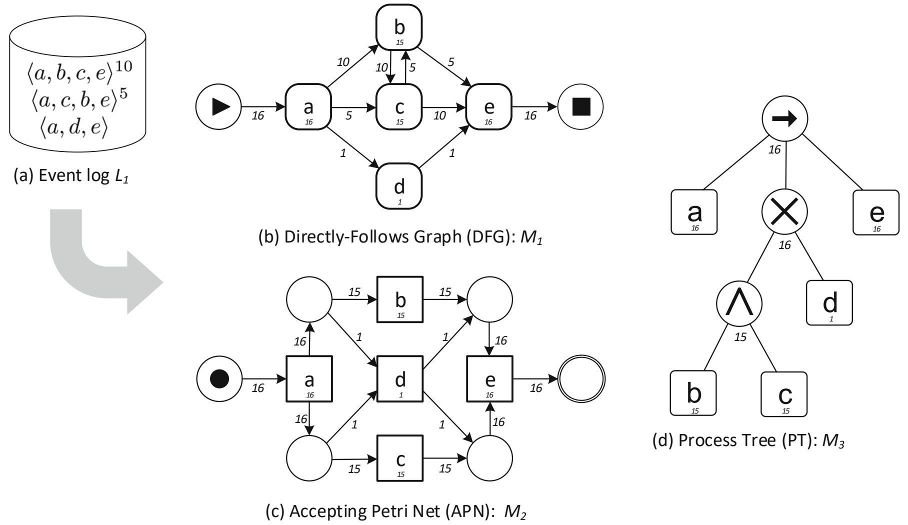
refers to a *case*, an *activity*, and has a *timestamp*. There may be many other attributes (e.g., resource), but we ignore these. Initially, we assume that timestamps are only used for the ordering of events corresponding to the same case. This implies that each case is represented by a *sequence of activities*. We call this a *trace*. For example, $\sigma = \langle a, b, c, e \rangle$ represents a case for which the activities a, b, c, and e occurred. Note that there may be many cases that have the same trace. Therefore, we represent an event log as a multiset of traces. For example, $L_1 = [\langle a, b, c, e \rangle^{10}, \langle a, c, b, e \rangle^5, \langle a, d, e \rangle]$ is an event log describing 16 cases and $10 \times 4 + 5 \times 4 + 1 \times 3 = 63$ events. Note that trace $\sigma = \langle a, b, c, e \rangle$ appears 10 times. In [3], we use the term *simplified* event log. Here we drop the adjective "simplified" since the representation will be used throughout the chapter.

Definition 1 (Event Log). *$\mathcal{U}_{act}$ is the universe of activity names. A trace $\sigma = \langle a_1, a_2, \dots, a_n \rangle \in \mathcal{U}_{act}{}^*$ is a sequence of activities. An event log $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ is a multiset of traces.*

Note that $L(\sigma)$ is the number of times trace σ appears in event log L. For example, $L_1(\langle a, b, c, e \rangle) = 10$, $L_1(\langle a, c, b, e \rangle) = 5$, $L_1(\langle a, d, e \rangle) = 1$, $L_1(\langle b, a \rangle) = 0$, $L_1(\langle c \rangle) = 0$, $L_1(\langle\,\rangle) = 0$, etc.

Given an event log $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$, we would like to learn a process model adequately capturing the observed behavior. Figure 2 shows four process models discovered for $L_1 = [\langle a, b, c, e \rangle^{10}, \langle a, c, b, e \rangle^5, \langle a, d, e \rangle]$. The models also show frequencies.

Figure 2(b) shows a Directly-Follows Graph (DFG). The start, end, and five activities are the nodes of the graph. Activities a and e occurred 16 times, b and c occurred 15 times, and d only once. The arcs in Fig. 2(b) show how often an activity is *directly* followed by another activity. For example, a is 10 times directly followed by b, a is 5 times directly followed by c, and a is once directly followed by d. To indicate the start



(a) Event log L_1

(b) Directly-Follows Graph (DFG): M_1

(c) Accepting Petri Net (APN): M_2

(d) Process Tree (PT): M_3

Fig. 2. Three process models learned from event log $L_1 = [\langle a,b,c,e\rangle^{10}, \langle a,c,b,e\rangle^{5}, \langle a,d,e\rangle]$.

and end of cases, we use a start node ▶ and an end node ■. One can view ▶ and ■ as "dummy" activities or states. Although they do not present real activities, they are needed to describe the process adequately. Since all 16 cases start with a, the arc connecting ▶ to a has a frequency of 16. Note that due to the cycles in the DFG, also traces such as $\langle a,b,c,b,c,b,c,b,e\rangle$ are possible according to the DFG (but did not appear in the event log).

Figure 2(c) shows a *Petri net* discovered using the same event log L_1. The transitions (i.e., squares) correspond to the five activities in the event log. The places (i.e., circles) constrain the behavior. The Petri net allows for the three traces in the event log and nothing more. Initially, only transition a is enabled. When a fires (i.e., occurs), a token is consumed from the input place and a token is produced for each of the two output places. As a result, transitions b, c, and d become enabled. If d fires, both tokens are removed and two tokens are produced for the input places of e. If b fires, only one token is consumed and one token is produced. After b fires, c is still enabled, and c will fire to enable e. Transition c can also occur before b, i.e., b and c are concurrent and can happen at the same time or in any order. There is a choice between d and the combination of b and c. The start of the process is modeled by the token in the source place. The end of the process is modeled by the double-bordered sink place.

Also, the *process tree* discovered for event log L_1 shown in Fig. 2(d) allows for the three traces in the event log and nothing more. The root node is a sequence ($\rightarrow$) with three "child nodes": activity a, a choice, and activity e. These nodes are visited 16 times (once for each case). The choice node ($\times$) has two "child nodes": a parallel node $\wedge$ and an activity node e. The parallel node ($\wedge$) has two "child nodes": activity b and activity c. The whole process tree can be represented by the expression $\rightarrow(a, \times(\wedge(b,c),d),e)$. Note that the d node is visited only once. The $\wedge$, b, and c nodes are visited 15 times.

In this example, each node has a unique label allowing us to refer easily. Often a tree has multiple nodes with the same label, e.g., $\rightarrow(a, \times(\rightarrow(a, a), a), a)$ where a appears five times and → two times.

In Fig. 2, we just show example results. In the remainder, we will see how such process models can be learned from event data. The goal of this chapter is not to give a complete survey (see also [10] for a recent survey). Instead, we would like to bring forward the essence of process discovery from event data, and introduce the main principles in an intuitive manner.

The remainder of this chapter is organized as follows. Section 2 presents a baseline approach that computes a Directly-Follows Graph (DFG). This approach is simple and highly scalable, but has many limitations (e.g., producing complex underfitting process models) [2]. In Sect. 3, we elaborate on the challenges of process discovery. Section 4 discusses higher-level representations such as Petri nets (Subsect. 4.1), process trees (Subsect. 4.2), and BPMN (Subsect. 4.3). Section 5 introduces "bottom-up" process discovery using the Alpha algorithm [1,9] as an example. Section 6 introduces "top-down" process discovery using the basic inductive mining algorithm [22–24] as an example. Finally, Sect. 7 concludes the chapter with pointers to other discovery approaches (e.g., using state-based or language-based regions).

2 Directly-Follows Graphs: A Baseline Approach

In this chapter, we present a very simple discovery approach that is supported by most (if not all) process mining tools: Constructing a so-called Directly-Follows Graph (DFG) by simply counting how often one activity is followed by another activity (see Fig. 2(b)). We use this to also introduce filtering techniques to remove infrequent activities, infrequent variants, and infrequent arcs. The more advanced techniques presented later in this chapter build upon the simple notions introduced in this section.

Let us first try to describe the process discovery problem in abstract terms, independent of the selected process modeling notation. Therefore, we describe a model's behavior as a set of traces.

Definition 2 (Process Model). *$\mathcal{U}_M$ is the universe of process models. A process model $M \in \mathcal{U}_M$ defines a set of traces $lang(M) \subseteq \mathcal{U}_{act}{}^*$.*

Examples of process models defined later are DFGs $\mathcal{U}_G \subseteq \mathcal{U}_M$ (Sect. 2.1), accepting Petri nets $\mathcal{U}_{AN} \subseteq \mathcal{U}_M$ (Sect. 4.1), process trees $\mathcal{U}_Q \subseteq \mathcal{U}_M$ (Sect. 4.2), and BPMN models $\mathcal{U}_{BPMN} \subseteq \mathcal{U}_M$ (Sect. 4.3). Consider, for example, the process models M_1 (DFG), M_2 (Petri net), and M_3 (process tree) in Fig. 2. $lang(M_2) = lang(M_3) = \{\langle a, b, c, e\rangle, \langle a, c, b, e\rangle, \langle a, d, e\rangle\}$. $lang(M_1) = \{\langle a, b, e\rangle, \langle a, c, e\rangle, \langle a, d, e\rangle, \ldots, \langle a, b, c, b, c, b, c, e\rangle, \ldots\}$ contains infinitely many traces due to the cycle involving b and c.

The goal of a process discovery algorithm is to produce a model that explains the observed behavior.

Definition 3 (Process Discovery Algorithm). *A process discovery algorithm is a function $disc \in \mathcal{B}(\mathcal{U}_{act}{}^*) \rightarrow \mathcal{U}_M$, i.e., based on a multiset of traces, a model is produced.*

Given an event log L, a process discovery algorithm $disc$ returns a model allowing for the traces $lang(disc(L))$. A discovery algorithm $disc$ guarantees *perfect replay fitness* if for any $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$: $\{\sigma \in L\} \subseteq lang(disc(L))$. We write $\{\sigma \in L\}$ to turn a multiset of traces into a set of traces and make the model and the log comparable. All three models in Fig. 2 have perfect replay fitness (also called perfect recall).

2.1 Directly-Follows Graphs: Basic Concepts

We already informally introduced DFGs, but now we formalize the concepts needed to precisely describe the corresponding discovery algorithm.

Definition 4 (Directly-Follows Graph). *A Directly-Follows Graph (DFG) is a pair* $G = (A, F)$ *where* $A \subseteq \mathcal{U}_{act}$ *is a set of activities and* $F \in \mathcal{B}((A \times A) \cup (\{\blacktriangleright\} \times A) \cup (A \times \{\blacksquare\}) \cup (\{\blacktriangleright\} \times \{\blacksquare\}))$ *is a multiset of arcs.* $\blacktriangleright$ *is the start node and* $\blacksquare$ *is the end node (*$\{\blacktriangleright, \blacksquare\} \cap \mathcal{U}_{act} = \emptyset$*).* $\mathcal{U}_G \subseteq \mathcal{U}_M$ *is the set of all DFGs.*

$\blacktriangleright$ and $\blacksquare$ can be viewed as artificially added activities to clearly indicate the start and end of the process. The nodes of a DFG are $\blacktriangleright$ to denote the beginning, $\blacksquare$ to denote the end, and the activities in set A. Note that $\blacktriangleright \not\in A$ and $\blacksquare \not\in A$ (this is also important in later sections). There are four types of arcs: $(\blacktriangleright, a)$, (a_1, a_2), $(a, \blacksquare)$, and $(\blacktriangleright, \blacksquare)$ (with $a, a_1, a_2 \in A$). $F((\blacktriangleright, a))$ indicates how many cases start with a, $F((a_1, a_2))$ indicates how often activity a_1 is directly followed by activity a_2, $F((a, \blacksquare))$ indicates how many cases end with a, and $F((\blacktriangleright, \blacksquare))$ counts the number of empty cases. In the directly-follows graph, we only consider directly-follows within the same case. For example, $F((a, b)) = (10 \times 0) + (10 \times 0) + (10 \times 1) + (10 \times 2) + (10 \times 3) = 60$ given some event log $[\langle a \rangle^{10}, \langle b \rangle^{10}, \langle a, b \rangle^{10}, \langle a, b, a, b \rangle^{10}, \langle a, b, a, b, a, b \rangle^{10}]$.

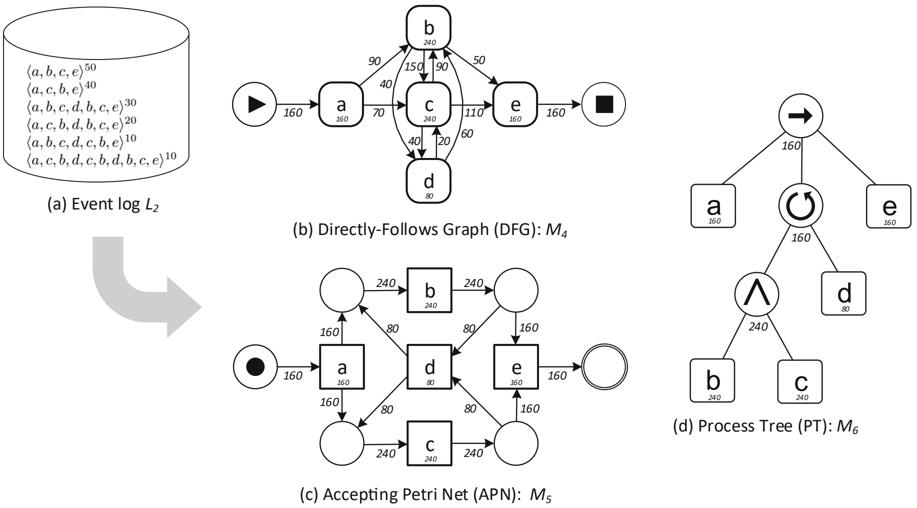
The DFG in Fig. 2(b) can be described as follows: $M_1 = (A, F)$ with $A = \{a, b, c, d, e\}$ and $F = [(\blacktriangleright, a)^{16}, (a, b)^{10}, (a, c)^5, (a, d)^1, (b, c)^{10}, (b, e)^5, (c, b)^5, (c, e)^{10}, (d, e)^1, (e, \blacksquare)^{16}]$.

Figure 3 shows process models discovered for another event log $L_2 = [\langle a, b, c, e \rangle^{50}, \langle a, c, b, e \rangle^{40}, \langle a, b, c, d, b, c, e \rangle^{30}, \langle a, c, b, d, b, c, e \rangle^{20}, \langle a, b, c, d, c, b, e \rangle^{10}, \langle a, c, b, d, c, b, d, b, c, e \rangle^{10}]$. The fact that b, c, and d occur a variable number of times per case suggests that there is a loop. Figure 3(b) shows the corresponding DFG. This DFG can be described as follows: $M_4 = (A, F)$ with $A = \{a, b, c, d, e\}$ and $F = [(\blacktriangleright, a)^{160}, (a, b)^{90}, (a, c)^{70}, (b, c)^{150}, (b, d)^{40}, (b, e)^{50}, (c, b)^{90}, (c, d)^{40}, (c, e)^{110}, (d, b)^{60}, (d, c)^{20}, (e, \blacksquare)^{160}]$.

Definition 5 (Traces of a DFG). *Let* $G = (A, F) \in \mathcal{U}_G$ *be a DFG. The set of possible traces described by* G *is* $lang(G) = \{\langle a_2, a_3, \ldots, a_{n-1} \rangle \mid a_1 = \blacktriangleright \wedge a_n = \blacksquare \wedge \forall_{1 \leq i < n}\ (a_i, a_{i+1}) \in F\}$.

Note that $\blacktriangleright$ and $\blacksquare$ have been added to the DFG to have a clear start and end. However, these "dummy activities" are not part of the language of the DFG.

Consider the DFG M_1 shown in Fig. 2(b): $lang(M_1) = \{\langle a, b, e \rangle, \langle a, c, e \rangle, \langle a, d, e \rangle, \langle a, b, c, e \rangle, \langle a, c, b, e \rangle, \langle a, b, c, b, e \rangle, \langle a, c, b, c, e \rangle, \langle a, b, c, b, c, e \rangle, \ldots\}$. Also the DFG M_4 in Fig. 3(b) has an infinite number of possible traces: $lang(M_4) = \{\langle a, b, e \rangle, \langle a, c, e \rangle, \langle a, b, c, e \rangle, \langle a, c, b, e \rangle, \langle a, b, c, b, e \rangle, \langle a, c, b, c, e \rangle, \langle a, b, d, b, e \rangle, \ldots\}$. Whenever the DFG has a cycle, then the number of possible traces is unbounded.



(a) Event log L_2

(b) Directly-Follows Graph (DFG): M_4

(c) Accepting Petri Net (APN): M_5

(d) Process Tree (PT): M_6

Fig. 3. Three process models learned from event log $L_2 = [\langle a,b,c,e\rangle^{50}, \langle a,c,b,e\rangle^{40}, \langle a,b,c,d,b,c,e\rangle^{30}, \langle a,c,b,d,b,c,e\rangle^{20}, \langle a,b,c,d,c,b,e\rangle^{10}, \langle a,c,b,d,c,b,d,b,c,e\rangle^{10}]$.

2.2 Baseline Discovery Algorithm

Since the event log only contains example traces, it is natural that the discovery algorithm aims to generalize the observed behavior to avoid over-fitting. Therefore, we start with a *baseline discovery algorithm* that ensures that all observed behavior is possible according to the discovered process model. The algorithm used to discover the DFGs in Fig. 2(b) and Fig. 3(b) is defined as follows.

Definition 6 (Baseline Discovery Algorithm). *Let $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ be an event log. $disc_{_{DFG}}(L) = (A, F)$ is the DFG based on L with:*

- $A = \{a \in \sigma \mid \sigma \in L\}$ *and*
- $F = [(\sigma_i, \sigma_{i+1}) \mid \sigma \in L' \ \wedge \ 1 \leq i < |\sigma|]$ *with* $L' = [\langle \blacktriangleright \rangle \cdot \sigma \cdot \langle \blacksquare \rangle \mid \sigma \in L]$.

Note that L, L', and F in Definition 6 are multisets. Each trace in the event log L is extended with the artificially added activities. L' adds $\blacktriangleright$ at the start and $\blacksquare$ at the end of each trace in L. $M_1 = disc_{_{DFG}}(L_1)$ is depicted in Fig. 2(b) and $M_4 = disc_{_{DFG}}(L_2)$ is depicted in Fig. 3(b).

A DFG can be viewed as a first-order Markov model (i.e., the state is determined by the last activity executed). The baseline discovery algorithm (Definition 6) tends to lead to underfitting process models. Whenever two activities are not executed in a fixed order, a loop is introduced.

2.3 Footprints

A DFG can also be represented as a matrix, as shown in Table 1. This is simply a tabular representation of the graph and the arc frequencies, e.g., $F((\blacktriangleright, \blacktriangleright)) = 0$,

$F((\blacktriangleright, a)) = 16$, and $F((c, e)) = 10$. To capture the relations between activities, we can also create a so-called *footprint matrix* [1]. Table 2 shows the footprint matrix for the DFG in Fig. 2(b). Between two activities a_1 and a_2, precisely one of four possible relations holds:

- $a_1 \rightarrow a_2$ (i.e., a_1 is sometimes directly followed by a_2, but a_2 is never directly followed by a_1),
- $a_1 \leftarrow a_2$ (i.e., a_2 is sometimes directly followed by a_1, but a_1 is never directly followed by a_2),
- $a_1 \| a_2$ (i.e., a_1 is sometimes directly followed by a_2 and a_2 is sometimes directly followed by a_1), and
- $a_1 \# a_2$ (i.e., a_1 is never directly followed by a_2 and a_2 is never directly followed by a_1).

Table 1. Matrix representation of the DFG in Fig. 2(b).

	$\blacktriangleright$	a	b	c	d	e	$\blacksquare$
$\blacktriangleright$	0	16	0	0	0	0	0
a	0	0	10	5	1	0	0
b	0	0	0	10	0	5	0
c	0	0	5	0	0	10	0
d	0	0	0	0	0	1	0
e	0	0	0	0	0	0	16
$\blacksquare$	0	0	0	0	0	0	0

Table 2. The footprint of the DFG in Fig. 2(b).

	$\blacktriangleright$	a	b	c	d	e	$\blacksquare$
$\blacktriangleright$	$\#$	$\rightarrow$	$\#$	$\#$	$\#$	$\#$	$\#$
a	$\leftarrow$	$\#$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\#$	$\#$
b	$\#$	$\leftarrow$	$\#$	$\|$	$\#$	$\rightarrow$	$\#$
c	$\#$	$\leftarrow$	$\|$	$\#$	$\#$	$\rightarrow$	$\#$
d	$\#$	$\leftarrow$	$\#$	$\#$	$\#$	$\rightarrow$	$\#$
e	$\#$	$\#$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\#$	$\rightarrow$
$\blacksquare$	$\#$	$\#$	$\#$	$\#$	$\#$	$\leftarrow$	$\#$

Table 2 (based on Fig. 2(b)) shows, for example, that $a \rightarrow b$, $b \leftarrow a$, $b \| c$, and $c \# d$. The creation of the footprint can be formalized as follows.

Definition 7 (Footprint). *Let $G = (A, F) \in \mathcal{U}_G$ be a DFG. G defines a footprint $fp(G) \in (A' \times A') \rightarrow \{\rightarrow, \leftarrow, \|, \#\}$ such that $A' = A \cup \{\blacktriangleright, \blacksquare\}$ and for any $(a_1, a_2) \in A' \times A'$:*

- *$fp(G)((a_1, a_2)) = \rightarrow$ if $(a_1, a_2) \in F$ and $(a_2, a_1) \notin F$,*
- *$fp(G)((a_1, a_2)) = \leftarrow$ if $(a_1, a_2) \notin F$ and $(a_2, a_1) \in F$,*
- *$fp(G)((a_1, a_2)) = \|$ if $(a_1, a_2) \in F$ and $(a_2, a_1) \in F$, and*
- *$fp(G)((a_1, a_2)) = \#$ if $(a_1, a_2) \notin F$ and $(a_2, a_1) \notin F$.*

We write $a_1 \rightarrow_G a_2$ if $fp(G)((a_1, a_2)) = \rightarrow$, $a_1 \#_G a_2$ if $fp(G)((a_1, a_2)) = \#$, etc.

We can also create the footprint of an event log by first applying the baseline discovery algorithm: $fp(L) = fp(disc_{DFG}(L))$. Hence, Table 2 also shows $fp(L_1) = fp(disc_{DFG}(L_1)) = fp(M_1)$. This allows us to write $b \rightarrow_{L_1} e$, $b \|_{L_1} e$, $b \#_{L_1} d$, etc.

2.4 Filtering

Using the baseline discovery algorithm, an activity a appears in the discovered DFG when it occurs at least once and two activities a_1 and a_2 are connected by a directed arc if a_1 is directly followed by a_2 at least once in the log. Often, we do not want to see the process model that captures all behavior. Instead, we would like to see the dominant behavior. For example, we are interested in the most frequent activities and paths. Therefore, we would like to *filter* the event log and model. Here, we consider the three basic types of filtering:

- *Activity-based filtering*: project the event log on a subset of activities (e.g., remove the least frequent activities).
- *Variant-based filtering*: remove selected traces (e.g., only keep the most frequent variants).
- *Arc-based filtering*: remove selected arcs in the DFG (e.g., delete arcs with a frequency lower than a given threshold).

To describe the different types of filtering, we introduce some notations for traces and event logs.

Definition 8 (Frequency and Projection Functions). *Let $L \in \mathcal{B}({\mathcal{U}_{act}}^*)$ be an event log.*

- *$act(L) = \{a \in \sigma \mid \sigma \in L\}$ are the activities in event log L,*
- *$var(L) = \{\sigma \in L\}$ are the trace variants in event log L,*
- *$\#_L^{act}(a) = \sum_{\sigma \in L} |\{i \in \{1, \ldots |\sigma|\} \mid \sigma_i = a\}|$ is the frequency of activity $a \in act(L)$ in event log L,*
- *$\#_L^{var}(\sigma) = L(\sigma)$ is the frequency of variant $\sigma \in var(L)$ in event log L,*
- *for a subset of activities $A \subseteq act(L)$ and trace $\sigma \in L$, we define $\sigma \upharpoonright A$ such that $\langle\rangle \upharpoonright A = \langle\rangle$ and $(\sigma \cdot \langle a \rangle) \upharpoonright A = \sigma \upharpoonright A \cdot \langle a \rangle$ if $a \in A$, and $(\sigma \cdot \langle a \rangle) \upharpoonright A = \sigma \upharpoonright A$ if $a \notin A$,*
- *$L \upharpoonright A = [\sigma \upharpoonright A \mid \sigma \in L]$ is the projection of L on a subset of activities $A \subseteq act(L)$,*
- *$L \Uparrow V = [\sigma \in L \mid \sigma \in V]$ is the projection of L on a subset of trace variants $V \subseteq var(L)$,*

First, we define *activity-based filtering* using a threshold $\tau_{act} \in \mathbb{N} = \{1, 2, 3, \ldots\}$. All activities with a frequency lower than τ_{act} are removed from the event log, but all cases are retained.

Definition 9 (Activity-Based Filtering). *Let $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ be an event log and $\tau_{act} \in \mathbb{N}$. $\mathit{filter}^{act}(L, \tau_{act}) = L{\uparrow}A$ with $A = \{a \in act(L) \mid \#^{act}_L(a) \geq \tau_{act}\}$.*

Again we use $L_1 = [\langle a, b, c, e\rangle^{10}, \langle a, c, b, e\rangle^5, \langle a, d, e\rangle]$ and $L_2 = [\langle a, b, c, e\rangle^{50}, \langle a, c, b, e\rangle^{40}, \langle a, b, c, d, b, c, e\rangle^{30}, \langle a, c, b, d, b, c, e\rangle^{20}, \langle a, b, c, d, c, b, e\rangle^{10}, \langle a, c, b, d, c, b, d, b, c, e\rangle^{10}]$ to illustrate the definition. If $\tau_{act} = 10$, then $\mathit{filter}^{act}(L_1, \tau_{act}) = [\langle a, b, c, e\rangle^{10}, \langle a, c, b, e\rangle^5, \langle a, e\rangle]$ (only activity d is removed). If $\tau_{act} = 16$, then $\mathit{filter}^{act}(L_1, \tau_{act}) = [\langle a, e\rangle^{16}]$ (only activities a and e remain). If $\tau_{act} > 16$, then $\mathit{filter}^{act}(L_1, \tau_{act}) = [\langle\rangle^{16}]$. Note that the number of traces is not affected by activity-based filtering (even when all activities are removed). If $\tau_{act} = 200$, then $\mathit{filter}^{act}(L_2, \tau_{act}) = [\langle b, c\rangle^{50}, \langle c, b\rangle^{40}, \langle b, c, b, c\rangle^{30}, \langle c, b, b, c\rangle^{20}, \langle b, c, c, b\rangle^{10}, \langle c, b, c, b, b, c\rangle^{10}]$ (only activities b and c remain).

Next, we define *variant-based filtering* using a threshold $\tau_{var} \in \mathbb{N}$. All trace variants with a frequency lower than τ_{var} are removed from the event log.

Definition 10 (Variant-Based Filtering). *Let $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ be an event log and $\tau_{var} \in \mathbb{N}$. $\mathit{filter}^{var}(L, \tau_{var}) = L{\Uparrow}V$ with $V = \{\sigma \in var(L) \mid \#^{var}_L(\sigma) \geq \tau_{var}\}$.*

If $\tau_{var} = 5$, then $\mathit{filter}^{var}(L_1, \tau_{var}) = [\langle a, b, c, e\rangle^{10}, \langle a, c, b, e\rangle^5]$. If $\tau_{var} = 10$, then $\mathit{filter}^{var}(L_1, \tau_{var}) = [\langle a, b, c, e\rangle^{10}]$. If $\tau_{var} > 10$, then $\mathit{filter}^{var}(L_1, \tau_{var}) = [\,]$. Note that (unlike activity-based filtering) the number of traces may decrease.

Finally, we define *arc-based filtering* using a threshold $\tau_{arc} \in \mathbb{N}$. Whereas activity-based filtering and variant-based filtering operate on event logs, arc-based filtering modifies the DFG and not the event log used to generate it. All arcs with a frequency lower than τ_{arc} are removed from the graph.

Definition 11 (Arc-Based Filtering). *Let $G = (A, F) \in \mathcal{U}_G$ be a DFG and $\tau_{arc} \in \mathbb{N}$. $\mathit{filter}^{arc}(G, \tau_{arc}) = (A, F')$ with $F' = [(x, y) \in F \mid F((x, y)) \geq \tau_{arc}]$.*

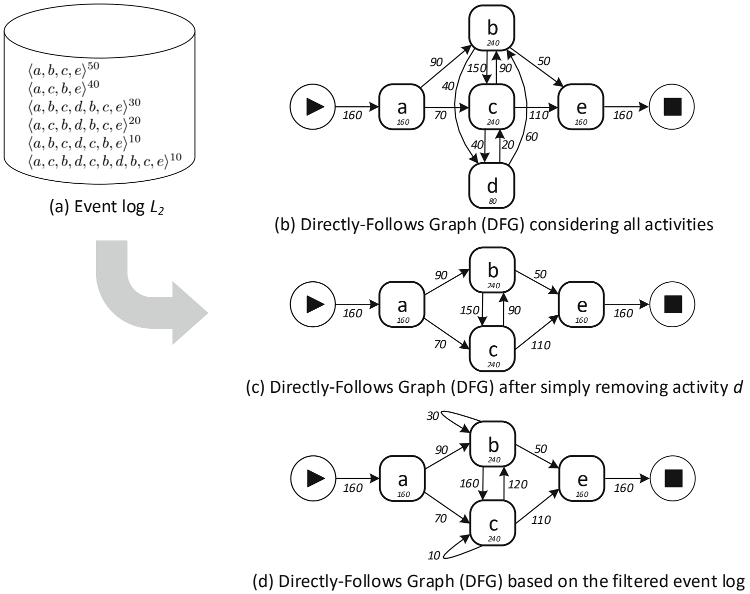
In its basic form τ_{arc} retains all nodes even when they become fully disconnected from the rest. Consider the DFG $M_1 = (A, F)$ in Fig. 2(b) with $A = \{a, b, c, d, e\}$ and $F = [(\blacktriangleright, a)^{16}, (a, b)^{10}, (a, c)^5, (a, d)^1, (b, c)^{10}, (b, e)^5, (c, b)^5, (c, e)^{10}, (d, e)^1, (e, \blacksquare)^{16}]$. If $\tau_{var} = 10$, then $\mathit{filter}^{arc}(M_1, \tau_{arc}) = (A, F')$ with $F' = [(\blacktriangleright, a)^{16}, (a, b)^{10}, (b, c)^{10}, (c, e)^{10}, (e, \blacksquare)^{16}]$. If $\tau_{var} = 15$, then $\mathit{filter}^{arc}(M_1, \tau_{arc}) = (A, F'')$ with $F'' = [(\blacktriangleright, a)^{16}, (e, \blacksquare)^{16}]$. Note that the DFG is no longer connected.

The three types of filtering can be combined. Because arc-based filtering operates on the DFG, it should be done last. It is also better to conduct activity-based filtering before variant-based filtering. There are several reasons for this. The number of traces is affected by variant-based filtering. Moreover, activity-based filtering may lead to variants with a higher frequency. Consider L_1 with $\tau_{act} = 16$ and $\tau_{var} = 10$. If we first apply variant-based filtering, one variant remains after the first step and none of the activities is frequent enough to be retained in the second step: $\mathit{filter}^{act}(\mathit{filter}^{var}(L_1, \tau_{var}), \tau_{act}) = [\langle\rangle^{10}]$. If we first apply activity-based filtering, then the two most frequent activities are retained and all 16 traces are considered in the second step: $\mathit{filter}^{var}(\mathit{filter}^{act}(L_1, \tau_{act}), \tau_{var}) = [\langle a, e\rangle^{16}]$. For L_2 with $\tau_{act} = 200$ and $\tau_{var} = 40$, we find that $\mathit{filter}^{act}(\mathit{filter}^{var}(L_2, \tau_{var}), \tau_{act}) = [\langle\rangle^{90}]$ and $\mathit{filter}^{var}(\mathit{filter}^{act}(L_2, \tau_{act}), \tau_{var}) = [\langle b, c\rangle^{50}, \langle c, b\rangle^{40}]$.

These examples show that the order of filtering matters. We propose a *refined baseline discovery algorithm using filtering*. The algorithm first applies activity-based filtering followed by variant-based filtering. Then the original baseline algorithm is applied to the resulting event log to get a DFG (see Definition 6). Finally, arc-based filtering is used to prune the DFG.

Definition 12 (Baseline Discovery Algorithm Using Filtering). *Let $L \in \mathcal{B}({\mathcal{U}_{act}}^*)$ be an event log. Given the thresholds $\tau_{act} \in \mathbb{N}$, $\tau_{var} \in \mathbb{N}$, and $\tau_{arc} \in \mathbb{N}$:* $disc_{DFG}^{\tau_{act},\tau_{var},\tau_{arc}}(L) = \mathit{filter}^{arc}(disc_{DFG}(\mathit{filter}^{var}(\mathit{filter}^{act}(L,\tau_{act}),\tau_{var})),\tau_{arc})$.

$disc_{DFG}^{\tau_{act},\tau_{var},\tau_{arc}}(L)$ returns a DFG using the three filtering steps. Only the last filtering step is specific for DFGs. Activity-based filtering and variant-based filtering can be used in conjunction with any discovery technique, because they produce filtered event logs. The footprint notion can also be extended to include these two types of filtering: $fp^{\tau_{act},\tau_{var}}(L) = fp(disc_{DFG}(\mathit{filter}^{var}(\mathit{filter}^{act}(L,\tau_{act}),\tau_{var})))$ is the footprint matrix considering only frequent activities and variants.


$\langle a,b,c,e\rangle^{50}$
$\langle a,c,b,e\rangle^{40}$
$\langle a,b,c,d,b,c,e\rangle^{30}$
$\langle a,c,b,d,b,c,e\rangle^{20}$
$\langle a,b,c,d,c,b,e\rangle^{10}$
$\langle a,c,b,d,c,b,d,b,c,e\rangle^{10}$
a 160
b 240
c 240
d 80
e 160
160
90
70
40
150
90
50
110
60
20
120
30
10

(a) Event log L_2

(b) Directly-Follows Graph (DFG) considering all activities

(c) Directly-Follows Graph (DFG) after simply removing activity *d*

(d) Directly-Follows Graph (DFG) based on the filtered event log

Fig. 4. Three DFGs learned from event log $L_2 = [\langle a,b,c,e\rangle^{50}, \langle a,c,b,e\rangle^{40}, \langle a,b,c,d,b,c,e\rangle^{30}, \langle a,c,b,d,b,c,e\rangle^{20}, \langle a,b,c,d,c,b,e\rangle^{10}, \langle a,c,b,d,c,b,d,b,c,e\rangle^{10}]$: (b) the original DFG considering all activities, (c) the problematic DFG obtained by simply removing activity d from the graph, and (d) the desired DFG obtained by removing activity d from the event log first.

Most process mining tools provide sliders to interactively set one or more thresholds. This makes it easy to seamlessly simplify the discovered DFG. However, it is vital that the user understands the different filtering approaches. Therefore, we highlight the following risks.

- The ordering of filters may greatly impact the result. As shown before: $\mathit{filter}^{var}(\mathit{filter}^{act}(L, \tau_{act}), \tau_{var}) \neq \mathit{filter}^{act}(\mathit{filter}^{var}(L, \tau_{var}), \tau_{act})$. If a tool provides multiple sliders, it is important to understand how these interact and what was left out.
- Applying projections to event logs is computationally expensive. Therefore, process mining tools may provide shortcuts that operate directly on the DFG without filtering the event log. Consider, for example, Fig. 4 showing (a) the event log and (b) the original DFG without filtering. Activity d has the lowest frequency. Simply removing node d from the graph leads to interpretation problems. Figure 4(c) shows the problem, e.g., b occurs 240 times but the frequencies of the input arcs add up to $90 + 90 = 180$ and the frequencies of output arcs add up to $50 + 150 = 200$. If we apply activity-based filtering using Definition 9, we obtain the DFG in Fig. 4(d). Now we see the loops involving b and c. Moreover, the frequencies of the input arcs of b add up to $90+120+30 = 240$ and the frequencies of output arcs also add up to $50+160+30 = 240$. Clearly, this is the DFG one would like to see after abstracting from d.
- Using activity-based filtering and variant-based filtering as defined in this section yields models where the frequency of a node matches the sum of the frequencies of the input arcs and the sum of the frequencies of the output arcs. As long as the resulting event log is not empty, the graph is connected and all activities are on a path from start to end. This leads to models that are easy to interpret. Arc-based filtering may lead to models that have disconnected parts and frequencies do not add up as expected (similar to the problems in Fig. 4(c)). Therefore, arc-based filtering should be applied with care.
- The above risks are not limited to control-flow (e.g., connectedness of the graph and incorrect frequencies). When adding *timing information* (e.g., the average time between two activities), the results are highly affected by filtering. Process mining tools using shortcuts that operate directly on the DFG without filtering the event log, quickly lead to misleading performance diagnostics [2].

2.5 A Larger Example

To further illustrate the concepts, we now consider a slightly larger event log $L_3 = [\langle ie, cu, lt, xr, fe\rangle^{285}, \langle ie, cu, lt, ct, fe\rangle^{260}, \langle ie, cu, ct, lt, fe\rangle^{139}, \langle ie, lt, cu, xr, fe\rangle^{137}, \langle ie, lt, cu, ct, fe\rangle^{124}, \langle ie, cu, xr, lt, fe\rangle^{113}, \langle ie, xr, cu, lt, fe\rangle^{72}, \langle ie, ct, cu, xr, fe\rangle^{72}, \langle ie, cu, om, am, cu, lt, xr, fe\rangle^{29}, \langle ie, cu, om, am, cu, lt, ct, fe\rangle^{28}, \ldots]$. We use the following abbreviations: ie = initial examination, xr = X-ray, ct = CT scan, cu = checkup, om = order medicine, am = administer medicine, lt = lab tests, and fe = final examination. The event log contains 11761 events corresponding to 1856 cases. Each case represents the treatment of a patient. There are 187 trace variants and 8 unique activities. For example, $\langle ie, cu, lt, xr, fe\rangle$ is the most frequent variant, i.e., 285 patients first get an initial examination (ie), followed by a checkup (cu), lab tests (lt), X-ray (xr), and a final examination (fe).

Figure 5 shows the DFG for L_3 using the baseline discovery algorithm described in Definition 6. The DFG was produced by ProM's "Mine with Directly Follows visual Miner". Using a slider, it is possible to remove infrequent activities. Figure 6 shows the DFG $disc_{DFG}(\mathit{filter}^{act}(L_3, \tau_{act}))$ with the activity threshold τ_{act} set to 1000, i.e.,

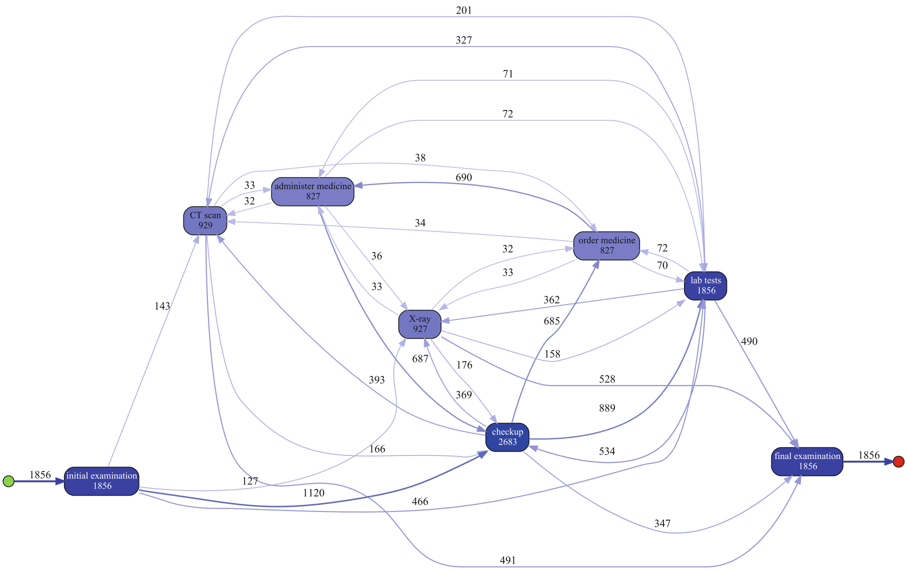


Fig. 5. The discovered DFG $disc_{DFG}(L_3)$ generated by ProM.

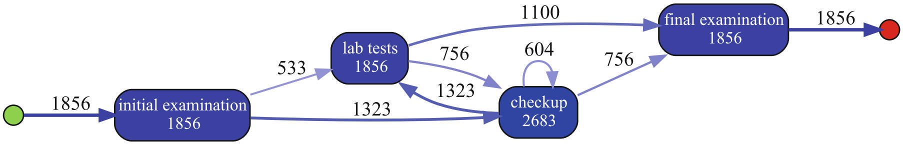


Fig. 6. The DFG $disc_{DFG}(\mathit{filter}^{act}(L_3, \tau_{act}))$ generated by ProM using $\tau_{act} = 1000$.

all activities with a frequency of less than 1000 are removed from the event log using projection. In the resulting DFG, four of the eight activities remain.

The discovery of DFGs (as defined in this section) is supported by almost all process mining tools. Figure 7 shows the DFGs discovered using the Celonis EMS using the same settings as used in ProM. Although the layout is different, the Celonis-based DFG in Fig. 7 (left) is identical to the ProM-based DFG in Fig. 5. The DFG in Fig. 7 (right) is identical to the DFG in Fig. 6.

Figure 8 shows variant-based filtering using the Celonis “Variant Explorer”. The six most frequent variants are selected. These are the variants that have a frequency above 100, i.e., the depicted DFG is $disc_{DFG}(\mathit{filter}^{var}(L_3, \tau_{var}))$ with $\tau_{var} = 100$. There are 1856 cases distributed over 197 variants. The top six variants (i.e., 3% of all variants) cover 1058 cases (i.e., 57%). We also computed the DFG $disc_{DFG}(\mathit{filter}^{var}(L_3, \tau_{var}))$ with $\tau_{var} = 10$. There are 22 variants meeting this lower threshold (i.e., 11% of all variants) covering 1483 cases (i.e., 80%). Most event logs follow such a *Pareto distribution*, i.e., a small fraction of variants explains most of the cases observed. This is also referred to as the “80/20 rule”, although the numbers 80 and 20 are arbitrary. For our

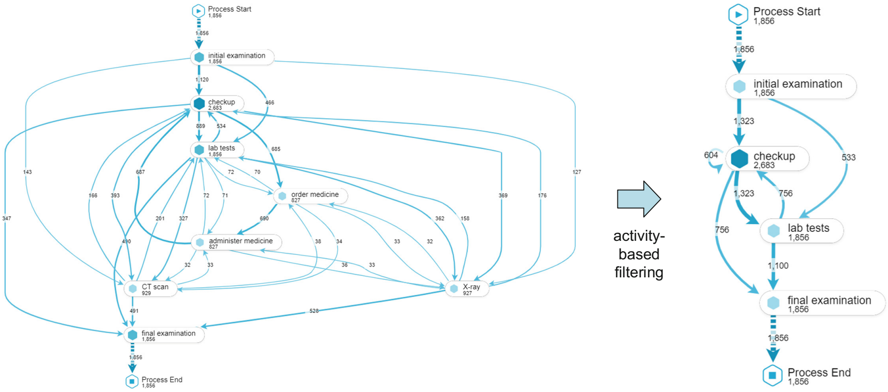


Fig. 7. The discovered DFG in Celonis before and after activity-based filtering, i.e., $disc_{DFG}(L_3)$ (left) and $disc_{DFG}(\mathit{filter}^{act}(L_3, \tau_{act}))$ with $\tau_{act} = 1000$ (right).

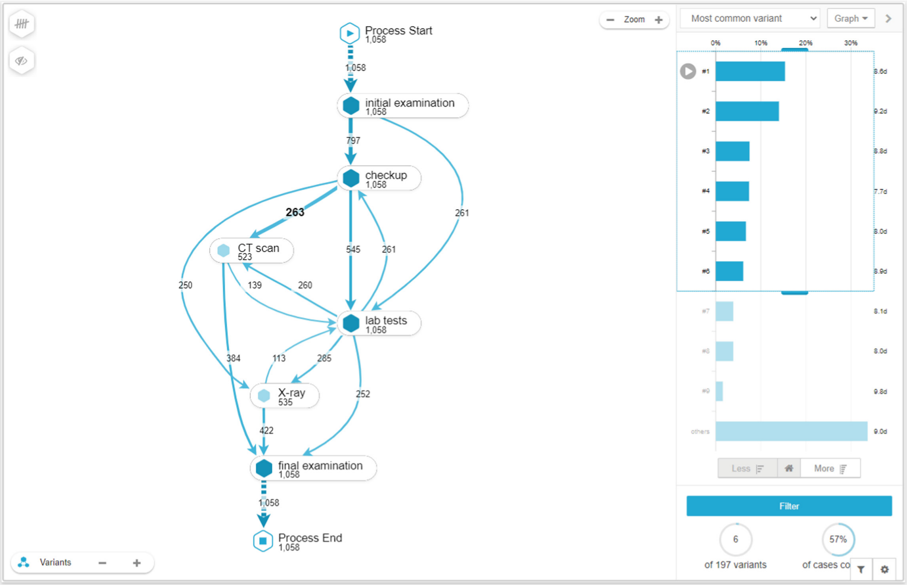


Fig. 8. A discovered DFG in Celonis using variant-based filtering: $disc_{DFG}(\mathit{filter}^{var}(L_3, \tau_{var}))$ with $\tau_{var} = 100$. There are six variants having a frequency above 100. These cover 57% of all cases, but only 3% of all variants.

example event log L_3, we could state that it satisfies the “80/11 rule” (but also the “57/3 rule”, “84/16 rule”, etc.).

If the distribution of cases over variants does not follow a Pareto distribution, then it is best to first apply activity-based filtering. If we project L_3 onto the top four most frequent activities, only 20 variants remain. The most frequent variant explains already 51% of all cases. The DFG $disc_{DFG}(\mathit{filter}^{var}(\mathit{filter}^{act}(L, \tau_{act}), \tau_{var}))$ with $\tau_{act} = 1000$ and $\tau_{var} = 100$ combines the activity-based filter used in Fig. 7 and the variant-based filter used in Fig. 8. The resulting DFG (not shown) explains 1672 of the 1856 cases (90%) and 7065 of 11761 events (60%) using only five variants.

The above examples show that, using filtering, it is possible to separate the normal (i.e., frequent) from the exceptional (i.e., infrequent) behavior. This is vital in the context of process discovery and can be combined with the later bottom-up and top-down discovery approaches.

3 Challenges

After introducing a baseline discovery algorithm and various filtering approaches, it is possible to better explain why process discovery is so challenging. In Definition 3, we stated that a process discovery algorithm is a function $disc \in \mathcal{B}({\mathcal{U}_{act}}^*) \rightarrow \mathcal{U}_M$, i.e., based on a multiset of traces L, a process model $M = disc(L)$ allowing for $lang(M) \subseteq {\mathcal{U}_{act}}^*$ is produced.

The first challenge is that *the discovered process model may serve different goals*. Should the model summarize past behavior, or is the model used for predictions and recommendations? Also, should the process model be easy to read and understand by end-users? Answers to these questions are needed to address the trade-offs in process discovery. We already mentioned that most event logs follow a Pareto distribution. Hence, the process model can focus on the dominant behavior or also include exceptional behavior.

The second challenge is that different process model representations can be used. These may or may not be able to capture certain behaviors. This is the so-called *representational bias* of process discovery. Consider, for example, event log $L = [\langle a, b, c, d\rangle^{1000}, \langle a, c, b, d\rangle^{1000}]$. There is no DFG that is able to adequately describe this behavior. The DFG will always need to introduce a loop involving b and c. Another example is $L = [\langle a, b, c\rangle^{1000}, \langle a, c\rangle^{1000}]$. It is easy to create a DFG describing this behavior. However, when representing this as a Petri net or process tree, it is vital that one can use so-called silent activities (to skip b) or duplicate activities (to have a c activity following a and another c activity following b).

Another challenge is that the event log contains just *example behavior*. Most event logs have a Pareto distribution. Typically, a few trace variants are frequent and many trace variants are infrequent. Actually, there are often trace variants that are unique (i.e., occur only once). If one observes the process longer, new variants will appear. Conversely, if one observes the process in a different period, some variants may no longer appear. An event log is a *sample* and should be treated as such. Just like in statistics, the goal is to use the sample to say something about the whole population (here, the process). For example, when throwing a dice ten times, one may have the following sequence observations $\sigma = \langle 4, 5, 2, 3, 6, 5, 4, 1, 2, 3\rangle$. If we do not know that two subsequent throws are independent, the expected value is 3.5, the minimum is 1, the maximum is 6, and the probabilities of all six values are equal, then what can be concluded from the sample σ? We could conclude that even numbers are always followed by odd numbers. Real-life processes have many more behaviors, and the observed sample rarely covers all possibilities.

Although processes are *stochastic*, most process discovery techniques aim to discover process models that are "binary", i.e., a trace is possible or not. This complicates analysis. Another challenge is that event logs *do not contain negative examples*. Process discovery can be seen as a classification problem: A trace σ is possible ($\sigma \in lang(M)$) or not ($\sigma \notin lang(M)$). In real applications, we never witness traces that are impossible. The event log only contains positive examples. If we also want to incorporate infrequent behavior in the discovered model, we may require $var(L) \subseteq lang(M)$. However, we cannot assume the reverse $lang(M) \subseteq var(L)$. For example, loops in models would be impossible, and for concurrent processes we would need a factorial number of cases.

Related to the above are the challenges imposed by *concept drift*. The behavior of the process that we are trying to discover may change over time in unforeseen ways. Certain traces may increase or decrease in likelihood. New trace variants may emerge while other variants no longer occur. Since process models already describe dynamic behavior, concept drift introduces second-order dynamics. Various techniques for concept-drift detection have been developed. However, this for sure complicates process discovery. If we cannot assume that the process itself is in steady-state, then what is the process we are trying to discover? Do we want to have a process model describing the past week or the past year?

Next to concept drift, there are the usual *data quality problems* [1]. Events may have been logged incorrectly and attributes may be missing or are imprecise. In some applications it may be difficult to *correlate events* and group them into cases. There may be different identifiers used for the same case and events may be shared by different cases. Since process discovery depends on the ordering of events in the event log, *high-quality timestamps* are important. However, the timestamp resolution may be too low (e.g., just a date) and different source systems may use different timestamp granularities or formats. Often the day and the month are swapped, e.g., 8/7/2022 is entered as 7/8/2022.

It is important to distinguish the *evaluation of a process discovery algorithm* $disc \in \mathcal{B}(\mathcal{U}_{act}{}^*) \rightarrow \mathcal{U}_M$ from the *evaluation of a specific process model* M in the context of a *specific event log* L. To evaluate a process discovery algorithm $disc$, one can use cross-validation, i.e., split an event log into a training part and an evaluation part. The process model is trained using the *training log* and evaluated using the *evaluation log*. Ideally, the evaluation log has both positive and negative examples. This is unrealistic in real settings. However, it is possible to create synthetic event data with positive and negative cases using, for example, simulation. If we assume that the *evaluation log* is a multiset of positive traces $L^+_{eval} \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ and a multiset of negative traces $L^-_{eval} \in \mathcal{B}(\mathcal{U}_{act}{}^*)$, then evaluation is simple. Let $M = disc(L^+_{train})$ be the discovered process model using only positive training examples. Now, we can use standard notions such as $recall = \frac{\left|[\sigma \in L^+_{eval} \mid \sigma \in lang(M)]\right|}{\left|L^+_{eval}\right|}$ and $precision = \frac{\left|[\sigma \in L^-_{eval} \mid \sigma \notin lang(M)]\right|}{\left|L^-_{eval}\right|}$ using the evaluation log. Recall is high when most of the positive traces in the evaluation log are indeed possible according to the process model. Precision is high when most of the negative traces in the evaluation log are indeed not possible according to the process model.

Unfortunately, the above view is very naïve considering process discovery in practical settings. We *cannot* assume negative examples when evaluating a *specific* model M

in the context of a *specific* event log L observed in *reality*. Splitting L into a training log and an evaluation log does not make any sense since the model is given and we want to use the whole event log.

In spite of these problems, there is consensus in the process mining community that there are the following four *quality dimensions* to evaluate a process model M in the context of an event log L with observed behavior [1].

- *Recall*, also called (replay) fitness, aims to quantify the fraction of observed behavior that is allowed by the model.
- *Precision* aims to quantify the fraction of behavior allowed by the model that was actually observed (i.e., avoids "underfitting" the event data).
- *Generalization* aims to quantify the probability that new unseen cases will fit the model (i.e., avoids "overfitting" the event data).
- *Simplicity* refers to Occam's Razor and can be made operational by quantifying the complexity of the model (number of nodes, number of arcs, understandability, etc.).

There exist various measures for recall. The simplest one computes the fraction of traces in event log L possible according to the process model M. It is also possible to define such a notion at the level of events. There are many simplicity notions. These do not depend on the behavior of the model, but measure its understandability and complexity. Most challenging are the notions of precision and generalization. Also, these notions can be quantified, but there is less consensus on what they should measure. The goal is to strike a balance between precision (avoiding "underfitting" the sample event data) and generalization (avoiding "overfitting" the sample event data). A detailed discussion is outside the scope of this chapter. Therefore, we refer to [1,4,15,31] for further information.

4 Process Modeling Notations

We have formalized the notion of an event log and the behavior represented by a DFG. Now we focus on higher-level process models able to model sequences, choices, loops, and concurrency. We formalize Petri nets and process trees and provide an informal introduction to a relevant subset of BPMN.

4.1 Labeled Accepting Petri Nets

Figures 2(c) and 3(c) already showed example Petri nets. Since their inception in 1962 [28], Petri nets have been used in a wide variety of application domains. Petri nets were the first formalism to capture concurrency in a systematic manner. See [17,18] for a more extensive introduction. Other notations such as Business Process Model and Notation (BPMN), Event-driven Process Chains (EPCs), and UML activity diagrams all build on Petri nets and have semantics involving "playing the token game". For process mining, we need to use the so-called *labeled accepting Petri nets*. These are standard

Petri nets where transitions are labeled to refer to activities in the event log and, next to an initial marking, these nets also have a final marking. The behavior described by such nets are all the "paths" leading from the initial state to the final state. We explain these concepts step-by-step.

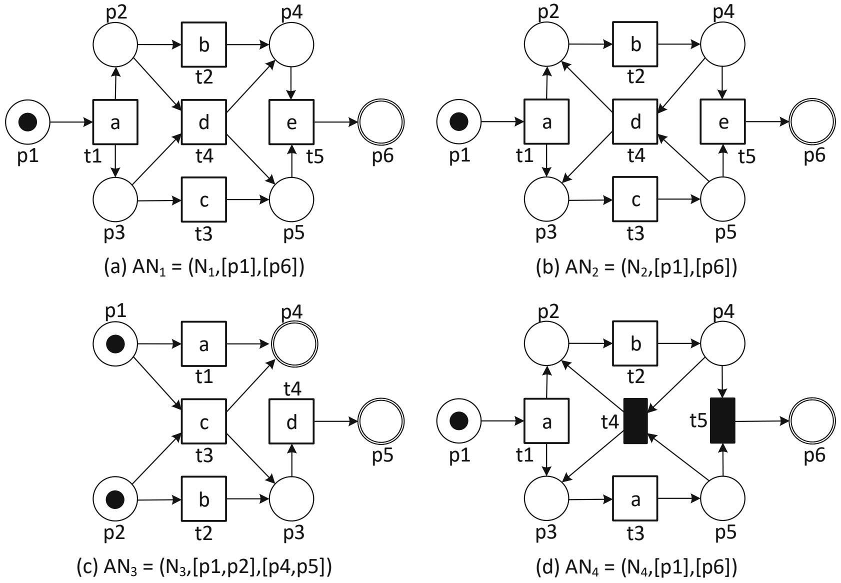


Fig. 9. Four accepting Petri nets: (a) $AN_1 = (N_1, [p1], [p6])$, (b) $AN_2 = (N_2, [p1], [p6])$, (c) $AN_3 = (N_3, [p1, p2], [p4, p5])$, and (d) $AN_4 = (N_4, [p1], [p6])$. AN_1 was discovered for L_1 (see Fig. 2(c)) and AN_2 was discovered for L_2 (see Fig. 3(c)).

States in Petri nets are called *markings* that mark certain *places* (represented by circles) with *tokens* (represented by black dots). *Transitions* (represented by squares) are the active components able to move the Petri net from one marking to another marking. Transitions may have a label referring to the corresponding activity. There may be multiple transitions that refer to the same activity and there may be transitions without an activity label. The former is needed if the same activity can occur at multiple stages in the process. The latter is needed if activities can be skipped. Later we will give examples illustrating the importance of the labeling function in the context of process mining.

Definition 13 (Labeled Petri Net). *A labeled Petri net is a tuple $N = (P, T, F, l)$ with P the set of places, T the set of transitions, $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ the flow relation, and $l \in T \not\rightarrow \mathcal{U}_{act}$ a labeling function. We write $l(t) = \tau$ if $t \in T \setminus dom(l)$ (i.e., t is a silent transition that cannot be observed).*

Figure 9 shows four accepting Petri nets. The first two were discovered for the event logs L_1 and L_2 used to introduce DFGs. Figure 9(a) shows the labeled Petri net $N_1 = (P_1, T_1, F_1, l_1)$ with $P_1 = \{p1, p2, p3, p4, p5, p6\}$ (six places),

$T_1 = \{t1, t2, t3, t4, t5\}$ (five transitions), $F_1 = \{(p1, t1), (t1, p2), (t1, p3), \ldots, (t5, p6)\}$ (fourteen arcs), and $l_1 = \{(t1, a), (t2, b), (t3, c), (t4, d), (t5, e)\}$ (labeling function).

As mentioned, there may be multiple transitions with the same label and there may be transitions that have no label (called "silent transitions"). This is illustrated by $N_4 = (P_4, T_4, F_4, l_4)$ in Fig. 9(d) with $l_4 = \{(t1, a), (t2, b), (t3, a)\}$. Note that $dom(l_4) = \{t1, t2, t3\}$ does not include $t4$ and $t5$ which are silent. This is denoted by the two black rectangles in Fig. 9(d). Also note that $l_4(t1) = l_4(t3) = a$, i.e., $t1$ and $t3$ refer to the same activity.

Since a place may have multiple tokens, markings are represented by multisets. Transitions may have input and output places. For example, $t1$ in Fig. 9(a) has one input place and two output places. A transition is called *enabled* if each of the input places has a token. An enabled transition may *fire* (i.e., occur), thereby consuming a token from each input place and producing a token for each output place.

An *accepting Petri net* has an initial marking $M_{init} \in \mathcal{B}(P)$ and a final marking $M_{final} \in \mathcal{B}(P)$. The accepting Petri nets $AN_1 = (N_1, [p1], [p6])$, $AN_2 = (N_2, [p1], [p6])$, and $AN_4 = (N_4, [p1], [p6])$ in Fig. 9 have the same initial and final marking. $AN_3 = (N_3, [p1, p2], [p4, p5])$ in Fig. 9(c) has an initial marking $M_{init} = [p1, p2]$ (denoted by the black tokens) and a final marking $M_{final} = [p4, p5]$ (denoted by the double-bordered places).

Definition 14 (Accepting Petri Net). *An accepting Petri net is a triplet $AN = (N, M_{init}, M_{final})$ where $N = (P, T, F, l)$ is a labeled Petri net, $M_{init} \in \mathcal{B}(P)$ is the initial marking, and $M_{final} \in \mathcal{B}(P)$ is the final marking. $\mathcal{U}_{AN} \subseteq \mathcal{U}_M$ is the set of accepting Petri nets.*

An accepting Petri net starts in the initial marking and may move from one marking to the next by firing enabled transitions. Consider, for example, $AN_3 = (N_3, [p1, p2], [p4, p5])$ in Fig. 9(c). Initially, three transitions are enabled in $[p1, p2]$: $t1$, $t2$, and $t3$. Firing $t1$ results in marking $[p2, p4]$, firing $t2$ results in marking $[p1, p3]$, and firing $t3$ results in marking $[p3, p4]$. If $t1$ fires (i.e., activity a occurs), then $t1$ and $t3$ are no longer enabled and only $t2$ remains enabled. If $t2$ fires in $[p2, p4]$, we reach the marking $[p3, p4]$. In this marking, only $t4$ is enabled. Firing $t4$ results in the marking $[p4, p5]$. This is also the final marking of AN_3. A *firing sequence* is a sequence of transition occurrences obtained by firing enabled transitions and moving from one marking to the next. A *complete* firing sequence starts in the initial marking and ends in the final marking. AN_3 has four possible complete firing sequences: $\langle t1, t2, t4\rangle$, $\langle t2, t1, t4\rangle$, $\langle t2, t4, t1\rangle$, and $\langle t3, t4\rangle$.

Definition 15 (Complete Firing Sequences). *Let $AN = (N, M_{init}, M_{final}) \in \mathcal{U}_{AN}$ be an accepting Petri net with $N = (P, T, F, l)$. $cfs(AN) \subseteq T^*$ is the set of complete firing sequences of AN, i.e., all firing sequences starting in the initial marking M_{init} and ending in the final marking M_{final}.*

$cfs(AN_1) = \{\langle t1, t2, t3, t5\rangle, \langle t1, t3, t2, t5\rangle, \langle t1, t4, t5\rangle\}$ and $cfs(AN_3) = \{\langle t1, t2, t4\rangle, \langle t2, t1, t4\rangle, \langle t2, t4, t1\rangle, \langle t3, t4\rangle\}$. Note that $cfs(AN_2)$ and $cfs(AN_4)$ contain an infinite number of complete firing sequences due to the loop involving $t4$.

As stated in Definition 2, a process model defines a set of traces. Earlier, we defined $lang(G) \subseteq \mathcal{U}_{act}{}^*$ for a DFG $G = (A, F)$. Now we need to define $lang(AN) \subseteq \mathcal{U}_{act}{}^*$ for an accepting Petri net $AN = (N, M_{init}, M_{final})$. For this purpose, we need to be able to apply the labeling function l to firing sequences. Let $\sigma \in T^*$ be a firing sequence and $l \in T \not\rightarrow \mathcal{U}_{act}$ a labeling function. Function l is generalized to sequences, i.e., transitions are replaced by their labels and are dropped if they do not have a label. Formally, $l(\langle\rangle) = \langle\rangle$, $l(\sigma \cdot \langle t\rangle) = l(\sigma) \cdot \langle l(t)\rangle$ if $t \in dom(l)$, and $l(\sigma \cdot \langle t\rangle) = l(\sigma)$ if $t \notin dom(l)$. Consider, for example, the complete firing sequence $\sigma = \langle t1, t2, t3, t4, t3, t2, t5\rangle \in cfs(AN_4)$ of the accepting Petri net in Fig. 9(d). $l(\sigma) = \langle a, b, a, a, b\rangle$, i.e., $t1$, $t2$, and $t3$ are mapped to the corresponding labels, and $t4$ and $t5$ are dropped.

Definition 16 (Traces of an Accepting Petri Net). *Let $AN = (N, M_{init}, M_{final}) \in \mathcal{U}_{AN}$ be an accepting Petri net. $lang(AN) = \{l(\sigma) \mid \sigma \in cfs(AN)\}$ are the traces possible according to AN.*

Now we can reason about the traces of the four accepting in Fig. 9. $lang(AN_1) = \{\langle a, b, c, e\rangle, \langle a, c, b, e\rangle, \langle a, d, e\rangle\}$. $lang(AN_2) = \{\langle a, b, c, e\rangle, \langle a, c, b, e\rangle, \langle a, b, c, d, b, c, e\rangle, \langle a, c, b, d, b, c, e\rangle, \ldots, \langle a, c, b, d, b, c, d, c, b, d, c, b, e\rangle, \ldots\}$. $lang(AN_3) = \{\langle a, b, d\rangle, \langle b, a, d\rangle, \langle b, d, a\rangle, \langle c, d\rangle\}$. $lang(AN_4) = \{\langle a, b, a\rangle, \langle a, a, b\rangle, \langle a, b, a, b, a\rangle, \langle a, a, b, b, a\rangle, \ldots, \langle a, a, b, b, a, a, b, a, b\rangle, \ldots\}$.

It is important to note the consequences of restricting $lang(AN)$ to the behavior of complete firing sequences. If AN has *livelocks* of *deadlocks*, then these are *not* considered to be part of the language. If we remove the arc from $p4$ to $t4$ in AN_2, then $lang(AN_2) = \{\langle a, b, c, e\rangle, \langle a, c, b, e\rangle\}$, because there are no complete firing sequences involving $t4$.

In literature, Petri nets are normally not equipped with a *labeling function* and a *final marking*. However, both the labeling function l and a defined final marking M_{final} are vital in the context of process mining. The final marking allows us to reason about complete firing sequences, just like traces in an event log have a clear ending. If we would consider ordinary Petri nets rather than accepting Petri nets, the language would also include all prefixes. This would make it impossible to describe the behavior found in an event log such as $L = [\langle a, b, c\rangle^{1000}]$, because the corresponding Petri net would also allow for traces $\langle a, b\rangle$, $\langle a\rangle$, and $\langle\rangle$.

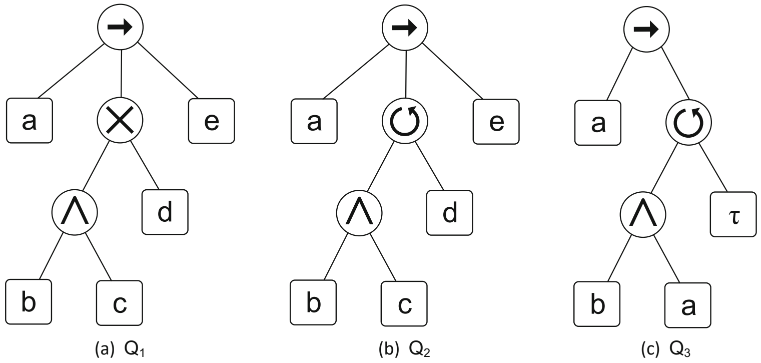
The labeling function $l \in T \not\rightarrow \mathcal{U}_{act}$ also greatly improves expressiveness. The alternative would be that transitions are uniquely identified by activities, i.e., $T \subseteq \mathcal{U}_{act}$. However, this would make it impossible to describe many behaviors seen in event logs. Consider, for example, an event log such as $L = [\langle a, b, c\rangle^{1000}, \langle a, c\rangle^{1000}]$ where b can be skipped. It is easy to model this behavior using a silent transition to skip b or by using two transitions with a c label. Although it is trivial to create a DFG G such that $lang(G) = \{\langle a, b, c\rangle, \langle a, c\rangle\}$ (simply apply the baseline algorithm described in Definition 6), it is impossible to create an accepting Petri net AN with $lang(AN) = \{\langle a, b, c\rangle, \langle a, c\rangle\}$ without using a labeling function allowing for silent or duplicate transitions.

4.2 Process Trees

The two process trees discovered for event logs L_1 and L_2 (see Fig. 2(c) and Fig. 3(c)) are depicted as $Q_1 = \rightarrow(a, \times(\wedge(b, c), d), e)$ and $Q_2 = \rightarrow(a, \circlearrowleft(\wedge(b, c), d), e)$ in Fig. 10. Their language is the same as AN_1 and AN_2 in Fig. 9.

Process trees are not commonly used as a modeling language. However, state-of-the-art process discovery techniques use process trees as an internal representation. The behavior of process trees can be visualized using Petri nets, BPMN, UML activity diagrams, EPCs, etc. However, they also have their own graphical representation, as shown in Fig. 10.

The main reason for using process trees is that they have a *hierarchical structure* and are *sound by construction*. This does not hold for other notations such as Petri nets and BPMN. For example, if we remove the arc $(t4, p2)$ in AN_2 shown in Fig. 9(b), then the process may *deadlock*. The process gets stuck in marking $[p5]$ making it impossible to reach the final marking. If we remove the arc $(p4, t4)$ in AN_2, then the process may *livelock*. It is possible to put an arbitrary number of tokens in $p2$ and $p4$, but after the occurrence of d it is impossible to reach the final marking. If both arcs are removed, the accepting Petri net is again sound (i.e., free of anomalies such as deadlocks and livelocks). When discovering process model constructs locally, these potential soundness problems are difficult to handle (see [6] for more details on analyzing soundness of process models). Therefore, a range of inductive mining techniques has been developed using process trees that are sound by construction [22–24].



(a) Q_1 (b) Q_2 (c) Q_3

Fig. 10. Three process trees: (a) $Q_1 = \rightarrow(a, \times(\wedge(b, c), d), e)$, (b) $Q_2 = \rightarrow(a, \circlearrowleft(\wedge(b, c), d), e)$, and (c) $Q_3 = \rightarrow(a, \circlearrowleft(\wedge(b, a), \tau))$.

A process tree is a tree-like structure with one root node. The leaf nodes correspond to activities (including the silent activity τ, which is similar to a silent transition in Petri nets). Four types of operators can be used in a process tree: → (sequential composition), × (exclusive choice), ∧ (parallel composition), and ↻ (redo loop). This way it is possible to construct process trees such as the ones shown in Fig. 10.

Definition 17 (Process Tree). *Let $PTO = \{\rightarrow, \times, \wedge, \circlearrowleft\}$ be the set of process tree operators and let $\tau \notin \mathcal{U}_{act}$ be the so-called silent activity. Process trees are defined as follows.*

- *if $a \in \mathcal{U}_{act} \cup \{\tau\}$, then $Q = a$ is a process tree,*
- *if $n \geq 1$, $Q_1, Q_2, \ldots, Q_n$ are process trees, and $\oplus \in \{\rightarrow, \times, \wedge\}$, then $Q = \oplus(Q_1, Q_2, \ldots Q_n)$ is a process tree, and*
- *if $n \geq 2$ and $Q_1, Q_2, \ldots, Q_n$ are process trees, then $Q = \circlearrowleft(Q_1, Q_2, \ldots Q_n)$ is a process tree.*

$\mathcal{U}_Q \subseteq \mathcal{U}_M$ is the set of all process trees.

Consider the process tree $Q_1 = \rightarrow(a, \times(\wedge(b, c), d), e)$ shown in Fig. 10(a). The leaf nodes correspond to the activities a, b, c, d, and e. The root node is a sequence operator ($\rightarrow$) having three children: a, $\times(\wedge(b, c), d)$, and e. The root node of the subtree $\times(\wedge(b, c), d)$ is a choice operator ($\times$) having two children: $\wedge(b, c)$ and d. The root node of the subtree $\wedge(b, c)$ is a parallel operator ($\wedge$) having two children: b and c.

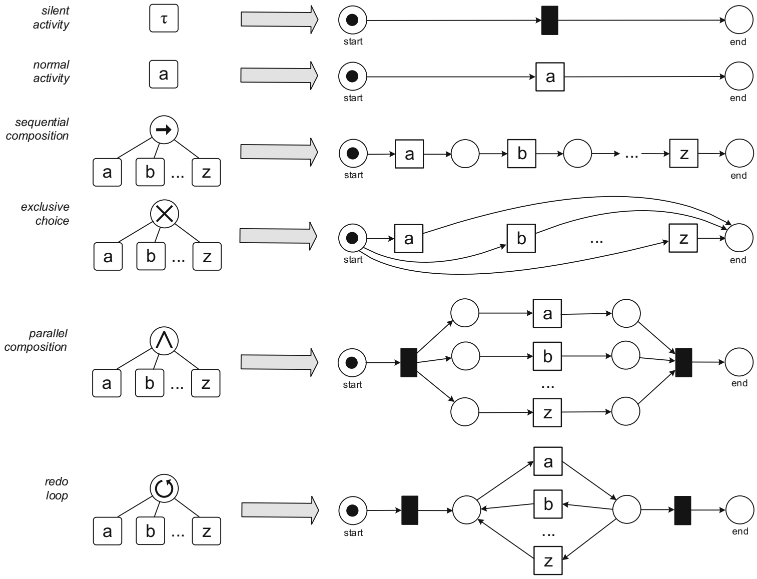


Fig. 11. The semantics of the four process tree operators, i.e., → (sequential composition), × (exclusive choice), ∧ (parallel composition), and $\circlearrowleft$ (redo loop), expressed in terms of Petri nets.

Although it is fairly straightforward to define the semantics of process trees directly in terms of traces, we can also use the mapping onto accepting labeled Petri nets shown in Fig. 11. A silent activity, i.e., a leaf node labeled τ, is mapped onto a silent transition. A normal activity a is mapped onto a transition t with label $l(t) = a$. Sequential composition $\rightarrow(a, b, c, \ldots, z)$ corresponds to the Petri net structure shown in Fig. 11, i.e., first

a occurs and only if a has finished, b may start, after b completes, c can start, etc. The sequential composition ends when the last element completes. Note that $a, b, c, \ldots, z$ do not need to be atomic activities. These elements may correspond to large subprocesses, each represented by a subtree of arbitrary complexity. Exclusive choice $\times(a, b, c, \ldots, z)$ and parallel composition $\wedge(a, b, c, \ldots, z)$ can be mapped onto Petri nets as shown in Fig. 11. Also here the elements do not need to be atomic and may correspond to subtrees of arbitrary complexity. Figure 11 also shows the semantics of the redo loop operator $\circlearrowleft$. In $\circlearrowleft(a, b, c, \ldots, z)$, first a is executed. This is called the "do" part (again a may be a subprocess). Then there is the option to stop (fire the silent transition to go to the end place) or one of the "redo elements" is executed. For example, b is executed. After the completion of b, we again execute the "do" part a after which there is again the choice to stop or pick one of the "redo elements", etc. Note that semantically $\circlearrowleft(a, b, c, \ldots, z)$ and $\circlearrowleft(a, \times(b, c, \ldots, z))$ are the same.

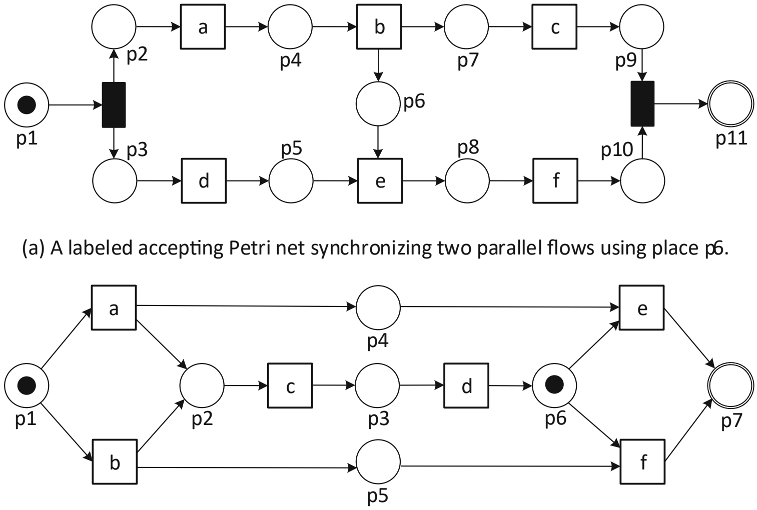
Definition 18 (Traces of a Process Tree). *Let $Q \in \mathcal{U}_Q$ be a process tree and $AN_Q \in \mathcal{U}_{AN}$ the corresponding accepting Petri net constructed by recursively applying the patterns depicted in Fig. 11. $lang(Q) = lang(AN_Q)$ are the traces possible according to Q.*

Using the above definition, we can compute the set of traces for the three process trees in Fig. 10: $Q_1 = \rightarrow(a, \times(\wedge(b, c), d), e)$, $Q_2 = \rightarrow(a, \circlearrowleft(\wedge(b, c), d), e)$, and $Q_3 = \rightarrow(a, \circlearrowleft(\wedge(b, a), \tau))$. $lang(Q_1) = \{\langle a, b, c, e\rangle, \langle a, c, b, e\rangle, \langle a, d, e\rangle\}$, $lang(Q_2) = \{\langle a, b, c, e\rangle, \langle a, c, b, e\rangle, \langle a, b, c, d, b, c, e\rangle, \langle a, c, b, d, b, c, e\rangle, \ldots, \langle a, c, b, d, b, c, d, c, b, d, c, b, e\rangle, \ldots\}$, and $lang(Q_3) = \{\langle a, b, a\rangle, \langle a, a, b\rangle, \langle a, b, a, b, a\rangle, \langle a, a, b, b, a\rangle, \ldots, \langle a, a, b, b, a, a, b, a, b\rangle, \ldots\}$.

Some additional examples to illustrate the expressiveness of process trees:

- $lang(\rightarrow(a, \times(b, \tau), c)) = \{\langle a, b, c\rangle, \langle a, c\rangle\}$ (ability to skip b).
- $lang(\rightarrow(a, a)) = \{\langle a, a\rangle\}$ (ability to specify that a should occur twice).
- $lang(\circlearrowleft(a, \tau)) = \{\langle a\rangle, \langle a, a\rangle, \langle a, a, a\rangle, \ldots\}$ (at least one a).
- $lang(\circlearrowleft(\tau, b)) = \{\langle\rangle, \langle b\rangle, \langle b, b\rangle, \ldots\}$ (any number of b's)
- $lang(\circlearrowleft(a, b)) = \{\langle a\rangle, \langle a, b, a\rangle, \langle a, b, a, b, a\rangle, \ldots\}$ (alternate a and b).
- $lang(\circlearrowleft(\tau, a, b, c, \ldots, z)) = \{a, b, c, \ldots, z\}^*$ (all traces over given set of activities).

There are also behaviors that are difficult to express in terms of a process tree. For example, it is difficult to synchronize between subtrees. Consider, for example, the process tree $Q = \wedge(\rightarrow(a, b, c), \rightarrow(d, e, f))$ with the additional requirement that b should be executed before e. This can only be handled by duplicating activities, e.g., $Q = \times(\rightarrow(\wedge(\rightarrow(a, b), d), \wedge(c, \rightarrow(e, f))), \rightarrow(a, b, c, d, e, f))$. Trying to capture arbitrary synchronizations between subprocesses leads to incomprehensible process trees whose behavior is still easy to express in terms of a BPMN model or a labeled accepting Petri net. Figure 12(a) shows how this can be expressed in terms of a labeled accepting Petri net. Similarly, process trees cannot capture long-term dependencies (e.g., a choice at the beginning of the process influences a choice later in the process). Figure 12(b) shows an example where the first choice depends on the second choice. This simple example can be modeled using the process tree $Q = \times(\rightarrow(a, c, d, e), \rightarrow(b, c, d, f))$, which enumerates the two traces and duplicates activities c and d. In general, process-tree based discovery techniques are unable to create such models. Nevertheless, process



(a) A labeled accepting Petri net synchronizing two parallel flows using place p6.

(b) A labeled accepting Petri net with long-term dependencies (p4 and p5).

Fig. 12. Two labeled accepting Petri nets with behaviors that are difficult to discover in terms of a process tree. The top model (a) corresponds to the process tree $Q = \wedge(\rightarrow(a, b, c), \rightarrow(d, e, f))$ with the additional requirement that b should be executed before e. The bottom model (b) corresponds to the process tree $Q = \rightarrow(\times(a, b), c, d, \times(e, f))$ with the additional requirement that a should be followed by e and b should be followed by f.

trees provide a powerful representational bias that can be exploited by process discovery techniques.

4.3 Business Process Model and Notation (BPMN)

Business Process Model and Notation (BPMN) is the de facto representation for business process modeling in industry [19,36]. The BPMN standard is maintained by the Object Management Group (OMG) [27], is supported by a wide range of vendors, and is used by numerous organizations. The OMG specification is 532 pages [27]. Given our focus on process discovery, the constructs for control-flow are most relevant. Moreover, most tools only support a small subset of the BPMN standard and an even smaller subset is actually used on a larger scale. When using the more advanced constructs like inclusive/complex gateways and multiple instance activities, the execution semantics are also not so clear (see Chapter 13 of [27]). Therefore, we only cover start and end events, activities, exclusive gateways, parallel gateways, and sequence flows. Constructs such as pools, lanes, data objects, messages, subprocesses, and inclusive gateways are relevant for more advanced forms of process mining, but outside the scope of this chapter.

Figure 13 shows three BPMN models (B_1, B_2, and B_3) and a limited set of BPMN notations. We (informally) refer to the class of BPMN models constructed using these building blocks as $\mathcal{U}_{BPMN}$. The behavior represented by the BPMN model

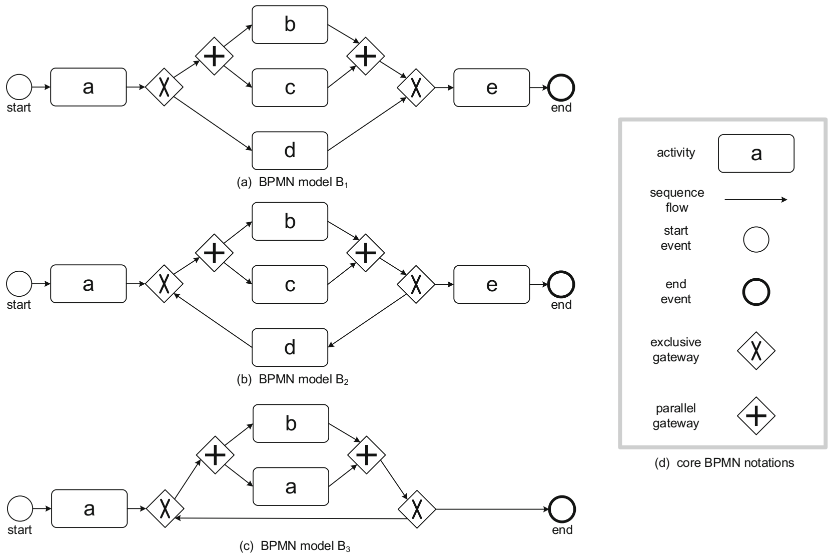


Fig. 13. Three BPMN models corresponding to the accepting Petri nets AN_1, AN_2, and AN_4, and the process trees Q_1, Q_2, and Q_3 used before.

$B_1 \in \mathcal{U}_{BPMN}$ is the same as the accepting Petri net $AN_1 = (N_1, [p1], [p6])$ in Fig. 9(a) and the process tree $Q_1 = \to(a, \times(\wedge(b, c), d), e)$ in Fig. 10(a). Hence, $lang(B_1) = \{\langle a, b, c, e\rangle, \langle a, c, b, e\rangle, \langle a, d, e\rangle\}$. BPMN model $B_2 \in \mathcal{U}_{BPMN}$ corresponds to AN_2 in Fig. 9(b) and the process tree Q_2 in Fig. 10(b). BPMN model $B_3 \in \mathcal{U}_{BPMN}$ corresponds to AN_4 in Fig. 9(d) and the process tree Q_3 in Fig. 10(c). We do not provide formal semantics for these BPMN constructs. However, the examples should be self-explaining and demonstrate that a BPMN model $B \in \mathcal{U}_{BPMN}$ defines indeed a set of traces $lang(B)$.

In this chapter, we have introduced four types of models: DFGs $\mathcal{U}_G \subseteq \mathcal{U}_M$, accepting Petri nets $\mathcal{U}_{AN} \subseteq \mathcal{U}_M$, process trees $\mathcal{U}_Q \subseteq \mathcal{U}_M$, and BPMN models $\mathcal{U}_{BPMN} \subseteq \mathcal{U}_M$. There exist discovery approaches for all of them. Since they all specify sets of possible complete traces, automated translations are often possible. For example, a discovery technique may use process trees internally, but use Petri nets or BPMN models to visualize the result.

5 Bottom-Up Process Discovery

In Sect. 2, we presented a baseline discovery approach to learn a DFG from an event log. As stated in Definition 3, a process discovery algorithm is a function $disc \in \mathcal{B}(\mathcal{U}_{act}{}^*) \to \mathcal{U}_M$ that, given an event log L, produces a model $M = disc(L)$ that allows for the traces in $lang(M)$. The DFG-based baseline approach has many limitations. One of the main limitations is the inability to represent concurrency. The DFG produced tends to have an excessive number of cycles leading to Spaghetti-like underfitting models. Therefore, we introduced higher-level process model notations such as accepting Petri nets (Sect. 4.1), process trees (Sect. 4.2), and a subset of the BPMN notation (Sect. 4.3).

In this chapter, we group the more advanced approaches into two groups: "bottom-up" process discovery and "top-down" process discovery. The first group aims to uncover local patterns involving a few activities. The second group aims to find a global structure that can be used to decompose the discovery problem into smaller problems. In this section, we introduce "bottom-up" process discovery using the Alpha algorithm [1,9] as an example. In Sect. 6, we introduce "top-down" process discovery using the basic inductive mining algorithm [22–24] as an example.

Both "bottom-up" and "top-down" process discovery can be combined with the filtering approaches presented in Sect. 2.4, in particular activity-based and variant-based filtering. Without filtering, the basic Alpha algorithm and basic inductive mining algorithm will not be very usable in real-life settings. Therefore, we assume that the event logs have been preprocessed before applying "bottom-up" or "top-down" discovery algorithms.

Definition 19 (Basic Log Preprocessing). *Let* $L \in \mathcal{B}({\mathcal{U}_{act}}^*)$ *be an event log. Given the thresholds* $\tau_{act} \in \mathbb{N}$ *and* $\tau_{var} \in \mathbb{N}$*:* $L^{\tau_{act},\tau_{var}} = \mathit{filter}^{var}(\mathit{filter}^{act}(L, \tau_{act}), \tau_{var})$*.*

In the remainder, we assume that the event log was preprocessed and that we want to discover a process model describing the filtered event log.

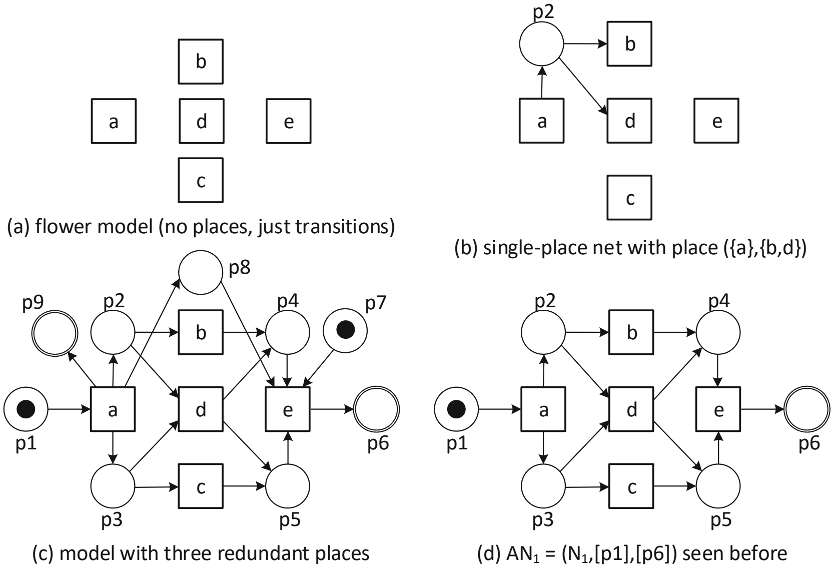
5.1 The Essence of Bottom-Up Process Discovery: Admissible Places

To explain "bottom-up" process discovery, we first introduce the notion of a "flower model" for an event log. This is the accepting Petri net without places. We use this as a basis and then add places one-by-one.

Definition 20 (Flower Model). *Let* $L \in \mathcal{B}({\mathcal{U}_{act}}^*)$ *be an event log with activities* $A = act(L)$*. The flower model of L is the accepting Petri net* $disc_{\mathit{flower}}(L) = (N, [\,], [\,])$ *with* $N = (\emptyset, A, \emptyset, \{(a, a) \mid a \in A\})$*.*

Note that $disc_{\mathit{flower}}(L)$ contains no places and one transition per activity. The flower model of L_1 is shown in Fig. 14(a). In a Petri net, a transition is enabled if all of its input places contain a token. Hence, a transition without an input place is always enabled. Moreover, the Petri net is always in the final marking $[\,]$. Therefore, $lang(disc_{\mathit{flower}}(L)) = A^*$, i.e., all traces over activities seen in the event log. Such a flower model can also be represented as a process tree. If $A = \{a_1, a_2, \ldots, a_n\} = act(L)$, then $Q = \circlearrowleft(\tau, a_1, a_2, \ldots, a_n)$ is the process tree that allows for any behavior over A, i.e., $lang(Q) = A^*$. Although it is easy to create such a process tree, it is not so clear how to add constraints to it. As mentioned earlier, it is impossible to synchronize activities in different subtrees. However, when looking at the flower Petri net $disc_{\mathit{flower}}(L)$, it is obvious that places can be added to constrain the behavior. Therefore, we use Petri nets to illustrate "bottom-up" process discovery.

Next, we consider a Petri net having a *single place* constraining the behavior of the flower model. The place $p = (A_1, A_2)$ is characterized by a set of input activities A_1 and a set of output activities A_2. We would like to add places that allow for the behavior seen in the event log. Such a place is called an *admissible place*.



(a) flower model (no places, just transitions)

(b) single-place net with place ({a},{b,d})

(c) model with three redundant places

(d) AN₁ = (N₁,[p1],[p6]) seen before

Fig. 14. Four accepting Petri nets: (a) a flower model, (b) AN_{p_2} with just one place $p_2 = (\{a\},\{b,d\})$, (c) an accepting Petri net with three additional redundant places $p_7 = (\emptyset,\{e\})$, $p_8 = (\{a\},\{e\})$, and $p_9 = (\{a\},\emptyset)$, and (d) the accepting Petri net AN_1 already shown in Fig. 9(a) (discovered by applying the original Alpha algorithm [1,9] to event log L_1).

Definition 21 (Admissible Place). *Let $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ be an event log with activities $A = act(L)$. $p = (A_1, A_2)$ is a candidate place if $A_1 \subseteq A$ and $A_2 \subseteq A$. The corresponding single place accepting Petri net is $AN_p = (N, M_{init}, M_{final})$ with $N = (P, T, F, l)$, $P = \{p\}$, $T = A$, $F = \{(a,p) \mid a \in A_1\} \cup \{(p,a) \mid a \in A_2\}$, $l = \{(a,a) \mid a \in A\}$), $M_{init} = [p \mid A_1 = \emptyset]$, and $M_{final} = [p \mid A_2 = \emptyset]$. Candidate place $p = (A_1, A_2)$ is admissible if $var(L) \subseteq lang(AN_p)$. $\overset{adm}{P}(L)$ is the set of all admissible places, given an event log L.*

Given a candidate place $p = (A_1, A_2)$, AN_p is the accepting Petri net consisting of one transition per activity and a single place p. The transitions in A_1 produce tokens for p and the transitions in A_2 consume tokens from p. If p is a source place (i.e., $A_1 = \emptyset$), then it has to be initially marked to be meaningful (otherwise, it would remain empty by definition). If p is a sink place (i.e., $A_2 = \emptyset$), then it has to be marked in the final marking to be meaningful (otherwise, it could never be marked on a path to the final marking). We also assume that all other places are empty both at the beginning and at the end. Hence, only source places are initially marked and only sink places are marked in the final marking. This explains the reason that $M_{init} = [p \mid A_1 = \emptyset]$ (p is initially marked if it is a source place) and $M_{final} = [p \mid A_2 = \emptyset]$ (p is marked in the final marking if it is a sink place).

A candidate place $p = (A_1, A_2)$ is admissible if the corresponding AN_p allows for all the traces seen in the event log, i.e., event log L and single-place net AN_p are perfectly fitting. Consider, for example, $L_1 = [\langle a,b,c,e\rangle^{10}, \langle a,c,b,e\rangle^{5}, \langle a,d,e\rangle]$.

Examples of admissible candidate places are $p_1 = (\emptyset, \{a\})$, $p_2 = (\{a\}, \{b, d\})$, $p_3 = (\{a\}, \{c, d\})$, $p_4 = (\{b, d\}, \{e\})$, $p_5 = (\{c, d\}, \{e\})$, $p_6 = (\{e\}, \emptyset)$. These are the places shown earlier in Fig. 9(a) (for convenience the accepting Petri net AN_1 is again shown in Fig. 14(d)). However, we now consider an accepting Petri net per place, i.e., $AN_{p_1}, AN_{p_2}, AN_{p_3}, \ldots, AN_{p_6}$. Figure 14(b) shows AN_{p_2} with $p_2 = (\{a\}, \{b, d\})$. Other admissible places (not shown in Fig. 9(a)) are $p_7 = (\emptyset, \{e\})$, $p_8 = (\{a\}, \{e\})$, $p_9 = (\{a\}, \emptyset)$. Examples of candidate places that are not admissible are $p_{10} = (\emptyset, \{b\})$ (the initial token in p_{10} is not consumed when replaying $\langle a, d, e \rangle$), $p_{11} = (\{a\}, \{b\})$ (the token produced for p_{11} by a is not consumed when replaying $\langle a, d, e \rangle$), $p_{12} = (\{b\}, \{e\})$ (it is impossible to replay $\langle a, d, e \rangle$ because of a missing token in p_{12}), and $p_{13} = (\{b\}, \emptyset)$ (the sink place is not marked when replaying $\langle a, d, e \rangle$).

Note that places correspond to *constraints*. Place $p_4 = (\{b, d\}, \{e\})$ allows for all the traces in L_1 but does not allow for traces such as $\langle a, e \rangle$, $\langle a, b, d, e \rangle$, $\langle a, b, e, e \rangle$, etc.

Assuming that we want to ensure perfect replay fitness (i.e., 100% recall), we *only add admissible places*. This is a reasonable premise if filtered the event log (cf. Definition 19) before conducting discovery. This means that process discovery is reduced to finding a subset of $P^{adm}(L)$ (i.e., a selection of admissible places given event log L).

Why not simply add all places in $P^{adm}(L)$ to the discovered process model? There are two reasons not to do this: *redundancy* and *overfitting*. A place is *redundant* if its removal does not change the behavior. Consider, for example, Fig. 14(c) with two source places, two sink places, and an additional place connecting a and e. The places $p_7 = (\emptyset, \{e\})$, $p_8 = (\{a\}, \{e\})$, and $p_9 = (\{a\}, \emptyset)$ are redundant, i.e., we can remove them without allowing for more behavior. Moreover, adding all possible places in $P^{adm}(L)$ may lead to overfitting. As explained in Sect. 3, the event log contains example behavior and it would be odd to assume that behaviors that have not been observed are not possible. Note that there are $2^n \times 2^n = 2^{2n}$ candidate places with $n = |act(L)|$. Hence, *for a log with just ten activities there are over one million candidate places* ($2^{2 \times 10} = 1048576$)). Many of these will be admissible by accident. This problem is comparable to "multiple hypothesis testing" in statistics. If one tests enough hypotheses, then one will find seemingly significant results by accident (cf. Bonferroni correction).

There are many approaches to select a suitable subset of $P^{adm}(L)$. For example, it is easy to remove redundant places and only consider places with a limited number of input and output arcs [7,26]. However, there is the additional problem that the above procedure requires evaluating each candidate place with respect to the whole event log. This means that a naïve approach quickly becomes intractable for larger event logs and processes.

5.2 The Alpha Algorithm

In the remainder of this section, we present the first process discovery technique able to discover concurrent models (e.g., Petri nets) from event logs: the *Alpha algorithm* [9]. The Alpha algorithm is completely based on the footprint of the (filtered) event log L. This implies that one pass through the event log is sufficient. Hence, the algorithm is linear in the size of the log (a naïve implementation is exponential in the number of unique activities, but this number is typically low). One can implement the Alpha algorithm efficiently by combining → relations that meet certain constraints. These constrains are monotonic, allowing for an apriori-style algorithm [1].

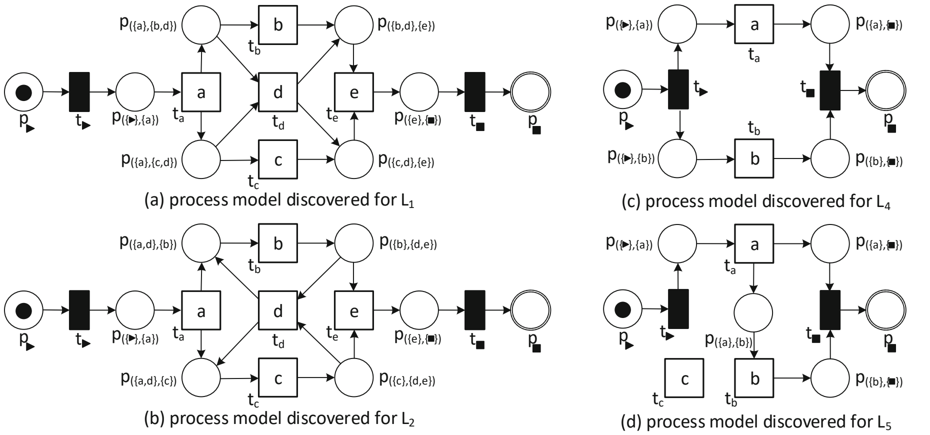
We have adapted the original presentation used in [9] to leverage the notations and insights already provided in this chapter. We use as input a DFG and as a result also add a dummy start (▶) and end (■) activity. However, in essence, the algorithm did not change. We elaborate on the differences with [9] later. The Alpha algorithm discovers an accepting Petri net for any event log L.

Definition 22 (Alpha Algorithm). *The alpha algorithm* $disc_{alpha} \in \mathcal{B}({\mathcal{U}_{act}}^*) \rightarrow \mathcal{U}_{AN}$ *returns an accepting Petri net* $disc_{alpha}(L)$ *for any event log* $L \in \mathcal{B}({\mathcal{U}_{act}}^*)$*. Let* $A = act(L)$ *and* $fp(L) = fp(disc_{DFG}(L))$ *the footprint of event log L. This allows us to write* $a_1 \rightarrow_L a_2$ *if* $fp(L)((a_1, a_2)) = \rightarrow$ *and* $a_1 \#_L a_2$ *if* $fp(L)((a_1, a_2)) = \#$ *for any* $a_1, a_2 \in A' = A \cup \{\blacktriangleright, \blacksquare\}$.

1. $Cnd = \{(A_1, A_2) \mid A_1 \subseteq A' \wedge A_1 \neq \emptyset \wedge A_2 \subseteq A' \wedge A_2 \neq \emptyset \wedge \forall_{a_1 \in A_1} \forall_{a_2 \in A_2}\ a_1 \rightarrow_L a_2 \wedge \forall_{a_1, a_2 \in A_1}\ a_1 \#_L a_2 \wedge \forall_{a_1, a_2 \in A_2}\ a_1 \#_L a_2\}$ *are the candidate places,*
2. $Sel = \{(A_1, A_2) \in Cnd \mid \forall_{(A_1', A_2') \in Cnd}\ A_1 \subseteq A_1' \wedge A_2 \subseteq A_2' \implies (A_1, A_2) = (A_1', A_2')\}$ *are the selected maximal places,*
3. $P = \{p_{(A_1, A_2)} \mid (A_1, A_2) \in Sel\} \cup \{p_{\blacktriangleright}, p_{\blacksquare}\}$ *is the set of all places,*
4. $T = \{t_a \mid a \in A'\}$ *is the set of transitions,*
5. $F = \{(t_a, p_{(A_1, A_2)}) \mid (A_1, A_2) \in Sel \wedge a \in A_1\} \cup \{(p_{(A_1, A_2)}, t_a) \mid (A_1, A_2) \in Sel \wedge a \in A_2\} \cup \{(p_{\blacktriangleright}, t_{\blacktriangleright}), (t_{\blacksquare}, p_{\blacksquare})\}$ *is the set of arcs,*
6. $l = \{(t_a, a) \mid a \in A\}$ *is the labeling function,*
7. $M_{init} = [p_{\blacktriangleright}]$ *is the initial marking,* $M_{final} = [p_{\blacksquare}]$ *is the final marking, and*
8. $disc_{alpha}(L) = ((P, T, F, l), M_{init}, M_{final})$ *is the discovered accepting Petri net.*

The complexity of the algorithm is in the first two steps building the sets Cnd and Sel that are used to create the places in Step 3. The rest builds on the ideas and notions introduced before. The Alpha algorithm creates a transition t_a for each activity a in the event log and also adds a start transition $t_{\blacktriangleright}$ and an end transition $t_{\blacksquare}$ (Step 4). Transitions are labeled with the corresponding activity (Step 6). Transitions $t_{\blacktriangleright}$ and $t_{\blacksquare}$ are silent, $t_{\blacktriangleright}$ has a source place $p_{\blacktriangleright}$ as input and $t_{\blacksquare}$ has a sink place $p_{\blacksquare}$ as output. The initial marking only marks the source place $p_{\blacktriangleright}$ and the final marking only marks the sink place $p_{\blacksquare}$ (Step 7). Steps 3–8 can be seen as "bookkeeping". The essence of the algorithm is in the first two steps.

Step 1 of the algorithm creates candidate places similar to the construction of candidate places used in Definition 21. (A_1, A_2) corresponds to a candidate place p such that activities in A_1 produce tokens for p and activities in A_2 consume tokens from p. Note that technically (A_1, A_2) is a pair of non-empty sets of activities (including start and end). The requirement $\forall_{a_1 \in A_1} \forall_{a_2 \in A_2}\ a_1 \rightarrow_L a_2$ states that any activity in A_1 can be directly followed by any activity in A_2, but no activity in A_2 can be directly followed by an activity in A_1. The requirements $\forall_{a_1, a_2 \in A_1}\ a_1 \#_L a_2$ and $\forall_{a_1, a_2 \in A_2}\ a_1 \#_L a_2$ state that activities in the sets A_1 and A_2 cannot directly follow any other member of the same activity set. As a consequence, an activity that can follow itself directly (i.e., $a \|_L a$) cannot be in A_1 or A_2. This also implies that A_1 and A_2 are disjoint. Cnd is the set of all pairs of activity sets meeting these requirements. $Sel \subseteq Cnd$ retains the "maximal elements". Candidate $(A_1, A_2) \in Cnd$ is maximal if there is no other $(A'_1, A'_2) \in Cnd$ that is strictly larger, i.e., it cannot be that $A_1 \subseteq A'_1$, $A_2 \subseteq A'_2$, and $(A'_1, A'_2) \neq (A_1, A_2)$. Each selected maximal element, i.e., $(A_1, A_2) \in Sel$, corresponds to a place $p_{(A_1,A_2)}$ connecting the transitions corresponding to A_1 (i.e., $\{t_a \mid a \in A_1\}$) to the transitions corresponding to A_2 (i.e., $\{t_a \mid a \in A_2\}$).



(a) process model discovered for L₁

(c) process model discovered for L₄

(b) process model discovered for L₂

(d) process model discovered for L₅

Fig. 15. Four accepting Petri nets created using the Alpha algorithm from Definition 22. The place and transition names are as specified in Definition 22. The four event logs used are: $L_1 = [\langle a,b,c,e\rangle^{10}, \langle a,c,b,e\rangle^{5}, \langle a,d,e\rangle]$, $L_2 = [\langle a,b,c,e\rangle^{50}, \langle a,c,b,e\rangle^{40}, \langle a,b,c,d,b,c,e\rangle^{30}, \langle a,c,b,d,b,c,e\rangle^{20}, \langle a,b,c,d,c,b,e\rangle^{10}, \langle a,c,b,d,c,b,d,b,c,e\rangle^{10}]$, $L_4 = [\langle a,b\rangle^{35}, \langle b,a\rangle^{15}]$, and $L_5 = [\langle a\rangle^{10}, \langle a,b\rangle^{8}, \langle a,c,b\rangle^{6}, \langle a,c,c,b\rangle^{3}, \langle a,c,c,c,b\rangle]$. Note that unlike in [9] invisible start and end transitions are added to be more general.

Figure 15 shows some examples where the Alpha algorithm is applied to a smaller event log. The place names reflect the elements of the set Sel created in Step 2 of the algorithm. For $L_1 = [\langle a,b,c,e\rangle^{10}, \langle a,c,b,e\rangle^{5}, \langle a,d,e\rangle]$, $Sel = \{(\{\blacktriangleright\},\{a\}), (\{a\}, \{b,d\}), (\{a\},\{c,d\}), (\{b,d\},\{e\}), (\{c,d\},\{e\}), (\{e\},\{\blacksquare\})\}$. Note that $Cnd \setminus Sel = \{(\{a\},\{b\}), (\{a\},\{c\}), (\{a\},\{d\}), (\{b\},\{e\}), (\{c\},\{e\}), (\{d\},\{e\})\}$. These candidates were removed because they are not maximal. Figure 15(a) shows the resulting accepting Petri net $disc_{alpha}(L_1)$. Figure 15(b) shows $disc_{alpha}(L_2)$. Note that the Alpha algorithm is able to discover concurrency, choices, and loops. Comparing the process models for L_1 and L_2 with the accepting Petri nets in Fig. 2 (for L_1) and Fig. 3 (for L_2), we can see that $p_{\blacktriangleright}$, $t_{\blacktriangleright}$, $t_{\blacksquare}$, and $p_{\blacksquare}$ have been added. These can be removed if start and end activities happen only at the beginning or end. In L_1 and L_2, the only start activity is a and a can only happen in the first position. Also, the only end activity is e and e can only happen in the last position. If this is the case, we do not need to add an artificial start ▶ or end ■.

Figure 15(c) shows why it is sometimes necessary to add an artificial start or end. In $L_4 = [\langle a,b\rangle^{35}, \langle b,a\rangle^{15}]$, a is a start activity in trace $\langle a,b\rangle$, but can also happen at the second position (cf. $\langle b, a\rangle$). The same holds for activity b. Therefore, we need to add an artificial start $\blacktriangleright$. a and b are also end activities, but do not appear just at the end, e.g., b may also happen in the first position. Therefore, we need to add an artificial end $\blacksquare$. Note that Definition 22 is slightly different from the original algorithm in [9] due to the addition of the dummy start and end activities. For logs where the traditional algorithm already produces the correct result, one can simply remove $p_{\blacktriangleright}$, $t_{\blacktriangleright}$, $t_{\blacksquare}$, and $p_{\blacksquare}$. However, the algorithm in Definition 22 is able to handle start and end activities that can also appear in the middle of a trace. Hence, it is more general.

Figure 16 shows the model discovered for the larger event log $L_3 = [\langle ie, cu, lt, xr, fe\rangle^{285}, \langle ie, cu, lt, ct, fe\rangle^{260}, \langle ie, cu, ct, lt, fe\rangle^{139}, \langle ie, lt, cu, xr, fe\rangle^{137}, \langle ie, lt, cu, ct, fe\rangle^{124}, \langle ie, cu, xr, lt, fe\rangle^{113}, \langle ie, xr, cu, lt, fe\rangle^{72}, \langle ie, ct, cu, xr, fe\rangle^{72}, \langle ie, cu, om, am, cu, lt, xr, fe\rangle^{29}, \langle ie, cu, om, am, cu, lt, ct, fe\rangle^{28}, \ldots]$ using the full activity names, i.e., ie = initial examination, xr = X-ray, ct = CT scan, cu = checkup, om = order medicine, am = administer medicine, lt = lab tests, and fe = final examination. The model was generated using the Alpha algorithm implemented in ProM. Note that there was no need to add artificial start or end activities because ie happens only at the beginning and fe happens only at the end.

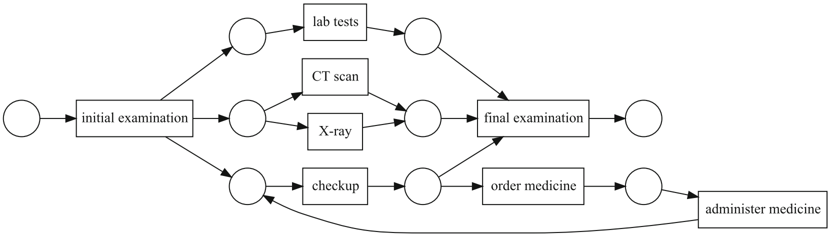


Fig. 16. The accepting Petri net that was discovered by the Alpha algorithm implemented in ProM, based on the larger event log L_3 introduced in Sect. 2.5. Note that the artificial start and end activities have not been added, and the full activity names are used.

The Alpha algorithm should be seen as a baseline algorithm to discover concurrency. It has many limitations, as pointed out in the original paper presenting the algorithm [9]. Event log $L_5 = [\langle a\rangle^{10}, \langle a, b\rangle^{8}, \langle a, c, b\rangle^{6}, \langle a, c, c, b\rangle^{3}, \langle a, c, c, c, b\rangle]$ is used to illustrate two of these problems: *skipping* and *self-loops*. Figure 15(d) shows the discovered process model $disc_{alpha}(L_5)$. The selected maximal elements are $Sel = \{(\{\blacktriangleright\}, \{a\}), (\{a\}, \{b\}), (\{a\}, \{\blacksquare\}), (\{b\}, \{\blacksquare\})\}$. Note that $(\{a\}, \{b, \blacksquare\}) \notin Sel$, because $b \rightarrow_{L_5} \blacksquare$ and not $b \#_{L_5} \blacksquare$. Because $c \|_{L_5} c$ (c can be directly followed by c) and not $c \#_{L_5} c$, activity c does not appear in Sel, implying that t_c remains disconnected from the rest of the model. Activity b can be seen as a "skippable" activity and the Alpha algorithm cannot handle such activities, because these require silent transitions. The basic Alpha algorithm can also not discover the self-loop involving c. The Alpha algorithm has been extended to address these problems, and there exist variants to deal with self-loops, skipping, long-term dependencies, etc. See [1] for more information on the limitations of the basic algorithm and pointers to extensions addressing these problems.

6 Top-Down Process Discovery

The Alpha algorithm is an example of a bottom-up discovery approach that tries to add places to the Petri net to locally constrain behavior. *Top-down discovery approaches try to recursively decompose the event log into smaller event logs until the problem gets trivial.* The whole event log L is decomposed into smaller event logs $L_1, L_2, \ldots, L_n$ that have a clear relationship, e.g., L_i may contain events that occur before L_j if $i < j$, or L_i and L_j are fully disjoint for all $i \neq j$. Each event in L ends up in *precisely one* of the sublogs. However, cases may be distributed over multiple sublogs. Each of the smaller event logs is analyzed and (if needed) decomposed into smaller event logs, e.g., L_i is in turn decomposed into $L_{i,1}, L_{i,2}, \ldots, L_{i,m}$, etc. Again the events in L_i are partitioned over $L_{i,1}, L_{i,2}, \ldots, L_{i,m}$. This is repeated until we encounter a so-called *base case*, i.e., a sublog containing just one activity, e.g., $[\langle a \rangle^{160}]$, $[\langle a \rangle^{80}, \langle \rangle^{80}]$, or $[\langle a \rangle^{80}, \langle a, a \rangle^{60}, \langle a, a, a \rangle^{20}]$.

Due to the recursive decomposition of logs into smaller event logs, we automatically get a tree-like structure where the root corresponds to the original event log and the leaves correspond to trivial event logs (the so-called base cases). This fits well with the process tree formalism introduced in Sect. 4.2.

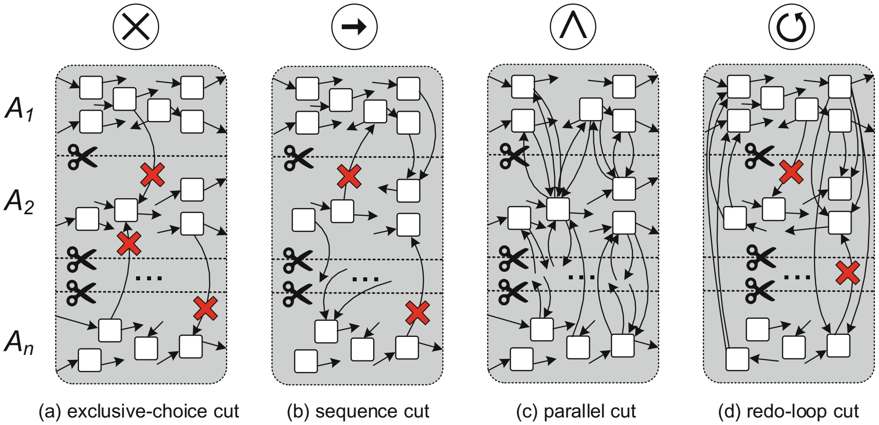
Before introducing a particular approach, let's use a few simple event logs to illustrate the idea of splitting an event log.

- Event log $L = [\langle a, b, c \rangle^{100}]$ is decomposed into base cases $L_1 = [\langle a \rangle^{100}]$, $L_2 = [\langle b \rangle^{100}]$, and $L_3 = [\langle c \rangle^{100}]$ leading to the discovery of $Q = \rightarrow(a, b, c)$.
- Event log $L = [\langle a \rangle^{50}, \langle b \rangle^{25}, \langle c \rangle^{25}]$ is decomposed into base cases $L_1 = [\langle a \rangle^{50}]$, $L_2 = [\langle b \rangle^{25}]$, and $L_3 = [\langle c \rangle^{25}]$ leading to the discovery of $Q = \times(a, b, c)$.
- Event log $L = [\langle a, b, c \rangle^{30}, \langle a, c, b \rangle^{20}, \langle b, a, c \rangle^{20}, \langle b, c, a \rangle^{10}, \langle c, a, b \rangle^{10}, \langle c, b, a \rangle^{10}]$ is decomposed into base cases $L_1 = [\langle a \rangle^{100}]$, $L_2 = [\langle b \rangle^{100}]$, and $L_3 = [\langle c \rangle^{100}]$ leading to the discovery of $Q = \wedge(a, b, c)$.
- Event log $L = [\langle a \rangle^{50}, \langle a, b, a \rangle^{25}, \langle a, b, a, b, a \rangle^{25}]$ is decomposed into base cases $L_1 = [\langle a \rangle^{175}]$ and $L_2 = [\langle b \rangle^{75}]$ leading to the discovery of $Q = \circlearrowleft(a, b)$.
- Event log $L = [\langle a, c \rangle^{50}, \langle a, b, c \rangle^{50}]$ is decomposed into base cases $L_1 = [\langle a \rangle^{100}]$, $L_2 = [\langle \rangle^{50}, \langle b \rangle^{50}]$, and $L_3 = [\langle c \rangle^{100}]$ leading to the discovery of $Q = \rightarrow(a, \times(b, \tau), c)$.
- Event log $L = [\langle a, c \rangle^{50}, \langle a, b, c \rangle^{20}, \langle a, b, b, c \rangle^{20}, \langle a, b, b, b, c \rangle^{10}]$ is decomposed into base cases $L_1 = [\langle a \rangle^{100}]$, $L_2 = [\langle \rangle^{50}, \langle b \rangle^{20}, \langle b, b \rangle^{20}, \langle b, b, b \rangle^{10}]$, and $L_3 = [\langle c \rangle^{100}]$ leading to the discovery of $Q = \rightarrow(a, \circlearrowleft(\tau, b), c)$.

In this section, we use the *basic inductive mining algorithm* to illustrate top-down discovery [22–24]. This algorithm uses DFGs to find so-called *cuts* partitioning the set of observed activities into subsets of activities. Set $A = act(L)$ is partitioned into pairwise disjoint sets of activities $A_1, A_2, \ldots, A_n$. These activity sets are used to distribute the events in L over $L_1, L_2, \ldots, L_n$ such that $A_1 = act(L_1)$, $A_2 = act(L_2)$, etc. There are cuts for all four process tree operators, i.e., $\rightarrow$ (sequential composition), $\times$ (exclusive choice), $\wedge$ (parallel composition), and $\circlearrowleft$ (redo loop).

Definition 23 (Sequence, Exclusive-Choice, Parallel, and Redo-Loop Cuts). *Let $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ be an event log having a DFG $disc_{_{DFG}}(L) = (A, F)$ based on L (note that $A = act(L)$) with start activities $A^{start} = \{a \in A \mid (\blacktriangleright, a) \in F\}$ and end activities $A^{end} = \{a \in A \mid (a, \blacksquare) \in F\}$. An n-ary $\oplus$-cut of L is a partition of A into $n \geq 2$ pairwise disjoint subsets $A_1, A_2, \ldots, A_n$ (i.e., $A = \bigcup_{i \in \{1,\ldots,n\}} A_i$ and $A_i \cap A_j = \emptyset$ for $i \neq j$) with $\oplus \in \{\rightarrow, \times, \wedge, \circlearrowleft\}$. Such a $\oplus$-cut is denoted $(\oplus, A_1, A_2, \ldots A_n)$. For each type of operator $\oplus \in \{\rightarrow, \times, \wedge, \circlearrowleft\}$ specific conditions apply:*

- *An* exclusive-choice cut *of L is a cut $(\times, A_1, A_2, \ldots A_n)$ such that*
  - $\forall_{i,j \in \{1,\ldots n\}} \forall_{a \in A_i} \forall_{b \in A_j}\ i \neq j \ \Rightarrow \ (a, b) \notin F$.
- *A* sequence cut *of L is a cut $(\rightarrow, A_1, A_2, \ldots A_n)$ such that*
  - $\forall_{i,j \in \{1,\ldots n\}} \forall_{a \in A_i} \forall_{b \in A_j}\ i < j \ \Rightarrow \ ((a, b) \in F^+ \ \wedge \ (b, a) \notin F^+)$.
    (Note that F^+ is the non-reflexive transitive closure of F, i.e., $(a, b) \in F^+$ means that there is a path from a to b in the DFG.)
- *A* parallel cut *of L is a cut $(\wedge, A_1, A_2, \ldots A_n)$ such that*
  - $\forall_{i \in \{1,\ldots n\}}\ A_i \cap A^{start} \neq \emptyset \ \wedge \ A_i \cap A^{end} \neq \emptyset$ *and*
  - $\forall_{i,j \in \{1,\ldots n\}} \forall_{a \in A_i} \forall_{b \in A_j}\ i \neq j \ \Rightarrow (a, b) \in F$.
- *A* redo-loop cut *of L is a cut $(\circlearrowleft, A_1, A_2, \ldots A_n)$ such that*
  - $A^{start} \cup A^{end} \subseteq A_1$,
  - $\forall_{i,j \in \{2,\ldots n\}} \forall_{a \in A_i} \forall_{b \in A_j}\ i \neq j \ \Rightarrow \ (a, b) \notin F$,
  - $\{a \in A_1 \mid (a, b) \in F \ \wedge \ b \notin A_1\} = A^{end}$,
  - $\{a \in A_1 \mid (b, a) \in F \ \wedge \ b \notin A_1\} = A^{start}$,
  - $\forall_{(a,b) \in F}\ a \in A_1 \ \wedge \ b \notin A_1 \ \Rightarrow \ \forall_{a' \in A^{end}}\ (a', b) \in F$, *and*
  - $\forall_{(b,a) \in F}\ a \in A_1 \ \wedge \ b \notin A_1 \ \Rightarrow \ \forall_{a' \in A^{start}}\ (b, a') \in F$.



(a) exclusive-choice cut (b) sequence cut (c) parallel cut (d) redo-loop cut

Fig. 17. Four types of cuts: $(\oplus, A_1, A_2, \ldots A_n)$ with $\oplus \in \{\times, \rightarrow, \wedge, \circlearrowleft\}$ (based on [1]).

Figure 17 illustrates the four types of cuts. There is an *exclusive-choice cut* when the DFG can be split into disconnected parts after leaving out the artificial start ▶ and end ■. (Recall that ▶ $\notin A$ and ■ $\notin A$.) There is a *sequence cut* when the DFG can be split into sequential parts where only "forward connections" are possible. Note that we need to use the non-reflexive transitive closure of F. There is a *parallel cut* when the DFG can be split into concurrent parts where any activity in one part can be followed by any activity in another part. The *redo-loop cut* has the most complex definition. All start and end activities should be in A_1 (the "do part") and none of the "redo parts" can have start or end activities. Moreover, the "redo parts" $(A_2, A_3, \ldots, A_n)$ are only connected through the "do part" (A_1). $B^{start} = \{b \mid (a,b) \in F \ \wedge \ a \in A_1 \ \wedge \ b \notin A_1\}$ are the start activities of the "redo parts" connected to end activities in the "do part" and $B^{end} = \{b \mid (b,a) \in F \ \wedge \ a \in A_1 \ \wedge \ b \notin A_1\}$ are the end activities of the "redo parts" connected to start activities in the "do part". The requirements in Definition 23 imply that $A^{end} \times B^{start} \subseteq F$ and $B^{end} \times A^{start} \subseteq F$. This implies that all end activities of the "do part" are connected to all start activities of the "redo parts" and all end activities of the "redo parts" are connected to all start activities of the "do part". For more explanations, see [1].

How the event log L is decomposed into $L_1, L_2, \ldots, L_n$ based on $\oplus$-cut $(\oplus, A_1, A_2, \ldots A_n)$ depends on the type of cut $\oplus \in \{\rightarrow, \times, \wedge, \circlearrowleft\}$. In all log decompositions, each event ends up in precisely one event log, i.e., the number of events remains invariant through decomposition. We use the previously introduced event logs to illustrate this.

First, we consider $L_1 = [\langle a,b,c,e\rangle^{10}, \langle a,c,b,e\rangle^5, \langle a,d,e\rangle]$ and construct the corresponding DFG to find one of the four cuts. We check the presence of a cut using the order in Definition 23, i.e., (1) $\times$, (2) $\rightarrow$, (3) $\wedge$, (4) $\circlearrowleft$. There is no exclusive-choice cut for L_1, but there is a sequence cut $(\rightarrow, \{a\}, \{b,c,d\}, \{e\})$. Using this cut, L_1 is split into $L_a = [\langle a\rangle^{16}]$, $L_{b,c,d} = [\langle b,c\rangle^{10}, \langle c,b\rangle^5, \langle d\rangle]$, and $L_e = [\langle e\rangle^{16}]$. L_a and L_e correspond to base cases since there is just one activity left: L_a is modeled by a single occurrence of activity a, and L_e is modeled by a single occurrence of activity e. Hence, the process tree starts with $\rightarrow(a, ?, e)$, where $?$ corresponds to the subtree describing $L_{b,c,d}$. Next, we create a DFG for $L_{b,c,d}$ and see that we can apply an exclusive-choice cut $(\times, \{b,c\}, \{d\})$. Using this cut, $L_{b,c,d}$ is split into $L_{b,c} = [\langle b,c\rangle^{10}, \langle c,b\rangle^5]$ and $L_d = [\langle d\rangle]$. L_d corresponds to a base case since there is just one activity left. Hence, the subtree for $L_{b,c,d}$ has the following structure $\times(?, d)$, where $?$ corresponds to the subtree describing $L_{b,c}$. The overall tree created thus far is $\rightarrow(a, \times(?, d), e)$. Next, we create a DFG for $L_{b,c}$ and see that we can apply a parallel cut $(\wedge, \{b\}, \{c\})$. It is not possible to apply an exclusive-choice cut or a sequence cut. Using cut $(\wedge, \{b\}, \{c\})$ sublog $L_{b,c}$ is split into $L_b = [\langle b\rangle^{15}]$ and $L_c = [\langle c\rangle^{15}]$. Both correspond to base cases. Hence, the subtree for $L_{b,c}$ is $\wedge(b,c)$. The overall tree is $\rightarrow(a, \times(\wedge(b,c), d), e)$. This is process tree Q_1 in Fig. 10(a) shown before.

Next, we consider $L_2 = [\langle a,b,c,e\rangle^{50}, \langle a,c,b,e\rangle^{40}, \langle a,b,c,d,b,c,e\rangle^{30}, \langle a,c,b,d,b,c,e\rangle^{20}, \langle a,b,c,d,c,b,e\rangle^{10}, \langle a,c,b,d,c,b,d,b,c,e\rangle^{10}]$. Again, we construct the corresponding DFG to find one of the four cuts. The first cut we find is a sequence cut $(\rightarrow, \{a\}, \{b,c,d\}, \{e\})$. Using this cut, L_2 is split into $L_a = [\langle a\rangle^{160}]$, $L_{b,c,d} = [\langle b,c\rangle^{50}, \langle c,b\rangle^{40}, \langle b,c,d,b,c\rangle^{30}, \langle c,b,d,b,c\rangle^{20}, \langle b,c,d,c,b\rangle^{10}, \langle c,b,d,c,b,d,b,c\rangle^{10}]$, and $L_e = [\langle e\rangle^{160}]$. L_a and L_e correspond to base cases suggesting that the process has the following structure $\rightarrow(a, ?, e)$, with $?$ corresponding to the subtree describing $L_{b,c,d}$. Again we check the presence of a cut. The first cut we find is the redo loop cut $(\circlearrowleft, \{b,c\}, \{d\})$.

Using this cut, $L_{b,c,d}$ is split into $L_{b,c} = [\langle b,c\rangle^{150}, \langle c,b\rangle^{90}]$ and $L_d = [\langle d\rangle^{80}]$. Note that $L_{b,c}$ has 240 cases because the "do part" happened $50+40+(2\times 30)+(2\times 20)+(2\times 10)+(3\times 10) = 240$ times. The "redo part" happened $30+20+10+(2\times 10) = 80$ times. The redo part is trivial since d is always executed once. Hence, the subtree for $L_{b,c,d}$ has the following structure $\circlearrowleft(?, d)$, where ? corresponds to the subtree describing $L_{b,c}$. For $L_{b,c}$, we find the subtree $\wedge(b,c)$. The overall tree is, therefore, $\rightarrow(a, \circlearrowleft(\wedge(b,c), d), e)$. This is process tree Q_2 in Fig. 10(b) shown before.

To explain the Alpha algorithm, we also used L_4 and L_5 in Fig. 15. Applying the basic inductive mining algorithm to $L_4 = [\langle a,b\rangle^{35}, \langle b,a\rangle^{15}]$ yields the process tree $\wedge(a,b)$. For $L_5 = [\langle a\rangle^{10}, \langle a,b\rangle^{8}, \langle a,c,b\rangle^{6}, \langle a,c,c,b\rangle^{3}, \langle a,c,c,c,b\rangle]$, we find the process tree $\rightarrow(a, \circlearrowleft(\tau, c), \times(b, \tau))$. Note that the subtree $\circlearrowleft(\tau, c)$ is created for the sublog involving just c, because c happens 0, 1, 2, or 3 times. The subtree $\times(b,\tau)$ is created for the sublog involving just b, because b happens at most once.

It is possible that none of the cuts in Definition 23 can be applied while the sublog still has multiple activities. In this case, one can always apply so-called *fallthroughs*, e.g., use $\circlearrowleft(\tau, a_1, a_2, \ldots, a_n)$ that allows for any behavior. Note that such fallthroughs are not needed when the original process was expressible in terms of a process tree (for the exact conditions, see [1,22]). Moreover, it is also possible to use smarter fallthroughs that separate the problematic activities or behavior from the rest. Suppose that there is a cut $(\oplus, A_1, A_2, \ldots A_k)$ possible considering only activities $A_{good} = A_1 \cup A_2 \cup \ldots \cup A_k$ and leaving out $A_{bad} = A \backslash A_{good} = \{a_1, a_2, \ldots, a_n\}$. Then one can first apply the parallel cut $(\wedge, A_{good}, A_{bad})$ followed by cut $(\oplus, A_1, A_2, \ldots A_k)$ and cut $\circlearrowleft(\tau, a_1, a_2, \ldots, a_n)$ applied to the two sublogs. There are many other fallthroughs, e.g., separating the empty traces from the rest.

Definition 24 (Inductive Mining Algorithm). *The basic inductive mining algorithm* $disc_{IM} \in \mathcal{B}(\mathcal{U}_{act}{}^*) \rightarrow \mathcal{U}_Q$ *returns a process tree* $disc_{IM}(L)$ *for any event log* $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ *using the four types of cuts, log decomposition, and fallthroughs described before.*

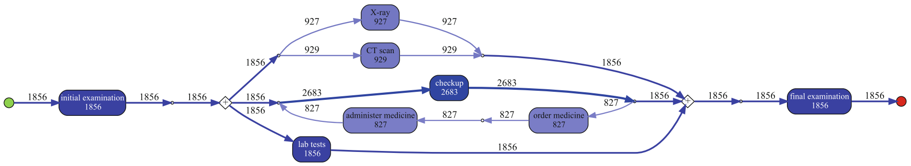


Fig. 18. Process tree $disc_{IM}(L_3) = \rightarrow(ie, \wedge(\times(xr, ct), \circlearrowleft(cu, \rightarrow(om, am)), lt), fe)$ discovered and visualized using ProM's Inductive Visual Miner.

Earlier, we introduced event log L_3, containing 11761 events corresponding to 1856 cases. Using the following abbreviations ie = initial examination, xr = X-ray, ct = CT scan, cu = checkup, om = order medicine, am = administer medicine, lt = lab tests, and fe = final examination, we find $disc_{IM}(L_3) = \rightarrow(ie, \wedge(\times(xr, ct), \circlearrowleft(cu, \rightarrow(om, am)), lt), fe)$. Figure 18 shows a screenshot of ProM's Inductive Visual

Miner while analyzing $disc_{IM}(L_3)$ using a BPMN-like notation. No fallthroughs were needed. Note that also the frequencies are shown. It is also possible to show timing information, e.g., average waiting times.

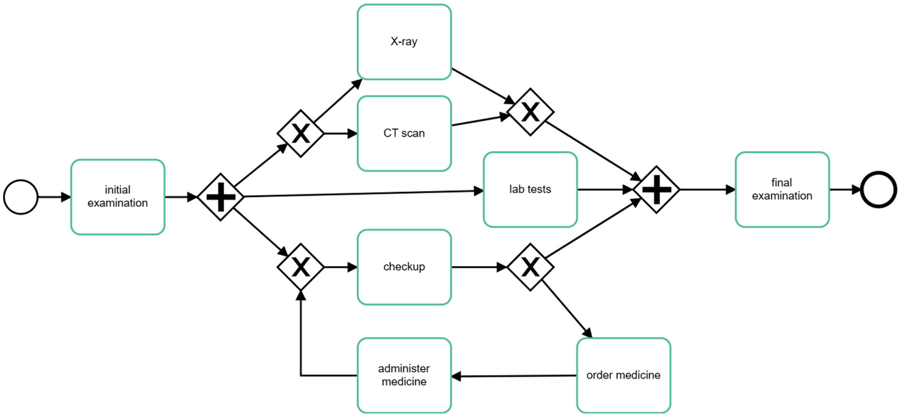


Fig. 19. Process tree $disc_{IM}(L_3) = \rightarrow(ie, \wedge(\times(xr, ct), \circlearrowleft(cu, \rightarrow(om, am)), lt), fe)$ discovered and visualized as a BPMN model using the Celonis EMS.

Figure 19 shows $disc_{IM}(L_3)$ discovered using Celonis. Celonis also uses a BPMN-like visualization of the process tree. The translation of process trees to BPMN or Petri nets is rather straightforward, and the resulting models are easier to interpret by most users.

In this section, we only introduced the basic inductive mining algorithm. We assume that the event log was filtered in advance to remove infrequent behavior. However, there are also extended versions of the inductive mining algorithm dealing with infrequent behavior [23]. The basic inductive mining algorithm may become intractable for huge event logs, because repeatedly sublogs need to be created. There are also more scalable variants that make a single pass through the event log and use a single overall DFG [24]. These provide fewer formal guarantees. The basic inductive mining algorithm has strong guarantees. For example, $disc_{IM}(L)$ guarantees perfect replay fitness (i.e., 100% recall). Formally, $var(L) \subseteq lang(disc_{IM}(L))$. See [22–24] for additional formal guarantees provided by these top-down approaches.

Next two the process discovery techniques presented this chapter, there are dozens of other techniques. In [12] additional techniques are presented.

7 Conclusion

The goal of this chapter is to introduce the foundations of process discovery without aiming to provide a complete survey or details on specific algorithms (see also [10]). After reading this chapter, it should be clear that process discovery is a challenging topic with many competing requirements. We started by introducing a baseline approach that produces a *Directly-Follows Graph* (DFG) for an event log converted into a multiset of traces. For real-life event logs, the DFG may have an excessive number of arcs making the model incomprehensible. Therefore, we discussed three *filtering* approaches that can also be combined to create simpler DFGs. We also showed that the interpretation of such process models highly depends on the log preprocessing [2].

After presenting the baseline DFG discovery approach, we focused on process representations able to capture *concurrency*: Petri nets, process trees, and BPMN models. This is needed because, if activities do not occur in a fixed order due to concurrency, then the discovered DFGs are underfitting and contain many loops. This allowed us to introduce more advanced process discovery approaches. We characterized these as (1) *bottom-up* approaches and (2) *top-down* approaches. Bottom-up approaches try to find local process patterns constraining the process model to better fit the event log. Top-down approaches tackle the problem differently and try to partition larger event logs into smaller ones that can be analyzed more easily. Two representative approaches we described in more detail: the *Alpha algorithm* and the *inductive mining algorithm*. These should be seen as representative examples of both categories. However, there are dozens of process discovery techniques, and it is impossible to name them all.

For example, there exist many extensions of the Alpha algorithm, e.g., variants that can discover silent transitions (e.g., skipping) [34] and non-free choice constructs (e.g., long-term dependencies) [33]. The heuristic mining approach [32] can be seen as another bottom-up approach that incorporates frequency information. The approach can discover complex process structures, but often leads to models that are not sound. Region-based process-discovery approaches provide formal guarantees, but are often not very applicable (e.g., they may produce huge and overfitting process models or take too long to compute). There are two types of regions: *state-based regions* (which require the construction of a transition system) and *language-based regions* (that work on sets of traces). State-based regions were introduced by Ehrenfeucht and Rozenberg [20] in 1989 and generalized by Cortadella et al. [16]. In [8], it is shown how these state-based regions can be applied to process mining by first creating a log-based transition system using different abstractions. In [14,30], refinements are proposed to tailor state-based regions towards process discovery. In parallel, several authors applied language-based regions to process mining [13,35,37]. There are also numerous bottom-up approaches combining different ideas. An example is the so-called split-miner [11] which aims to balance recall and precision. This approach also starts from a filtered DFG, but identifies combinations of splits that capture the concurrency, conflict and causal relations between neighbors in the DFG. As mentioned, there also exist different variants of the inductive mining approach presented in this chapter [22–24].

In this chapter, we only considered a simple event log $L \in \mathcal{B}({\mathcal{U}_{act}}^*)$, ignoring additional event and case attributes (e.g., resources, data, transactional information). However, other logging formats may be considered. There are process discovery approaches that exploit timing information, data attributes, object references, partial order information (e.g., events happening on the same day), explicit uncertainty (e.g., imprecise timestamps or missing case identifiers), etc. We also only focused on mainstream representations such as DFGs, Petri nets, and BPMN. However, there are also discovery techniques that aim to discover stochastic process models [29], declarative process models (using Declare or DCR graphs) [25], or object/artifact-centric models (e.g., object-centric Petri nets) [5,21].

The above illustrates that the topic of process discovery has many facets, providing interesting scientific challenges. Moreover, there are several open-source tools (e.g., ProM, bupaR, PM4Py, and RapidProM) and over 40 commercial process mining tools (e.g., Celonis, Disco/Fluxicon, Lana/Appian, Minit, Apromore, myInvenio/IBM, PAFnow, Signavio/SAP, Timeline/Abby and ProcessGold/UiPath) that already provide solid discovery approaches, and are sometimes applied to processes with billions of events. However, as applications of process mining become more demanding, new discovery approaches are needed that are better scalable and can deal with more complex processes and data structures. Therefore, process discovery is not just a great research topic, but also of great practical relevance.

Acknowledgment. Funded by the Alexander von Humboldt (AvH) Stiftung and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2023 Internet of Production – 390621612.

References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer, Berlin (2016). https://doi.org/10.1007/978-3-662-49851-4
2. van der Aalst, W.M.P.: A practitioner's guide to process mining: limitations of the directly-follows graph. In: International Conference on Enterprise Information Systems (Centeris 2019), Volume 164 of Procedia Computer Science, pp. 321–328. Elsevier (2019)
3. van der Aalst, W.M.P.: Process mining: a 360 degrees overview. In: van der Aalst, W.M.P., Carmona, J. (eds.) PMSS 2022. LNBIP, vol. 448, pp. 3–34. Springer, Cham (2022)
4. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Min. Knowl. Discov. **2**(2), 182–192 (2012)
5. van der Aalst, W.M.P., Berti, A.: Discovering object-centric Petri nets. Fund. Inform. **175**(1–4), 1–40 (2020)
6. van der Aalst, W.M.P., et al.: Soundness of workflow nets: classification, decidability, and analysis. Formal Aspects Comput. **23**(3), 333–363 (2011). https://doi.org/10.1007/s00165-010-0161-4
7. van der Aalst, W.M.P., De Masellis, R., Di Francescomarino, C., Ghidini, C.: Learning hybrid process models from events. In: Carmona, J., Engels, G., Kumar, A. (eds.) BPM 2017. LNCS, vol. 10445, pp. 59–76. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65000-5_4
8. van der Aalst, W.M.P., Rubin, V., Verbeek, H.M.W., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Softw. Syst. Model. **9**(1), 87–111 (2010). https://doi.org/10.1007/s10270-008-0106-z
9. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
10. Augusto, A., et al.: Automated discovery of process models from event logs: review and benchmark. IEEE Trans. Knowl. Data Eng. **31**(4), 686–705 (2019)
11. Augusto, A., Conforti, R., Marlon, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowl. Inf. Syst. **59**(2), 251–284 (2019). https://doi.org/10.1007/s10115-018-1214-x

12. Augusto, A., Carmona, J., Verbeek, E.: Advanced process discovery techniques. In: van der Aalst, W.M.P., Carmona, J. (eds.) PMSS 2022. LNBIP, vol. 448, pp. 76–107. Springer, Cham (2022)
13. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 375–383. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_27
14. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering Petri nets from event logs. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 358–373. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85758-7_26
15. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking: Relating Processes and Models. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-99414-7
16. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri nets from finite transition systems. IEEE Trans. Comput. **47**(8), 859–882 (1998)
17. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science, vol. 40. Cambridge University Press, Cambridge (1995)
18. Desel, J., Reisig, W.: Place/transition Petri nets. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 122–173. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_15
19. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.: Fundamentals of Business Process Management. Springer, Berlin (2018). https://doi.org/10.1007/978-3-662-56509-4
20. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-structures - part 1 and part 2. Acta Informatica **27**(4), 315–368 (1989)
21. Fahland, D.: Describing behavior of processes with many-to-many interactions. In: Donatelli, S., Haar, S. (eds.) PETRI NETS 2019. LNCS, vol. 11522, pp. 3–24. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21571-2_1
22. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_17
23. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann, N., Song, M., Wohed, P. (eds.) BPM 2013. LNBIP, vol. 171, pp. 66–78. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06257-0_6
24. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery and conformance checking. Softw. Syst. Model. **17**(2), 599–631 (2018). https://doi.org/10.1007/s10270-016-0545-x
25. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 270–285. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31095-9_18
26. Mannel, L.L., van der Aalst, W.M.P.: Finding complex process-structures by exploiting the token-game. In: Donatelli, S., Haar, S. (eds.) PETRI NETS 2019. LNCS, vol. 11522, pp. 258–278. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21571-2_15
27. OMG: Business Process Model and Notation (BPMN), Version 2.0.2. Object Management Group (2014). http://www.omg.org/spec/BPMN/
28. Petri, C.A.: Kommunikation mit Automaten. Ph.D. thesis, Institut für instrumentelle Mathematik, Bonn (1962)
29. Rogge-Solti, A., van der Aalst, W.M.P., Weske, M.: Discovering stochastic Petri nets with arbitrary delay distributions from event logs. In: Lohmann, N., Song, M., Wohed, P. (eds.)

BPM 2013. LNBIP, vol. 171, pp. 15–27. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06257-0_2

30. Solé, M., Carmona, J.: Process mining from a basis of state regions. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 226–245. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13675-7_14
31. Syring, A.F., Tax, N., van der Aalst, W.M.P.: Evaluating conformance measures in process mining using conformance propositions. In: Koutny, M., Pomello, L., Kristensen, L.M. (eds.) Transactions on Petri Nets and Other Models of Concurrency XIV. LNCS, vol. 11790, pp. 192–221. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-662-60651-3_8
32. Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering workflow models from event-based data using little thumb. Integr. Comput.-Aided Eng. **10**(2), 151–162 (2003)
33. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Min. Knowl. Disc. **15**(2), 145–180 (2007). https://doi.org/10.1007/s10618-007-0065-y
34. Wen, L., Wang, J., van der Aalst, W.M.P., Huang, B., Sun, J.: Mining process models with prime invisible tasks. Data Knowl. Eng. **69**(10), 999–1021 (2010)
35. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. Fundam. Informaticae **94**, 387–412 (2010)
36. Weske, M.: Business Process Management: Concepts, Languages, Architectures, 3rd edn. Springer, Berlin (2019). https://doi.org/10.1007/978-3-642-28616-2
37. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P., Verbeek, H.M.W.: Discovering workflow nets using integer linear programming. Computing **100**(5), 529–556 (2018). https://doi.org/10.1007/s00607-017-0582-5

Advanced Process Discovery Techniques

Adriano Augusto[1], Josep Carmona[2], and Eric Verbeek[3](✉)

[1] The University of Melbourne, Melbourne, Australia
[2] Universitat Politècnica de Catalunya, Barcelona, Spain
[3] Eindhoven University of Technology, Eindhoven, The Netherlands
h.m.w.verbeek@tue.nl

Abstract. Given the challenges associated to the process discovery task, more than a hundred research studies addressed the problem over the past two decades. Despite the richness of proposals, many state-of-the-art automated process discovery techniques, especially the oldest ones, struggle to systematically discover accurate and simple process models. In general, when the behavior recorded in the input event log is simple (e.g., exhibiting little parallelism, repetitions, or inclusive choices) or noise free, some basic algorithms such as the alpha miner can output accurate and simple process models. However, as the complexity of the input data increases, the quality of the discovered process models can worsen quickly. Given that oftentimes real-life event logs record very complex and unstructured process behavior containing many repetitions, infrequent traces, and incomplete data, some state-of-the-art techniques turn unreliable and not purposeful. Specifically, they tend to discover process models that either have limited accuracy (i.e., low fitness and/or precision) or are syntactically incorrect. While currently there exists no perfect automated process discovery technique, some are better than others when discovering a process model from event logs recording complex process behavior. In this chapter, we introduce four of such techniques, discussing their underlying approach and algorithmic ideas, reporting their benefits and limitation, and comparing their performance with the algorithms introduced in the previous chapter.

1 Introduction

The previous chapter has introduced the alpha algorithm and the inductive mining algorithm as basic algorithms that discover an accepting Petri net from a (simplified) event log. It has also shown a number of example event logs for which these two basic algorithms work excellently. However, these two basic algorithms do not always perform well, often depending on the characteristics of the given event log.

In this chapter, we first introduce an example event log where the recorded process behavior features intertwined parallel compositions and exclusive choices. Second, we discuss the results of the alpha algorithm and the inductive mining algorithm on this example event log, showing that there is room for improvement. Third, we introduce four advanced process mining algorithms, discussing the results of using these algorithms on the example event log – highlighting their benefits and limitations. The first two advanced algorithms use region-based techniques to discover accepting Petri nets,

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 76–107, 2022.
https://doi.org/10.1007/978-3-031-08848-3_3

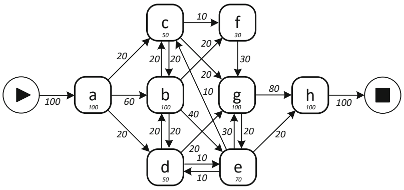


Fig. 1. The directly-follows graph of the event log L_6.

where the first algorithm uses state-based regions and the second uses language-based regions. The third algorithm relies on sophisticated approaches to pre-process the DFG prior the identification of splits and joins behavioral semantics, and it natively outputs BPMN models. Whereas these three algorithms produce imperative process models, the fourth algorithm generates declarative process models (like Declare) called *log skeletons*. As we shall see in this chapter, thanks to their advanced approaches, these mining algorithms are capable of handling event logs recording very complex process behavior better than the basic mining algorithms do. At the same time, also these algorithms should not be considered bullet-proof solutions for addressing exercises of automated process discovery as, in general, their results vary depending on the input event log.

2 Motivation

For motivating the need for advanced process discovery algorithm, we introduce the event log $L_6 = [\langle a,b,c,g,e,h\rangle^{10}, \langle a,b,c,f,g,h\rangle^{10}, \langle a,b,d,g,e,h\rangle^{10}, \langle a,b,d,e,g,h\rangle^{10}, \langle a,b,e,c,g,h\rangle^{10}, \langle a,b,e,d,g,h\rangle^{10}, \langle a,c,b,e,g,h\rangle^{10}, \langle a,c,b,f,g,h\rangle^{10}, \langle a,d,b,e,g,h\rangle^{10}, \langle a,d,b,f,g,h\rangle^{10}]$. At first sight, there seems to be a choice between c and d, followed by a choice between e and f. However, it is more complicated than that, as traces like $\langle a,c,b,g,e,h\rangle$ and $\langle a,b,d,f,g,h\rangle$ are not included in L_6.

Figure 1 shows the DFG that results from the event log L_6. Clearly, this DFG is not as symmetric as we would have thought after a first glance at L_6. For example, e can be directly followed by c or d, but f is always directly followed by g.

Figure 2 shows the accepting Petri net that results from running the alpha algorithm on event log L_6. The places with the $>$ sign are places with a larger inflow than outflow, whereas the places with the $<$ symbol are places with a smaller inflow than outflow. This is a clear indication that this net has quality problems, which is also confirmed by the fact that in this net the final marking is not reachable from the initial marking. It is possible to put a token in the final place, but then there would be other tokens in the net as well. Precisely, there would be tokens in the place that is the output of a and e and the input of c.

Figure 3 shows the process tree that results from running the inductive mining algorithm on event log L_6. Although the process tree guarantees that the final marking is always reachable from the initial marking, this process tree allows for too much behavior. As an example it is possible to do both e and f, or neither, even though in L_6 always exactly one of these two activities is observed per trace. Also the fact that f is always directly followed by g is not captured by this process tree.

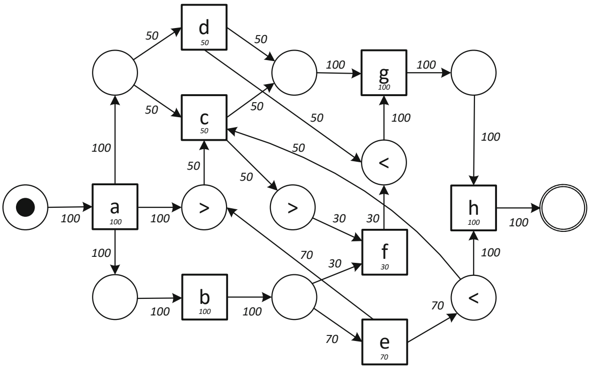


Fig. 2. The accepting Petri net discovered by the alpha algorithm from the event log L_6.

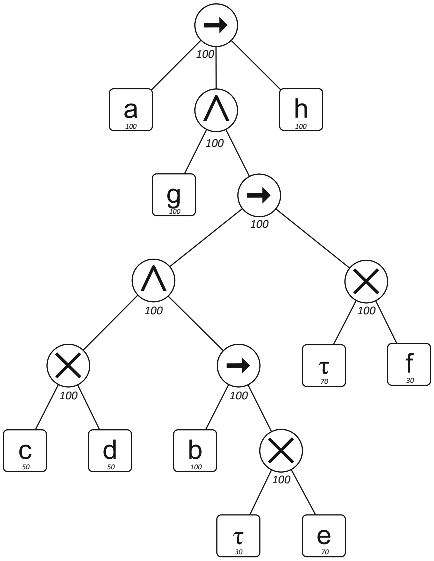


Fig. 3. The process tree discovered by the Inductive Mining Algorithm for event log L_6.

This shows that, for more complex event logs, we need more advanced algorithms than the alpha algorithm and the inductive mining algorithm. This chapter, introduces four of such advanced algorithms each having more success in discovering a process model from the event log L_6 than the basic algorithms from the previous chapter, they are:

1. The State-based Region Miner, which produces accepting Petri nets like the basic algorithms do;
2. The Language-based Region Miner, which also produces accepting Petri nets;
3. The Split Miner, which produces BPMN models;

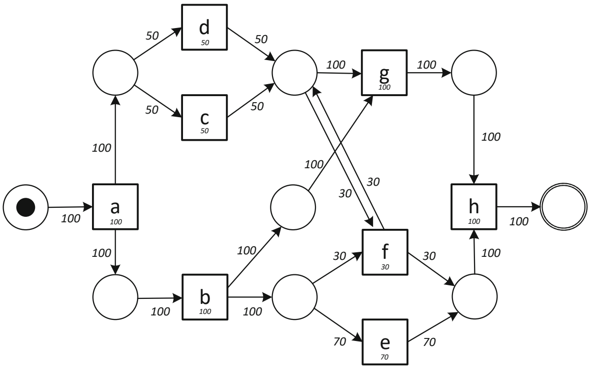


Fig. 4. The accepting Petri net discovered by the State-based Region Algorithm for event log L_6.

4. The Log Skeleton Miner, which produces declarative process models (like Declare [45]) called *log skeletons*.

These four advanced algorithms are discussed in the next sections, as the first two algorithms both use the theory of regions, they are discussed in a single section. Then, we continue with split miner, and lastly we conclude with the log skeleton miner.

3 The Theory of Regions

The *theory of regions* [30] was proposed in the early nineties to define a formal correspondence between behavior and structure. In particular, several region-based algorithms have been proposed in the last decades to synthesize specifications into Petri nets using this powerful theory.

As mining is a form of synthesis, several approaches have appeared to mine process models from event logs. Regardless of the region based technique applied, the approaches that rely on the notion of region theory search for a process model that is both fitting and precise [17]. This section shows two branches of region-based approaches for process discovery: state and language-based approaches.

3.1 State-Based Region Approach for Process Discovery

Figure 4 shows the accepting Petri net that results from running the State-based Region Algorithm on event log L_6. Note that for all places the inflow equals the outflow. In the remainder of this section we will provide an overview of the main ingredients of state-based region discovery.

State-based region approaches for process discovery need to convert the event log into a state-based representation, that will be used to discover the Petri net. This representation, is formalized in the following definition.

Definition 1 (Transition System). *A* transition system *(*TS*) is a tuple* (S, Σ, A, s_{in})*, where* S *is a set of* states*,* Σ *is an alphabet of* activities*,* $A \subseteq S \times \Sigma \times S$ *is a set of* (labeled) arcs*, and* $s_{in} \in S$ *is the* initial state*. We will use* $s \xrightarrow{e} s'$ *as a shortcut for* $(s, e, s') \in A$*, and the transitive closure of this relation will be denoted by* $\xrightarrow{*}$*.*

Figure 5(a) presents an example of a transition system.

Definition 2 (Multiset representation of traces). *We denote by* $\#(\sigma, \mathsf{e})$ *the number of times that event* e *occurs in* σ*, that is* $\#(\langle \mathsf{e}_1 \ldots \mathsf{e}_n \rangle, \mathsf{e}) = |\{\mathsf{e}_i \mid \mathsf{e}_i = \mathsf{e}\}|$*. Given an alphabet* Σ*, the* Parikh vector of a sequence σ *with respect to* Σ *is a vector* $p_\sigma \in \mathbb{N}^{|\Sigma|}$ *such that* $p_\sigma(\mathsf{e}) = \#(\sigma, e)$*.*

The techniques described in [62] present different variants for generating a transition system from an event log. For the most common variant, the basic idea to incorporate state information is to look at the set of multiset of events included in a subtrace in the event log:

Definition 3 (Multiset State Representation of an Event Log). *Given an event log* $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$*, the* TS *corresponding to the* multiset *conversion of* L*, denoted as* $\mathsf{TS}_{mset}(L)$*, is* $\langle S, \Sigma, T, \mathsf{s}_{p_\epsilon} \rangle$*, such that:* S *contains one state* s_{p_w} *for each Parikh vector* p_w *of a prefix* w *in* L*, with* ϵ *denoting the empty prefix, and* $T = \{\mathsf{s}_{p_w} \xrightarrow{\mathsf{e}} \mathsf{s}_{p_{w\mathsf{e}}} \mid w\mathsf{e}$ *is a prefix of* $L\}$*.*

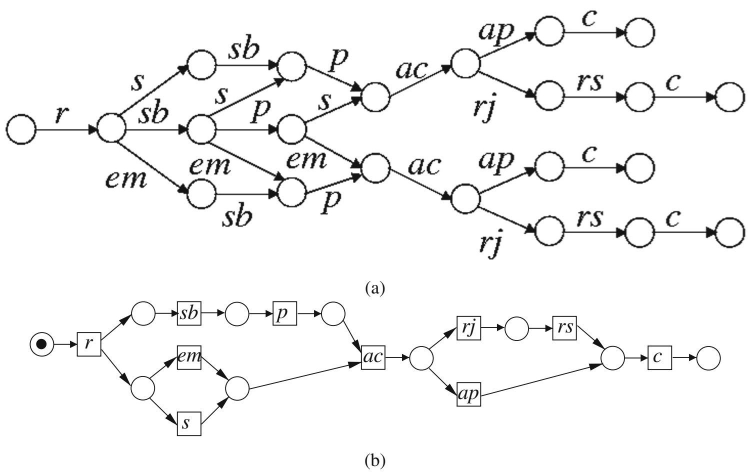
In the *sequence* conversion, two traces lead to the same state if they fire the same events in exactly the same order.

Example 1. Let us use along this section an example extracted from [61]. The event log contains the following activities: *r=register*, *s=ship*, *sb=send_bill*, *p=payment*, *ac=accounting*, *ap=approved*, *c=close*, *em=express_mail*, *rj=rejected*, and *rs=resolve*. Given the event log $L_7 = [\langle r, s, sb, p, ac, ap, c\rangle^{10}, \langle r, sb, em, p, ac, ap, c\rangle^{10}, \langle r, sb, p, em, ac, rj, rs, c\rangle^{10}, \langle r, em, sb, p, ac, ap, c\rangle^{10}, \langle r, sb, s, p, ac, rj, rs, c\rangle^{10}, \langle r, sb, p, s, ac, ap, c\rangle^{10}, \langle r, sb, p, em, ac, ap, c\rangle^{10}]$, Fig. 5(a) show an example of TS constructed according to Definition 3.

A *region*[1] in a transition system is a set of states that satisfy an homogeneous relation with respect to the set of arcs. In the simplest case, this relation can be described by a predicate on the set of states considered. Formally:

Definition 4 (Region). *Let* S' *be a subset of the states of a* TS*,* $S' \subseteq S$*. If* $s \notin S'$ *and* $s' \in S'$*, then we say that transition* $s \xrightarrow{a} s'$ enters S'*. If* $s \in S'$ *and* $s' \notin S'$*, then transition* $s \xrightarrow{a} s'$ exits S'*. Otherwise, transition* $s \xrightarrow{a} s'$ does not cross S'*: it is completely inside (*$s \in S'$ *and* $s' \in S'$*) or completely outside (*$s \notin S'$ *and* $s' \notin S'$*). A set of states* $r \subseteq S$ *is a region iff for each event* $\mathsf{e} \in E$*, exactly one of the three predicates (*enters, exits *or* does not cross*) holds for each of its arcs.*

[1] In this paper we will use region to denote a 1-bounded region. However, when needed we will use k-bounded region to extend the notion, necessary to account for k-bounded Petri nets.



(a)

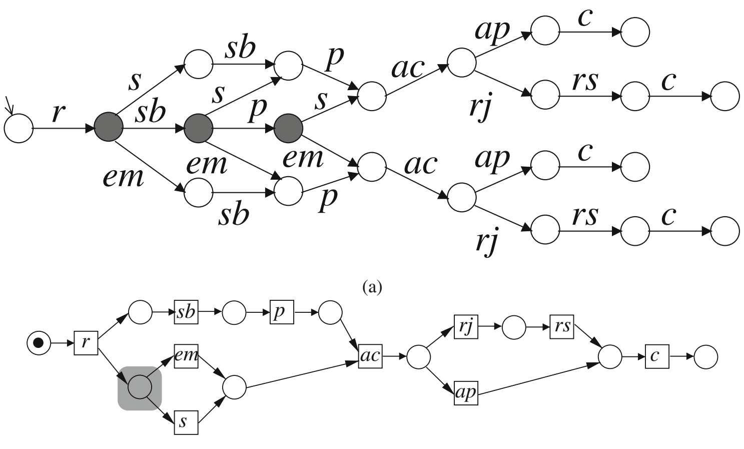
(b)

Fig. 5. State-based region discovery: (a) transition system corresponding to L_7, (b) derived Petri net.

An example of region is presented in Fig. 6 on the TS of our running example. In the highlighted region, event r enters the region, s and em exit the region, and the rest of labels do not cross the region.

A region corresponds to a place in the Petri net, and the role of the arcs determine the Petri net flow relation: when an event e enters the region, there is an arc from the corresponding transition for e to the place, and when e exits the region, there is an arc from the region to the transition for e. Events satisfying the do not cross relation are not connected to the corresponding place. For instance, the region shown in Fig. 6(a) corresponds to the shadowed place in Fig. 6(b), where event r belongs to the set of input transitions of the place whereas events em and s belong to the set of output transitions. Hence, the algorithm for Petri net derivation from a transition system consists in finding regions and constructing the Petri net as illustrated with the previous example. In [26] it was shown that only a minimal set of regions was necessary, whereas further relaxations to this restriction can be found in [17]. The Petri net obtained by this method is guaranteed to accept the language of the transition system, and satisfy the *minimal language containment property*, which implies that if all the minimal regions are used, the Petri net derived is the one whose language difference with respect to the log is minimal, hence being the most precise Petri net for the set of transitions considered.

In any case, the algorithm that searches for regions in a transition system must explore the lattice of sets (or multisets, in the case for k-*bounded* regions), thus having a high complexity: for a transition system with n states, the lattice for k-bounded regions is of size $\mathcal{O}(k^n)$. For instance, the lattice of sets of states for the toy TS used in this article (which has 22 states) has 2^{22} possibles sets to check for the region conditions. Although many simplification properties, efficient data structures and algorithms, and heuristics are used to prune this search space [17], they only help to alleviate the problem. Decomposition alternatives, which for instance use partitions of the state



(a)

(b)

Fig. 6. (a) Example of region (three shadowed states). The predicates are r enters, s and em exits, and the rest of events do not cross, (b) Corresponding place shadowed in the Petri net.

space to guide the search for regions, significantly alleviate the complexity of the state-based region algorithm, at the expense of not guaranteeing the derivation of precise models [15]. Other state-based region approaches for discovery have been proposed, which complement the approach described in this section [54–56].

3.2 Language-Based Region Approach for Process Discovery

In language-based region theory [6,8,9,22,37,38] the goal is to construct the smallest Petri net such that the behaviour of the net is equal to the given input language (or minimally larger). [41] provides an overview for language-based region theory for different classes of languages: step languages, regular languages, and (infinite) partial languages.

Figure 7 shows the accepting Petri net that results from running the Language-based Region Algorithm on event log L_6. As it happened with state-base regions, again for all places the inflow equals the outflow.

More formally, let $L \in \mathcal{B}(\mathcal{U}_{act}{}^{*})$ be an event log, then language based region theory constructs a Petri net with the set of transitions equals to Σ and in which all traces of L are a firing sequence. The Petri net should have only minimal firing sequences not in the language L (and all prefixes in L). This is achieved by adding places to the Petri net that restrict unobserved behavior, while allowing for observed behavior. The theory of regions provides a method to identify these places, using *language regions*.

Definition 5 (Prefix Closure). *Let $L \in \mathcal{B}(\mathcal{U}_{act}{}^{*})$ be an event log. The prefix closed language $\mathcal{L} \subseteq \Sigma^*$ of L is defined as: $\mathcal{L} = \{\sigma \in \Sigma^* \mid \exists_{\sigma' \in \Sigma^*} \sigma \circ \sigma' \in L\}$.*

The prefix closure of a log is simply the set of all prefixes in the log (including the empty prefix).

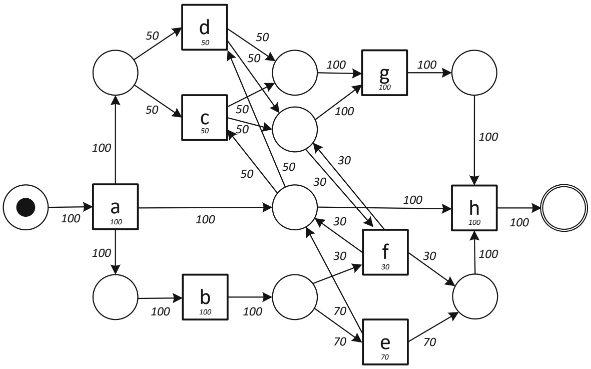


Fig. 7. The accepting Petri net discovered by the Language-based Region Algorithm for event log L_6.

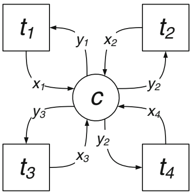


Fig. 8. Region for a language over four activities [63].

Definition 6 (Language Region). *Let Σ be a set of activities. A region of a prefix-closed language $\mathcal{L} \in \Sigma^*$ is a triple $(\vec{x}, \vec{y}, c)$ with $\vec{x}, \vec{y} \in \{0,1\}^{\Sigma}$ and $c \in \{0,1\}$, such that for each non-empty sequence $w = w' \circ a \in \mathcal{L}$, $w' \in \mathcal{L}$, $a \in \Sigma$:*

$$c + \sum_{t \in \Sigma} \left(\vec{w'}(t) \cdot \vec{x}(t) - \vec{w}(t) \cdot \vec{y}(t) \right) \geq 0$$

This can be rewritten into the inequation system:

$$c \cdot \vec{1} + M' \cdot \vec{x} - M \cdot \vec{y} \geq \vec{0}$$

where M and M' are two $|\mathcal{L}| \times |\Sigma|$ matrices with $M(w,t) = \vec{w}(t)$, and $M'(w,t) = \vec{w'}(t)$, with $w = w' \circ a$. The set of all regions of a language is denoted by $\Re(\mathcal{L})$ and the region $(\vec{0}, \vec{0}, 0)$ is called the trivial region.

Intuitively, vectors $\vec{x}, \vec{y}$ denote the set of incoming and outgoing arcs of the place corresponding to the region, respectively, and c sets if it is initially marked. Figure 8 shows a region for a language over four activities, i.e. each solution $(\vec{x}, \vec{y}, c)$ of the inequation system can be regarded in the context of a Petri net, where the region corresponds to a feasible place with preset $\{t | t \in T, \vec{x}(t) = 1\}$ and postset $\{t | t \in T, \vec{y}(t) = 1\}$, and initially marked with c tokens. Note that we do not assume arc-weights here, while the authors of [6,7,22,38] do.

Since the place represented by a region is a place which can be added to a Petri net, without disturbing the fact that the net can reproduce the language under consideration, such a place is called a *feasible* place.

Definition 7 (Feasible place). *Let $\mathcal{L}$ be a prefix-closed language over Σ and let $N = ((P, \Sigma, F), m)$ be a marked Petri net. A place $p \in P$ is called* feasible *if and only if there exists a* corresponding *region $(\vec{x}, \vec{y}, c) \in \Re(\mathcal{L})$ such that $m(p) = c$, and $\vec{x}(t) = 1$ if and only if $t \in {}^{\bullet}p$, and $\vec{y}(t) = 1$ if and only if $t \in p^{\bullet}$.*

In general, there are many feasible places for any given event log (when considering arc-weights in the discovered Petri net, there are even infinitely many). Several methods exist for selecting an appropriate subset of these places. The authors of [7,38] present two ways of finitely representing these places, namely a *basis representation* and a *separating representation*. Both representations maximize precision, i.e. they select a set of places such that the behavior of the model outside of the log is minimal.

In contrast, the authors of [63,65,66,68] focus on those feasible places that express some causal dependency observed in the event log, and/or ensure that the entire model is a connected workflow net. They do so by introducing various cost functions favouring one solution of the equation system over another and then selecting the top candidates.

3.3 Strengths and Limitations of Region Theory

The goal of region theory is to find a Petri net that perfectly describes the observed behavior (where this behavior is specified in terms of a language or a statespace). As a result the Petri nets are perfectly fitting and maximally precise. Consequently, the assumption on the input event log is that it records a *full behavioral specification*, i.e., that the input is *complete* and *noise free*. While the assumption on the output is that it is a *compact* and *exact* representation of the behavior recorded in the input event log. To this end, we note that, although in this section we have focused on safe nets, the theory of regions can represent general k-bounded Petri nets – a feature that is not yet provided by any other automated process discovery technique.

When applying region theory in the context of process mining, it is therefore very important to perform any required generalization of the behavior recorded in the input event log *before* calling region theory algorithms. For state-based regions, the challenges are in the construction of the statespace from the event log, while in language-based regions it is in the selection of the appropriate prefixes to include in the final prefix-closed language in order to ensure some level of generalization.

In the next section, we will see that split miner relaxes the requirement of having the full behavioral specification recorded in the input event log, striving to discover BPMN process models that only maximizes the *balance* between its fitness and precision.

4 Split Miner

In the following, we describe how Split Miner (hereinafter, SM) discovers a BPMN model starting from an event log. SM operates in six steps (cf. Fig. 9). In the first step, it constructs the DFG and analyses it to detect self-loops and short-loops. In the second step, it discovers concurrency relations between pairs of activities in the DFG. In the third step, the DFG is filtered by applying a filtering algorithm designed to strike balanced fitness and precision of the final BPMN model while maintaining a low control-flow complexity. The fourth and fifth steps focus (respectively) on the discovery of split and join gateways, activities having multiple outgoing edges are turned into a hierarchy of split gateways, while activities have multiple incoming edges are turned into a hierarchy of join gateways. Lastly, if any OR-joins were discovered, they are analyzed and turned (whenever possible) into either XOR-gateways or AND gateways.

Although some of the steps executed by SM are typical of basic automated process discovery techniques such as alpha miner and inductive miner (e.g., the filtering of the DFG), the steps of SM were designed to overcome the limitations of such techniques. Most notably, to increase precision without compromising fitness and/or structural complexity. Furthermore, in SM, each step can operate as a black-box, allowing for additional future improvements by redesign or enhancing a step at a time [5].

We now provide a brief overview of each step of SM in a tutorial-like fashion, by leveraging the example log $L_6 = [\langle a, b, c, g, e, h\rangle^{10}, \langle a, b, c, f, g, h\rangle^{10}, \langle a, b, d, g, e, h\rangle^{10}, \langle a, b, d, e, g, h\rangle^{10}, \langle a, b, e, c, g, h\rangle^{10}, \langle a, b, e, d, g, h\rangle^{10}, \langle a, c, b, e, g, h\rangle^{10}, \langle a, c, b, f, g, h\rangle^{10}, \langle a, d, b, e, g, h\rangle^{10}, \langle a, d, b, f, g, h\rangle^{10}]$ (introduced in Sect. 2). Given that an in-depth analysis of the algorithms behind SM would be out of the scope of this chapter and book, we refer the interested reader to the original work [3].

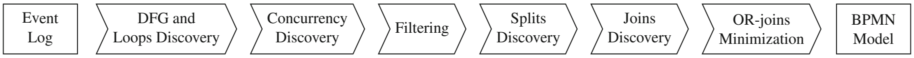


Fig. 9. Overview of the Split Miner algorithm.

4.1 Step 1: DFG and Loops Discovery

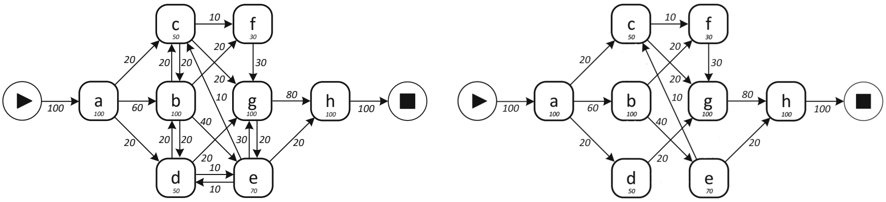
Given the input event log L_6, SM immediately builds its DFG, as shown in Fig. 10a. In this example, all the traces have the same start and end activity, however, SM automatically adds artificial start and end activities (represented by the nodes ▶ and ■).

Then, SM detects *self-loops* and *short-loops*, i.e., loops involving only one and two activities (respectively). Loops are known to cause problems when detecting concurrency [60], hence, we want to detect loops before detecting concurrency.

The simplest of the loops is the *self-loop*, a self-loop exists if a node is both source and target of one arc of the DFG, i.e., $a \rightarrow a$. Short-loops and their frequencies are detected in the log as follows. Given two activities a and b, for SM, a short-loop ($a \circlearrowright b$) exists if and only if (iff) the following two conditions hold:

i. both a and b are not self-loops;
ii. there exists at least one log trace containing the subtrace $\langle a, b, a\rangle$ or $\langle b, a, b\rangle$.

Condition (i) is necessary because otherwise the short-loop evaluation may not be reliable. In fact, if we consider a process that allows a concurrency between a self-loop activity a and a normal activity b, we could observe log traces containing the subtrace



(a) Initial DFG.

(b) After the Pruning (Step 2).

Fig. 10. Processing of the directly-follows graph.

$\langle a, b, a \rangle$, which can also characterize $a \circlearrowleft b$. Condition (ii) guarantees that we have observed (in at least one trace of the log) a short-loop between the two activities. In fact, short-loops are characterized by subtraces of the type $\langle a, b, a \rangle$ or $\langle b, a, b \rangle$.

The detected self-loops are trivially removed from the DFG and restored only in the output BPMN model. While the detected short-loops are saved and used in the next step. In our example (Fig. 10a), there are no self-loops or short-loops.

4.2 Step 2: Concurrency Discovery

Given a DFG and any two activities a and b, such that neither a nor b is a self-loop, for SM, a and b are considered concurrent (noted as $a \| b$) iff three conditions hold:

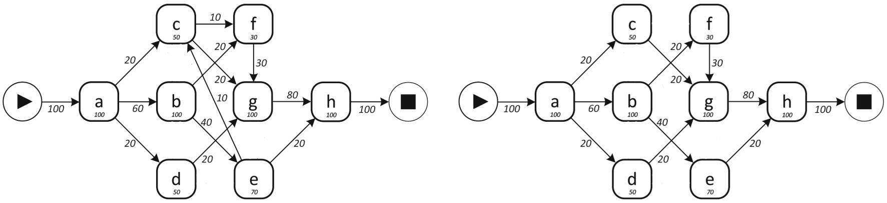
iii. there exist two arcs in the DFG: (a, b) and (b, a);
iv. both a and b are not in a short-loop;
v. the arcs (a, b) and (b, a) have similar frequency: $\frac{||a \rightarrow b| - |b \rightarrow a||}{|a \rightarrow b| + |b \rightarrow a|} < \varepsilon$ ($\varepsilon \in [0, 1]$).

These three conditions define the heuristic-based concurrency oracle of SM. The rationale behind the conditions is the following. Condition (iii) captures the basic requirement for $a \| b$: the existence of the arcs (a, b) and (b, a) entails that a and b can occur in any order. However, Condition (iii) is not sufficient to postulate concurrency because it may hold in three cases: a and b form a short-loop; (a, b) *or* (b, a) is an infrequent observation (e.g., noise in the data); a and b are concurrent. We are interested in identifying when the third case holds. To this end, we check Conditions (iv) and (v). When Condition (iv) holds, we can exclude the first case because a and b do not form a short-loop. When Condition (v) holds, we can exclude the second case because (a, b) and (b, a) are both observed frequently and have similar frequencies. At this point, we are left with the third case and we assume $a \| b$. The variable ε becomes a user input parameter, the smaller is its value the more similar have to be the number of observations of (a, b) and (b, a). Instead, setting $\varepsilon = 1$, Condition (v) would always hold.

Whenever we find $a \| b$, we remove the arcs (a, b) and (b, a) from the DFG, since we assume there is no causality but instead there is concurrency. On the other hand, if we find that either (a, b) or (b, a) represents an infrequent directly-follows relation, we remove the least frequent of the two edges. We call the output of this step a *Pruned DFG* (PDFG).

In the example in Fig. 10a, we identify four possible cases of concurrency: (b, c), (b, d), (d, e), (e, g). Setting $\varepsilon = 0.25$, we capture the following concurrency relations: $b \| c$, $b \| d$, $d \| e$, $e \| g$. The resulting PDFG is shown in Fig. 10b.

4.3 Step 3: Filtering



(a) After the Pruning (Step 2).

(b) After the Filtering (Step 3).

Fig. 11. Processing of the directly-follows graph.

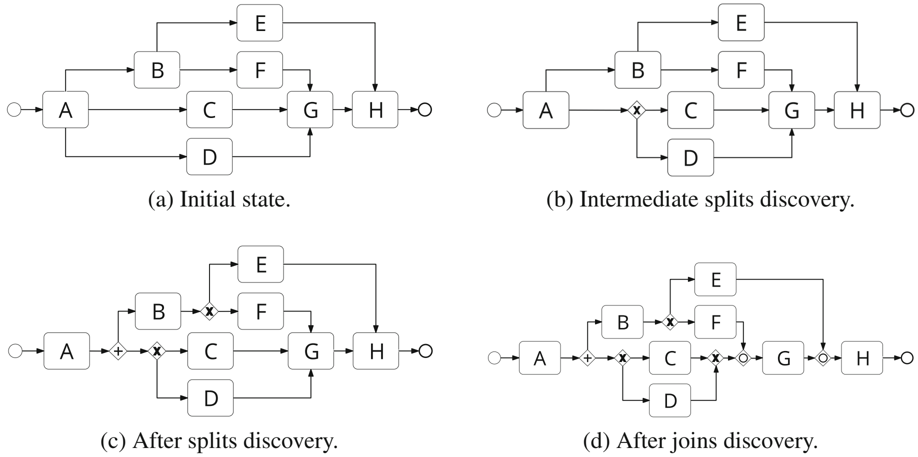
The filtering algorithm applied by SM on the PDFG is based on three criteria. First, each node of the PDFG must be on a path from the single start node (source) to the single end node (sink). Second, for each node, (at least one of) its path(s) from source to sink must be the one having maximum capacity. In our context, the capacity of a path is the frequency of the least frequent arc of the path. Third, the number of edges of the PDFG must be minimal. The three criteria aim to guarantee that the discovered BPMN process model is accurate and simple at the same time.

The filtering algorithm performs a double breadth-first exploration: forward (source to sink) and backward (sink to source). During the forward exploration, for each node of the PDFG, we discover its maximum source-to-node capacity, and its incoming edge granting such capacity (*best incoming edge*). During the backward exploration, for each node of the PDFG, we discover its maximum node-to-sink capacities, and the *best outgoing edges*. Then, we remove from the PDFG all the edges that are not best incoming edges or best outgoing edges. In doing so, we may reduce the amount of behavior that the final model can replay, and consequently its fitness. Therefore, we introduce a frequency threshold that allows the user to strike a balance fitness and precision. Precisely, we compute the η percentile over the frequencies of the best incoming and outgoing edges of each node, and we add to the PDFG the edges with a frequency exceeding the threshold. It is important to note that the percentile is not taken over the frequencies of *all* the edges, otherwise we would simply retain η percentage of all the edges. Also, this means that even by setting $\eta = 0$, SM will still apply a certain amount of filtering.

Figure 11b shows the output of the filtering algorithm when applied to the PDFG of our working example (Fig. 11a). As a consequence of retaining the best incoming and outgoing edges for each node, we would drop the arcs: (e, c) and (c, f); and they would not be retained regardless of the value assigned to η.

4.4 Step 4: Splits Discovery

Before discovering the split gateways, the filtered PDFG is converted into a BPMN process model by turning the start (▶) and end (■) nodes of the graph into the start and end events of the BPMN model, and each other node of the graph into a BPMN activity. Figure 12a shows the BPMN model[2] generated from the filtered PDFG of our working example (Fig. 11b). Now, let us focus on the discovery of the split gateways by considering the example in Fig. 13a. Given an activity with multiple outgoing edges (e.g., activity z), the splits discovery is based on the idea that all the activities directly following (successors of) the same split gateway must have the same *concurrency* and/or *mutually exclusive* relations with the activities that do not directly follow their preceding split gateway. With hindsight and reference to Fig. 13b, we see that since activities c and d are successors of gateway and_1, both c and d are concurrent to e, f, g, due to gateway and_3 (i.e., $c \| e$, $c \| f$, $c \| g$, and $d \| e$, $d \| f$, $d \| g$). At the same time, both c and d are mutually exclusive with a and b, due to gateway xor_3. Considering activities by pairs, and analyzing which concurrency or mutually exclusive relations they have in common, we can generate the appropriate splits hierarchy.



(a) Initial state.

(b) Intermediate splits discovery.

(c) After splits discovery.

(d) After joins discovery.

Fig. 12. Processing of the BPMN model.

With this in mind, we continue our working example. Let us consider activity A (Fig. 12a), it has three successors: B, C, and D. From the outcome of *Step 2*, we know that both C and D are concurrent to B, while C and D are not concurrent (hence, mutually exclusive with each other). Since C and D share the same relations to other

[2] Labels are capitalised to distinguish them from the DFG nodes.

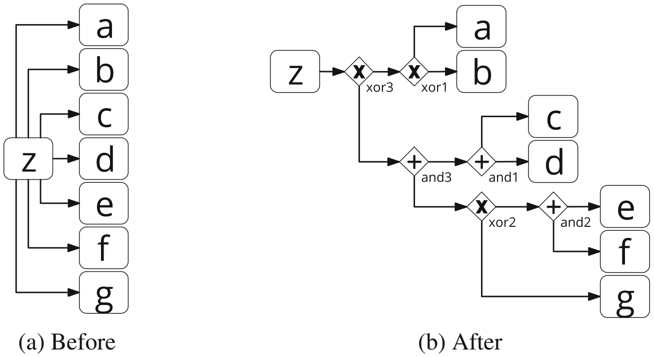


Fig. 13. Splits discovery example.

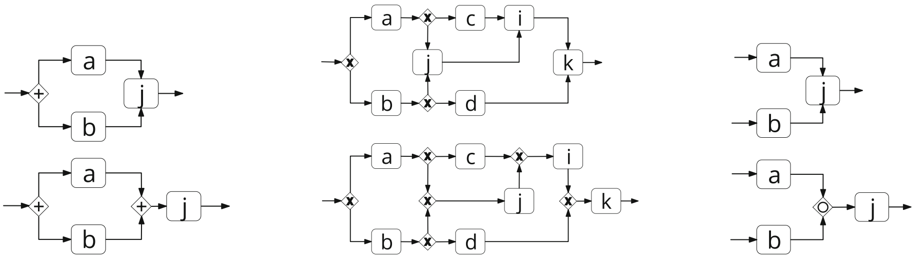
activities (both are concurrent to B), they can be selected as successors of the same gateway, which in this case would be an XOR-gateway because C and D are mutually exclusive. After we add the XOR-gateway, the successors of activity A will be two: B and the newly added XOR-gateway (see Fig. 12b). The algorithm becomes trivial when an activity with multiple outgoing edges has only two successors, indeed, it is enough to add a split gateway matching the relation between the two successors. Continuing the example of activity A, the successor B is in parallel with the newly added XOR-gateway or, more precisely, with all the activities following the XOR-gateway (activities C an D). Therefore, we can add an AND gateway preceding B and the XOR-gateway. Similarly, if we consider activity B and its two successors, activities E and F, given that they are not concurrent, they must be mutually exclusive and therefore an XOR-gateway is placed before them. The result of the splits discovery is shown in Fig. 12c.

4.5 Step 5: Joins Discovery

Once all the split gateways have been placed, we can discover the join gateways. To do so, we rely on the Refined Process Structure Tree (RPST) [46] of the current BPMN model. The RPST of a process model is a tree data structure where the tree nodes represent the single-entry single-exit (SESE) fragments of the process model, and the tree edges denote a containment relation between SESE fragments. Precisely, the children of a SESE fragment are its directly contained SESE fragments, whilst SESE fragments on different branches of the tree are disjoint. Each SESE fragment represents a *subgraph* of the process model, and the partition of the process model into SESE fragments is made in terms of edges. A SESE fragment can be of one of the following four types: a *trivial* fragment, which consists of a single edge; a *polygon*, which consists of a sequence of fragments; a *bond*, which is a fragment where all the children fragments share two common nodes, one being the entry and the other being the exit of the *bond*; and a *rigid*, which represents any other fragment. Each SESE fragment is classified as *homogeneous*, if the gateways it contains (and are not contained in any of its SESE children) are all of the same type (e.g., only XOR-gateways), or *heterogeneous* if its gateways have different types. Figure 14a and Fig. 14b show two examples of homogeneous SESE fragments: a *bond* and a *rigid*.

We note that, at this stage, in the BPMN model (Fig. 12c) all the SESE fragment's exits correspond to activities with multiple incoming edges, which we aim to turn into join gateways. Starting from the leaves of the RPST, i.e., the innermost SESE fragments of the process model, we explore the RPST bottom-up. For each SESE fragment we encounter in this exploration, we select the activities it contains that have multiple incoming edges (there is always at least one, the SESE fragment exit). For each of the selected activities, we add a join gateway preceding it. The join gateway type will depend on whether the SESE fragment is homogeneous or heterogeneous. In the former case, the join gateway will have the same type of the other gateways in the SESE fragment, in the latter case, the join gateway will be an OR-gateway. Figure 14 shows in brief how our approach works for SESE bonds (Fig. 14a), for homogeneous SESE rigids (Fig. 14b), and for all other cases, i.e. heterogeneous SESE rigids (Fig. 14c).

Returning to our working example (Fig. 12c), we can discover three joins. The first one is the XOR-join in the SESE bond containing activities C, D and G, with G as the exit of the bond and the XOR-split as the entry. The bond is XOR-homogeneous, so that the type of the join is set to XOR. The remaining two joins are in the parent SESE fragment of the bond, which is a heterogeneous rigid, hence, we place two OR-joins. The resulting model is shown in Fig. 12d.



(a) Bond. (b) (XOR-)Homogeneous Rigid. (c) Generic case.

Fig. 14. Joins discovery examples.

4.6 Step 6: OR-joins Minimization

The previous step may leave several OR-join gateways in the discovered BPMN model. Since OR-gateways can be difficult to interpret [42], SM tries to remove them by analyzing the process behavior and turning OR-gateways into AND- or XOR-gateways whenever the behavior is interchangeable.

4.7 Strengths and Limitations of Split Miner

SM was designed to bring together the strengths of older and basic automated process discovery algorithms while addressing their limitations. An example of this design strategy is the filtering algorithm. Past filtering algorithms were either based on heuristics [73,79] that risk to compromise the correctness of the output model, or driven by structural requirements [35]. While SM retains the idea of an integrated filtering algorithm, it focuses on balancing fitness, precision, and simplicity of the output process model.

Past automated discovery algorithms favored either accuracy [73,79] or simplicity [11,35], SM aims to strike a trade-off between the two. The splits and joins discovery steps do not impose any structural constraint on the output process model, as opposed to inductive miner [35] and evolutionary tree miner [11], which enforce block-structuredness, allowing SM to pursue accuracy. Yet, the discovery of the split gateways is designed to produce hierarchies of gateway which foster simplicity and structuredness, while the join discovery and the use of OR-gateways allow for simplicity without compromising accuracy.

However, also SM has its own limitations. First, SM was designed for real-life contexts, and it operates under the assumption that there is always some infrequent behavior to filter out. Second, SM may discover unsound processes, indeed, hitherto soundness has been guaranteed only by enforcing block-structuredness, a trend that SM does not adhere to. While SM guarantees to discover deadlock-free process models [3], it does not guarantee that such process models respect the soundness property of *proper completion*, so that when a token reaches the end event of the process model, more tokens may be left behind. Nonetheless, the chances of SM discovering an unsound process model are very low [2] and in most cases it can discover accurate yet simple and sound process models.

5 Log Skeletons

The previous sections introduced three advanced mining algorithms that tackle the example event log L_6 with more success than the basic algorithms as introduced in Sect. 2. Like these basic algorithms, these advanced algorithms all result in an imperative process model, that is, a process model that indicates what the next possible steps are. However, next to these imperative models, we also have declarative models, like Declare [45]. Unlike an imperative model, a declarative model does not specify what the next possible steps are, instead it provides a collection of constraints that any process instance in the end should adhere to.

This Section introduces an advanced mining algorithm that results in a declarative process model, called a *log skeleton*. [75]. This algorithm has been implemented as the "Visualize Log as Log Skeleton" visualizer plugin in ProM 6 [76]. Provided an event log L, the algorithm first extends the provided event log with the artificial start activity ▶ and the artificial end activity ■. In accordance with Sect. 2, we use L' to denote the event log L extended with these artificial activities. Second, the algorithm discovers from this extended event log L' the collection of initial specific constraints it adheres to. Third, it reduces some of these constraints, keeping only those constraints that are considered to be relevant. Fourth, it shows the most-relevant constraints to the user as a graph. These last three steps are detailed in the next sections.

5.1 Discovering the Log Skeleton

The specific constraints in a log skeleton are the following three activity frequencies and six binary activity relations.

Definition 8 (Log Skeleton Frequencies and Relations). *Let $L' \in \mathcal{B}({\mathcal{U}_{act}}^*)$ be an extended event log and let $a, b \in act(L')$ be two different activities.*

$$c_{L'}(a) = \#^{act}_{L'}(a)$$

is the frequency *of activity a in event log L'.*

$$l_{L'}(a) = \underline{min}\{|\sigma \uparrow \{a\}| \mid \sigma \in L'\}$$

is the lowest frequency *of activity a in any trace in event log L'.*

$$h_{L'}(a) = \underline{max}\{|\sigma \uparrow \{a\}| \mid \sigma \in L'\}$$

is the highest frequency *of activity a in any trace in event log L'.*

$$(a, b) \in E_{L'} \Leftrightarrow \forall_{\sigma \in L'} |\sigma \uparrow \{a\}| = |\sigma \uparrow \{b\}|$$

denotes that for every trace in event log L' the frequencies of activities a and b are the same. Note that the relation $E_{L'}$ induces an equivalence relation *over the activities. We use $r_{L'}(a)$ to denote the* representative activity *for the equivalence class of activity a (by definition, $(r_{L'}(a), a) \in E_{L'}$).*

$$(a, b) \in R_{L'} \Leftrightarrow \forall_{\sigma \in L'} \forall_{i \in \{1,\ldots,|\sigma|\}} (\sigma_i = a \Rightarrow \exists_{j \in \{i+1,\ldots,|\sigma|\}} \sigma_j = b)$$

denotes that for every trace in event log L' an occurrence of activity a is always followed *by an occurrence of activity b. This corresponds to the* response *relation in Declare.*

$$(a, b) \in P_{L'} \Leftrightarrow \forall_{\sigma \in L'} \forall_{i \in \{1,\ldots,|\sigma|\}} (\sigma_i = a \Rightarrow \exists_{j \in \{1,\ldots,i-1\}} \sigma_j = b)$$

denotes that for every trace in event log L' an occurrence of activity a is always preceded by an occurrence of activity b. This corresponds to the precedence *relation in Declare.*

$$(a, b) \in \overline{R}_{L'} \Leftrightarrow \forall_{\sigma \in L'} \forall_{i \in \{1,\ldots,|\sigma|\}} (\sigma_i = a \Rightarrow \nexists_{j \in \{i+1,\ldots,|\sigma|\}} \sigma_j = b)$$

denotes that for every trace in event log L' an occurrence of activity a is never followed by an occurrence of activity b.

$$(a, b) \in \overline{P}_{L'} \Leftrightarrow \forall_{\sigma \in L'} \forall_{i \in \{1,\ldots,|\sigma|\}} (\sigma_i = a \Rightarrow \nexists_{j \in \{1,\ldots,i-1\}} \sigma_j = b)$$

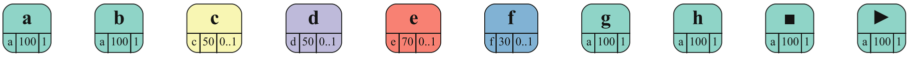


Fig. 15. The nodes of the log skeleton discovered from the event log L_6.

denotes that for every trace in event log L' an occurrence of activity a is never preceded by an occurrence of activity b.

$$(a, b) \in \overline{C}_{L'} \Leftrightarrow \forall_{\sigma \in L'} \forall_{i \in \{1,\ldots,|\sigma|\}} (\sigma_i = a \Rightarrow \nexists_{j \in \{1,\ldots,|\sigma|\}} \sigma_j = b)$$

denotes that for every trace in event log L' an occurrence of activity a never co-occurs with an occurrence of activity b.

Figure 15 shows that we can easily visualize the frequencies and the equivalence relation in the nodes of the log skeleton. The activity, the representative of the equivalence class and the frequencies are simply shown at the bottom of the node, whereas equivalent nodes also have the same background color. For example, Fig. 15 immediately shows that the activities a, b, g, h, $\blacksquare$, and $\blacktriangleright$ are equivalent.

The remaining five activity relations will be visualized by edges between these nodes. However, there could be many such relations, which could very well result in a model that is often called a spaghetti model: A model that contains way too many edges to make any sense of it. Consider, for example, Table 1, which shows that for event log L_6 there are relations between 80 out of 90 possible pairs of different activities, like $(f, b) \in P_{L_6} \cap \overline{R}_{L_6}$. For this reason, the algorithm reduces the collection of these remaining five relations to a collection of *relevant* relations.

Table 1. An overview of the initial non-Equivalence relations for event log L_6.

L_6	$\blacktriangleright$	a	b	c	d	e	f	g	h	$\blacksquare$
$\blacktriangleright$		$R \cap \overline{P}$	$R \cap \overline{P}$	$\overline{P}$	$\overline{P}$	$\overline{P}$	$\overline{P}$	$R \cap \overline{P}$	$R \cap \overline{P}$	$R \cap \overline{P}$
a	$P \cap \overline{R}$		$R \cap \overline{P}$	$\overline{P}$	$\overline{P}$	$\overline{P}$	$\overline{P}$	$R \cap \overline{P}$	$R \cap \overline{P}$	$R \cap \overline{P}$
b	$P \cap \overline{R}$	$P \cap \overline{R}$				$\overline{P}$	$\overline{P}$	$R \cap \overline{P}$	$R \cap \overline{P}$	$R \cap \overline{P}$
c	$P \cap \overline{R}$	$P \cap \overline{R}$			$\overline{R} \cap \overline{P} \cap \overline{C}$		$\overline{P}$	$R \cap \overline{P}$	$R \cap \overline{P}$	$R \cap \overline{P}$
d	$P \cap \overline{R}$	$P \cap \overline{R}$		$\overline{R} \cap \overline{P} \cap \overline{C}$			$\overline{P}$	$R \cap \overline{P}$	$R \cap \overline{P}$	$R \cap \overline{P}$
e	$P \cap \overline{R}$	$P \cap \overline{R}$	$P \cap \overline{R}$				$\overline{R} \cap \overline{P} \cap \overline{C}$		$R \cap \overline{P}$	$R \cap \overline{P}$
f	$P \cap \overline{R}$	$P \cap \overline{R}$	$P \cap \overline{R}$	$\overline{R}$	$\overline{R}$	$\overline{R} \cap \overline{P} \cap \overline{C}$		$R \cap \overline{P}$	$R \cap \overline{P}$	$R \cap \overline{P}$
g	$P \cap \overline{R}$	$P \cap \overline{R}$	$P \cap \overline{R}$	$\overline{R}$	$\overline{R}$		$\overline{R}$		$R \cap \overline{P}$	$R \cap \overline{P}$
h	$P \cap \overline{R}$	$P \cap \overline{R}$	$P \cap \overline{R}$	$\overline{R}$	$\overline{R}$	$\overline{R}$	$\overline{R}$	$P \cap \overline{R}$		$R \cap \overline{P}$
$\blacksquare$	$P \cap \overline{R}$	$P \cap \overline{R}$	$P \cap \overline{R}$	$\overline{R}$	$\overline{R}$	$\overline{R}$	$\overline{R}$	$P \cap \overline{R}$	$P \cap \overline{R}$	

Definition 9 (Relevant Log Skeleton Relations). *Let $L' \in \mathcal{B}({\mathcal{U}_{act}}^*)$ be an extended event log and let $a, b \in act(L')$ be two different activities.*

$$\begin{aligned}(a, b) \in \mathcal{R}_{L'} \Leftrightarrow ((a, b) &\in R_{L'} \\ &\wedge \nexists_{\,c \in act(L')}((a, c) \in R_{L'} \wedge (c, b) \in R_{L'}) \\ &)\end{aligned}$$

that is, $\mathcal{R}_{L'}$ is the transitively reduced version of $R_{L'}$. Clearly, if a is always followed by c and c is always followed by b, then a must be always followed by b.

$$\begin{aligned}(a, b) \in \mathcal{P}_{L'} \Leftrightarrow ((a, b) &\in P_{L'} \\ &\wedge \nexists_{\,c \in act(L')}((a, c) \in P_{L'} \wedge (c, b) \in P_{L'}) \\ &)\end{aligned}$$

that is, $\mathcal{P}_{L'}$ is the transitively reduced version of $P_{L'}$. Clearly, if a is always preceded by c and c is always preceded by b, then a must be always preceded by b.

$$\begin{aligned}(a, b) \in \overline{\mathcal{R}}_{L'} \Leftrightarrow ((a, b) &\in \overline{R}_{L'} \\ &\wedge (a, b) \notin \overline{C}_{L'} \\ &\wedge \nexists_{\,c \in act(L')}((a, c) \in \overline{R}_{L'} \wedge (c, b) \in \overline{R}_{L'}) \\ &)\end{aligned}$$

that is, $\overline{\mathcal{R}}_{L'}$ is the transitively reduced version of $\overline{R}_{L'}$, on top of which the fact that a is never followed by b is also considered irrelevant if a and b do not co-occur. It is not true that if a is never followed by c and c is never followed by b, that then a is never followed by b. Consider, for example the event log containing the traces $\langle a, b\rangle$, $\langle b, c\rangle$, and $\langle c, a\rangle$. We are aware of this, but believe the benefits of doing the transitive reduction outweighs the fact that we may remove relevant relations.

$$\begin{aligned}(a, b) \in \overline{\mathcal{P}}_{L'} \Leftrightarrow ((a, b) &\in \overline{P}_{L'} \\ &\wedge (a, b) \notin \overline{C}_{L'} \\ &\wedge \nexists_{\,c \in act(L')}((a, c) \in \overline{P}_{L'} \wedge (c, b) \in \overline{P}_{L'}) \\ &)\end{aligned}$$

that is, $\overline{\mathcal{P}}_{L'}$ is the transitively reduced version of $\overline{P}_{L'}$, on top of which the fact that a is never preceded by b is also considered irrelevant if a and b do not co-occur. Like with $\overline{\mathcal{R}}_{L'}$, it is not true that if a is never preceded by c and c is never preceded by b, that then a is never preceded by b.

$$\begin{aligned}(a, b) \in \overline{\mathcal{C}}_{L'} \Leftrightarrow ((a, b) &\in \overline{C}_{L'} \\ &\wedge \nexists_{\,c \in act(L')}((a, c) \in P_{L'} \wedge (c, b) \in \overline{C}_{L'}) \\ &\wedge \nexists_{\,c \in act(L')}((b, c) \in P_{L'} \wedge (c, a) \in \overline{C}_{L'}) \\ &)\end{aligned}$$

Table 2. An overview of the relevant non-Equivalence relations for event log L_6.

L_6	▶	a	b	c	d	e	f	g	h	■
▶		$\mathcal{R} \cap \overline{\mathcal{P}}$								
a	$\mathcal{P} \cap \overline{\mathcal{R}}$		$\mathcal{R} \cap \overline{\mathcal{P}}$	$\overline{\mathcal{P}}$	$\overline{\mathcal{P}}$					
b		$\mathcal{P} \cap \overline{\mathcal{R}}$				$\overline{\mathcal{P}}$	$\overline{\mathcal{P}}$	$\mathcal{R}$		
c		$\mathcal{P} \cap \overline{\mathcal{R}}$			$\overline{\mathcal{C}}$		$\overline{\mathcal{P}}$	$\mathcal{R}$		
d		$\mathcal{P} \cap \overline{\mathcal{R}}$		$\overline{\mathcal{C}}$			$\overline{\mathcal{P}}$	$\mathcal{R}$		
e			$\mathcal{P} \cap \overline{\mathcal{R}}$				$\overline{\mathcal{C}}$		$\mathcal{R} \cap \overline{\mathcal{P}}$	
f			$\mathcal{P} \cap \overline{\mathcal{R}}$	$\overline{\mathcal{R}}$	$\overline{\mathcal{R}}$	$\overline{\mathcal{C}}$		$\mathcal{R} \cap \overline{\mathcal{P}}$		
g			$\mathcal{P}$				$\overline{\mathcal{R}}$		$\mathcal{R} \cap \overline{\mathcal{P}}$	
h						$\overline{\mathcal{R}}$		$\mathcal{P} \cap \overline{\mathcal{R}}$		$\mathcal{R} \cap \overline{\mathcal{P}}$
■									$\mathcal{P} \cap \overline{\mathcal{R}}$	

Clearly, if b is always preceded by c and c does not co-occur with a, then b cannot co-occur with a. Note that we could also have used the always-follows relation $R_{L'}$ here instead of the always-precedes relation $P_{L'}$, but using the latter relation results in the relevant never-co-occurs relations being more at the beginning of the process, that is, towards the point where the actual decision was made to choose one or the other.

Table 2 shows the results for the event log L_6: Of the 80 initial relations, only 32 are considered to be relevant. Finally, the algorithm shows the log skeleton as a graph to the user, where this graph contains only edges for the relevant relations.

5.2 Visualizing the Log Skeleton

The discovered log skeleton is visualized using a log skeleton graph, which is a graph showing the relevant relations, the equivalence classes, and the frequencies as discovered from the event log.

Definition 10 (Log Skeleton Graph). *Let $L' \in \mathcal{B}({\mathcal{U}_{act}}^*)$ be an extended event log and let $a, b \in act(L')$. The log skeleton graph for L' is the graph $G = (V, E, t)$ where:*

$$V = \{(a, r_{L'}(a), c_{L'}(a), l_{L'}(a), h_{L'}(a)) | a \in act(L')\}$$

is the set of nodes, where every node contains the activity, the representative of the activity within its equivalence class, the frequency of the activity in the log, and the minimal and maximal frequencies of the activity in any trace. If $l(a) = h(a)$ then only $l(a)$ is shown, otherwise $l(a)..h(a)$ is shown.

$$\begin{aligned} E = &(\mathcal{R}_{L'} \cup \mathcal{P}_{L'} \cup \overline{\mathcal{R}}_{L'} \cup \overline{\mathcal{P}}_{L'} \cup \overline{\mathcal{C}}_{L'}) \\ &\cup (\mathcal{R}_{L'} \cup \mathcal{P}_{L'} \cup \overline{\mathcal{R}}_{L'} \cup \overline{\mathcal{P}}_{L'} \cup \overline{\mathcal{C}}_{L'})^{-1} \end{aligned} \tag{1}$$

is the set of edges, where we have an edge from one activity to another activity if we have a relevant relation between these activities (either way).

$$d \in E \to \{\blacktriangleright, \blacktriangleleft, \triangleright, \triangleleft, |, \perp\}$$

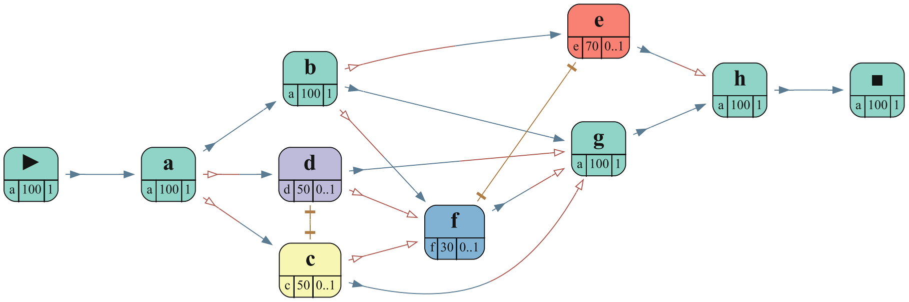


Fig. 16. The full log skeleton discovered from the event log L_6 (shown using a left-right orientation).

denotes the decorator to be used to show the relation from the activity at the tail to the activity at the head:

- *if* $(a, b) \in R_{L'}$ *then* $d((a, b)) = \blacktriangleright$*, indicating that* a *is always followed by* b*,*
- *else if* $(a, b) \in P_{L'}$ *then* $d((a, b)) = \blacktriangleleft$*, indicating that* a *is always preceded by* b*,*
- *else if* $(a, b) \in \overline{C}_{L'}$ *then* $d((a, b)) = |$*, indicating that* a *does not co-occur with* b*,*
- *else if* $(a, b) \in \overline{R}_{L'}$ *then* $d((a, b)) = \triangleleft$*, indicating that* a *is never followed by* b*,*
- *else if* $(a, b) \in \overline{P}_{L'}$ *then* $d((a, b)) = \triangleright$*, indicating that* a *is never preceded by* b*, and*
- *otherwise* $d((a, b)) = \perp$*, indicating that no relation was discovered from* a *to* b*.*

These decorations are shown on the tail of the corresponding edge.

Table 3 shows which decorators will be shown for the event log L_6, and Fig. 16 shows the resulting log skeleton[3]. Note that the edges (a, b) and (b, a) are visualized by a single edge, with the decorator for (a, b) near a and the decorator for (b, a) near b.

Table 3. An overview of the decorators used for the non-Equivalence relations for event log L_6.

L_6	▶	a	b	c	d	e	f	g	h	■
▶		▶								
a	◀		▶	▷	▷					
b		◀				▷	▷	▶		
c		◀			|		▷	▶		
d		◀		|			▷	▶		
e			◀				|		▶	
f			◀	◁	◁	|		▶		
g			◀	◁	◁		◁		▶	
h						◁		◀		▶
■									◀	

[3] For sake of completeness, we mention that we are using version 6.12.5 of the LogSkeleton package, which is available in the Nightly Build of ProM, see https://www.promtools.org/doku.php?id=nightly.

As example relations, activity b is never preceded by e (that is, if both b and e occur, then e occurs after b), e is is always preceded by b, and e and f do not co-occur. Also note that although 32 relations were considered to be relevant, 34 are now shown: The relations $(g, c) \in \overline{R}$ and $(g, d) \in \overline{R}$ were not considered relevant as these relations can be induced using f. However, as $(c, g) \in R$ and $(d, g) \in R$ are considered relevant, the relations for (g, c) and (g, d) are shown as well.

Using the log skeleton shown in Fig. 16, we can deduce the following facts on the example event log:

- The activities a, b, g, and h are always executed exactly once, and always in the given order.
- In parallel with b, there is a $50/50$ choice between c and d.
- There is a $70/30$ choice between e and f, but the position of this choice in the process is less clear. If e is chosen, it is executed after b but in parallel with c, d, and g. However, if f is chosen it is executed after b, c, and d, and before g.

5.3 Handling Noise

So far, we have assumed that the event log does not contain any noise. As a result, a constraint like $(a, b) \in R_{L'}$ may be invalid because a single instance of a in the entire event log is not followed by a b. To be able to handle noisy logs, the log skeletons allow the user to set a percentage for which the constraint should hold. We recall here the definition of the Response constraint as provided earlier:

$$(a, b) \in R_{L'} \Leftrightarrow \forall_{\sigma \in L'} \forall_{i \in \{1,\ldots,|\sigma|\}} (\sigma_i = a \Rightarrow \exists_{j \in \{i+1,\ldots,|\sigma|\}} \sigma_j = b)$$

When dealing with noise, we are interested in the percentage of cases for which the left-hand side of the implication ($\sigma_i = a$) holds, for which then also the right-hand side ($\exists_{j \in \{i+1,\ldots,|\sigma|\}} \sigma_j = b$) holds. As such, we can divide the instances of the left-hand side into positive instances (for which the right-hand side holds) and negative instances (for which the right-hand side does not hold). If the user allows for a noise level of l (where $0 \leq l \leq 1$), then the number of negative instances should be at most l times the number of total instances:

$$\left(\sum_{\sigma \in L'} \left| \{ i \in \{1, \ldots, |\sigma|\} \mid \sigma_i = a \wedge \nexists_{\, j \in \{i+1,\ldots,|\sigma|\}} \sigma_j = b \} \right| \right) \leq l \times \#_{L'}^{act}(a)$$

This way of handling noise can also be used for the relations $P_{L'}$, $\overline{R}_{L'}$, $\overline{P}_{L'}$, and $\overline{C}_{L'}$, because these constraint are structured in a similar way. However, this way will not work for the equivalence relation $E_{L'}$. To decide whether two different activities a_1 and a_n (where $n \geq 2$) are considered to be equivalent given a certain noise level l (where again $0 \leq l \leq 1$), we use the following condition for equivalence:

$$\forall_{i \in \{1,\ldots,n-1\}} \left(\left(\sum_{\sigma \in L'} ||\sigma \uparrow \{a_i\}| - |\sigma \uparrow \{a_{i+1}\}|| \right) \leq l \times |L'| \right)$$

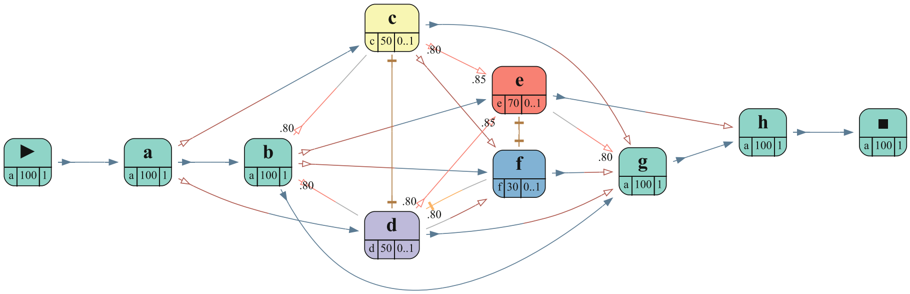


Fig. 17. The full log skeleton discovered from the event log L_6 allowing for 20% noise.

That is, there is a series of activities a_1, a_2, ..., a_n such that for every subsequent pair (a_i, a_{i+1}) the *distance* between both activity counts over all traces should at most be l times the number of traces in the event log. Clearly, setting a noise level of $l = 0$ results in a condition that the activity counts should match perfectly, which is exactly what we want.

Figure 17 shows the log skeleton that results from event log L_6 when setting the noise level to 0.2. For example, this shows that 80% of the instances of activity c are never preceded by e, that 85% of the instances of e are never followed by c, and that 80% of the instances of activity d do not co-occur with f.

5.4 Strengths and Limitations

Clearly, a log skeleton is not an imperative process model like a Petri net or a BPMN diagram. Instead, it is a declarative process model like Declare [45]. Some of the relations in the log skeletons exist in Declare as well like $R_{L'}$ (Response) and $P_{L'}$ (Precedence). But Declare contains many relations that are unknown in a log skeleton, while the Equivalence relation $E_{L'}$ does not have a counterpart in Declare. As a result, a log skeleton can be considered as a Declare model restricted to only some relations but with an additional equivalence relation.

Of course, limitations also exists for log skeletons. Known process constructs that are hard for log skeletons are loops and duplicate tasks. Furthermore, noise in an event log may be a problem, as a single misplaced activity may prevent discovery of some relations. As attempts to alleviate the problems with these constructs and noise, The visualizer plugin allows the user to specify *boundary activities* (to tackle loops), to *split activities over activities* (to tackle duplicates), and various *noise levels* (to tackle noise). Although our experience with the noise levels is very positive, our experience with the boundary activities and splitting of activities shows that they only can solve some of the problems related to the hard process constructs. As a result, more research is needed in this direction to improve on this.

6 Related Work

Discovering accurate and simple process models is extremely important to reduce the time spent to enhance them and avoid mistakes during process analysis [28].

While extensive research effort was spent in designing the perfect automated process discovery algorithm, in parallel, researchers have investigated the problem of improving the quality of the input data, proposing techniques for data filtering and data repairing [19,21,32,50–52,57,59,69,70,78]; as well as the problem of predicting what would be the process discovery algorithm yielding the best process model from a given event log [47–49]. A few research studies also explored divide-and-conquer strategies, designing approaches to divide the input data into smaller chunks and separately feed each chunk to a discovery algorithm – in order to facilitate the discovery task. The set of process models discovered from the data chunks would then be reassembled into a unique process model. Among these techniques we find Genet [15,16], C-net miner [55], Stage Miner [43], BPMN Miner [20], and Decomposed Process Mining [77].

It is also worth mentioning techniques that have the ability to deal with negative examples [23,24,33], i.e., to accept also traces that are known to not be part of the underlying process. Of course, this is an information that is not often available, unless domain knowledge can be used, or some automated techniques can be applied for generating it [71,72]. These techniques seem to be better positioned to also consider generalization when searching for the best process model.

Optimization metaheuristics have also been extensively applied in the context of automated process discovery, aiming to incrementally discover and refine the process model to reach a trade-off between accuracy and simplicity. The most notorious, among this type of approaches, are those based on evolutionary (genetic) algorithms [11,25]. However, several other metaheuristics have been explored, such as the imperialist competitive algorithm [1], the swarm particles optimization [18,29,44], and simulated annealing [31,58].

Nonetheless, the latest literature review and benchmark in automated process discovery [2] highlighted that many of the state-of-the-art automated process discovery algorithms [4,13,34,36,67,73,79] were affected by one (or more) of the following three limitations when discovering process models from real-life event logs: i) they achieve limited accuracy; ii) they are computationally inefficient to be used in practice; iii) they discover syntactically incorrect process models. In practice, when the behavior of the process recorded in the input event log varies little, most of the state-of-the-art automated process discovery algorithms can output accurate and simple process models. However, as the behavioral complexity of the process increases, the quality of the discovered process models can worsen quickly. Given that oftentimes real-life event logs are highly complex (i.e., containing complex process behavior, noise, and incomplete data), discovering highly accurate and simple process models with traditional state-of-the-art algorithms can be challenging.

On the other hand, achieving in a robust and scalable manner the best trade-off between accuracy and simplicity, while ensuring behavioral correctness (i.e., process soundness), has proved elusive. In particular, it is possible to group automated process discovery algorithms in two categories: those focusing more on the *simplicity*, the *soundness* and either the *precision* [13] or the *fitness* [36] of the discovered process model, and those focusing more on its *fitness* and its *precision* at the cost of *simplicity* and/or *soundness* [4,73,79]. The first kind of algorithms strive for simplicity and soundness by enforcing block-structured behavior on the discovered process model. However, since real-life processes are not always block-structured, a direct consequence of doing that is an approximation of the behavior which leads to a loss of accuracy (either fitness of precision). The second kind of algorithms do not adopt any strategy to deal with process simplicity and soundness, focusing only on capturing its behavior in a process model, but in doing so they can produce unsound process models.

Alongside techniques that discover imperative process models, it is important to mention that there exists many discovery algorithm that produce *declarative models* [10,27,39,40,53,74]. Declare models capture the processes' behavior through a set of rules, also known as *declarative* constraints. Even though each declarative constraint is precise, capturing the whole process behavior in a declarative model can be very difficult, especially because declarative models do not give any information about "undeclared" behavior, e.g., any behavior that does not break the declarative constraint is technically allowed behavior. Hence, imperative process models are usually preferred in practice.

7 Challenges Ahead

Process Mining started about 20 years ago with the development of control-flow miners like the Alpha Miner [64] and the Little Thumb Miner [80]. Although the field has advanced in these 20 years with many others control-flow miners, this does not mean that control-flow mining is already a done deal.

Consider, for example, the results of the latest Process Discovery Contest (PDC 2020) [14], which are shown by Fig. 18. The PDC 2020 was a contest for fully-automated control-flow miners, and shows the then-current state of the field on these miners. In this contest, every miner was used to discover a control-flow model from a training event log, after which this model was used to classify every trace from a test event log. As the ground truth for this classification is known, we can compute both the average positive accuracy and the average negative accuracy for all of the algorithms on this data set. The results show that there is still some ground to cover for the imperative miners, as none of these miners was able to achieve both an average positive accuracy and an average negative accuracy exceeding 80.0%.

Table 4 shows the weaknesses of several algorithms submitted to the PDC 2020 contest. As an example, the weaknesses of the Inductive IMfa Miner included loops: It scored 59.2%[4] on the event logs in the PDC 2020 data set that do not contain loops, and only 19.3% on the event logs that do contain loops. This table indicates that noise and loops but also optional tasks and duplicate tasks can be considered as challenges for control-flow miners in the near future.

[4] This score is computed as the average over $\frac{2 \cdot P_L \cdot N_L}{P_L + N_L}$, where P_L is the positive accuracy and N_L is the negative accuracy for (1) the model discovered from a training log L and (2) the corresponding test log.

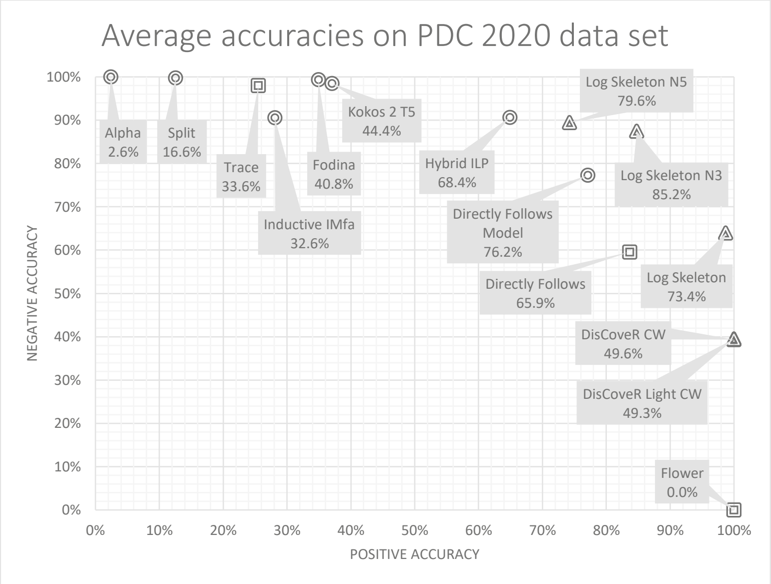


Fig. 18. The results of the PDC 2020. The squares correspond to base miners, the circles to imperative miners (that result in an imperative model, like a Petri net or a BPMN diagram), and the triangles to declarative miners (that result in a declarative model, like a DCR graph or a log skeleton). The percentage mentioned with a miner is the score (see footnote 4) of that miner.

Table 4. Weaknesses and scores of miners submitted to PDC 2020 and their scores on the event logs that do not contain the weakness (**No**) or that do contain it (**Yes**). Only weaknesses where the **No** and **Yes** scores differ at least 10.0% are listed.

Algorithm	Score	Weakness	No	Yes
DisCoveR CW	49.6%	Noise	78.0%	21.3%
DisCoveR Light CW	49.3%	Noise	79.9%	18.7%
Inductive IMfa	32.6%	Loops	59.2%	19.3%
		Duplicate tasks	39.4%	25.7%
Kokos 2 T5	44.4%	Loops	61.8%	35.6%
		Noise	50.4%	38.3%
		Optional tasks	66.5%	22.2%

In these 20 years, algorithms have been developed that discover perspectives other than the control-flow perspective. However, many of these other perspectives are added on top of the discovered control-flow model, and hence depend on the discovery of a control-flow model of high-enough quality. Nevertheless, even if assuming that the quality of the control-flow model is indeed high enough, challenges remain for these other perspectives as well.

As a first example, consider the data perspective, which would add expressions (guards) to the control-flow model that would guide the execution of the model: Certain parts of the control-flow model may be only valid if a certain guard evaluates true. Challenges here include the discovery of sensible guards with sensible values. As an example, if based on some value the control-flow 'goes either left or right', then the data in the event log may not contain this precise value. As a result, this value needs to be guessed based on the data that is in the event log.

A second example is the organizational perspective, which would add organizational entities (like users, groups, or roles) to certain parts of the control-flow model: Only resources (like users and automated services) that qualify for these entities can be involved in these parts. Challenges here include the discovery of the correct organizational entities at the correct level. As an example, if some activity was performed by some user according to the event log, then what is the correct organizational level (like user, role, group) for this activity?

8 Conclusion

In this chapter, we have introduced four advanced process discovery techniques: the state-based region miner; the language-based region miner; the split miner; the log skeleton miner. Each of the four techniques aims to alleviate shortcomings of the more basic process discovery techniques as introduced in the previous chapter.

First, the region-based miners can lift the shortcoming of having to assume that activities only occur once in the model. When using regions, different contexts of an activity can be found, and the activity can then be divided over these contexts, leading to a model with an activity for every different context. This is a feature that is not shared by any of the other miners, and this feature can be very important in case we have an event log of a system where these "duplicate activities" occur. Where other miners need to assume there is only one activity, which may lead to discovered models that are incomprehensible, these region-based miners do not need to make this assumption, which may result in more precise models.

Second, the split miner aims to discover process models that simultaneously maximize and balance fitness and precision, while at the same time minimizing the control-flow complexity of the resulting model. This approach brings precision and complexity into the equation, something that previously could be done only by using genetic miners like the evolutionary tree miner [12]. However, differently than genetic miners, split miner typically takes seconds to discover a process model from the event log, as opposed to the hour-long execution times required by genetic miners [2].

Third, the log skeleton miner is not limited to using only the directly-follows relations, which are heavily leveraged by many existing discovery algorithm. This miner discovers a declarative model from the event log that contains facts like "95% of the instances of activity a is always followed by activity b", or "90% of the instances of activity a do not co-occur with an instance of activity b". As such, it is not limited to just the directly-follows relations, and it can discover relations between activities that cannot be discovered if only considering the directly-follows relations.

It is clear that each of these advanced techniques can be used effectively on certain event logs, and may produce better models than those produced by basic techniques.

However, ultimately, there is no technique yet that is effective on all (or even almost all) event logs regardless of the process behavior features. Such an ideal process discovery technique should be able to maximize accuracy and simplicity of the discovered process model while at the same time guaranteeing its simplicity and soundness. While, hitherto, the design of such a technique has proved to be challenging and elusive, it has become clear that each process discovery technique can be useful on some event logs. Hence, while we hope that future research endeavors will lead to the ideal process discovery technique, until it materializes, we just have to rely on educated choices based on the process data at hand (i.e., in the form of event log), and select the most appropriate technique for discovering the best process model.

Acknowledgements. This work has been supported by MCIN/AEI funds under grant PID2020-112581GB-C21.

References

1. Alizadeh, S., Norani, A.: ICMA: a new efficient algorithm for process model discovery. Appl. Intell. **48**(11) (2018)
2. Augusto, A., et al.: Automated discovery of process models from event logs: Rev. Benchmark. IEEE TKDE **31**(4) (2019)
3. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowl. Inf. Syst. **59**(2), 251–284 (2018). https://doi.org/10.1007/s10115-018-1214-x
4. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Bruno, G.: Automated discovery of structured process models: discover structured vs. discover and structure. In: Comyn-Wattiau, I., Tanaka, K., Song, I.-Y., Yamamoto, S., Saeki, M. (eds.) ER 2016. LNCS, vol. 9974, pp. 313–329. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46397-1_25
5. Augusto,A., Dumas, M., La Rosa, M.:Automated discovery of process models with true concurrency and inclusive choices. In: International Conference on Process Mining, pp. 43–56. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-98581-3_1
6. Badouel, E., Bernardinello, L., Darondeau. Ph.: Polynomial algorithms for the synthesis of bounded nets. In: TAPSOFT, pp. 364–378 (1995)
7. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 375–383. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_27
8. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of petri nets from infinite partial languages. In: Billington, J., Duan, J., Koutny, M. (eds.) ACSD, pp. 170–179. IEEE (2008)
9. Bergenthum, R., Desel, J., Mauser, S., Lorenz, R.: Synthesis of petri nets from term based representations of infinite partial languages. Fundam. Inform. **95**(1), 187–217 (2009)
10. Bernardi, M.L., Cimitile, M., Di Francescomarino, C., Maggi, F.M.: Do activity lifecycles affect the validity of a business rule in a business process? Inf. Syst. **62** (2016)
11. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: On the role of fitness, precision, generalization and simplicity in process discovery. In: Meersman, R., et al. (eds.) OTM 2012. LNCS, vol. 7565, pp. 305–322. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33606-5_19
12. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A genetic algorithm for discovering process trees. In: IEEE Congress on Evolutionary Computation (CEC), 2012, pp. 1–8. IEEE (2012)

13. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Quality dimensions in process discovery: the importance of fitness, precision, generalization and simplicity. Int. J. Cooperat. Inf. Syst. **23**(01),1440001 (2014)
14. Carmona, J., Depaire, B., Verbeek, H.M.W.: Process discovery contest 2020 (2019). https://icpmconference.org/2020/process-discovery-contest/. Accessed 23 Apr 2021
15. Carmona, J.: Projection approaches to process mining using region-based techniques. Data Min. Knowl. Discov. **24**(1), 218–246 (2012)
16. Carmona, J., Cortadella, J., Kishinevsky, M.: Divide-and-conquer strategies for process mining. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 327–343. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03848-8_22
17. Carmona, J., Cortadella, J., Kishinevsky, M.: New region-based algorithms for deriving bounded Petri nets. IEEE Trans. Comput. **59**(3), 371–384 (2009)
18. Chifu, V.R., Pop, C.B., Salomie, I., Balla, I., Paven, R.: Hybrid particle swarm optimization method for process mining. In: ICCP, IEEE (2012)
19. Conforti, R., La Rosa, M., ter Hofstede, A.H.M.: Filtering out infrequent behavior from business process event logs. IEEE Trans. Knowl. Data Eng. **29**(2), 300–314 (2016)
20. Conforti, R., Dumas, M., García-Bañuelos, L., La Rosa, M.: BPMN miner: automated discovery of BPMN process models with hierarchical structure. Inf. Syst. **56**, 284–303 (2016)
21. Conforti, R., La Rosa, M., ter Hofstede, A.H.M., Augusto, A.: Automatic repair of same-timestamp errors in business process event logs. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNCS, vol. 12168, pp. 327–345. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58666-9_19
22. Darondeau, P.: Deriving unbounded Petri nets from formal languages. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 533–548. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055646
23. Ponce de León, H., Nardelli, L., Carmona, J., vanden Broucke, S.K.L.M.: Incorporating negative information to process discovery of complex systems. Inf. Sci. **422**, 480–496 (2018)
24. Ponce-de-León, H., Rodríguez, C., Carmona, J., Heljanko, K., Haar, S.: Unfolding-based process discovery. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) ATVA 2015. LNCS, vol. 9364, pp. 31–47. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24953-7_4
25. Alves de Medeiros, A.K.: Genetic process mining. Ph.D. thesis, Eindhoven University of Technology (2006)
26. Desel, J., Reisig, W.: The synthesis problem of Petri nets. Acta Inf. **33**(4), 297–315 (1996)
27. Di Ciccio, C., Mecella, M.: A two-step fast algorithm for the automated discovery of declarative workflows. In: 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), pp. 135–142. IEEE (2013)
28. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer, Berlin (2013). https://doi.org/10.1007/978-3-662-56509-4
29. Effendi, Y.A., Sarno, P.: Discovering optimized process model using rule discovery hybrid particle swarm optimization. In: 2017 3rd International Conference on Science in Information Technology (ICSI Tech), pp. 97–103. IEEE (2017)
30. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-structures. Part I, II. Acta Inform. **27**, 315–368 (1990)
31. Gao, D., Liu, Q.: An improved simulated annealing algorithm for process mining. In: CSCWD, IEEE (2009)
32. Ghionna, L., Greco, G., Guzzo, A., Pontieri, L.: Outlier detection techniques for process mining applications. In: An, A., Matwin, S., Ras, Z.W., Slezak, D. (eds.) ISMIS 2008. LNCS (LNAI), vol. 4994, pp. 150–159. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68123-6_17
33. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. J. Mach. Learn. Res. **10**, 1305–1340 (2009)

34. Guo, Q., Wen, L., Wang, J., Yan, Z., Yu, P.S.: Mining invisible tasks in non-free-choice constructs. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 109–125. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_7
35. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann, N., Song, M., Wohed, P. (eds.) BPM 2013. LNBIP, vol. 171, pp. 66–78. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06257-0_6
36. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from incomplete event logs. In: Ciardo, G., Kindler, E. (eds.) PETRI NETS 2014. LNCS, vol. 8489, pp. 91–110. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07734-5_6
37. Lorenz, R.: Towards synthesis of petri nets from general partial languages. In: Lohmann, N., Wolf, K. (eds.) AWPN, vol. 380 of CEUR Workshop Proceedings, pp. 55–62. CEUR-WS.org (2008)
38. Lorenz, R., Juhás, R.: How to synthesize nets from languages - a survey. In: Proceedings of the Wintersimulation Conference (WSC) 2007 (2007)
39. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 270–285. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31095-9_18
40. Maggi, F.M., Dumas, M., García-Bañuelos, L., Montali, M.: Discovering data-aware declarative process models from event logs. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 81–96. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40176-3_8
41. Mauser, S., Lorenz, S.: Variants of the language based synthesis problem for petri nets. In: ACSD, pp. 89–98 (2009)
42. Mendling, J., Reijers, H.A., van der Aalst, W.M.P.: Seven process modeling guidelines (7PMG). Inform. Softw. Technol. **52**(2), 127–136 (2010)
43. Nguyen, H., Dumas, M., ter Hofstede, A.H.M., La Rosa, M., Maggi, F.M.: Mining business process stages from event logs. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 577–594. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_36
44. Nurlaili, A.L., Sarno, R.: A combination of the evolutionary tree miner and simulated annealing. In: 2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), pp. 1–5. IEEE (2017)
45. Pesic, M., Schonenberg, H., van der Aalst, W.I.P.: DECLARE: full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), 15–19 October 2007, Annapolis, Maryland, USA, pp. 287–300 (2007)
46. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. In: WS-FM, pp. 25–41 (2010)
47. Ribeiro, J., Carmona, J.: RS4PD: a tool for recommending control-flow algorithms. In: BPM (Demos), pp. 66. Citeseer (2014)
48. Ribeiro, J., Carmona, J., Mısır, M., Sebag, M.: A recommender system for process discovery. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 67–83. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_5
49. Ribeiro, J., Carmona Vargas, J.: A method for assessing parameter impact on control-flow discovery algorithms. In: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data: Brussels, Belgium, 22–23 June 2015, pp. 83–96. CEUR-WS. org (2015)

50. Rogge-Solti, A., Mans, R.S., van der Aalst, W.M.P., Weske, M.: Improving Documentation by repairing event logs. In: Grabis, J., Kirikova, M., Zdravkovic, J., Stirna, J. (eds.) PoEM 2013. LNBIP, vol. 165, pp. 129–144. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41641-5_10
51. Sani, M.F., van Zelst, S.J., van der Aalst, W.M.P.: Improving process discovery results by filtering outliers using conditional behavioural probabilities. In: International Workshop on Business Process Intelligence (BPI 2017) (2017)
52. Sani, M.F., van Zelst, S.J., van der Aalst, W.M.P.: Repairing outlier behaviour in event logs using contextual behaviour. EMISAJ **14**, 1–24 (2019)
53. Schönig, S., Rogge-Solti, A., Cabanillas, C., Jablonski, S., Mendling, J.: Efficient and customisable declarative process mining with SQL. In: Nurcan, S., Soffer, P., Bajec, M., Eder, J. (eds.) CAiSE 2016. LNCS, vol. 9694, pp. 290–305. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39696-5_18
54. Solé, M., Carmona, J.: Light region-based techniques for process discovery. Fundam. Inform. **113**(3–4), 343–376 (2011)
55. Solé, M., Carmona, J.: Incremental process discovery. Trans. Petri Nets Other Models of Concurr. **5**, 221–242 (2012)
56. Solé, M., Carmona, J.: Region-based foldings in process discovery. IEEE Trans. Knowl. Data Eng. **25**(1), 192–205 (2013)
57. Song, S., Cao, Y., Wang, J.: Cleaning timestamps with temporal constraints. VLDB Endow. **9**(10), 708–719 (2016)
58. Song, W., Liu, S., Liu, Q.: Business process mining based on simulated annealing. In: ICYCS, IEEE (2008)
59. Tax, N., Sidorova, N., van der Aalst, W.M.P.: Discovering more precise process models from event logs by filtering out chaotic activities. J. Intell. Inf. Syst., **52**(1), 107–139 (2019)
60. van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9) (2004)
61. van der Aalst, W.M.P., Günther, C.W.: Finding structure in unstructured processes: the case for process mining. In: ACSD, pp. 3–12 (2007)
62. van der Aalst, W.M.P., Rubin, V., (Eric) Verbeek, H.M.W., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Softw. Syst. Model. **9**, 87–111 (2009)
63. van der Aalst, W.M.P., van Dongen, B.F.: Discovering petri nets from event logs. Trans. Petri Nets Other Models Concurr. **7**, 372–422 (2013)
64. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
65. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. Fundam. Inform. **94**(3–4), 387–412 (2009)
66. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: ILP-based process discovery using hybrid regions. In van der Aalst, W.M.P., Bergenthum, R., Carmona, J. (eds.) Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED 2015, Satellite Event of the Conferences: 36th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2015 and 15th International Conference on Application of Concurrency to System Design ACSD 2015, Brussels, Belgium, 22–23 June 2015, vol. 1371 of CEUR Workshop Proceedings, pp. 47–61. CEUR-WS.org (2015)
67. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: ILP-based process discovery using hybrid regions. In: International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED 2015, vol. 1371 of CEUR Workshop Proceedings, pp. 47–61. CEUR-WS.org (2015)

68. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P., Verbeek, H.M.W.: Discovering workflow nets using integer linear programming. Computing **100**(5), 529–556 (2017). https://doi.org/10.1007/s00607-017-0582-5
69. van Zelst, S.J., Fani Sani, M., Ostovar, A., Conforti, R., La Rosa, M.: Filtering spurious events from event streams of business processes. In: Krogstie, J., Reijers, H.A. (eds.) CAiSE 2018. LNCS, vol. 10816, pp. 35–52. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91563-0_3
70. van Zelst, S.J., Fani Sani, M., Ostovar, A., Conforti, R., La Rosa, M.: Detection and removal of infrequent behaviour from event streams of business processes. Inf. Syst. **90** (2019)
71. vanden Broucke, S.K.L.M., De Weerdt, J., Baesens, B., Vanthienen, J.: Improved artificial negative event generation to enhance process event logs. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 254–269. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31095-9_17
72. vanden Broucke, S.K.L.M., De Weerdt, J., Vanthienen, J., Baesens, B.: Determining process model precision and generalization with weighted artificial negative events. IEEE Trans. Knowl. Data Eng, **26**(8), 1877–1889 (2014)
73. vanden Broucke, S.K.L.M., De Weerdt, J.: Fodina: a robust and flexible heuristic process discovery technique. Decis. Supp. Syst. **100**, 109–118 (2017)
74. vanden Broucke, S.K.L.M., Vanthienen, J., Baesens, B.: Declarative process discovery with evolutionary computing. In: 2014 IEEE Congress on Evolutionary Computation (CEC), pp. 2412–2419. IEEE (2014)
75. Verbeek, H.M.W.: The Log Skeleton Visualizer in ProM 6.9: the winning contribution to the process discovery contest 2019. Int. J. Softw. Tools Technol. Trans. **339** (2021). https://doi.org/10.1007/s10009-021-00618-y
76. Verbeek, H.M.W. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: ProM 6: the process mining toolkit. In: Proceedings of BPM Demonstration Track 2010, vol. 615, pp. 34–39. CEUR-WS.org (2010)
77. Verbeek, H.M.W., van der Aalst, W.M.P.: Decomposed process mining: the ILP case. In: Fournier, F., Mendling, J. (eds.) BPM 2014. LNBIP, vol. 202, pp. 264–276. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15895-2_23
78. Wang, J., Song, S., Lin, X., Zhu, X., Pei, J.: Cleaning structured event logs: a graph repair approach. In: Proceedings of IEEE ICDE, pp. 30–41. IEEE (2015)
79. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: 2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), pp. 310–317. IEEE (2011)
80. Weijters, A.J.M.M., van der Aalst, W.: Rediscovering workflow models from event-based data using little thumb. Integr. Comput.-Aid. Eng. **10**(2) (2003)

Declarative Process Specifications: Reasoning, Discovery, Monitoring

Claudio Di Ciccio[1](✉) and Marco Montali[2]

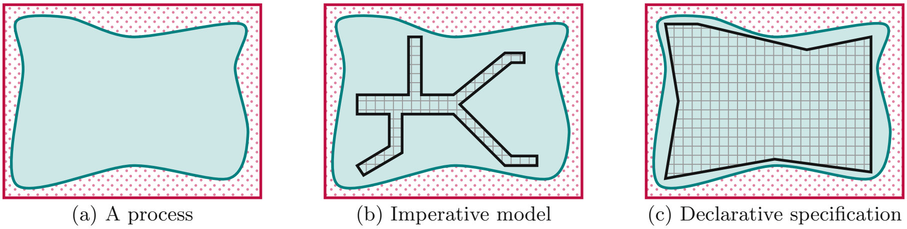
[1] Sapienza University of Rome, Rome, Italy
claudio.diciccio@uniroma1.it
[2] Free University of Bozen-Bolzano, Bolzano, Italy
montali@inf.unibz.it

Abstract. The declarative specification of business processes is based upon the elicitation of behavioural rules that constrain the legal executions of the process. The carry-out of the process is up to the actors, who can vary the execution dynamics as long as they do not violate the constraints imposed by the declarative model. The constraints specify the conditions that require, permit or forbid the execution of activities, possibly depending on the occurrence (or absence) of other ones. In this chapter, we review the main techniques for process mining using declarative process specifications, which we call *declarative process mining*. In particular, we focus on three fundamental tasks of (1) reasoning on declarative process specifications, which is in turn instrumental to their (2) discovery from event logs and their (3) monitoring against running process executions to promptly detect violations. We ground our review on Declare, one of the most widely studied declarative process specification languages. Thanks to the fact that Declare can be formalized using temporal logics over finite traces, we exploit the automata-theoretic characterization of such logics as the core, unified algorithmic basis to tackle reasoning, discovery, and monitoring. We conclude the chapter with a discussion on recent advancements in declarative process mining, considering in particular multi-perspective extensions of the original approach.

1 Introduction

Finding a suitable balance between flexibility and control is a long-standing problem in the management of work processes [83]. Among the different approaches striving to achieve this balance, *flexibility by design* suggests to infuse flexibility in the process modeling language at hand. Declarative process modeling languages take this to the extreme: they support the specification of *what* are the relevant constraints on the temporal evolution of the process, without explicitly indicating *how* process instances should be routed to satisfy such constraints. In comparison with imperative approaches that produce "closed" representations (i.e., only those process executions explicitly foreseen in the model are allowed), declarative approaches yield "open" representations (i.e., every process execution is implicitly allowed, as long as it does not incur in the violation of some constraint).

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 108–152, 2022.
https://doi.org/10.1007/978-3-031-08848-3_4


(a) A process

(b) Imperative model

(c) Declarative specification

Fig. 1. Intuitive representation of the difference between imperative process models and declarative process specifications in the space of all execution traces. Diagram (a) represents a real process, which isolates the *allowed* (green, solid fill) behaviors from the *forbidden* (red, dotted fill) ones. Diagram (b) shows an *imperative process model* that stays within the boundaries of the process, but misses many allowed behaviors. Diagram (c) shows a *declarative process specification* that well approximates the boundaries of the process: it accepts only traces that are allowed by the process, and includes all the traces accepted by the imperative model in (b). (Color figure online)

Figure 1 depicts an intuitive representation of the difference between classical imperative process models and declarative process specifications, considering execution traces that are forbidden by the real process, allowed by the real process, and captured by the designed process specification. Imperative models (such as those based on Petri nets and related formalisms) are suited to explicitly capture control-flow patterns like sequences, choices, concurrent sections, and loops. Those patterns, in turn, lend themselves to characterize a subset of the allowed traces, but struggle in covering the whole space of execution paths in the case of loosely structured, flexible processes. In other words, they favor control over flexibility. Contrariwise, declarative specifications strive to balance flexibility and control by attempting to characterize constraints that well-separate the allowed behaviors from the forbidden ones. In other words, declarative process specifications allow us to capture not only *what is expected* to occur, but also *what should not happen*. This helps in better approximating the boundaries of the real process, containing (and extending) those captured via imperative process models.

The idea of adopting a constraint-based, declarative approach to regulate dynamic systems has been originally brought forward in different communities: in data management, to express cascaded transactional updates [26]; in multiagent systems, to regulate agent interaction protocols [88]; and in business process management, to capture subprocesses that foresee loosely-coupled control-flow conditions on their activities [85]. This idea was further developed within BPM in consequent years, leading to a series of declarative, constraint-based process modeling languages, with two prominent exponents: Declare [76] and Dynamic Condition-Response Graphs [49]. Common to all such approaches is the usage of *linear temporal/dynamic logics* (i.e., temporal/dynamic logics for sequences of events) to formally describe specifications, and the exploitation of corresponding reasoning mechanisms to tackle a variety of concrete tasks along the entire process lifecycle, from design and model analysis to runtime execution and data analysis.

In this chapter, we focus on *declarative process mining*, that is, process mining where the input or output models are specified using declarative, constraint-based languages. Concretely, we employ the Declare language, but all the presented ideas seamlessly apply any language that can be formalized using logics over *finite* traces [30], which are indeed at the core of Declare. Focusing on finite traces reflects the intuition that every process instance is expected to complete in a finite number of steps. This aspect has a significant impact on the corresponding operational techniques, as these logics admit an automata-theoretic characterization that is based on standard *finite-state automata* [27,30], instead of automata on infinite structures, which are needed when such logics are interpreted over infinite traces.

Leveraging automata-based techniques paired with suitable measures relating traces, events and constraints, we review three interconnected fundamental declarative process mining tasks:

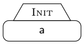
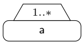
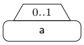
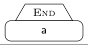
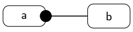
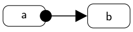
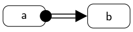
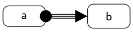
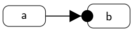
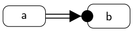
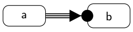
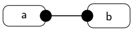
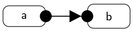
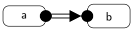
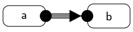
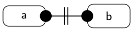
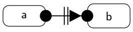
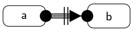
Reasoning – to uncover relationships among different constraints, and check key properties of Declare specifications;
Discovery – to extract a Declare specification that suitably characterizes the traces contain in an event log;
Monitoring – to provide operational decision support [63] by checking at run-time whether a running process execution satisfies a Declare specification, promptly detecting and reporting violations.

All the presented techniques are integrated in the MINERful process discovery technique[1] [40] and the RuM toolkit[2] [4].

The chapter is organized as follows. Section 2 introduces the declarative process specification language Declare alongside a running example to which we will refer throughout the remainder of the chapter. Section 3 provides the fundamental notions upon which the core techniques for reasoning, discovery and monitoring on declarative specifications are based. We define the formal semantics of Declare and discuss the core reasoning tasks for declarative specifications in Sect. 4. Section 5 explains the core notions of declarative process discovery and monitoring. Section 6 discusses the latest advances in the field of declarative process specification mining. Finally, Sect. 7 concludes this chapter with final remarks and a summary of the core concepts illustrated herein.

[1] https://github.com/cdc08x/MINERful.
[2] https://rulemining.org.

Table 1. A set of Declare constraints among those that are typically used for process mining, with their textual description, graphical notation, and examples fulfilling or violating them.

Constraint	**Explanation**	**Examples**				**Notation**
Existence constraints						
Init(a)	a is the *first* to occur	✓⟨a, c, c⟩	✓⟨a, b, a, c⟩	×⟨c, c⟩	×⟨b, a, c⟩	
AtLeastOne(a)	a occurs at least *once*	✓⟨b, c, a, c⟩	✓⟨b, c, a, a, c⟩	×⟨b, c, c⟩	×⟨c⟩	
AtMostOne(a)	a occurs at most *once*	✓⟨b, c, c⟩	✓⟨b, c, a, c⟩	×⟨b, c, a, a, c⟩	×⟨b, c, a, c, a, a⟩	
End(a)	a is the *last* to occur	✓⟨b, c, a⟩	✓⟨b, a, c, a⟩	×⟨b, c⟩	×⟨b, a, c⟩	
Relation constraints						
RespondedExistence(a, b)	If a occurs in the trace, then b occurs as well	✓⟨b, c, a, a, c⟩	✓⟨b, c, c⟩	×⟨c, a, a, c⟩	×⟨a, c, c⟩	
Response(a, b)	If a occurs, then b occurs after a	✓⟨c, a, a, c, b⟩	✓⟨b, c, c⟩	×⟨c, a, a, c⟩	×⟨b, a, c, c⟩	
AlternateResponse(a, b)	Each time a occurs, then b occurs afterwards, and no other a recurs in between	✓⟨c, a, c, b⟩	✓⟨a, b, c, a, c, b⟩	×⟨c, a, a, c, b⟩	×⟨b, a, c, a, c, b⟩	
ChainResponse(a, b)	Each time a occurs, then b occurs immediately afterwards	✓⟨c, a, b, b⟩	✓⟨a, b, c, a, b⟩	×⟨c, a, c, b⟩	×⟨b, c, a⟩	
Precedence(a, b)	b occurs only if preceded by a	✓⟨c, a, c, b, b⟩	✓⟨a, c, c⟩	×⟨c, c, b, b⟩	×⟨b, a, c, c⟩	
AlternatePrecedence(a, b)	Each time b occurs, it is preceded by a and no other b can recur in between	✓⟨c, a, c, b, a⟩	✓⟨a, b, c, a, a, c, b⟩	×⟨c, a, c, b, b, a⟩	×⟨a, b, b, a, b, c, b⟩	
ChainPrecedence(a, b)	Each time b occurs, then a occurs immediately beforehand	✓⟨a, b, c, a⟩	✓⟨a, b, a, a, b, c⟩	×⟨b, c, a⟩	×⟨b, a, a, c, b⟩	
Mutual relation constraints						
CoExistence(a, b)	If b occurs, then a occurs, and vice versa	✓⟨c, a, c, b, b⟩	✓⟨b, c, c, a⟩	×⟨c, a, c⟩	×⟨b, c, c⟩	
Succession(a, b)	a occurs if and only if it is followed by b	✓⟨c, a, c, b, b⟩	✓⟨a, c, c, b⟩	×⟨b, a, c⟩	×⟨b, c, c, a⟩	
AlternateSuccession(a, b)	a and b if and only if the latter follows the former, and they alternate each other in the trace	✓⟨c, a, c, b, a, b⟩	✓⟨a, b, c, a, b, c⟩	×⟨c, a, a, c, b, b⟩	×⟨b, a, c⟩	
ChainSuccession(a, b)	a and b occur if and only if the latter immediately follows the former	✓⟨c, a, b, a, b⟩	✓⟨c, c, c⟩	×⟨c, a, c, b⟩	×⟨c, b, a, c⟩	
Negative relation constraints						
NotCoExistence(a, b)	a and b never occur together	✓⟨c, c, c, b, b, b⟩	✓⟨c, c, a, c⟩	×⟨a, c, c, b, b⟩	×⟨b, c, a, c⟩	
NotSuccession(a, b)	b cannot occur after a	✓⟨b, b, c, a, a⟩	✓⟨c, b, b, c, a⟩	×⟨a, a, c, b, b⟩	×⟨a, b, b⟩	
NotChainSuccession(a, b)	a and b cannot occur contiguously	✓⟨a, c, b, a, c, b⟩	✓⟨b, b, a⟩	×⟨a, b, c, a, b⟩	×⟨c, a, b, c⟩	

2 Declare: A Gentle Introduction

Declare is a language and graphical notation providing an extendible repertoire of templates to formulate constraints. The origin of the approach traces back to the PhD work by Pesic [75], and the parallel and consequent study in the PhD work by Montali [67]. Notably, Declare actually stems from three initial lines of research, respectively focused on the declarative specification of business processes (cf. the ConDec language [78]), service choreographies (cf. the DecSerFlow language [70,94]), and clinical guidelines (cf. the CigDec language [72]). These lines were then unified into a single research thread. The term Declare was used for the first time in [76].

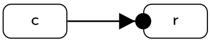
Table 1 shows a set of Declare constraints we use throughout this chapter. The whole, core set of Declare templates has been inspired by a catalogue of temporal logic patterns used in model checking for a variety of dynamic systems from different application domains [41].

Formally, we define a declarative process specification as follows.

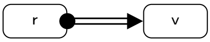
Definition 1 (Declarative process specification). *A* declarative process specification *is a tuple* $\mathrm{DS} = (\textsc{Rep}, \mathrm{Act}, K)$ *where*

- $\textsc{Rep}$ *is a finite non-empty set of* templates*, where each template is a predicate* $\textsc{k}(x_1, \ldots, x_m) \in \textsc{Rep}$ *on variables* $x_1, \ldots, x_m$ *(with* $m \in \mathbb{N}$ *the* arity *of* $\textsc{k}$*),*
- Act *is a finite non-empty set of* activities*,*
- K *is a finite set of* constraints*, namely pairs* $(\textsc{k}(x_1, \ldots, x_m), \kappa)$ *where* $\textsc{k}(x_1, \ldots, x_m)$ *is a template from* $\textsc{Rep}$*, and* κ *is a mapping that, for every* $i \in \{1, \ldots, m\}$ *assigns variable* x_i *with an activity* $\kappa(x_i) = a_i \in \mathrm{Act}$*; we compactly denote such a constraint with* $\textsc{k}(a_1, \ldots, a_m)$*.* ◁

Example 1 (A Declare process specification). Figure 2 portrays an example of declarative specification for the admission process of an international Bachelor's program. This example considers the Declare repertoire of templates. The process begins with the creation of an account in the university portal (henceforth, c). To specify that c is the initial task, we write Init(c), graphically depicted with the Init label in the tag on top of the activity box. Init is a unary template and Init(c) assigns its variable with activity c. Unary templates in Declare are also known as *existence* templates. We indicate that not more than one account can be created per process run with AtMostOne(c). In the diagram, it is indicated with the 0..1 label in the tag.

To register for a selection round (r), an account must have been created before (Precedence(c, r)). Precedence is a binary template and Precedence(c, r), graphically depicted as 

, assigns c and r to its first and second variable, respectively. Binary templates in Declare are commonly named as *relation* templates.

Every registration to a selection round (r) gives access to a uniquely corresponding evaluation phase (v). After r, v eventually follows and no other registrations are allowed until v completes. We write AlternateResponse(r, v), graphically depicted as 

. The evaluation requires r to

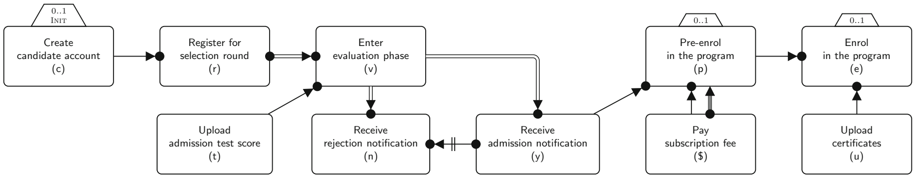
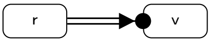
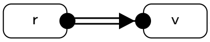
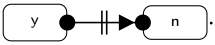
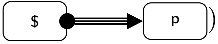


Fig. 2. The DECLARE map of the admission process at a university.

be completed before and v will not recur unless a new registration is issued: ALTERNATEPRECEDENCE(r, v), . Typically, if both ALTERNATERESPONSE(r, v) and ALTERNATEPRECEDENCE(r, v) hold true, we compactly represent them jointly with the *mutual* relation constraint ALTERNATESUCCESSION(r, v) . An admission test score has to be uploaded in the platform to access the evaluation phase: PRECEDENCE(t, v). Evaluation phases are necessary for the committee to return rejections (n) and notifications of admission (y), thus ALTERNATEPRECEDENCE(v, y) and ALTERNATEPRECEDENCE(v, n) hold.

After the admission has been notified, the candidate will not receive a rejection any longer – NOTRESPONSE(y, n), drawn in Fig. 2 as . NOTRESPONSE(y, n) falls under the category of the *negative* relation constraints, as the occurrence of y *disables* n in the remainder of the process execution.

Only if candidates receive a notification of admission, they will be entitled to pre-enrol in the program (PRECEDENCE(y, p)). The candidates are considered as pre-enrolled immediately after they pay the subscription fee (CHAINRESPONSE($, p), shown as follows in the diagram: ). Also, candidates cannot be considered as pre-enrolled if they have not paid the subscription fee: PRECEDENCE($, p). Not more than one pre-enrolment is allowed per candidate: ATMOSTONE(p). To enrol in the program (e), the candidate must have pre-enrolled – PRECEDENCE(p, e) – and uploaded the necessary school and language certificates – PRECEDENCE(u, e).

So far, we have been attaching an informal semantics to DECLARE and its templates. In the next section, we provide a more systematic and formal characterization.

3 Formal Background

Considering that DECLARE templates have been originally defined starting from a catalogue of Linear Temporal Logic (LTL) patterns [41], it is not surprising that temporal logics have been used to characterize the semantics of DECLARE since the very beginning. However, the fact that DECLARE specifications are interpreted over finite-length executions calls for the use of Linear Temporal Logic on Finite Traces (LTL$_f$) [30]. This indeed leads to a setting that is radically different, both semantically and algorithmically, from the traditional one where formulae are interpreted using LTL over infinite, recurring behaviors [29].

A complete formalization of Declare templates, also including an alternative formalization using a logic programming-based approach, can be found in [68]. It was later refined in [29]. In his PhD thesis, Di Ciccio was the first to provide a semantics based on regular expressions [36]. These two themes were later unified in [28], leading to a richer framework that is able to declaratively capture constraints and metaconstraints, that is, constraints predicating over the possible/certain satisfaction and violation of other constraints.

In this section, we provide some necessary background on LTL$_f$ and its extension with past-tense temporal operators, as well as on the automata-theoretic characterization for this logic. We then use this framework to formalize Declare and reason automatically on Declare specifications. Thereupon, we reflect upon the most recent advances of research in attempting at capturing not only the formal semantics of constraints, but also how they pragmatically interact with relevant events.

3.1 Linear Temporal Logic on Finite Traces

LTL$_f$ has the same syntax of LTL [80], but is interpreted on finite traces. In this chapter, in particular, we consider the LTL dialect including past modalities [56] for declarative process specifications as in [18].

From now on, we fix a finite set Σ representing an alphabet of propositional symbols describing (names of) activities available in the domain under study. A (finite) *trace* $t = \langle a_1, \dots, a_n \rangle \in \Sigma$ of length $|t| = n$ is a finite sequence of activities, where the presence of activity a_i at instant i of the trace represents an *event* that witnesses the occurrence of a_i at instant i – which we also write $t(i) = a_i$. Notice that *at each instant we assume that one and only one activity occurs.* Using standard notation from regular expressions, the set Σ^* denotes the overall set of traces whose constitutive events refer to activities in Σ.

Definition 2 (Syntax of LTL$_f$). *Well-formed formulae are built from Σ, the unary temporal operators $\bigcirc$ (next) and $\ominus$ (yesterday), and the binary temporal operators* $\mathbf{U}$ *(until) and* $\mathbf{S}$ *(since) as follows:*

$$\varphi ::= \mathsf{a} \mid (\neg\varphi) \mid (\varphi_1 \wedge \varphi_2) \mid (\bigcirc\varphi) \mid (\varphi_1 \ \mathbf{U} \ \varphi_2) \mid (\ominus\varphi) \mid (\varphi_1 \ \mathbf{S} \ \varphi_2)$$

where $\mathsf{a} \in \Sigma$. ◁

Definition 3 (Semantics of LTL$_f$, satisfaction, validity, entailment). *An* LTL$_f$ *formula φ is inductively* satisfied *in some instant i $(1 \leq i \leq n)$ of a trace t of length $n \in \mathbb{N}$, written $t, i \vDash \varphi$, if the following holds:*

- $t, i \vDash \mathsf{a}$ *iff* $t(i)$ *is assigned with* a;
- $t, i \vDash \neg\varphi$ *iff* $t, i \nvDash \varphi$;
- $t, i \vDash \varphi_1 \wedge \varphi_2$ *iff* $t, i \vDash \varphi_1$ *and* $t, i \vDash \varphi_2$;
- $t, i \vDash \bigcirc\varphi$ *iff* $i < n$ *and* $t, i+1 \vDash \varphi$;

- *$t, i \vDash \ominus \varphi$ iff $i > 1$ and $t, i-1 \vDash \varphi$;*
- *$t, i \vDash \varphi_1 \ \mathbf{U}\ \varphi_2$ iff $t, j \vDash \varphi_2$ with $i \leq j \leq n$, and $t, k \vDash \varphi_1$ for all k s.t. $i \leq k < j$;*
- *$t, i \vDash \varphi_1 \ \mathbf{S}\ \varphi_2$ iff $t, j \vDash \varphi_2$ with $1 \leq j \leq i$, and $t, k \vDash \varphi_1$ for all k s.t. $j < k \leq i$.*

A formula φ is satisfied by *a trace t (equivalently, t* satisfies *φ), written $t \vDash \varphi$, iff $t, 1 \vDash \varphi$. A formula φ is: (i)* satisfiable *if it has a satisfying trace from Σ^*; (ii)* valid *if every trace in Σ^* satisfies it. A formula φ_1* entails *formula φ_2, written $\varphi_1 \models \varphi_2$, if, for every trace t of length $n \in \mathbb{N}$ and every i s.t. $1 \leq i \leq n$, if $t, i \models \varphi$ then $t, i \models \psi$.* ◁

Since LTL$_f$ is closed under negation, it is easy to see that a formula φ is valid if and only if $\neg\varphi$ is unsatisfiable.

It is worth noting that, in LTL$_f$, the next operator is interpreted as the so-called *strong* next: $\bigcirc \varphi$ requires that the next instant exists within the trace, and that at such next instant φ holds. This has an important consequence: differently from LTL, in LTL$_f$ formula $\neg \bigcirc \varphi$ is *not* equivalent to $\bigcirc \neg\varphi$. This is because $\neg \bigcirc \varphi$ is true in an instant of a finite trace either when that instant has no successor, or the next instant exists and in such a next instant φ does not hold. More on this can be found in [29].

From the basic operators above, the following can be derived:

- Classical boolean abbreviations **true**, **false**, $\vee$, $\rightarrow$;
- Constant **end** $\equiv \neg \bigcirc$ **true**, denoting the last instant of a trace;
- Constant **start** $\equiv \neg \ominus$ **true**, denoting the first instant of a trace;
- $\Diamond \varphi \equiv$ **true U** φ indicating that φ eventually holds true in the trace (hence, before or at **end**);
- $\varphi_1 \ \mathbf{W}\ \varphi_2 \equiv (\varphi_1 \ \mathbf{U}\ \varphi_2) \vee \Box \varphi_1$, which relaxes **U** as φ_2 may never hold true;
- $\diamondminus \varphi \equiv$ **true S** φ indicating that φ holds true at some instant before the current one (i.e., after **start** in the trace);
- $\Box \varphi \equiv \neg \Diamond \neg\varphi$ indicating that φ holds true from the current instant till **end**;
- $\boxminus \varphi \equiv \neg \diamondminus \neg\varphi$ indicating that φ holds true from **start** to the current instant.

Example 2. Let $t = \langle a, b, b, c, d, e\rangle$ be a trace and φ_1, φ_2 and φ_3 three LTL$_f$ formulae defined as follows: $\varphi_1 \doteq d$; $\varphi_2 \doteq \Diamond b$; $\varphi_3 \doteq \Box(b \rightarrow \Diamond d)$. We have that $t, 1 \nvDash \varphi_1$ whereas $t, 5 \vDash \varphi_1$; $t, 1 \vDash \varphi_2$ whereas $t, 5 \nvDash \varphi_2$; $t, 1 \vDash \varphi_3$ and $t, 5 \vDash \varphi_3$ (in fact, $t, i \vDash \varphi_3$ for any instant $1 \leq i \leq n$). ◁

3.2 Finite-State Automata

One of the central features of LTL$_f$ is that a finite state automaton (FSA) [22] $\mathscr{A}(\varphi)$ can be computed such that for every trace t we have that $t \vDash \varphi$ iff t is in the language recognized by $\mathscr{A}(\varphi)$, as illustrated in [18,28,30,38]. We include the main notions next, recalling that focusing on deterministic FSAs is without loss of generality, as over finite traces every non-deterministic FSAs can be determinized [50].

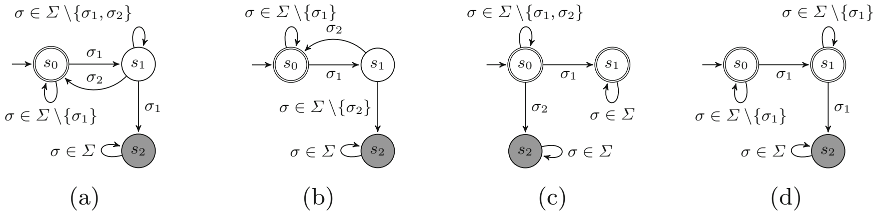

$\sigma \in \Sigma \setminus \{\sigma_1, \sigma_2\}$
s_0
σ_1
s_1
σ_2
$\sigma \in \Sigma \setminus \{\sigma_1\}$
σ_1
$\sigma \in \Sigma$
s_2
(a)
$\sigma \in \Sigma \setminus \{\sigma_1\}$
σ_2
s_0
s_1
σ_1
$\sigma \in \Sigma \setminus \{\sigma_2\}$
$\sigma \in \Sigma$
s_2
(b)
$\sigma \in \Sigma \setminus \{\sigma_1, \sigma_2\}$
s_0
s_1
σ_1
σ_2
$\sigma \in \Sigma$
s_2
$\sigma \in \Sigma$
(c)
$\sigma \in \Sigma \setminus \{\sigma_1\}$
s_0
s_1
σ_1
$\sigma \in \Sigma \setminus \{\sigma_1\}$
σ_1
$\sigma \in \Sigma$
s_2
(d)

Fig. 3. Examples of constraint FSAs.

Definition 4 (Finite state automaton (FSA)). *A (deterministic) finite state automaton (FSA) is a tuple $A = (\Sigma, S, \delta, s_0, S_F)$, where:*

- *Σ is a finite set of symbols;*
- *S is a finite non-empty set of states;*
- *$\delta : S \times \Sigma \to S$ is the* transition function, *i.e., a partial function that, given a starting state and a (labeled) transition, returns the target state;*
- *s_0 is the initial state;*
- *$S_F \subseteq S$ is the set of final (accepting) states $s_F \in S_F$*

◁

In the remainder of the chapter, we assume that δ is left-total and surjective on $S \setminus \{s_0\}$, that is, the transition function is defined for every state and symbol, and every state is on a path from the initial one – with the possible exception of the initial state itself. An FSAs that is left-total is called *untrimmed*. Notice that these two requirements are without loss of generality: every FSA can be converted into an equivalent FSA that is left-total and surjective. In particular, to make an FSAs untrimmed, it is sufficient to: *(i)* introduce a non-final trap state $s_\perp$; *(ii)* for every state s and symbol a' such that $\delta(s, a')$ is *not* defined, enforce $\delta(s, a') = s_\perp$; *(iii)* connect $s_\perp$ to itself for every symbol, setting $\delta(s_\perp, a) = s_\perp$ for every $a \in \Sigma$.

Example 3. Figure 3 depicts four FSAs. States are represented as circles and transitions as arrows. Accepting states are decorated with a double line. The initial state is indicated with a single, unlabeled incoming arc. For instance, Fig. 3(a) is such that $\Sigma \supseteq \{\sigma_1, \sigma_2\}$, $S = \{s_0, s_1, s_2\}$, $S_\mathrm{F} = \{s_0\}$, $\delta(s_0, \sigma_1) = s_1$ and $\delta(s_1, \sigma_1) = s_2$. ◁

Definition 5 (Runs and traces of an FSA). *Let $A = (\Sigma, S, \delta, s_0, S_F)$ be an FSA as per Definition 4. A* computation *π of A is a finite sequence alternating states and activities $s_0 \xrightarrow{\sigma_0} \ldots \xrightarrow{\sigma_{n-1}} s_n$ that starts from the initial state s_0 is such that for every $0 \leq i < n$, we have $\delta(s_i, \sigma_i) = s_{i+1}$. If π terminates in a final state, that is, $s_n \in S_F$, then it is a* run, *and* induces *a corresponding trace $\sigma_0, \ldots, \sigma_{n-1}$ over Σ^* obtained from π by only keeping the symbols that label the transitions.* ◁

Example 4. In Fig. 3(a), $\pi_1 = s_0 \xrightarrow{\sigma_1} s_1$, $\pi_2 = s_0 \xrightarrow{\sigma_2} s_0 \xrightarrow{\sigma_1} s_1 \xrightarrow{\sigma_1} s_2$, and $\pi_3 = s_0 \xrightarrow{\sigma_1} s_1 \xrightarrow{\sigma_2} s_2 \xrightarrow{\sigma_1} s_0$ are three examples of computations. However, only π_3 is a run because $s_0 \in S_\text{F}$ whereas $s_1, s_2 \notin S_\text{F}$. Notice that, in Fig. 3, we additionally highlight with a grey background colour those states that cannot be in a step of a run – that is, from which accepting states cannot be reached (e.g., s_2 in Fig. 3(a)). ◁

Definition 6 (Accepted trace, language of an FSA). *A trace $t \in \Sigma^*$ is* accepted *by FSA $A = (\Sigma, S, \delta, s_0, s_F)$ if there is a run of A inducing t. The* language *$\mathscr{L}(A)$ of A is the set of traces accepted by A.* ◁

Example 5. For the FSA in Fig. 3(a), the language contains the trace $t_1 = \langle \sigma_1, \sigma_2, \sigma_1 \rangle$, since a run exists over this sequence of labels (i.e., π_3 above), whereas $t_2 = \langle \sigma_2, \sigma_1 \rangle$ is not part of the language. ◁

Automata Product. FSAs are closed under the (synchronous) product operation × [81]. The (cross-)product $A \times A'$ of two FSAs A and A' is an FSA that accepts the intersection of languages (sets of accepted traces) of each operand: $\mathscr{L}(A \times A') = \mathscr{L}(A) \bigcap \mathscr{L}(A')$. It is defined as follows.

Definition 7 (Automata product). *The* product FSA *of two FSAs $A = (\Sigma, S, \delta, s_0, S_F)$ and $A' = (\Sigma, S', \delta', s'_0, S'_F)$ over the same alphabet Σ is the FSA $A \times A' = (\Sigma, S^\times, \delta^\times, s_0^\times, S_F^\times)$, where the set $S^\times \subseteq S \times S'$ of states (obtained from the cartesian product of the states in A and A'), its initial state $s_0^\times$, its final states $S_F^\times$, and the transition function $\delta^\times$, are defined by simultaneous induction as follows:*

- *$s_0^\times = \langle s_0, s'_0 \rangle \in S^\times$;*
- *For every state $\langle s_1, s'_1 \rangle \in S^\times$, state $s_2 \in S$, state $s'_2 \in S'$, and label $\ell \in \Sigma$, if $\delta(s_1, \ell) = s_2$ and $\delta'(s'_1, \ell) = s'_2$ then: (i) $\langle s_2, s'_2 \rangle \in S^\times$, (ii) $\delta^\times(\langle s_1, s'_1 \rangle, \ell) = \langle s_2, s'_2 \rangle$, (iii) if $s_2 \in S_F$ and $s'_2 \in S'_F$, then $\langle s_2, s'_2 \rangle \in S_F^\times$.*
- *Nothing else is in $S_F^\times$, $S^\times$, and $\delta^\times$.*

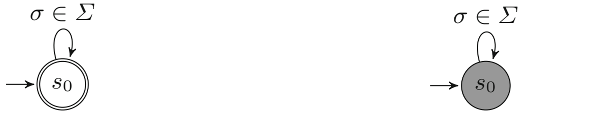
◁

Notice that the FSA constructed with Definition 7 can be manipulated using language-preserving automata operations, such as in particular *minimization* [50].

The product operation × is commutative and associative. The identity element for × over alphabet Σ is $A^\text{I} = (\Sigma, \{s_0\}, s_0, \{s_0\} \times \Sigma \times \{s_0\}, \{s_0\})$ – depicted in Fig. 4(a). It accepts all traces over Σ: $\mathscr{L}\big(A^\text{I}\big) = \mathbb{P}(\Sigma^*)$ as any sequence of transitions labeled by symbols in Σ corresponds to a run for A^I. The absorbing element is $A^\emptyset = (\Sigma, \{s_0\}, s_0, \{s_0\} \times \Sigma \times \{s_0\}, \emptyset)$ and is illustrated in Fig. 4(b). It does not accept any trace at all: $\mathscr{L}\big(A^\emptyset\big) = \emptyset$ as any sequence of transitions labeled by symbols in Σ corresponds to a computation ending in a non-accepting state.

4 Reasoning

Equipped with the notions acquired thus far, we can now discuss the core reasoning tasks that are associated to declarative process specifications. To this end, we begin this section by describing the semantics of Declare in detail.



(a) Identity element (b) Absorbing element

Fig. 4. Finite state automata acting as identity element and absorbing element for the automata cross-product operation.

4.1 Semantics of Declare

The semantics of a Declare template $\text{K}(x_1, \ldots, x_m)$ is given as an LTL$_f$ formula $\varphi_{\text{K}(x_1,\ldots,x_m)}$ defined over variables $x_1, \ldots, x_m$ instead of activities. Given the free variables x and y, e.g., Response(x, y) corresponds to $\square(x \rightarrow \lozenge\, y)$, witnessing that whenever x occurs, then y is expected to occur at some later instant. Table 2 shows the LTL$_f$ formulae of some templates of the Declare repertoire. The formalization of a constraint is then obtained by grounding the LTL$_f$ formula of its template.

Definition 8 (Constraint formula, satisfying trace). *The* formula *of constraint* $\text{K}(a_1, \ldots, a_m)$*, written* $\varphi_{\text{K}(a_1,\ldots,a_m)}$*, is the* LTL$_f$ *formula obtained from* $\varphi_{\text{K}(x_1,\ldots,x_m)}$ *by replacing* x_i *with* a_i *for each* $1 \leq i \leq m$*. A trace* t satisfies $\text{K}(a_1, \ldots, a_m)$ *if* $t \models \varphi_{\text{K}(a_1,\ldots,a_m)}$*; otherwise, we say that* t violates $\text{K}(a_1, \ldots, a_m)$*.* ◁

Example 6. Considering Table 2, we have $\varphi_{\text{Response}(\mathsf{a},\mathsf{b})} = \square(\mathsf{a} \rightarrow \lozenge\, \mathsf{b})$, and $\varphi_{\text{Response}(\mathsf{b},\mathsf{c})} = \square(\mathsf{b} \rightarrow \lozenge\, \mathsf{c})$. Traces $\langle \mathsf{b} \rangle$ and $\langle \mathsf{a}, \mathsf{b}, \mathsf{a}, \mathsf{a}, \mathsf{c}, \mathsf{b} \rangle$ satisfy Response(a, b), while $\langle \mathsf{a} \rangle$ and $\langle \mathsf{a}, \mathsf{b}, \mathsf{a}, \mathsf{a}, \mathsf{c} \rangle$ do not. ◁

A Declare specification is then formalized by conjoining all its constraint formulae, thus obtaining a direct, declarative notion of *model trace*, that is, a trace that is accepted by the specification.

Definition 9 (Specification formula, model trace). *The* formula *of* Declare *specification* $\text{DS} = (\text{Rep}, \text{Act}, K)$*, written* φ_{DS}*, is the* LTL$_f$ *formula* $\bigwedge_{\text{K} \in K} \varphi_{\text{K}}$*. A trace* $t \in \text{Act}^*$ *is a* model trace *of* DS *if* $t \models \varphi_{\text{DS}}$*; in this case, we say that* t *is* accepted by DS*, otherwise that* t *is* rejected by DS*.* ◁

Constructing constraint and specification formulae is, however, not enough. When one reads $\square(\mathsf{a} \rightarrow \lozenge\, \mathsf{b})$ following the textual description given above, the formula gets intepreted as "whenever a occurs, then b is expected to occur at some later instant". This formulation intuitively hints at the fact that the occurrence of a *activates* the Response(a, b) constraint, requiring the *target* b to occur. In turn, we get that a trace not containing any occurrence of a is *less interesting* than a trace containing occurrences of a, each followed by one or more occurrences of b: even though both traces satisfy Response(a, b), the first trace never "interacts"

Table 2. Semantics of some Declare constraints.

Template	**LTL$_f$expression [18,30]**	**Activation**	**Target**
Existence constraints			
AtLeastOne(x)	$\Box(\textbf{start} \rightarrow \Diamond x)$	**start**	$\Diamond x$
AtMostOne(x)	$\Box(x \rightarrow \neg \bigcirc \Diamond x)$	x	$\neg \bigcirc \Diamond x$
Init(x)	$\Box(\textbf{start} \rightarrow x)$	**start**	x
End(x)	$\Box(\textbf{end} \rightarrow x)$	**end**	x
Relation constraints			
RespondedExistence(x, y)	$\Box(x \rightarrow \Diamond y \vee \diamondminus y)$	x	$\Diamond y \vee \diamondminus y$
Response(x, y)	$\Box(x \rightarrow \Diamond y)$	x	$\Diamond y$
AlternateResponse(x, y)	$\Box(x \rightarrow \bigcirc(\neg x \;\mathbf{U}\; y))$	x	$\bigcirc(\neg x \;\mathbf{U}\; y)$
ChainResponse(x, y)	$\Box(x \rightarrow \bigcirc y)$	x	$\bigcirc y$
Precedence(x, y)	$\Box(y \rightarrow \diamondminus x)$	y	$\diamondminus x$
AlternatePrecedence(x, y)	$\Box(y \rightarrow \ominus(\neg y \;\mathbf{S}\; x))$	y	$\ominus(\neg y \;\mathbf{S}\; x)$
ChainPrecedence(x, y)	$\Box(y \rightarrow \ominus x)$	y	$\ominus x$
Negative relation constraints			
NotRespondedExistence(x, y)	$\Box(x \rightarrow (\Box \neg y \wedge \boxminus \neg y))$	x	$\Box \neg y \wedge \boxminus \neg y$
NotResponse(x, y)	$\Box(x \rightarrow \Box \neg y)$	x	$\Box \neg y$
NotChainResponse(x, y)	$\Box(x \rightarrow \neg \bigcirc y)$	x	$\neg \bigcirc y$
NotPrecedence(x, y)	$\Box(y \rightarrow \boxminus \neg x)$	y	$\boxminus \neg x$
NotChainPrecedence(y, x)	$\Box(y \rightarrow \neg \ominus x)$	y	$\neg \ominus x$

with Response(a, b), while the second does. This relates to the notion of vacuous satisfaction in LTL [51] and that of interestingness of satisfaction in LTL$_f$ [39].

The point is, all such considerations are not captured by the formula $\Box(\mathsf{a} \rightarrow \Diamond \mathsf{b})$, but are related to pragmatic interpretation of how it relates to traces. To see this aspect, let us consider that we can equivalently express the formula above as $\Box \neg\mathsf{a} \vee \Diamond(\mathsf{b} \wedge \Box \neg\mathsf{a})$, which now reads as follows: "Either a never happens at all, or there is some occurrence of b after which a never happens". This equivalent reformulation does not put into evidence the activation or the target.

This problem can be tackled in two possible ways. One option is to attempt at an automated approach where activation, target, and interesting satisfaction are semantically, implicitly characterized once and for all at the logical level; this is the route followed in [39]. The main drawback of this approach is that the user cannot intervene at all in deciding how to fine-tune the activation and target conditions. An alternative possibility is instead to ask the user to explicitly indicate, together with the LTL$_f$ formula φ of the template, also two related LTL$_f$ formulae expressing activation and target conditions for φ. This latter approach, implicitly adopted in [69] and then explicitly formalized in [18], gives more control to the user on how to *pragmatically* interpret constraints. We follow this latter approach.

Intuitively, the *activation* of a constraint is a triggering condition that, once made true, expects that the *target* condition is satisfied by the process execution.

Contrariwise, if the constraint is not activated, the satisfaction of the target is not enforced. All in all, to properly constitute an activation-target pair for an LTL$_f$ formula φ, we need them to satisfy the condition that whenever the current instant is such that the activation is satisfied, φ must behave equivalently to the target (thus requiring its satisfaction). This is formally captured as follows.

Definition 10 (Activation and target of a constraint). *The* activation *and* target *of a constraint* $\textsc{k}$ *over activities* Act *are two* LTL$_f$ *formulae* ${}_{\square}\textsc{k}$ *and* $\textsc{k}_{\blacktriangleright}$ *such that for every trace* $t \in \mathrm{Act}^*$ *we have that:*

$$t \vDash \varphi_{\textsc{k}} \quad \textit{iff} \quad t \vDash \square\left({}_{\square}\textsc{k} \rightarrow (\textsc{k}_{\blacktriangleright})\right)$$

Table 2 shows activations and targets for each constraint, inspired by the work of Cecconi et al. [18]. In the next example, we explain the rationale behind some of the constraint formulations in the table.

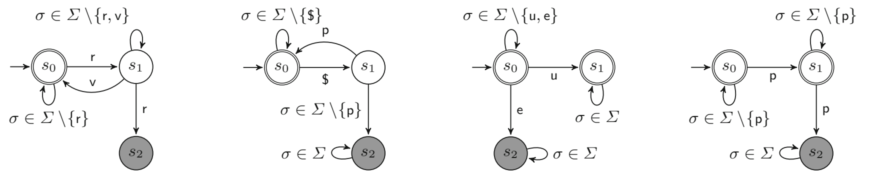
Example 7. Consider $\textsc{ChainResponse}(\$, \mathsf{p})$, dictating that whenever $\$$ occurs, then p is the activity occurring next. We have $\varphi_{\textsc{ChainResponse}(\$,\mathsf{p})} = \square(\$ \rightarrow \bigcirc \mathsf{p})$. Then, by Definition 10, we can directly fix ${}_{\square}\textsc{ChainResponse}(\$, \mathsf{p}) = \$$, and $\textsc{ChainResponse}(\$, \mathsf{p})_{\blacktriangleright} = \bigcirc \mathsf{p}$, respectively witnessing that every occurrence of $\$$ triggers the constraint, with a target requiring the consequent execution of p in the next instant. Similarly, for $\textsc{Precedence}(\$, \mathsf{p})$ we have $\varphi_{\textsc{Precedence}(\$,\mathsf{p})} = \square\,(\mathsf{p} \rightarrow \diamondminus\,\$)$, and in turn, by Definition 10, ${}_{\square}\varphi_{\textsc{Precedence}(\$,\mathsf{p})} = \mathsf{p}$ and $\varphi_{\textsc{Precedence}(\$,\mathsf{p})\blacktriangleright} = \diamondminus\,\$$. The case of $\textsc{AtMostOne}(\mathsf{p})$ is also similar. In this case, $\varphi_{\textsc{AtMostOne}(\mathsf{p})}$ formalizes that p cannot occur twice, which in LTL$_f$ can be directly captured by $\neg\Diamond(\mathsf{p} \wedge \bigcirc \Diamond \mathsf{p})$. This is logically equivalent to $\square(\mathsf{p} \rightarrow \neg \bigcirc \Diamond \mathsf{p})$, which directly yields ${}_{\square}\textsc{AtMostOne}(\mathsf{p}) = \mathsf{p}$ and $\textsc{AtMostOne}(\mathsf{p})_{\blacktriangleright} = \neg \bigcirc \Diamond \mathsf{p}$.

A quite different situation holds instead for the other *existence* constraints. Take, for example, $\textsc{AtLeastOne}(\mathsf{a})$, requiring that a occurs at least once in the execution. This can be directly encoded in LTL$_f$ as $\Diamond \mathsf{a}$. This formulation, however, does not help to individuate the activation and target of the constraint. Intuitively, we may disambiguate this by capturing that since the constraint requires the presence of a from the very beginning of the execution, the constraint is indeed activated at the beginning, i.e., when **start** holds, imposing the satisfaction of the target $\Diamond \mathsf{a}$. This intuition is backed up by Definition 10, using the semantics of **start** and noticing the following logical equivalences:

$$\Diamond \mathsf{a} = \textbf{start} \rightarrow \Diamond \mathsf{a} = \square\,(\textbf{start} \rightarrow \Diamond \mathsf{a})$$

This explains why the latter formulation is employed in Table 2. ◁

Declarative Constraints as FSAs. Crucial for our techniques is that every LTL$_f$ formula φ can be encoded into a corresponding FSA (in the sense of Definition 4) A_φ that recognizes all and only those traces that satisfy the formula. This can be done through different algorithmic techniques. A direct approach


$\sigma \in \Sigma \setminus \{\mathsf{r},\mathsf{v}\}$
s_0
s_1
r
v
r
$\sigma \in \Sigma \setminus \{\mathsf{r}\}$
s_2
$\sigma \in \Sigma \setminus \{\$\}$
p
s_0
s_1
$
$\sigma \in \Sigma \setminus \{\mathsf{p}\}$
$\sigma \in \Sigma$
s_2
$\sigma \in \Sigma \setminus \{\mathsf{u},\mathsf{e}\}$
s_0
s_1
u
e
$\sigma \in \Sigma$
s_2
$\sigma \in \Sigma$
$\sigma \in \Sigma \setminus \{\mathsf{p}\}$
s_0
s_1
p
$\sigma \in \Sigma \setminus \{\mathsf{p}\}$
p
$\sigma \in \Sigma$
s_2

(a) Alt.Resp.(r, v) (b) Chn.Resp.($\$, \mathsf{p}$) (c) Prec.($\mathsf{u}, \mathsf{e}$) (d) AtMostOne($\mathsf{p}$)

Fig. 5. Example FSAs of Declare constraints.

that transforms an input formula into a non-deterministic FSAs is presented in [28,29]; notice that the so-obtained FSAs can then be determinized and minimized using standard techniques [50,99]. A fortiori, given a Declare specification $\text{DS} = (\text{Rep}, \text{Act}, K)$, we proceed as follows:

- We pair each constraint $\kappa \in K$ to a corresponding, so-called *local automaton* A_κ. This automaton is the FSA A_{φ_κ} of the constraint formula φ_κ, and is used to characterize all and only those traces that satisfy κ;
- We pair the whole specification to a so-called *global automaton* A_{DS}, that is, the FSA $A_{\varphi_{\text{DS}}}$ of the constraint formula φ_{DS}. It thus recognizes all and only the model traces of DS. Recall that, as introduced in Definition 9, φ_{DS} is the conjunction of the formulae of the constraints in K, and thus the language $\mathscr{L}(A_{\text{DS}})$ corresponds to $\bigcap_{\kappa \in K} \mathscr{L}(A_\kappa)$. By definition of automata product, this means that $\mathscr{L}(A_{\text{DS}})$ can be obtained by computing the product of the local automata of the constraints in K.

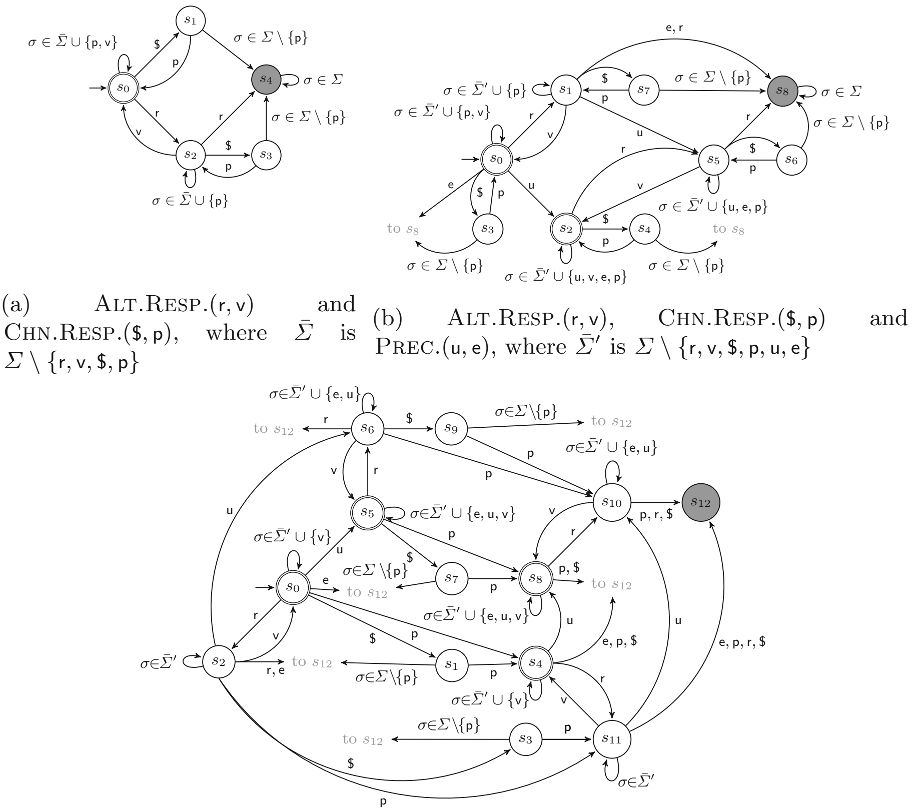
Figure 5 shows four local automata for constraints taken from our running example: AlternateResponse(r, v), ChainResponse($\$, \mathsf{p}$), Precedence($\mathsf{u}, \mathsf{e}$) and AtMostOne($\mathsf{p}$). Examples of global automata are instead given in Fig. 6.

In the remainder of this chapter, we will extensively use local and global automata for reasoning, discovery, and monitoring. Though out of scope for this chapter, it is also worth mentioning that the automata-based approach has also been used for simulation of Declare models and thereby the production of event logs from declarative specifications [37], and also to define enactment engines for Declare specifications [76,97].

4.2 Reasoning on Declare Specifications

Reasoning on a Declare specification is necessary to understand which model traces are supported and, in turn, to ascertain its correctness. Reasoning is also key to unveil how constraints interact with each other, and check whether activations and targets are properly defined. As we will see, this is instrumental not only to analyze specifications, but it is also an integral part of declarative process mining.

In general, reasoning on declarative specifications is of particular importance: while they enjoy flexibility, they typically do not explicitly indicate how execu-


s_0
s_1
s_2
s_3
s_4
$\sigma \in \bar{\Sigma} \cup \{p, v\}$
$\sigma \in \Sigma \setminus \{p\}$
$\sigma \in \Sigma$
$\sigma \in \bar{\Sigma} \cup \{p\}$
$, p, r, v
s_5
s_6
s_7
s_8
e, r
$\sigma \in \bar{\Sigma}' \cup \{p\}$
$\sigma \in \bar{\Sigma}' \cup \{p, v\}$
$\sigma \in \bar{\Sigma}' \cup \{u, e, p\}$
$\sigma \in \bar{\Sigma}' \cup \{u, v, e, p\}$
to s_8
u, e
s_9
s_{10}
s_{11}
s_{12}
to s_{12}
$\sigma \in \bar{\Sigma}' \cup \{e, u\}$
$\sigma \in \bar{\Sigma}' \cup \{e, u, v\}$
$\sigma \in \bar{\Sigma}' \cup \{v\}$
$\sigma \in \bar{\Sigma}'$
$\sigma \in \Sigma \setminus \{p\}$
p, r, $
p, $
e, p, $
e, p, r, $
r, e

(a) Alt.Resp.(r, v) and Chn.Resp.($, p), where $\bar{\Sigma}$ is $\Sigma \setminus \{r, v, \$, p\}$

(b) Alt.Resp.(r, v), Chn.Resp.($, p) and Prec.(u, e), where $\bar{\Sigma}'$ is $\Sigma \setminus \{r, v, \$, p, u, e\}$

(c) Alt.Resp.(r, v), Chn.Resp.($, p), Prec.(u, e) and AtMostOne(p), where $\bar{\Sigma}'$ is $\Sigma \setminus \{r, v, \$, p, u, e\}$ (for the sake of readability, a few transitions to s_{12} are omitted)-

Fig. 6. Global automata for the interplay of Declare constraints.

tion has to be controlled. We have seen how this phenomenon concretely manifests itself in the context of Declare: traces conforming to the specification (that is, *model traces*) are only implicitly described as those that satisfy all the given constraints. Constraints, in turn, may be quite diverse from each other (e.g., indicating what *is expected* to occur, but also what should *not* happen) and, even more importantly, may affect each other in subtle, difficult to detect ways. This phenomenon is known, in the literature that studies the cognitive impact of languages and notations, under the name of *hidden dependencies* [47]. Hidden dependencies in Declare have been studied in [32, 70], and their impact on understandability and interpretability of declarative process models has spawned a dedicated line of research, started in [48].

We detail next key reasoning tasks in the context of Declare, substantiating how hidden dependencies enter into the picture. We show that all such reasoning tasks can be homogeneously tackled by a single check on the global automaton of the specification under study.

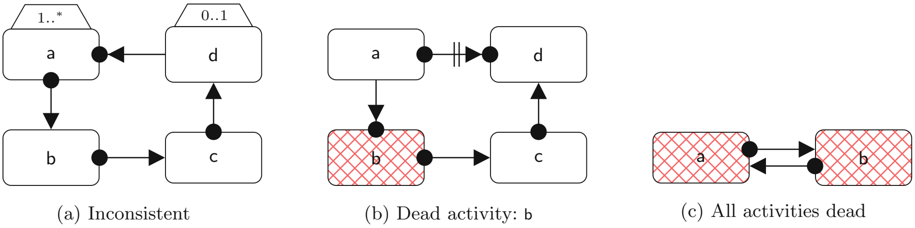
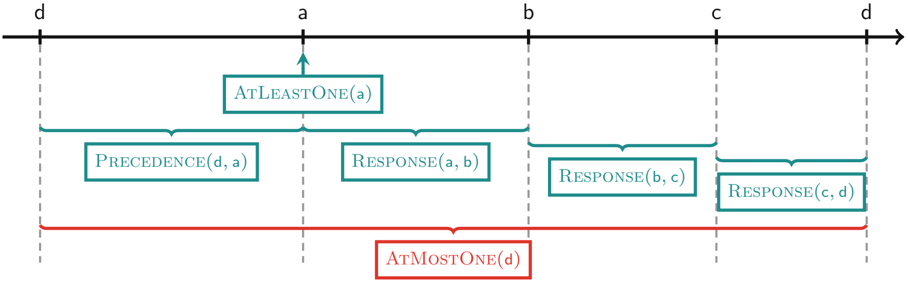


Fig. 7. Examples of incorrect DECLARE specifications.

Specification Consistency. This is the most fundamental task, defined as follows.

Definition 11 (Consistent specification). *A* DECLARE *specification* DS *is* consistent *if there exists at least one model trace for* DS*.* ◁

Example 8. Consider the DECLARE specification in Fig. 7(a). The specification is inconsistent. This is not due to conflicting constraints insisting on the same activity, but due to hidden dependencies arising from the interplay of multiple constraints. To see why the specification is inconsistent, we can try to construct a trace that satisfies some of the constraints in the model, until we reach a contradiction (i.e., the "trace pattern" constructed so far violates a constraint of the specification). This is graphically shown next:



The picture clearly depicts that ATLEASTONE(a) triggers:

- on the one hand PRECEDENCE(d, a), calling for a preceding occurrence of d;
- on the other hand, in cascade, RESPONSE(a, b), RESPONSE(b, c), and RESPONSE(c, d), calling for a later occurrence of d.

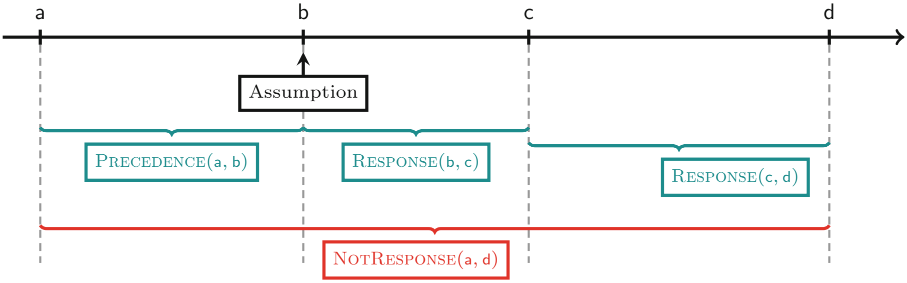
Considering the interplay of the involved constraints, d is required to occur in different instants, hence twice, in turn violating ATMOSTONE(d). ◁

By definition of model trace, it is immediate to see that DS *is consistent if and only if the* LTL$_f$ *specification formula* φ_{DS} *is satisfiable.* This, in turn, can be algorithmically verified by first constructing the global automaton A_{DS}, and then checking whether such an automaton is empty (i.e., it does not recognize any trace). Specifically, φ_{DS} *is* satisfiable *if and only if* A_{DS} *is non-empty.*

Detection of Dead Activities. This task amounts to check whether a Declare specification is over-constrained, in the sense that it contains an activity that can never be executed (in that case, such an activity is called *dead*).

Definition 12 (Dead activity). *Let* $\mathrm{DS} = (\textsc{Rep}, \mathrm{Act}, K)$ *be a* Declare *specification. An activity* $\mathsf{a} \in \mathrm{Act}$ *is* dead *in* DS *if there is no model trace of* DS *where* a *occurs.* ◁

Example 9. Consider the Declare specification in Fig. 7(b). The specification is consistent; as an example, trace $\langle \mathsf{c}, \mathsf{d} \rangle$ is a model trace. However, none of its model traces can foresee the execution of b. This can be seen if one tries to construct a trace containing an occurrence of b. The result is the following:



It is apparent that the presence of b requires a previous occurrence of a and, indirectly, a future occurrence of d, violating $\textsc{NotResponse}(\mathsf{a}, \mathsf{d})$. This shows that b is a dead activity.

Consider now the specification in Fig. 7(c). The situation here is trickier. The specification is consistent, as it accepts the empty trace (where no activity is executed, and hence none of the two response constraints present in the specification gets activated). However, none of the two activities a and b present therein can occur. As soon as this happens, the combination of the two response constraints cannot be *finitely* satisfied. In fact, an occurrence of a requires a later occurrence of b, which in turn requires a later occurrence of a, and so on and so forth, indefinitely. In other words, in every instant, one between $\textsc{Response}(\mathsf{a}, \mathsf{b})$ and $\textsc{Response}(\mathsf{b}, \mathsf{a})$ must be active and waiting for a later occurrence of its target, in a future instant. Since every instant must have a next instant, it is not possible to construct a satisfying (finite) trace. ◁

Dead activity detection can be directly reduced to (in)consistency of a specification. Specifically, activity a is dead in a Declare specification $\mathrm{DS} = (\textsc{Rep}, \mathrm{Act}, K)$ if and only if the specification $(\textsc{Rep}, \mathrm{Act}, K \cup \{\textsc{AtLeastOne}(\mathsf{a})\})$, obtained from DS by forcing the existence of a is inconsistent (i.e., its specification formula is not satisfiable).

Valid Activation and Target. To ensure that a Declare constraint $\textsc{k}$ comes with a valid activation ${}_{\square}\textsc{k}$ and target $\textsc{k}_{\blacktriangleright}$ for its formula $\varphi_{\textsc{k}}$, we can directly apply Definition 10 and check whether the LTL$_f$ formula $\varphi_{\textsc{k}} \leftrightarrow \Box({}_{\square}\textsc{k} \rightarrow \textsc{k}_{\blacktriangleright})$ is valid, that is, whether its negation is not satisfiable.

Checking Relations Between Constraints/Specifications. We establish two key relations between constraints/specifications. The first is that of *subsumption* between templates, leveraging the entailment relation between LTL$_f$ formulae to constraints. We formally define it as follows.

Definition 13 (Subsumption). *Let* $\textsc{k}(x_1,\ldots,x_m), \textsc{k}'(x_1,\ldots,x_m) \in \textsc{Rep}$ *two templates.* $\textsc{k}(x_1,\ldots,x_m)$ subsumes $\textsc{k}'(x_1,\ldots,x_m)$ *(in symbols,* $\textsc{k}(x_1,\ldots,x_m) \sqsubseteq \textsc{k}'(x_1,\ldots,x_m)$*) if, given any mapping* κ *assigning* $x_1,\ldots,x_m$ *with activities* $a_1,\ldots,a_m \in \mathrm{Act}$*,* $\varphi_{\textsc{k}(a_1,\ldots,a_m)} \models \varphi_{\textsc{k}'(a_1,\ldots,a_m)}$. ◁

This relation can be checked by verifying that $\varphi_{\textsc{k}(a_1,\ldots,a_m)} \rightarrow \varphi_{\textsc{k}'(a_1,\ldots,a_m)}$ is valid, that is, the negated formula $\varphi_{\textsc{k}(a_1,\ldots,a_m)} \wedge \neg\varphi_{\textsc{k}'(a_1,\ldots,a_m)}$ is not satisfiable for any $a_1,\ldots,a_m \in \mathrm{Act}$. For example, $\textsc{Alt.Prec.}(x,y) \sqsubseteq \textsc{Precedence}(x,y)$ as the former requires that y can occur only if preceded by x (just as the latter) *and* y does not recur in between. Therefore, every event that satisfies the former must satisfy the latter too. In the following, we shall lift this notion to constraints too (e.g., we say that $\textsc{AlternatePrecedence}(\mathsf{y},\mathsf{p})$ subsumes $\textsc{Precedence}(\mathsf{y},\mathsf{p})$).

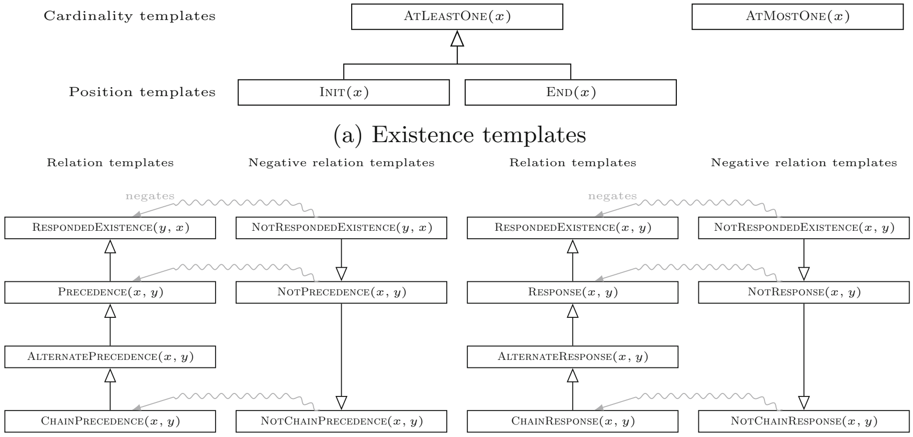
By Definition 8 and Definition 9, since both Declare constraints and specifications correspond to LTL$_f$ formulae, we can use subsumption for a twofold purpose:

- Consider two candidate constraints $\textsc{k}_1$ and $\textsc{k}_2$. If $\textsc{k}_1 \sqsubseteq \textsc{k}_2$, then we know that adding $\textsc{k}_1$ to a Declare specification will make the addition of $\textsc{k}_2$ irrelevant, and that adding $\textsc{k}_1$ or $\textsc{k}_2$ will determine whether the specification is more or less constraining.
- Consider a candidate constraint $\textsc{k}$ and a target specification DS. If the former logically entails the latter, $\varphi_{\mathrm{DS}} \models \varphi_{\textsc{k}}$, then $\textsc{k}$ is *redundant* in DS, and it makes no sense to include it in DS.

The second relation characterizes constraints that are the negated version of each other. Let $\textsc{k}_1$ and $\textsc{k}_2$ be two Declare constraints, coming with activation formulae ${}_{\square}\textsc{k}_1$ and ${}_{\square}\textsc{k}_2$ and target formulae $\textsc{k}_{1\blacktriangleright}$ and $\textsc{k}_{2\blacktriangleright}$, respectively. We say that $\textsc{k}_1$ and $\textsc{k}_2$ are the *negated versions* of one another if their activations are logically equivalent, that is ${}_{\square}\textsc{k}_1 \leftrightarrow {}_{\square}\textsc{k}_2$, and their targets are incompatible, that is, $\textsc{k}_{1\blacktriangleright} \wedge \textsc{k}_{2\blacktriangleright}$ is false. An example is that of Response vs NotResponse.

Consider now the situation where a decision must be taken concerning which of two candidate constraints $\textsc{k}_1$ and $\textsc{k}_2$ can be added to a Declare specification. Knowing that $\textsc{k}_1$ and $\textsc{k}_2$ are the negated versions of one another indicates that they should not *both* be added to the specification, as including them both would make the specification inconsistent as soon as the two constraints are activated.

As we will see in the next section, these notions become key when dealing with declarative process mining, and in particular the discovery of Declare specifications from event logs. Figure 8 graphically depicts how the main Declare constraint templates relate to each other in terms of subsumption and negated versions.


Cardinality templates
AtLeastOne(x)
AtMostOne(x)
Position templates
Init(x)
End(x)
Relation templates
Negative relation templates
Relation templates
Negative relation templates
negates
RespondedExistence(y, x)
NotRespondedExistence(y, x)
Precedence(x, y)
NotPrecedence(x, y)
AlternatePrecedence(x, y)
ChainPrecedence(x, y)
NotChainPrecedence(x, y)
negates
RespondedExistence(x, y)
NotRespondedExistence(x, y)
Response(x, y)
NotResponse(x, y)
AlternateResponse(x, y)
ChainResponse(x, y)
NotChainResponse(x, y)

(a) Existence templates

(b) Relation templates

Fig. 8. The subsumption map of Declare templates. Templates are indicated with solid boxes. The subsumption relation is depicted as a line starting from the subsumed template and ending in the subsuming one, with an empty triangular arrow recalling the UML IS-A graphical notation. The negative templates are graphically linked to the corresponding relation templates by means of wavy grey arcs.

5 Declarative Process Mining

Declarative process constraints depict the interplay of every activity in the process with the rest of the activities. As a consequence, the behavioural relationships that hold among activities can be analysed with a local focus on each one [9], as a projection of the whole process behaviour on a single element thereof. The constraints pertaining to a single activity thus be seen as its footprint in the global behaviour of the process. We shall interchangeably interpret Declare constraints as *(i)* behavioural relations between activities in a process specification or *(ii)* rules exerted on the occurrence of events in traces. Notice that the latter is a different approach than the former, typically used for process modelling as originally conceived by the seminal work of Pesic et al. [77]. The former is instead the basis for declarative process mining. In the following, we describe how process specifications can be discovered and monitored.

5.1 Declarative Process Discovery

Declarative process discovery refers to the inference of those constraints that significantly rule the behaviour of a process, based upon an input event log. The problem can be framed in two distinct ways:

- A *discriminative discovery problem*, reminiscent of a classification task. This requires to split the input event log in two partitions, one containing "positive" examples and the second containing "negative" examples. Discovery

Algorithm 1: Overview of the discovery algorithm

```
Input: $L \in \mathcal{B}(\mathcal{U}_{act}^{*})$, the event log to be analyzed;
       Rep, a finite set of Declare templates to be considered to express the discovered specification;
       $\mathrm{Act} \subseteq \mathcal{U}_{act}$, a finite set of activities to be included in the discovered specification;
       $\mathrm{conf}_{\mathrm{t}}^{\mathrm{min}}$, $\mathrm{supp}_{\mathrm{t}}^{\mathrm{min}}$, $\mathrm{conf}_{\mathrm{e}}^{\mathrm{min}}$, $\mathrm{supp}_{\mathrm{e}}^{\mathrm{min}}$, the minimum thresholds for trace-based confidence and
       support, and event-based confidence and support, respectively (default for all four parameters: 0.0);
Output: DS, a declarative process specification

$K \leftarrow \left\{ \kappa(a_1, \ldots, a_m) : \kappa \in \mathrm{Rep}, a_1; \ldots, a_m \in \mathrm{Act}, a_i \neq a_j \text{ with } 1 \leq i, j \leq m \right\}$
                 /* candidate constraints: templates assigned with any pair of distinct activities */
foreach $\kappa \in K$                                                        /* compute measures */
 do
    $\mathrm{c_t} \leftarrow \mathrm{conf_t}(\kappa, L)$; $\mathrm{s_e} \leftarrow \mathrm{supp_t}(\kappa, L)$; $\mathrm{c_e} \leftarrow \mathrm{conf_e}(\kappa, L)$; $\mathrm{s_e} \leftarrow \mathrm{supp_e}(\kappa, L)$
    if $c_t \leq \mathrm{conf}_{\mathrm{t}}^{min}$ or $s_t \leq \mathrm{supp}_{\mathrm{t}}^{min}$ or $c_e \leq \mathrm{conf}_{\mathrm{e}}^{min}$ or $s_e \leq \mathrm{supp}_{\mathrm{e}}^{min}$ then
        $K \leftarrow K \setminus \{\kappa\}$                  /* remove constraints with a measure below the threshold */

foreach $\kappa \in K$                  /* remove constraints as per subsumption hierarchy and negated v. */
 do
    foreach $\kappa' \in K$ s.t. $\kappa' \sqsubseteq \kappa$                         /* for every $\kappa'$ that subsumes $\kappa$ in $K$ */
     do
        if $\mathrm{allm}\left(\kappa', L\right) \leq \mathrm{allm}(\kappa, L)$                   /* if the measures of $\kappa'$ are $\leq$ those of $\kappa$ */
         then
            $K \leftarrow K \setminus \{\kappa\}$
        else $K \leftarrow K \setminus \{\kappa'\}$
    foreach $\kappa' \in \mathrm{DS}$ s.t. $\kappa'$ is the negated version of $\kappa$ do
        if $\mathrm{allm}\left(\kappa', L\right) < \mathrm{allm}(\kappa, L)$ then $K \leftarrow K \setminus \{\kappa\}$
        else $K \leftarrow K \setminus \{\kappa'\}$

return $\mathrm{DS} = (\mathrm{Rep}, \mathrm{Act}, K)$
```

amounts to find a suitable Declare specification that correctly reconstructs the classification, that is, accepts all positive examples and reject all negative ones.

- A *standard discovery problem* – also known as *specification mining* in the software engineering literature [53]. This calls for the individuation of which Declare constraints best describe the traces in the log, considering all of them as "positive" examples.

The first discovery algorithm for Declare treated discovery as a discriminative problem, exploiting inductive logic programming to tackle it [20,52]. In parallel, Goedertier et al. [46] brought forward techniques to generate negative examples from positive ones. Interestingly, this line of investigation recently received again the attention of the community [19,89].

Declarative process discovery framed as a standard discovery problem finds its two main exponents in *Declare Miner* [58] and *MINERful* [40], which have been then extended with an arsenal of techniques to improve the quality and correctness of the discovered specifications. We follow the second thread, summarizing the main ideas exploited therein, though reshaping the core concepts in an attempt to embrace the wider plethora of declarative process discovery techniques and the advancements they brought [8,18,59].

Process discovery in a declarative setting typically consists of the following phases:

1) The initial setup, i.e., the selection of *(i)* the templates to be sought for, *(ii)* the activities to be considered for the candidate constraints instantiating those templates, and *(iii)* the minimum thresholds for constraint interestingness measures to retain a candidate constraint;

2) The computation of interestingness measures for all the constraints that instantiate the given templates;
3) The simplification of the returned specification, through *(i)* the removal of constraints whose measures do not reach the user-specified thresholds, *(ii)* the pruning of the redundant constraints from the set, and *(iii)* the removal of one constraint for every pair of constraints that are the negated version of one another.

Algorithm 1 gives a bird-eye view of the approach in pseudocode. As we can observe, interestingness measures are crucial to determine the degree to which constraints are satisfied in the log. They have been introduced to indicate the level of reliability and relevance of constraints discovered from event logs, originally devised in the field of association rule mining [3] and adapted to the declarative process discovery context [17,65]. Among them, we recall support and confidence. Intuitively, support is a normalized measure quantifying how often the constraint is satisfied in the event log. Confidence considers the number of satisfactions with respect to the occurrences of the activations. We define them formally as follows.

Definition 14 (Trace-based measures). *Let L be a non-empty simplified event log with at least a non-empty trace, and* $\textsc{k}$ *a declarative constraint as per Definition 1. We define the trace-based support* $\mathrm{supp_t}$ *and the trace-based confidence* $\mathrm{conf_t}$ *as follows:*

$$\mathrm{supp_t}(\textsc{k}, L) = \frac{\sum\limits_{t \in L : t \,\models\, \Diamond({}_{\square}\textsc{k}) \wedge \textsc{k}} L(t)}{\sum\limits_{t \in L} L(t)}; \tag{1}$$

$$\mathrm{conf_t}(\textsc{k}, L) = \frac{\sum\limits_{t \in L : t \,\models\, \Diamond({}_{\square}\textsc{k}) \wedge \textsc{k}} L(t)}{\max\left\{1, \sum\limits_{t \in L : t \,\models\, \Diamond({}_{\square}\textsc{k})} L(t)\right\}}. \tag{2}$$

$\triangleleft$

We remark that the condition at the numerator that the trace has to satisfy not only the constraint $\textsc{k}$ but also eventually its activation, i.e., $t \models \Diamond({}_{\square}\textsc{k}) \wedge \textsc{k}$, serves the purpose of avoiding to count "vacuous satisfactions" discussed in Sect. 4.1. For example, while trace $\langle \mathsf{b}, \mathsf{c} \rangle$ satisfies ChainResponse(a, b), it does so vacuously, in the sense that it never activates the constraint. This intuitively means that ChainResponse(a, b), albeit satisfied, it cannot be interestingly used to describe the behaviour encoded in the trace. We recall that with $L(t)$ denotes the multiplicity of occurrences of t in the log L (see [1], Sect 3.1). The max term at the denominator of the formulation of confidence serves the purpose of avoiding a division by zero in case no trace satisfies $\Diamond({}_{\square}\textsc{k})$.

Declare Miner first introduced the trace-based measures to discover specifications from logs, counting traces that (non-vacuously) satisfy constraints as a whole. MINERful, instead, advocated also the adoption of measures that lie at the level of granularity of events. The similarities and differences between the two measuring schemes and the role of explicit activations and targets to tackle vacuity has been later systematized in [18]. The motivation behind the use of event-based measures is the ability to give a differently weight to traces violating the constraints in more than one instant: with trace-based measures, e.g., both traces $\langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{c}, \mathsf{a}, \mathsf{b}, \mathsf{a}, \mathsf{b}, \mathsf{a}, \mathsf{b}, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{a}, \mathsf{b}, \mathsf{a}, \mathsf{b}, \mathsf{a}, \mathsf{c} \rangle$ and $\langle \mathsf{b}, \mathsf{a}, \mathsf{c}, \mathsf{a}, \mathsf{c}, \mathsf{a}, \mathsf{a}, \mathsf{a}, \mathsf{a}, \mathsf{a}, \mathsf{a}, \mathsf{c} \rangle$ would count as single violations for $\textsc{ChainResponse}(\mathsf{a}, \mathsf{b})$. However, only the last occurrence of a out of ten leads to violation in the first trace, whereas all eight occurrences of a lead to violation in the second trace. Next, we formally capture the notion of event-based measures.

Definition 15 (Event-based measures). *Let L be a non-empty simplified event log with at least a non-empty trace, and $\textsc{k}$ a declarative constraint as per Definition 1. We define the event-based support* $\mathrm{supp}_{\mathrm{e}}$ *and the event-based confidence* $\mathrm{conf}_{\mathrm{e}}$ *as follows:*

$$\mathrm{supp}_{\mathrm{e}}(\textsc{k}, L) = \frac{\sum\limits_{t \in L} |\{a_i \in t : a, i \models ({}_{\square}\textsc{k} \wedge \textsc{k}_{\blacktriangleright})\}| \times L(t)}{\sum\limits_{t \in L} |t| \times L(t)}; \tag{3}$$

$$\mathrm{conf}_{\mathrm{e}}(\textsc{k}, L) = \frac{\sum\limits_{t \in L} |\{a_i \in t : a, i \models ({}_{\square}\textsc{k} \wedge \textsc{k}_{\blacktriangleright})\}| \times L(t)}{\max\left\{1, \sum\limits_{t \in L} |\{a_i \in t : a, i \models {}_{\square}\textsc{k}\}| \times L(t)\right\}}. \tag{4}$$

◁

Again, the condition at the numerator that events satisfy both activation and target of the constraint is intended to avoid including vacuous satisfactions in the sum. The max term at the denominator of confidence is intended to avoid a division by zero in case no event satisfies ${}_{\square}\textsc{k}$.

For the sake of readability, we shall denote with $\mathrm{allm}(\textsc{k}, L)$ the tuple containing all computed measures for a constraint $\textsc{k}$ on the event log L: $\mathrm{allm}(\textsc{k}, L) = (\mathrm{supp}_{\mathrm{t}}(\textsc{k}, L), \mathrm{conf}_{\mathrm{t}}(\textsc{k}, L), \mathrm{supp}_{\mathrm{e}}(\textsc{k}, L), \mathrm{conf}_{\mathrm{e}}(\textsc{k}, L))$. Given two constraints $\textsc{k}_1$ and $\textsc{k}_2$, we write $\mathrm{allm}(\textsc{k}_1, L) \leq \mathrm{allm}(\textsc{k}_2, L)$ if $\mathrm{supp}_{\mathrm{t}}(\textsc{k}_1, L) \leq \mathrm{supp}_{\mathrm{t}}(\textsc{k}_2, L)$, $\mathrm{conf}_{\mathrm{t}}(\textsc{k}_1, L) \leq \mathrm{conf}_{\mathrm{t}}(\textsc{k}_2, L)$, $\mathrm{supp}_{\mathrm{e}}(\textsc{k}_1, L) \leq \mathrm{conf}_{\mathrm{t}}(\textsc{k}_2, L)$, and $\mathrm{conf}_{\mathrm{e}}(\textsc{k}_1, L) \leq \mathrm{conf}_{\mathrm{t}}(\textsc{k}_2, L)$. We write $\mathrm{allm}(\textsc{k}_1, L) \leq \mathrm{allm}(\textsc{k}_2, L)$ if $\mathrm{allm}(\textsc{k}_1, L) \leq \mathrm{allm}(\textsc{k}_2, L)$ and $\mathrm{allm}(\textsc{k}_2, L) \leq \mathrm{allm}(\textsc{k}_1, L)$.

Example 10 (An event log for the specification in Example 1). Let $\mathcal{U}_{act} \doteq \{\mathsf{c}, \mathsf{r}, \mathsf{v}, \mathsf{t}, \mathsf{n}, \mathsf{y}, \mathsf{\$}, \mathsf{p}, \mathsf{e}, \mathsf{u}\} \cup \{\mathsf{@}\}$ be an alphabet of activities. We interpret $\mathsf{@}$ as an email exchange, which can occur at any stage during the process. The other activities in $\mathcal{U}_{act}$ are those that were considered in the process specification in Example 1. Let the following event log be built on $\mathcal{U}_{act}$:

Table 3. Measures computed for the relation constraints of Example 1 from the event log of Example 10.

Constraint	Event-based		Trace-based	
	Confidence	Support	Confidence	Support
Precedence(c, r)	1	0.129	1	1
AlternatePrecedence(r, v)	1	0.129	1	1
AlternateResponse(r, v)	0.997	0.129	0.996	0.996
Precedence(t, v)	0.997	0.129	0.996	0.996
AlternatePrecedence(v, n)	1	0.059	1	0.461
AlternatePrecedence(v, y)	1	0.084	1	0.856
NotResponse(y, n)	1	0.084	1	0.856
Precedence(y, p)	1	0.07	1	0.715
Precedence($, p)	1	0.07	1	0.715
ChainResponse($, p)	1	0.07	1	0.715
AtMostOne(p)	1	1	1	1
Precedence(p, e)	1	0.07	1	0.715
AtMostOne(e)	1	1	1	1
Precedence(u, e)	0.985	0.069	0.985	0.704

$L = [t_1^{200}, t_2^{100}, t_3^{100}, t_4^{80}, t_5^{80}, t_6^{4}, t_7^{2}, t_8^{2}]$ where

$$t_1 = \langle \mathsf{c, t, r, v, y, \$, p, u, e} \rangle \qquad t_2 = \langle \mathsf{c, t, t, r, v, n, t, r, v, y, \$, p, u, e} \rangle$$
$$t_3 = \langle \mathsf{c, t, r, t, v, y, u, \$, p, e} \rangle \qquad t_4 = \langle \mathsf{c, t, @, t, r, v, n, @, r, v, n} \rangle$$
$$t_5 = \langle \mathsf{c, r, t, t, v, n, y, @} \rangle \qquad t_6 = \langle \mathsf{c, t, r, t, v, @, @, y, \$, p, @, e} \rangle$$
$$t_7 = \langle \mathsf{c, @, r, v, y, \$, p, @, e} \rangle \qquad t_8 = \langle \mathsf{c, t, r, r, v, @, n} \rangle$$

We observe that the log above does not fully comply with the specification. Indeed, *(i)* trace t_8 violates AlternateResponse(r, v), as the candidate managed to register twice before evaluation (notice the occurrence of two consecutive r's before v); *(ii)* t_7 violates Precedence(t, v) and Precedence(u, e), as the candidate must have sent the admission test score and the necessary enrolment documents via email rather than via the system (see the occurrence of @ in place of t in the second instant and in place of u later in the trace); finally, *(iii)* trace t_6 violates Precedence(u, e), as the candidate must have submitted the enrolment documents via email in that case too (notice the absence of task u and the presence of @ in its stance). ◁

Example 11. With the example above, we have that both the trace-based support and trace-based confidence of Alt.Prec.(r, v), e.g., equate to 1.0: $\mathrm{supp}_{\mathsf{t}}(\text{Precedence}(\mathsf{c}, \mathsf{r}), L) = \mathrm{conf}_{\mathsf{t}}(\text{Precedence}(\mathsf{c}, \mathsf{r}), L) = 1.0$. This is because in all traces the activator (i.e., r) occurs and the constraint is not violated in any trace. Instead, $\mathrm{supp}_{\mathsf{t}}(\text{Alt.Prec.}(\mathsf{v}, \mathsf{n}), L) = \frac{100+80+80+2}{568} \approxeq 0.461$

and $\mathrm{conf}_{\mathrm{t}}(\textsc{Alt.Prec.}(\mathsf{v},\mathsf{n}), L) = 1.0$. The trace-based support is lower than the trace-based confidence because the activator (n) occurs in 262 traces out of 568 (i.e., in the 100 instances of t_2, the 80 instances of t_4, the 80 instances of t_5, and the 2 instances of t_8). Similarly, $\mathrm{conf}_{\mathrm{e}}(\textsc{Precedence}(\mathsf{c},\mathsf{r}), L) = 1.0$ and $\mathrm{conf}_{\mathrm{e}}(\textsc{Alt.Prec.}(\mathsf{v},\mathsf{n}), L) = 1.0$. The measures do not change for event-based and trace-based confidence because every activation of the two constraints above leads to a satisfaction. In contrast, $\mathrm{supp}_{\mathrm{e}}(\textsc{Precedence}(\mathsf{c},\mathsf{r}), L) = \frac{1\times200+2\times100+1\times100+2\times80+1\times80+1\times4+1\times2+2\times2}{9\times200+14\times100+10\times100+11\times80+8\times80+12\times4+9\times2+7\times2} = \frac{750}{5800} \approx 0.129$. ◁

It is worth noting that discovery approaches such as Declare Miner [58] and Janus [18] adopt (variations of) local constraint automata to count the satisfactions of constraints. MINERful [40] and DisCoveR [8] resort to occurrence statistics of activities gathered from the event log, more closely to the procedural discovery algorithms discussed in [2].

By definition of confidence and support (trace- or event-based), and as exemplified above, we observe that trace-based confidence is an upper bound for trace-based support and event-based confidence is an upper bound for event-based support. Next, we illustrate how the discovery algorithm operates with our running example.

Example 12. Table 3 shows the event-based and trace-based measures computed on the basis of our running example for every constraint in the original specification – phase (2) of the discovery procedure described above. They belong to the output of the discovery algorithm running on the event log of Example 10 set at phase (1) to seek for *(i)* all templates from the Declare repertoire in Table 2 *(ii)* over activities $\{\mathsf{c},\mathsf{r},\mathsf{v},\mathsf{t},\mathsf{n},\mathsf{y},\mathsf{\$},\mathsf{p},\mathsf{e},\mathsf{u}\}$, with *(iii)* minimum event-based confidence of 0.95. We remark that also AlternatePrecedence(y, p), ChainPrecedence($, p), AlternatePrecedence(p, e) and AlternatePrecedence(c, p), NotChainPrecedence(y, p) and NotChainResponse(y, p), among others, fulfil those criteria and thus are part of the returned set. ◁

To increase the information brought by a discovered model, not only we prune the constraints whose measures lie below the given threshold values. Also, we take into account the subsumption hierarchy illustrated in Fig. 8. In addition, we retain in the constraint set only one among pairs that are a negated version of one another. If we kept both, the model would turn the activation in common into a dead activity (see Sect. 4.2).

Example 13. Figure 9 illustrates the result of the pruning phase (3) based on subsumption and choice of constraints that are the negated version of one another, based on the event log of Example 10. We observe that AlternatePrecedence(y, p) has the same measures as Precedence(y, p), and we know that Precedence(y, p) is subsumed by AlternatePrecedence(y, p) (see Sect. 4.2); as we are interested in more restrictive constraints that reduce the space of possible process runs to more closely define its behaviour, we retain the former and discard the latter. Keeping both would introduce a redundancy,

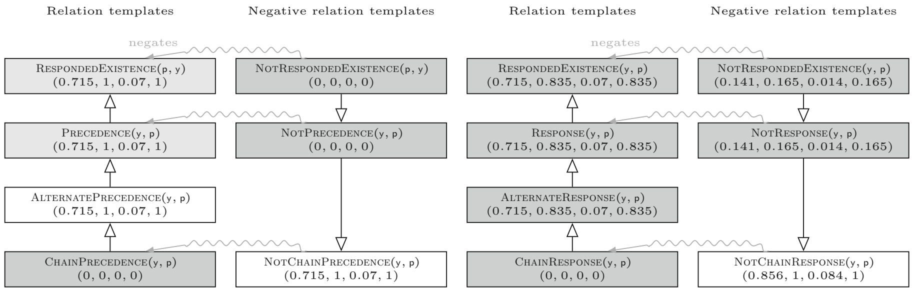


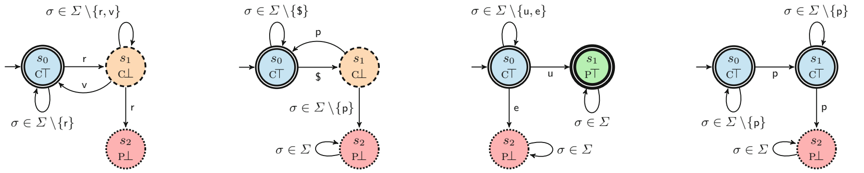
Fig. 9. The subsumption map of relation Declare constraints in a discovery context. The graphical notation follows Fig. 8. Gray boxes denote constraints that have measures below the minimum thresholds. Light-gray boxes indicate constraints that are subsumed by others with equivalent measures.

and retaining only the latter would omit detailed information as not only p must be preceded by y, but also p cannot recur unless y occurs again. By the same line of reasoning, we prefer retaining Init(c) to AtMostOne(c) in the result specification. The same concepts apply with ChainPrecedence($, p), to be preferred over Precedence($, p) and AlternatePrecedence(p, e) in place of Precedence(p, e), among others. Notice that Precedence(y, p), Precedence($, p) and Precedence(p, e) were in the given specification of our running example but, we conclude, are not the most restrictive constraints that could be used in the specification, as the discovery algorithm evidences. ◁

To conclude, we remark that not all redundancies can be found with the sole subsumption-hierarchy based pruning. The subsumption hierarchy, indeed, checks constraints that are exerted on the same activities – e.g., AlternatePrecedence(y, p) and Precedence(y, p). Therefore, we need a more powerful redundancy checking mechanism, seeking for constraints that are entailed by the remainder of the specification's constraint set (see Sect. 4.2).

Example 14. The confidence of AlternatePrecedence(v, p) is 1.0 in the event log of our running example. Yet, it does not add information to the discovered specification as it is redundant, logically entailed by the other constraints – in particular, AlternatePrecedence(r, v), AlternatePrecedence(v, y), Precedence(y, p) and AtMostOne(p). ◁

To verify this, we can resort to language inclusion via automata product as in [38]: the language of the product of the four constraint automata is not smaller than the language accepted by the intersection of the second, third and fourth constraint automata. Here, we do not enter the details of the algorithms that detect redundancies at such a deeper level but provide an example of its rationale. The interested reader can find further details in [24,38].



(a) ALT.RESP.(r, v) (b) CHN.RESP.($, p) (c) PREC.(u, e) (d) ATMOSTONE(p)

Fig. 10. Example FSAs adapted for the monitoring of constraints. Non-final states indicating current violation (c⊥) are dashed and filled in orange; non-final states indicating permanent violation (p⊥) are dotted and filled in red; final states indicating current satisfaction (c⊤) are thin-solid and filled in blue; final states indicating permanent satisfaction (p⊤) are thick-solid and filled in green. (Color figure online)

5.2 Declarative Process Monitoring

(Compliance) process monitoring aims at tracking running process executions to check their conformance to a reference process model, with the purpose of detecting and reporting deviations as soon as possible [57]. It constitutes one of the main tasks of operational decision support [92, Ch. 10], which characterizes process mining applied at runtime to running process executions.

Declarative process monitoring employs a declarative specification (in our case, described using DECLARE) as reference model for monitoring. The central fact in monitoring that process instances are running, that is, their generated traces evolve over time, calls for a finer-grained understanding of the state of constraints and of the whole specification. We illustrate this intuitively in the next example.

Example 15. Consider the excerpt in Fig. 11 of our admission process running example, and an evolving trace that, once completed, corresponds to the following sequence: $\langle \mathsf{\$}, \mathsf{p}, \mathsf{u}, \mathsf{\$}, \mathsf{p} \rangle$. Let us replay the trace from the beginning.

1. At the beginning, all constraints are *satisfied*, but they are so for sure only *currently*, as events may occur making them violated. For example, a registration without a consequent evaluation would lead to violating ALTERNATERESPONSE(r, v), whereas an enrolment without a prior upload of certificates would lead to a violation of PRECEDENCE(u, e).
2. Upon the occurrence of $, constraint CHAINRESPONSE($, p) becomes pending or, to be more precise, *currently violated*, as paying demands a pre-enrolment occurring immediately after.
3. The execution of p brings CHAINRESPONSE($, p) back to *currently satisfied*, as it does not require the occurrence of further events, but may do so in the future in case of another payment.
4. Upon the occurrence of u, constraint PRECEDENCE(u, e) becomes *permanently satisfied*, as enrolment is now enabled, and there is no way to continue the execution leading to a violation of the constraint.

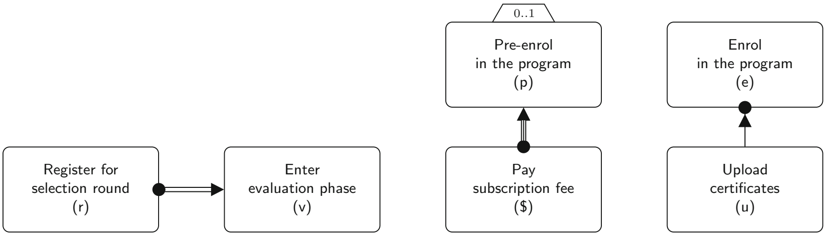

0..1
Pre-enrol in the program (p)
Enrol in the program (e)
Register for selection round (r)
Enter evaluation phase (v)
Pay subscription fee ($)
Upload certificates (u)

Fig. 11. Excerpt of the Declare specification in Fig. 2.

5. This is indeed what happens with the next occurrence of $, which makes ChainResponse($, p) *currently violated.*
6. The second pre-enrolment has the effect of bringing ChainResponse($, p) once again back to *currently satisfied.* However, it has also the effect of *permanently violating* AtMostOne(p), as the number of occurrences of p has exceeded the upper bound allowed by AtMostOne(p), and there is no way of fixing the violation.

◁

As witnessed by the example, the state of each constraint can be described in a fine-grained way by considering on the one hand the trace accumulated so far (i.e., the prefix of the whole, still unknown, execution), and by pondering on the other hand about the possible, future continuations. To do so in a formal way, we appeal to the literature on runtime-verification for linear temporal logics, and in particular to the RV-LTL semantics, originally introduced in [11] over infinite traces. This semantics was adopted for the first time in the context of LTL$_f$ over finite traces in [64,66], in order to define an operational technique for Declare monitoring. This led to deeper investigations on the usage of RV-LTLto characterize the relevance of a trace to a declarative specification [39], and to finally obtain a formally grounded, comprehensive framework for monitoring [27,28].

We now define the RV-LTL semantics for LTL$_f$. In the definition, we denote the concatenation of trace t_1 with t_2 as $t_1 \cdot t_2$.

Definition 16 (RV-LTL states). *Consider an* LTL$_f$ *formula* φ *over* Σ*, and a trace* t *over* Σ^**. We say that* φ is in (RV-LTL) state s after t, *written* $[t \models \varphi]_{\mathrm{RV}} = v$*, if:*

(Permanent satisfaction) *(i)* $v = \mathrm{P}\top$*, (ii) the current trace satisfies* φ $(t \models \varphi)$*, and (iii) every possible suffix keeps* φ *satisfied (for every trace* $t' \in \Sigma^*$*, we have* $t \cdot t' \models \varphi$*).*

(Permanent violation) *(i)* $v = \mathrm{P}\bot$*, (ii) the current trace violates* φ $(t \not\models \varphi)$*, and (iii) every possible suffix keeps* φ *violated (for every trace* $t' \in \Sigma^*$*, we have* $t \cdot t' \not\models \varphi$*).*

(Current satisfaction) *(i)* $v = \mathrm{C}\top$*, (ii) the current trace satisfies* φ $(t \models \varphi)$*, and (iii) there exists a suffix that leads to violate* φ *(for some trace* $t' \in \Sigma^*$*, we have* $t \cdot t' \not\models \varphi$*).*

(Current violation) *(i)* $v = \textsc{c}\bot$*, (ii) the current trace violates* φ *(*$t \not\models \varphi$*), and (iii) there exists a suffix that leads to satisfy* φ *(for some trace* $t' \in \Sigma^*$*, we have* $t \cdot t' \models \varphi$*).*

We also say that t conforms to φ *if* $[t \models \varphi]_{\text{RV}} = \textsc{p}\top$ *or* $[t \models \varphi]_{\text{RV}} = \textsc{c}\top$ *(i.e., stopping the execution in* t *satisfies the formula).* ◁

By inspecting the definition, we can directly see that monitoring is at least as hard as LTL$_f$ satisfiability/validity checking. To see this, consider what happens at the *beginning of an execution*, where the current trace is empty. By applying Definition 16 to this special case, and by recalling the notion of satisfiability/validity of an LTL$_f$ formula, we in fact get that an LTL$_f$ formula φ is:

- permanently satisfied if φ is valid;
- permanently violated if φ is unsatisfiable;
- currently satisfied if the two formulae $\varphi \wedge \textbf{end}$ and $\neg\varphi$ are both satisfiable;
- currently violated if the two formulae $\neg\varphi \wedge \textbf{end}$ and φ are both satisfiable.

To perform monitoring according to the RV-LTL states from Definition 16, we can once again exploit the automata-theoretic characterization of LTL$_f$. In particular, given an LTL$_f$ formula φ, we construct its FSA A_φ, and *color* the automaton states according to the RV-LTL semantics. As introduced in [64] and then formally verified in [28], this can be simply done as follows. Consider a state s in of A_φ. We label it by:

- $\textsc{p}\top$, if s is final and all the states reachable from s in A_φ are final as well; if A_φ is minimized, this means that s only reaches itself.
- $\textsc{p}\bot$, if s is non-final and all the states reachable from s in A_φ are non-final as well; if A_φ is minimized, this means that s only reaches itself.
- $\textsc{c}\top$, if s is final and can reach a non-final state in A_φ.
- $\textsc{c}\bot$, if s is non-final and can reach a final state in A_φ.

Figure 10 shows some examples of colored constraint automata, obtained by considering the constraint formulae of some Declare constraints from our running example. To monitor the state evolution of a constraint, one has simply to dynamically play the evolving trace on its colored local automaton, returning the updated RV-LTL label as soon as a new event is processed. Doing so on the local automata in Fig. 10 for trace $\langle \mathsf{\$}, \mathsf{p}, \mathsf{u}, \mathsf{\$}, \mathsf{p} \rangle$ formally reconstructs what discussed in Example 15.

However, this is not enough to *promptly detect violations* as soon as they manifest in the traces. This has been extensively discussed in [28,66], and is at the very core of the power of temporal logic-based techniques for monitoring. We use again Example 15 to illustrate the problem.

Example 16. Consider Example 15 and the following question: is step 6 the earliest at which a violation can be detected? Clearly, if we focus on each constraint in isolation, the answer is affirmative. To see this formally, we play trace $\langle \mathsf{\$}, \mathsf{p}, \mathsf{u}, \mathsf{\$}, \mathsf{p} \rangle$ on the four colored local automata of Fig. 10, obtaining the following runs:

- For AlternateResponse(r, v), we have $s_0 \xrightarrow{\mathsf{\$}} s_0 \xrightarrow{\mathsf{p}} s_0 \xrightarrow{\mathsf{u}} s_0 \xrightarrow{\mathsf{\$}} s_0 \xrightarrow{\mathsf{p}} s_0$; no violation is encountered.
- For ChainResponse($\mathsf{\$}, \mathsf{p}$), we have $s_0 \xrightarrow{\mathsf{\$}} s_1 \xrightarrow{\mathsf{p}} s_0 \xrightarrow{\mathsf{u}} s_0 \xrightarrow{\mathsf{\$}} s_1 \xrightarrow{\mathsf{p}} s_0$; no violation is encountered.
- For Precedence(u, e), we have $s_0 \xrightarrow{\mathsf{\$}} s_0 \xrightarrow{\mathsf{p}} s_0 \xrightarrow{\mathsf{u}} s_1 \xrightarrow{\mathsf{\$}} s_1 \xrightarrow{\mathsf{p}} s_1$; no violation is encountered.
- For AtMostOne(p), we have $s_0 \xrightarrow{\mathsf{\$}} s_0 \xrightarrow{\mathsf{p}} s_1 \xrightarrow{\mathsf{u}} s_1 \xrightarrow{\mathsf{\$}} s_1 \xrightarrow{\mathsf{p}} s_2$; a violation is encountered in the last reached state.

The answer changes if we consider the whole Declare specification that contains all such constraints at once. In fact, by taking into account the interplay of constraints, we can detect a violation already at step 5, i.e., after the second occurrence of payment. This is because, after that step, the two constraints ChainResponse($\mathsf{\$}, \mathsf{p}$) and AtMostOne($\mathsf{p}$) enter into a *conflict*, that is, no continuation of the current trace can lead to satisfy them both. In fact, after trace $\langle \mathsf{\$}, \mathsf{p}, \mathsf{u}, \mathsf{\$} \rangle$, constraint ChainResponse($\mathsf{\$}, \mathsf{p}$) is currently violated, waiting for a consequent occurrence of p; however, constraint AtMostOne(p), which is currently satisfied, becomes permanently violated upon a further occurrence of p. ◁

As we have seen, the early detection of violations cannot always be caught by considering the colored local automata of constraints in isolation. However, it can be systematically detected by taking into account the colored global automaton of the whole specification.

Example 17. Figure 12 shows the colored global automaton of the Declare specification in Fig. 11. By playing the trace $\langle \mathsf{\$}, \mathsf{p}, \mathsf{u}, \mathsf{\$}, \mathsf{p} \rangle$ therein, we obtain the following run: $s_0 \xrightarrow{\mathsf{\$}} s_1 \xrightarrow{\mathsf{p}} s_4 \xrightarrow{\mathsf{u}} s_8 \xrightarrow{\mathsf{\$}} s_{12} \xrightarrow{\mathsf{p}} s_{12}$. Clearly, the violation state s_{12} is already reached in step 5, i.e., just after the second payment. ◁

All in all, we can then monitor an evolving trace against a Declare specification as follows:

- Each constraint is encoded into the corresponding colored local automaton, used to track the state evolution of the constraint itself.
- The whole specification is encoded into the corresponding colored global automaon, used to track the evolution of the whole specification, and in particular to early-detect violations.
- At runtime, every new event occurrence is delivered in parallel to all the automata, updating each of them by executing the corresponding transition and entering into the next state, at the same time returning the associated RV-LTL label.

Figure 13 shows the result of applying this technique to our running example.

An alternative approach, which is exploited in [64], is to compute, as done before, the global automaton as the cross-product of local automata, remembering, in each global state, the RV-LTL labels of all local states from which such

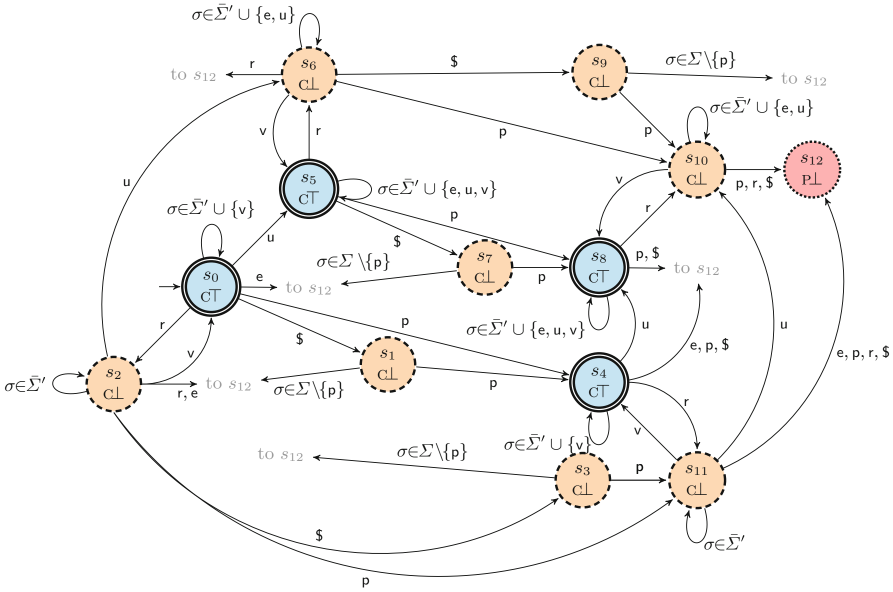

$\sigma \in \bar{\Sigma}' \cup \{e, u\}$
s_6 C⊥
r
to s_{12}
$
s_9 C⊥
$\sigma \in \Sigma \setminus \{p\}$
to s_{12}
$\sigma \in \bar{\Sigma}' \cup \{e, u\}$
v
r
p
p
s_{10} C⊥
s_{12} P⊥
p, r, $
u
s_5 C⊤
$\sigma \in \bar{\Sigma}' \cup \{e, u, v\}$
v
$\sigma \in \bar{\Sigma}' \cup \{v\}$
p
r
u
$
s_7 C⊥
$\sigma \in \Sigma \setminus \{p\}$
s_8 C⊤
p, $
to s_{12}
s_0 C⊤
e
to s_{12}
p
r
p
$\sigma \in \bar{\Sigma}' \cup \{e, u, v\}$
u
u
e, p, $
e, p, r, $
$
s_1 C⊥
v
$\sigma \in \bar{\Sigma}'$
s_2 C⊥
r, e
to s_{12}
$\sigma \in \Sigma \setminus \{p\}$
p
s_4 C⊤
r
v
$\sigma \in \bar{\Sigma}' \cup \{v\}$
to s_{12}
$\sigma \in \Sigma \setminus \{p\}$
s_3 C⊥
p
s_{11} C⊥
$
$\sigma \in \bar{\Sigma}'$
p

Fig. 12. The colored global automaton automaton obtained as the (colored) cross-product of constraints in Fig. 10 as shown in Fig. 6(c), the states of which are decorated with the four RV-LTL truth values.

a global state has been produced. In addition, no minimization step is applied on the resulting automaton. Once colored, this non-minimized, global colored automaton combines in a single device the contribution of all local monitors and that of the global monitor.

5.3 A Note on Conformance Checking

In this section, we have focused on monitoring evolving traces against Declare specifications. This can be seen as a form of *online conformance checking*, aiming at detecting deviations at execution time. This technique can be seamlessly lifted to handle the standard conformance checking task, where conformance is evaluated on an event log containing full traces of already completed process executions (cf. [16]). In this setting, the global automaton is not needed anymore, as a-posteriori it is not relevant to compute the earliest moment of a violation, but only to properly detect it at the trace level. The usage of local automata, one per constraint, is enough, and also has the advantage of producing an informative feedback that indicates, trace by trace, how many (and which) constraints are satisfied or violated. Finer-grained feedbacks like those based on the computation of trace alignments have been extensively applied for procedural models (cf. [16]), and can be also recasted in the declarative setting, aligning the log traces with the (closest) model traces accepted by the global automaton

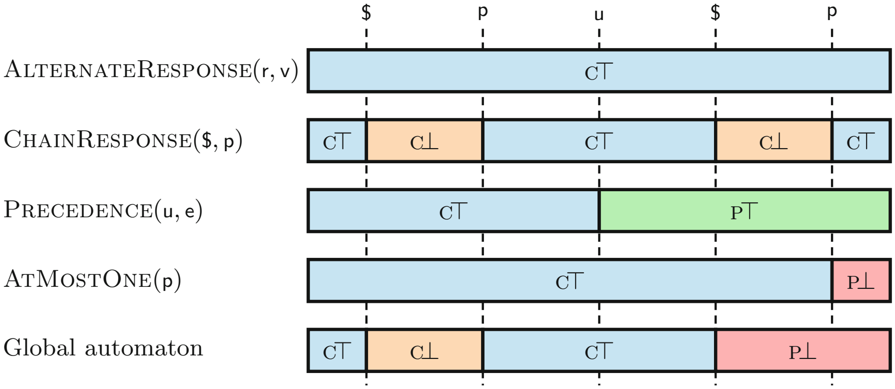


Fig. 13. Monitoring with local and global colored automata, showing a case where the global automaton detects a violation before it actually manifests on a single constraint.

of the Declare specification of interest. This is an active line of research, which started from the seminal approach in [31].

6 Recent Advances and Outlook

We close this chapter by reporting about the most recent advances in the field of declarative process mining revolving around Declare, describing the current frontier of research, and highlighting open challenges.

6.1 Beyond Declare Patterns

As we have seen in Sect. 3, a Declare specification consists of a repertoire of constraint templates grounded on specific activities. At the same time, such templates come with a logic-based semantics given in terms of LTL$_f$. A natural question is then: can the techniques described in this chapter be used for the *entire* LTL$_f$ logic? This means, more precisely, considering the situation where each constraint corresponds to an arbitrary LTL$_f$ formula while, as usual, the specification formula is constructed by putting in conjunction the LTL$_f$ formulae of all its constituting constraints.

To answer this question, one has to separate the *logical* and *pragmatic* aspects involved in the different tasks we have been introducing. We do so focusing on reasoning, discovery, and monitoring.

Reasoning. As discussed in Sect. 4.2, all the reasoning tasks we have considered in this chapter can be lifted to the whole LTL$_f$ logic. Indeed, they are reduced to LTL$_f$ satisfiability/validity checking, which in turn can be tackled by checking (non-)emptiness of FSAs. The situation may change if one wants to provide more advanced debugging or diagnosis functionalities – for example, to return the most relevant conflicting set(s) of constraints that are causing inconsistencies or dead activities. While these types of problem can also be attacked at the level of the entire logic [25,79], focusing only on pre-defined patterns becomes necessary if one wants to involve humans in the loop or define preferences over constraints in the case where multiple explanations exist [25]. Considering specific patterns is also relevant when studying the computational complexity of reasoning on pattern combinations [44,45,91], or the scalability and effectiveness of reasoning tools [44,45,71,97].

Discovery. As pointed out in Sect. 5.1, two distinct process discovery problems are typically tackled in a declarative setting: discriminative discovery and specification mining.

The case of discriminative discovery is tightly related to classification and machine learning, allowing one to rely on general learning algorithms for declarative process mining. Such algorithms tackle general logical frameworks, such as Horn clauses in inductive logic programming or full temporal logics in model learning, and can thus go far beyond a pre-defined set of templates, either targeting full LTL_f [15,82] or enriching the discoverable Declare templates with further key dimensions, such as metric temporal constraints, event attributes, and data conditions [21,23].

As shown in Sect. 5.1, standard discovery stands as a radically different problem, since the input event log provides a uniform set of (positive) examples, while no negative example is given. This calls for suitable metrics to measure *how well* a set of constraints characterizes the behaviour contained in the log. In the approach described in this chapter, such metrics are defined starting from the notions of constraint activation and target, which are template-specific. Attempts have been conducted to lift some of these notions (in particular that of activation and "relevant" satisfaction [39]) to full LTL_f, but further research is needed to target the discovery of arbitrary LTL_f formulae from event logs. Notice that while full LTL_f discovery would enrich the expressiveness of the discovered specifications, it would on the other hand pose the issue of *understandability*: end users may struggle when confronted with arbitrary temporal formulae, while they are facilitated when pre-defined templates are used.

Monitoring. As we have discussed in Sect. 5.2, Declare monitoring is tackled using automata, and consequently seamlessly work for arbitrary LTL_f formulae. As for advanced debugging techniques, the same considerations done for reasoning also hold for monitoring. For example, the detection of minimal conflicting sets of constraints in the case of early detection of violations caused by the interplay of multiple constraints can be tamed at the level of the full logic [66], but would require to focus on patterns if one wants to formulate preferences or incorporate human feedback [25].

Remarkably, working with FSAs allows us to define monitors for temporal formulae that go even beyond LTL_f. In fact, LTL_f is as expressive as star-free regular expressions, while automata are able to capture full regular expressions and, in turn, finite-trace temporal logics incorporating in a single formalism

LTL_f and regular expressions, such as Linear Dynamic Logic over finite traces (LDL_f) [30]. Working with LDL_f in our setting has the specific advantage that we can express and monitor *metaconstraints*, that is, constraints that predicate on the RV-LTL truth values of other constraints [27,28].

6.2 Dealing with Uncertainty

In the conventional definition of a Declare specification, constraints are interpreted as being *certain*: every model trace is expected to satisfy all constraints contained in the specification. Such an interpretation is too restrictive in scenarios where the specification should accommodate:

- constraints describing common behaviours, expected to hold in the majority, but not all cases;
- constraints describing exceptional, outlier behaviours that rarely occurs but should be not judged as violating the specification.

To deal with this form of *uncertainty*, Declare has been recently extended with *probabilistic constraints* [62]. In this framework, every probabilistic constraint comes with:

- a constraint formula φ (specified, as in the standard case, using LTL_f);
- a comparison operator $\odot \in \{=, \neq, <, \leq, >, \geq\}$;
- a number $p \in [0, 1]$.

The interpretation of this constraint is that φ holds in a random trace generated by the process with a probability that is $\odot p$. In frequentist terms, this can be in turn interpreted as follows: given a log of the process, the ratio of traces satisfying φ must be $\odot p$.

Since a Declare specification contains multiple constraints, one has to consider how different probabilistic constraints interact with each other. In particular, n probabilistic constraints yield up to 2^n possible so-called *scenarios*, each highlighting which probabilistic constraints hold and which are violated. Reasoning over such scenarios has to be conducted by suitably mixing their temporal and probabilistic dimensions. The former handles which combinations of constraints and their violations (i.e., which scenarios) are consistent, while the latter lifts the probability conditions attached of single constraints to discrete probability distributions over the possible scenarios.

To carry out this form of combined reasoning, probabilistic constraints are formalized in a well-behaved fragment of the logic introduced in [61]. As it turns out, logical and probabilistic reasoning are loosely coupled in this fragment, and can be carried out resorting to standard finite-state automata and systems of linear inequalities. This approach has been used as the basis for defining a new family of *probabilistic declarative process mining* techniques [6].

6.3 Mixed-Paradigm Models

In Fig. 1, we have intuitively contrasted declarative specifications and imperative models. The distinction of these two approaches is in reality not so crisp. In fact, a single process may contain parts that are more suitably captured using imperative languages, and parts that can be better described as declarative specifications. Take, for instance, a clinical guideline mixing administrative and therapeutic subprocesses [73].

To capture such *hybrid* processes, one needs a multi-paradigm approach that can combine imperative and declarative constructs in a single process model. One of the first proposals doing so is [85], where an imperative process can contain activities that are internally structured using so-called *pockets of flexibility* specified using declarative temporal constraints over a given set of tasks.

This layered approach has been further developed in [90], which brings forward a hierarchical model where each sub-process can be specified either as an imperative or declarative component. Discovery of hierarchical hybrid process models has been subsequently tackled in [87].

Multi-paradigm approaches providing a tighter integration between imperative and declarative components have also been studied. In [33], process models combining Petri nets and Declare constraints at the same modelling level are introduced and studied, singling out methodologies and techniques to handle the intertwined state space emerging from their interaction. Conformance checking for these mixed-paradigm models is extensively assessed in [95]. A different approach is brought forward in [5], where a Declare specification is used to express global constraints that “glue together” multiple imperative processes concurrently executed over the same instances. Automata-based techniques extending those illustrated in Sect. 5.2 are introduced to provide integrated monitoring functionalities dealing at once with the local processes and the global constraints.

At the current stage, further research is needed along the illustrated lines towards a solid theory and corresponding algorithmic techniques for *hybrid, mixed-paradigm process mining.*

6.4 Multi-perspective Declare Specifications

Throughout the chapter, we have considered pure control-flow specifications, where a process is captured solely in terms of its constitutive activities and of behavioural constraints separating legal from undesired executions. While the control-flow provides the main process backbone, other equally important perspectives should also be taken into account as suggested already in [1]:

- The *resource* perspective deals with the actors that are responsible for executing tasks within the process.
- The *time* perspective focusses on quantitative temporal conditions on when tasks can/must be scheduled and executed, and on their expected durations.
- The *data* perspective captures how data objects and their attributes influence and are manipulated during the process execution.

Several works have investigated the extension of Declare with additional perspectives. From the formal point of view, this requires to extend the logic-based formalization of Declare with features that can capture resources, metric time, data, and conditions thereof, in turn resorting to variants of metric and/or first-order formalisms over finite traces [10,14,69,74]. It is important to stress that such features may be blurred, considering that data support (if equipped with suitable datatypes and conditions) may be used to predicate over resources and time as well.

Such multi-perspective features have been extensively embedded into Declare or related approaches (see, for example, [13,69,98] for constraints with metric time and [42] for constraints with metric time and resources). Next, we focus in more detail on the data dimension.

When it comes to data, two main lines of research can be identified. The first one deals with standard "case-centric" processes extended with event and case data. The second one focuses instead on "multi-case" processes, wherein constraints are expressed over multiple objects and their mutual relations. We briefly discuss each line separately.

Declarative Process Specifications with Event/Case Data. Within a process, activities may be equipped with data attributes that, at execution time, are grounded to actual data values by the involved resources. This means that events witnessing the occurrence of task instances come with a data payload. In addition, each process instance may evolve its own case data in response to the execution of activities.[3] Such case data may be stored in different ways, e.g., as key-value pairs or a full-fledged relational database. In this setting, it becomes crucial to extend Declare with so-called *data-aware constraints*, that is, constraints enriched with data-aware conditions over activities. The simple but illustrative example described next motivates why this is needed.

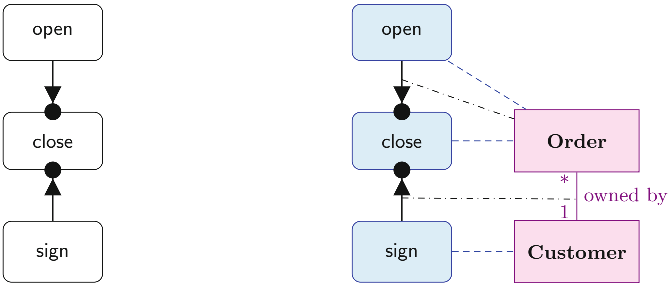
Example 18. We focus on a process where payments are issued by customers through a pay activity, which comes with an attribute indicating the paid amount, in Euros. Two consequent activities check and emit are executed to respectively inspect a payment and emit a receipt.

Let a log for this process contain multiple repetitions of the following traces:

$$\begin{array}{ll}
t_1 = \langle \mathsf{pay}(\mathsf{amount}{=}50), \mathsf{emit}\rangle & \qquad t_2 = \langle \mathsf{pay}(\mathsf{amount}{=}300), \mathsf{check}, \mathsf{emit}\rangle \\
t_3 = \langle \mathsf{pay}(\mathsf{amount}{=}20)\rangle & \qquad t_4 = \langle \mathsf{pay}(\mathsf{amount}{=}100), \mathsf{emit}, \mathsf{check}\rangle \\
t_5 = \langle \mathsf{pay}(\mathsf{amount}{=}90), \mathsf{emit}\rangle & \qquad t_6 = \langle \mathsf{pay}(\mathsf{amount}{=}800), \mathsf{check}\rangle
\end{array}$$

One may wonder whether Response($\mathsf{pay}, \mathsf{check}$) is a suitable constraint to explain (part of) the behaviour contained in the log. If considered unrestrictedly, this

[3] For conciseness of presentation, we will not distinguish between event and case data in our discussion, but technically they pose different, albeit tightly related, requirements.



(a) Conventional Declare specification (b) Object-centric Declare specification

Fig. 14. Comparison of conventional vs object-centric Declare.

is not the case, as there are many traces where payment is not followed by any inspection. The situation changes completely if one restricts the scope of the constraint activation only to those payments that involve an amount of 100 or more. ◁

A number of works has brought forward combined techniques to discover Declare constraints equipped with various forms of data conditions [54,60,86], to check conformance for data-aware constraints [12,13], and to handle their monitoring [5,69]. This passage has to be carried out with extreme care, as combining event data and time quickly leads to undecidability of reasoning [14,34,35]. Therefore, such techniques have to operate in a limited fashion or suitably controlling the expressiveness of data conditions and the way they interact with time.

Object-Centric Declarative Process Specifications. So far, we have discussed the extension of Declare with event or case data. In a more general setting, data may refer to more complex networks of objects and their mutual relations, simultaneously co-evolved by one or multiple processes. In this type of processes, known under the umbrella term of *object-centric processes*, there is no single, pre-defined notion of case, and process executions cannot consequently be represented as flat traces, but call for richer representations (cf. [43]). The following example illustrates why Declare, in its conventional version, cannot be used to capture *object-centric* processes.

Example 19. Consider the fragment of an order-to-cash process, containing three activities: sign (indicating the signature of a GDPR form by the customer), open (the opening of an order), and close (the closing of an order). Two constraints apply to close, defining under which conditions it becomes executable:

- An order can be closed only if *that order* has been opened before.
- An order can be closed only if *its owner* has signed the consent before.

Figure 14(a) shows how these two constraints can be captured in conventional Declare. This specification is satisfactory only in the case where each trace refers to a single customer and a single order by that customer. For example, consider the following two traces, respectively referring to an order o_1 by Anne, and an order o_2 by Bob:

$$t_1 = \langle \mathsf{sign}, \mathsf{open}, \mathsf{close} \rangle \qquad\qquad t_2 = \langle \mathsf{open}, \mathsf{close}, \mathsf{sign} \rangle$$

Clearly, t_1 is a model trace, while t_2 is not, as the latter violates Precedence(sign, close).

However, one may need to consider multiple orders owned by the same or distinct customers, in the common situation where distinct orders may be later bundled together to handle their shipment. In our example, assuming that o_1 and o_2 are later bundled together in a shipment, this would require to combine t_1 and t_2 in a single object-centric trace, suitably extending each event with a reference to the object(s) it operates on. Suppose this would result into:

$$t = \left\langle \begin{array}{l} \mathsf{sign(customer{=}Anne)}, \mathsf{open(order{=}o2)}, \mathsf{open(order{=}o1)}, \\ \mathsf{close(order{=}o1)}, \mathsf{close(order{=}o2)}, \mathsf{sign(customer{=}Bob)} \end{array} \right\rangle$$

The Declare specification of Fig. 14(a) becomes now inadequate. In fact, it cannot distinguish which events actually *co-refer* to one another and which do not, so it cannot identify that the first signature by Anne refers to the first occurrence of close, but not to the second one. Hence, it wrongly uses the first occurrence of sign to satisfy Precedence(sign, close) for both orders. ◁

Fixing the issue described in Example 19 requires the explicitly extension of Declare with the ability of expressing how events relate to objects, how objects relate to each other, and in turn to *scope* the application of constraints, expressing that they must be enforced over events that suitably co-refer to each other – either because they operate on the same object, or because they operate on related objects. In our running example, this would call for the following actions:

- introduce the classes of **Order** and **Customer**;
- capture that there is a many-to-one **owned by** association linking orders to customers;
- indicate that sign refers to a customer, and that open and close refer to an order;
- scope Precedence(open, close) by enforcing that the two involved activities must *co-refer to the same order* (i.e., that an event of activity close for order o can only occur if an event of activity open has previously occurred *for the same order*);
- scope Precedence(sign, close) by enforcing that the two involved activities must respectively operate with a customer and an order that *co-refer through the* ***owned by*** *association* (i.e., that an event of activity close for order o can only occur if an event of activity sign has previously occurred *for the customer who owns* o).

Object-centric behavioral constraints (OCBC) [93] have been brought forward to handle this type of scoping through the integration of Declare specifications and UML class diagrams. Figure 14(b) shows the OCBC specification correctly capturing the constraints of Example 19. The approach is still at its infancy: some first seminal works have been conducted to handle discovery of OCBC specifications from object-centric event logs recording full database transactions [55], and to formalize and reason upon OCBC specifications through temporal description logics [7]. Further research is being carried out to improve the performance of discovery and frame it in the context of object-centric event logs of the form of [1], and to tackle conformance checking and monitoring. This is particularly challenging, as integrating temporal constraints with data models quickly leads to undecidability [7].

7 Conclusion

Throughout this chapter, we have thoroughly reviewed the declarative approach to process specification and mining. The declarative approach aims at limiting the process behavior by defining the boundaries within which its executions can unfold, yet leaving process executors free to explore at runtime which specific executions are generated. This is in contrast with the imperative approach, where process models compactly depict all and only those traces that are admissible. In fact, notice that different (imperative) process models can comply with the same declarative specification, just like different dynamic systems can model ($\models$) a set of temporal rules. In the chapter, we have grounded our discussion on the Declare language, but the introduced concepts are broad enough to be seamlessly applicable to other related approaches.

Specifically, we have first discussed how declarative process specifications can be formalized using Linear Temporal Logic on Finite Traces (LTL$_f$), and in turn operationally characterized in terms finite state automata (FSAs) for their execution semantics. On this solid formal ground, we have examined the core reasoning tasks that relate to declarative specifications and then delved deeper into the discovery and monitoring of processes according to the declarative paradigm. Interestingly, we have observed that the reasoning tasks are pervasive in all stages of declarative process mining, such as within discovery to avoid producing redundant or inconsistent outputs, and within monitoring to speculatively consider the possible future continuations of the monitored execution. In the last part of the chapter, we have provided a summary of the most recent advances in declarative process mining, focusing in particular on: *(i)* the applicability of declarative process mining techniques and concepts to full temporal logics, going beyond predefined patterns; *(ii)* the incorporation of uncertainty within constraints; *(iii)* the analysis of hybrid models integrating imperative and declarative fragments; *(iv)* multi-perspective constraints incorporating additional dimensions beyond the control-flow, and supporting the declarative specification of object-centric (multi-case) processes. This bird-eye view provides a fair account of the open research challenges in declarative process mining.

Acknowledgments. The authors want to thank Fabrizio Maria Maggi, Wil van der Aalst, Alessio Cecconi, Federico Chesani, Giuseppe De Giacomo, Riccardo De Masellis, Johannes De Smedt, Massimo Mecella, Paola Mello, Jan Mendling, Maja Pesic, Johannes Prescher for the long-standing cooperation and years of joint work that led to this chapter. The work of the authors has received funding by the Italian Ministry of University and Research under the PRIN programme, grant B87G22000450001 (PINPOINT). The work of C. Di Ciccio was partly funded by the Italian Ministry of University and Research under grant "Dipartimenti di eccellenza 2018–2022" of the Department of Computer Science at the Sapienza University of Rome and the Sapienza research project SPECTRA. The work of M. Montali was partly funded by the UNIBZ projects WineID, SMART-APP, QUEST, and VERBA.

References

1. van der Aalst, W.M.P.: Process mining: a 360 degrees overview. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
2. van der Aalst, W.M.P.: Foundations of process discovery. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
3. Adamo, J.-M.: Data Mining for Association Rules and Sequential Patterns - Sequential and Parallel Algorithms. Springer, New York (2001). https://doi.org/10.1007/978-1-4613-0085-4
4. Alman, A., Di Ciccio, C., Maggi, F.M., Montali, M., van der Aa, H.: RuM: declarative process mining, distilled. In: Polyvyanyy, A., Wynn, M.T., Van Looy, A., Reichert, M. (eds.) BPM 2021. LNCS, vol. 12875, pp. 23–29. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85469-0_3
5. Alman, A., Maggi, F.M., Montali, M., Patrizi, F., Rivkin, A.: Multi-model monitoring framework for hybrid process specifications. In: Franch, X., Poels, G. (eds.) Proceedings of the 34th International Conference on Advanced Information Systems Engineering (CAiSE 2022). Lecture Notes in Computer Science (2022, to appear)
6. Alman, A., Maggi, F.M., Montali, M., Peñaloza, R.: Probabilistic declarative process mining. Inf. Syst. (2012, to appear)
7. Artale, A., Kovtunova, A., Montali, M., van der Aalst, W.M.P.: Modeling and reasoning over declarative data-aware processes with object-centric behavioral constraints. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNCS, vol. 11675, pp. 139–156. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26619-6_11
8. Back, C.O., Slaats, T., Hildebrandt, T.T., Marquard, M.: Discover: Accurate & efficient discovery of declarative process models. CoRR, abs/2005.10085 (2020)
9. Baier, T., Di Ciccio, C., Mendling, J., Weske, M.: Matching events and activities by integrating behavioral aspects and label analysis. Softw. Syst. Model. **17**(2), 573–598 (2018)
10. Basin, D.A., Klaedtke, F., Müller, S., Zalinescu, E.: Monitoring metric first-order temporal properties. J. ACM **62**(2), 15:1–15:45 (2015)
11. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. **20**(4), 14:1–14:64 (2011)

12. Bergami, G., Maggi, F.M., Marrella, A., Montali, M.: Aligning data-aware declarative process models and event logs. In: Polyvyanyy, A., Wynn, M.T., Van Looy, A., Reichert, M. (eds.) BPM 2021. LNCS, vol. 12875, pp. 235–251. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85469-0_16
13. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. **65**, 194–211 (2016)
14. Calvanese, D., De Giacomo, G., Montali, M., Patrizi, F.: Verification and monitoring for first-order LTL with persistence-preserving quantification over finite and infinite traces. In: De Raedt, L. (ed.) Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI 2022). ijcai.org (2022, to appear)
15. Camacho, A., McIlraith, S.A.: Learning interpretable models expressed in linear temporal logic. In: Benton, J., Lipovetzky, N., Onaindia, E., Smith, D.E., Srivastava, S. (eds.) Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2018), pp. 621–630. AAAI Press (2019)
16. Carmona, J., van Dongen, B., Weidlich, M.: Conformance checking: foundations, milestones and challenges. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
17. Cecconi, A., De Giacomo, G., Di Ciccio, C., Mendling, J.: A temporal logic-based measurement framework for process mining. In: van Dongen et al. [92]
18. Cecconi, A., Di Ciccio, C., De Giacomo, G., Mendling, J.: Interestingness of traces in declarative process mining: the janus LTLp$_f$ approach. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 121–138. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_8
19. Chesani, F., et al.: Process discovery on deviant traces and other stranger things. CoRR, abs/2109.14883 (2021)
20. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. In: Jensen, K., van der Aalst, W.M.P. (eds.) Transactions on Petri Nets and Other Models of Concurrency II. LNCS, vol. 5460, pp. 278–295. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00899-3_16
21. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. Trans. Petri Nets Other Model. Concurr. **2**, 278–295 (2009)
22. Chomsky, N., Miller, G.A.: Finite state languages. Inf. Control **1**(2), 91–112 (1958)
23. Corea, C., Deisen, M., Delfmann, P.: Resolving inconsistencies in declarative process models based on culpability measurement. In: Ludwig, T., Pipek, V. (eds.) WI, pp. 139–153. University of Siegen, Germany/AISeL (2019)
24. Corea, C., Delfmann, P.: Quasi-inconsistency in declarative process models. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNBIP, vol. 360, pp. 20–35. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26643-1_2
25. Corea, C., Nagel, S., Mendling, J., Delfmann, P.: Interactive and minimal repair of declarative process models. In: Polyvyanyy, A., Wynn, M.T., Van Looy, A., Reichert, M. (eds.) BPM 2021. LNBIP, vol. 427, pp. 3–19. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85440-9_1
26. Davulcu, H., Kifer, M., Ramakrishnan, C.R., Ramakrishnan, I.V.: Logic based modeling and analysis of workflows. In: PODS, pp. 25–33. ACM (1998)
27. De Giacomo, G., De Masellis, R., Grasso, M., Maggi, F.M., Montali, M.: Monitoring business metaconstraints based on LTL and LDL for finite traces. In: Sadiq, S.,

Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 1–17. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_1

28. De Giacomo, G., De Masellis, R., Maggi, F.M., Montali, M.: Monitoring constraints and metaconstraints with temporal logics on finite traces. ACM Trans. Softw. Eng. Methodol. (2022, to appear)
29. De Giacomo, G., De Masellis, R., Montali, M.: Reasoning on LTL on finite traces: insensitivity to infiniteness. In: Brodley, C.E., Stone, P. (eds.) AAAI, pp. 1027–1033. AAAI Press (2014)
30. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Rossi, F. (ed.) IJCAI, pp. 854–860. IJCAI/AAAI (2013)
31. De Leoni, M., Maggi, F.M., van der Aalst, W.M.: An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. Inf. Syst. **47**, 258–277 (2015)
32. De Smedt, J., De Weerdt, J., Serral, E., Vanthienen, J.: Discovering hidden dependencies in constraint-based declarative process models for improving understandability. Inf. Syst. **74**(Part 1), 40–52 (2018)
33. De Smedt, J., De Weerdt, J., Vanthienen, J., Poels, G.: Mixed-paradigm process modeling with intertwined state spaces. Bus. Inf. Syst. Eng. **58**(1), 19–29 (2016)
34. Demri, S., Lazic, R.: LTL with the freeze quantifier and register automata. ACM Trans. Comput. Log. **10**(3), 16:1–16:30 (2009)
35. Demri, S., Lazic, R., Nowak, D.: On the freeze quantifier in constraint LTL: decidability and complexity. Inf. Comput. **205**(1), 2–24 (2007)
36. Di Ciccio, C.: On the mining of artful processes. Ph.D. thesis, SAPIENZA, University of Rome, October 2013
37. Di Ciccio, C., Bernardi, M.L., Cimitile, M., Maggi, F.M.: Generating event logs through the simulation of declare models. In: Barjis, J., Pergl, R., Babkin, E. (eds.) EOMAS 2015. LNBIP, vol. 231, pp. 20–36. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24626-0_2
38. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. Inf. Syst. **64**, 425–446 (2017)
39. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: On the relevance of a business constraint to an event log. Inf. Syst. **78**, 144–161 (2018)
40. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Trans. Manag. Inf. Syst. **5**(4), 24:1–24:37 (2015)
41. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Boehm, B.W., Garlan, D., Kramer, J. (eds.) ICSE, pp. 411–420. ACM (1999)
42. Elgammal, A., Turetken, O., van den Heuvel, W.-J., Papazoglou, M.: Formalizing and appling compliance patterns for business process compliance. Softw. Syst. Model. **15**(1), 119–146 (2014). https://doi.org/10.1007/s10270-014-0395-3
43. Fahland, D.: Process mining over multiple behavioral dimensions with event knowledge graphs. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
44. Fionda, V., Greco, V.: LTL on finite and process traces: complexity results and a practical reasoner. J. Artif. Intell. Res. **63**, 557–623 (2018)
45. Fionda, V., Guzzo, A.: Control-flow modeling with declare: behavioral properties, computational complexity, and tools. IEEE Trans. Knowl. Data Eng. **32**(5), 98–911 (2020)
46. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. J. Mach. Learn. Res. **10**, 1305–1340 (2009)

47. Green, T.R.G., Petre, M.: Usability analysis of visual programming environments: a ‘cognitive dimensions’ framework. Vis. Comp. and Lang. **7**(2), 131–74 (1996)
48. Haisjackl, C., et al.: Understanding declare models: strategies, pitfalls, empirical results. Softw. Syst. Model. **15**(2), 325–352 (2016)
49. Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: PLACES. EPTCS, vol. 69, pp. 59–73 (2010)
50. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley Longman Publishing Co. Inc., Boston (2006)
51. Kupferman, O., Vardi, M.Y.: Vacuity detection in temporal model checking. Int. J. Softw. Tools Technol. Transfer **4**(2), 224–233 (2003)
52. Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Inducing declarative logic-based models from labeled traces. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 344–359. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_25
53. Lemieux, C., Park, D., Beschastnikh, I.: General LTL specification mining (T). In: Cohen, M.B., Grunske, L., Whalen, M. (eds.) 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, 9–13 November 2015, pp. 81–92. IEEE Computer Society (2015)
54. Leno, V., Dumas, M., Maggi, F.M., La Rosa, M., Polyvyanyy, A.: Automated discovery of declarative process models with correlated data conditions. Inf. Syst. **89**, 101482 (2020)
55. Li, G., de Carvalho, R.M., van der Aalst, W.M.P.: Automatic discovery of object-centric behavioral constraint models. In: Abramowicz, W. (ed.) BIS 2017. LNBIP, vol. 288, pp. 43–58. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59336-4_4
56. Lichtenstein, O., Pnueli, A., Zuck, L.: The glory of the past. In: Parikh, R. (ed.) Logic of Programs 1985. LNCS, vol. 193, pp. 196–218. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-15648-8_16
57. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.P.: Compliance monitoring in business processes: functionalities, application, and tool-support. Inf. Syst. **54**, 209–234 (2015)
58. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 270–285. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31095-9_18
59. Maggi, F.M., Di Ciccio, C., Di Francescomarino, C., Kala, T.: Parallel algorithms for the automated discovery of declarative process models. Inf. Syst. **74**, 136–152 (2018)
60. Maggi, F.M., Dumas, M., García-Bañuelos, L., Montali, M.: Discovering data-aware declarative process models from event logs. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 81–96. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40176-3_8
61. Maggi, F.M., Montali, M., Peñaloza, R.: Temporal logics over finite traces with uncertainty. In: Proceedings of the 34 AAAI Conference on Artificial Intelligence (AAAI 2020), pp. 10218–10225. AAAI Press (2020)
62. Maggi, F.M., Montali, M., Peñaloza, R., Alman, A.: Extending temporal business constraints with uncertainty. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNCS, vol. 12168, pp. 35–54. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58666-9_3

63. Maggi, F.M., Montali, M., van der Aalst, W.M.P.: An operational decision support framework for monitoring business constraints. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 146–162. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28872-2_11
64. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: an approach based on colored automata. In: Rinderle-Ma et al. [81], pp. 132–147
65. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: CIDM, pp. 192–199. IEEE (2011)
66. Maggi, F.M., Westergaard, M., Montali, M., van der Aalst, W.M.P.: Runtime verification of LTL-based declarative process models. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 131–146. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_11
67. Montali, M.: Specification and verification of declarative open interaction models - a logic-based framework. Ph.D. thesis, University of Bologna, Italy (2009)
68. Montali, M.: Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach. Lecture Notes in Business Information Processing, vol. 56. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14538-4
69. Montali, M., Maggi, F.M., Chesani, F., Mello, P., van der Aalst, W.M.P.: Monitoring business constraints with the event calculus. ACM TIST **5**(1), 17:1–17:30 (2013)
70. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographies. TWEB **4**(1), 1–62 (2010)
71. Montali, M., et al.: Verification from declarative specifications using logic programming. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 440–454. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89982-2_39
72. Mulyar, N., Pesic, M., van der Aalst, W.M.P., Peleg, M.: Declarative and procedural approaches for modelling clinical guidelines: addressing flexibility issues. In: ter Hofstede, A., Benatallah, B., Paik, H.-Y. (eds.) BPM 2007. LNCS, vol. 4928, pp. 335–346. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78238-4_35
73. Munoz-Gama, J., Martin, N., et al.: Process mining for healthcare: characteristics and challenges. J. Biomed. Inform. **127**, 103994 (2022)
74. Ouaknine, J., Worrell, J.: On the decidability and complexity of metric temporal logic over finite words. Log. Methods Comput. Sci. **3**(1) (2007)
75. Pesic, M.: Constraint-based workflow management systems: shifting control to users. Ph.D. thesis, Technische Universiteit Eindhoven (2008)
76. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: EDOC, pp. 287–300 (2007)
77. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: EDOC, pp. 287–300. IEEE Computer Society (2007)
78. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Eder, J., Dustdar, S. (eds.) BPM 2006. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006). https://doi.org/10.1007/11837862_18
79. Pill, I., Quaritsch, T.: Behavioral diagnosis of LTL specifications at operator level. In: Rossi, F. (ed.) Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013), pp. 1053–1059. IJCAI/AAAI (2013)

80. Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57. IEEE (1977)
81. Rabin, M.O., Scott, D.S.: Finite automata and their decision problems. IBM J. Res. Dev. **3**(2), 114–125 (1959)
82. Raha, R., Roy, R., Fijalkow, N., Neider, D.: Scalable anytime algorithms for learning fragments of linear temporal logic. In: Fisman, D., Rosu, G. (eds.) TACAS 2022. LNCS, vol. 13243, pp. 263–280. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_14
83. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30409-5
84. Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.): Business Process Management. LNCS, vol. 6896. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23059-2
85. Sadiq, S., Sadiq, W., Orlowska, M.: Pockets of flexibility in workflow specification. In: S.Kunii, H., Jajodia, S., Sølvberg, A. (eds.) ER 2001. LNCS, vol. 2224, pp. 513–526. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45581-7_38
86. Schönig, S., Di Ciccio, C., Maggi, F.M., Mendling, J.: Discovery of multi-perspective declarative process models. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) ICSOC 2016. LNCS, vol. 9936, pp. 87–103. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46295-0_6
87. Schunselaar, D.M.M., Slaats, T., Maggi, F.M., Reijers, H.A., van der Aalst, W.M.P.: Mining hybrid business process models: a quest for better precision. In: Abramowicz, W., Paschke, A. (eds.) BIS 2018. LNBIP, vol. 320, pp. 190–205. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93931-5_14
88. Singh, M.P.: Distributed enactment of multiagent workflows: temporal logic for web service composition. In: AAMAS, pp. 907–914. ACM (2003)
89. Slaats, T., Debois, S., Back, C.O.: Weighing the pros and cons: process discovery with negative examples. In: Polyvyanyy, A., Wynn, M.T., Van Looy, A., Reichert, M. (eds.) BPM 2021. LNCS, vol. 12875, pp. 47–64. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85469-0_6
90. Slaats, T., Schunselaar, D.M.M., Maggi, F.M., Reijers, H.A.: The semantics of hybrid process models. In: Debruyne, C., et al. (eds.) OTM 2016. LNCS, vol. 10033, pp. 531–551. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48472-3_32
91. Sun, Y., Su, J.: Conformance for DecSerFlow constraints. In: Franch, X., Ghose, A.K., Lewis, G.A., Bhiri, S. (eds.) ICSOC 2014. LNCS, vol. 8831, pp. 139–153. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45391-9_10
92. van der Aalst, W.M.P.: Process Mining - Data Science in Action, 2nd edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
93. van der Aalst, W.M.P., Artale, A., Montali, M., Tritini, S.: Object-centric behavioral constraints: integrating data and declarative process modelling. In: Artale, A., Glimm, B., Kontchakov, R. (eds.) DL. CEUR Workshop Proceedings, vol. 1879. CEUR-WS.org (2017)
94. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: towards a truly declarative service flow language. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 1–23. Springer, Heidelberg (2006). https://doi.org/10.1007/11841197_1
95. van Dongen, B.F., De Smedt, J., Di Ciccio, C., Mendling, J.: Conformance checking of mixed-paradigm process models. Inf. Syst. **102**, 101685 (2021)
96. van Dongen, B.F., Montali, M., Wynn, M.T. (eds.) 2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, 4–9 October 2020. IEEE (2020)

97. Westergaard, M.: Better algorithms for analyzing and enacting declarative workflow languages using LTL. In: Rinderle-Ma et al. [81], pp. 83–98

98. Westergaard, M., Maggi, F.M.: Looking into the future. In: Meersman, R., et al. (eds.) OTM 2012. LNCS, vol. 7565, pp. 250–267. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33606-5_16

99. Zhu, S., Tabajara, L.M., Pu, G., Vardi, M.Y.: On the power of automata minimization in temporal synthesis. In: Proceedings 12th International Symposium on Games, Automata, Logics, and Formal Verification (GandALF 2021). EPTCS, vol. 346, pp. 117–134 (2021)

Conformance Checking

Conformance Checking: Foundations, Milestones and Challenges

Josep Carmona[1], Boudewijn van Dongen[2(✉)], and Matthias Weidlich[3]

[1] Universitat Politècnica de Catalunya, Barcelona, Spain
[2] Eindhoven University of Technology, Eindhoven, The Netherlands
B.F.v.Dongen@tue.nl
[3] Humboldt-Universität zu Berlin, Berlin, Germany

Abstract. By relating observed and modelled behaviour, conformance checking unleashes the full power of process mining. Techniques from this discipline enable the analysis of the quality of a process model discovered from event data, the identification of potential deviations, and the projection of real traces onto process models. This way, the insights gained from the available event data can be transferred to a richer conceptual level, amenable for a human interpretation. The aforementioned functionalities are grounded on the use of conformance checking artefacts that explicit the relation between observed and modelled behaviour. This chapter describes these artefacts, and builds upon them to gain evidence-based insights on the processes of an organization. Moreover, we overview the applications of conformance checking and propose a general framework that incorporates these applications. Finally, milestones and challenges of the field are outlined.

1 Introduction

Organisations tend to define, by means of conceptual models, complex business processes that must be followed to achieve their objectives [22]. Sometimes the corresponding processes are distributed in different systems, and most of the cases include human tasks, enabling the occurrence of unexpected deviations with respect to the (normative) process model. This is aggravated by the appearance of more and more complex processes, where the observations are provided by heterogeneous sources, such as Internet-of-Things (IoT) devices involved in Cyber-physical Systems [46].

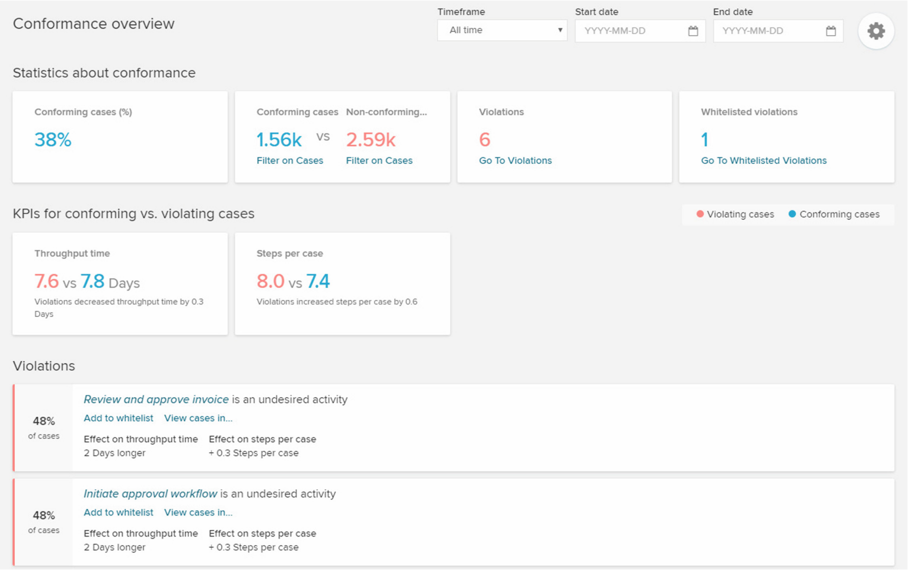
Conformance checking techniques provide mechanisms to relate modelled and observed behaviour, so the frictions between the footprints left by process executions, and the process models that formalise the expected behaviour, can be revealed [14]. As it has been already commented in the first chapters of this book, process executions are often materialized and stored by means of event logs. Table 1 shows an example of an event log for a loan application process.

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 155–190, 2022.
https://doi.org/10.1007/978-3-031-08848-3_5

Conformance checking is expected to be the fastest growing segment in process mining for the next years[1]. The main reason for this forthcoming industrial interest is the promise of having event data and process models aligned, thus increasing the value of process models within organizations.

Given an event log and a process model, conformance checking techniques yield some explicit description of their consistent and deviating parts, here referred to as a *conformance artefact*. In the first part of this chapter, we focus on three main conformance artefacts that are covering most of the spectrum of conformance checking:

- Behavioural rules such as ordering constraints for activities imposed by the model that are violated by some traces of the event log;
- Events of traces that could correctly be replayed by task executions in the process model, or for which the replay failed;
- An alignment between the events of a trace of the event log and the task executions of an execution sequence of the model.



Conformance overview

Timeframe: All time ▾ | Start date: YYYY-MM-DD | End date: YYYY-MM-DD

Statistics about conformance

Conforming cases (%)
38%

Conforming cases 1.56k Filter on Cases vs Non-conforming... 2.59k Filter on Cases

Violations
6
Go To Violations

Whitelisted violations
1
Go To Whitelisted Violations

KPIs for conforming vs. violating cases

● Violating cases ● Conforming cases

Throughput time
7.6 vs 7.8 Days
Violations decreased throughput time by 0.3 Days

Steps per case
8.0 vs 7.4
Violations increased steps per case by 0.6

Violations

48% of cases
Review and approve invoice is an undesired activity
Add to whitelist View cases in...
Effect on throughput time: 2 Days longer
Effect on steps per case: + 0.3 Steps per case

48% of cases
Initiate approval workflow is an undesired activity
Add to whitelist View cases in...
Effect on throughput time: 2 Days longer
Effect on steps per case: + 0.3 Steps per case

Fig. 1. Example of conformance checking in Celonis.

Remarkably, a conformance artefact enables conclusions on the relation between the event log and the process model. By interpreting the conformance artefact, for instance, the fitness and precision of the model regarding the given log is quantified. Such an interpretation may further involve decisions on how to weight and how to attribute any encountered deviation (see the end of this chapter

[1] https://www.marketsandmarkets.com/Market-Reports/process-analytics-market-254139591.html.

for a discussion on this topic). Since the log and the model are solely representations of the process, both of them may differ in how they abstract the process.

Differences in the representations of a process may, of course, be due to inaccuracies. For example, an event log may be recorded by an erroneous logging mechanism (see next chapter of this handbook for understanding this in depth), whereas a process model may be outdated. Yet, differences may also be due to different purposes and constraints that guide how the process is abstracted and therefore originate from the pragmatics of the respective representation of the process. Think of a logging mechanism that does not track the execution of a specific activity due to privacy considerations, or a model that outlines only the main flow of the process to clarify its high-level phases. Either way, the respective representations are not *wrong*, but differ because of their purpose and the constraints under which they have been derived.

By linking an event log and a process model through a conformance artefact, the understanding of the underlying process can be improved. That includes techniques for process enhancement (see [18]). For instance, traces of an event log can be replayed in the process model, while taking into account the deviations between the log and model as materialised in the conformance artefact. Commercial tools that include conformance checking nicely display these deviations on their dashboards, as can be seen in Fig. 1. Another example includes the inspection of the conditions that govern the decision points in a process. The conformance artefact can be used to derive a classification problem per decision point, which enables discovery of the respective branching conditions. Assuming that the model represents the desired behaviour of the process, the conformance artefact further enables conclusions on how the current realisation of the process needs to be adapted.

There exist different algorithmic perspectives to relate modelled and observed behaviour: rule checking, token-replay and alignments.

A process model defines a set of tasks along with causal dependencies for their execution. As such, a process model constrains the possible behaviour of a process in terms of its execution sequences. Instead of considering the set of possible execution sequences of a process model, however, the basic idea of rule-based conformance checking is to exploit rules that are satisfied by all these sequences as the basis for analysis. Such rules define a set of constraints that are imposed by the process model. Verification of these constraints with respect to the traces of an event log, therefore, enables the identification of conformance issues.

Unlike rule checking that is grounded in information derived from the process model, token replay takes the event log as the starting point for conformance analysis. As indicated already by its name, this technique replays each trace of the event log in the process model by executing tasks according to the order of the respective events. By observing the states of the process model during the replay, it can be determined whether, and to what extent, the trace indeed corresponds to a valid execution sequence of the model.

In spite of the two aforementioned class of techniques to relate modelled and observed behaviour, most conformance checking techniques rely on the notion of alignment [1]: given an observed trace σ, query the model to obtain the execution sequence γ that is most similar to σ. The computation of alignments is a computational challenge, since it encompasses the exploration of the model state space, an object that is worst-case exponential with respect to the size of the model or the trace.

Table 1. Example of a log of the loan application process.

Event	Application	Offer	Activity	Amount	Signed	Timestamp
...	...	...	...	...	...	...
e_{13}	$A5634$		Application submitted	€2,000		Jan 01, 12:31
e_{14}	$A5634$		Accept application	€2,000		Jan 01, 12:32
e_{15}	$A5635$		Application submitted	€5,000		Jan 02, 04:31
e_{16}	$A5635$		Accept application	€5,000		Jan 02, 04:32
e_{17}	$A5636$		Application submitted	€200		Jan 03, 06:59
e_{18}	$A5636$		Accept application	€200		Jan 03, 07:00
...	...	...	...	...	...	...
e_{22}	$A5634$		Finalise application			Jan 03, 09:00
e_{23}	$A5636$		Finalise application			Jan 03, 09:01
e_{24}	$A5635$		Decline application			Jan 03, 09:02
e_{25}	$A5635$		Decline application			Jan 03, 09:03
...	...	...	...	...	...	...
e_{30}	$A5636$	$O3521$	Select and send offer	€500		Jan 04, 16:32
...	...	...	...	...	...	...
e_{37}	$A5634$	$O3541$	Select and send offer	€1,500		Jan 05, 12:32
e_{38}	$A5636$	$O3521$	Receive offer		NO	Jan 05, 12:33
e_{38}	$A5636$	$O3521$	Cancel offer			Jan 05, 12:34
e_{39}	$A5636$	$O3542$	Select and send offer	€500		Jan 05, 13:29
e_{40}	$A5636$	$O3542$	Receive offer		YES	Jan 08, 08:33
e_{41}	$A5636$	$O3542$	Accept offer			Jan 08, 16:34
e_{42}	$A5634$	$O3541$	Receive offer		NO	Jan 10, 10:00
...	...	...	...	...	...	...
e_{54}	$A5634$	$O3541$	Decline offer			Jan 10, 10:04
...	...	...	...	...	...	...
e_{64}	$A5634$		Decline application			Jan 10, 10:05
e_{65}	$A5634$		Application finished			Jan 10, 10:06
e_{66}	$A5636$		Approve and activate application			Jan 10, 10:07
e_{67}	$A5636$		Application finished			Jan 10, 10:08
...	...	...	...	...	...	...

Once conformance artefacts are computed, the next natural step is to use them. The main applications arising from these artefacts are listed in this chapter as well, as a gentle introduction to some of the chapters devoted to this in this book. We highlight performance analysis and decision point analysis as natural examples of the application of conformance checking.

Furthermore, depending on the trust we put on the two main elements (trust on the log, trust on the model), conformance checking can be generalized as a framework that unifies diverse analysis techniques in the field of process mining [48]. As such, this framework includes several instantiations already known to the reader.

We finish the chapter by listing important milestones and challenges, some of them being already under consideration by the research community, like the computational feasibility of the underlying techniques.

2 Relating Observed and Modelled Behaviour: The Basics

In this section, we discuss the basic notions and techniques to relate observed and modelled behaviour. To this end, we first review generic quality dimensions on this relation (Sect. 2.1). Subsequently, we turn to three different types of conformance checking artefacts that capture the relation between a trace observed in the event log and a process model, namely artefacts grounded in rule checking (Sect. 2.2), token replay (Sect. 2.3), and alignments (Sect. 2.4), see also Fig. 2. A detailed explanation of the contents of this section can be found in [14].

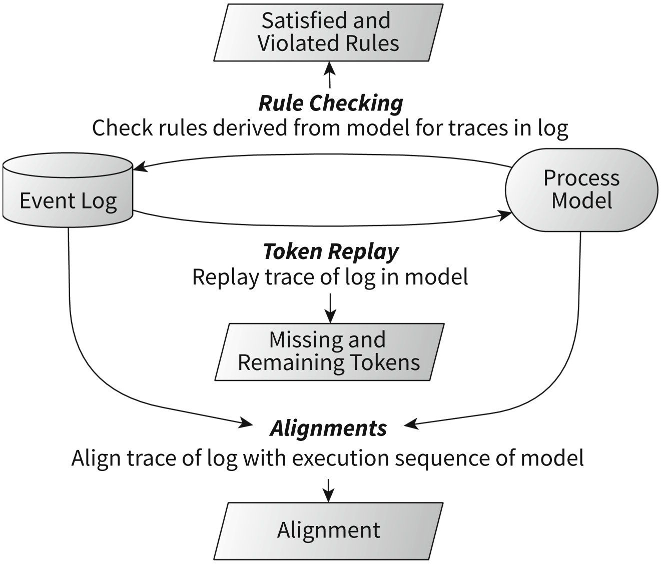

Satisfied and Violated Rules
Rule Checking
Check rules derived from model for traces in log
Event Log
Process Model
Token Replay
Replay trace of log in model
Missing and Remaining Tokens
Alignments
Align trace of log with execution sequence of model
Alignment

Fig. 2. General approaches to conformance checking and resulting conformance artefacts (from [14]): rule checking, token replay, and alignments. All techniques take a trace of an event log and a process model as input. However, conceptually, rule checking starts from the behaviour of the process model, extracting constraints to check for a trace. Token replay, in turn, starts from the behaviour of a single trace, trying to replay the trace in a process model. Alignments, in turn, adopt an inherently symmetric view.

2.1 Quality Dimensions to Relate Process Models and Event Logs

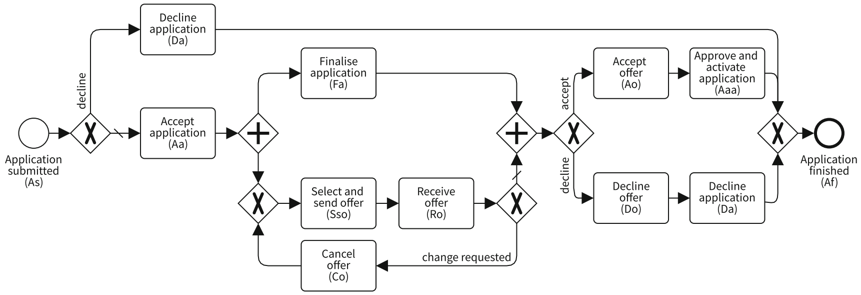


Fig. 3. Example process model of a loan application process in BPMN.

By relating observed and modelled behaviour, an organization can get insights on the execution of their processes with respect to the expectations as described in the models. If both process model M and event log L are considered as languages, their relation can be used to measure how good is a process model in describing the behaviour recorded in an event log.

Hence, confronting M and L can help into understanding the complicate relation between modelled and recorded behaviour. We now provide two views on this relation that represent two alternative perspectives: *fitness* and *precision*. To illustrate this, in this chapter we will be using a process for a loan application. A process model illustrating this process is described in Fig. 3. According to this model, a submitted application is either accepted or rejected, depending on the applicant's data. An accepted application is finalised by a worker, in parallel with the offer process. For each application, an offer is selected and sent to the customer. The customer reviews the offer and sends it back. If the offer is accepted, the process continues with the approval of the application and the activation of the loan. If the customer declines the offer, the application is also declined and the process ends. However, the customer can also request a new offer, in which case the offer is cancelled and a new offer is sent to the customer.

Fitness measures the ability of a model to explain the recorded execution of a process as recorded in an event log (see the example of Fig. 4 for an example of fitting behaviour). It is the main measure to assess whether a model is well-suited to explain the recorded behaviour. To explain a certain trace, the process model is queried to assess its ability in replaying the trace, taking into account the control flow logic expressed in the model.

In general, fitness is the fraction of the behaviour of the log that is also allowed by the model. It can be expressed as follows.

$$fitness = \frac{|L \cap M|}{|L|} \tag{1}$$

Let us have a look at this fraction in more detail by examining the extreme cases. Fitness is 1, if the entire behaviour that we see in the log L is covered by the model M. Conversely, fitness is 0, if no behaviour in the log L is captured by the model M. In the remainder of this section, we will describe three different algorithms deriving artefacts that can be used to evaluate fitness.

We define a trace to be either *fitting* (it corresponds to an execution sequence of the model) or *non-fitting* (there is some deviation with respect to all execution sequences of the model). For instance, the trace corresponding case $A5634$ in our running example is fitting, since there is an execution sequence of the model that perfectly reproduces this case, as shown in Fig. 4. In contrast, Fig. 5 shows the information for a trace that does not contain the event to signal that the application has been finalised (Fa).

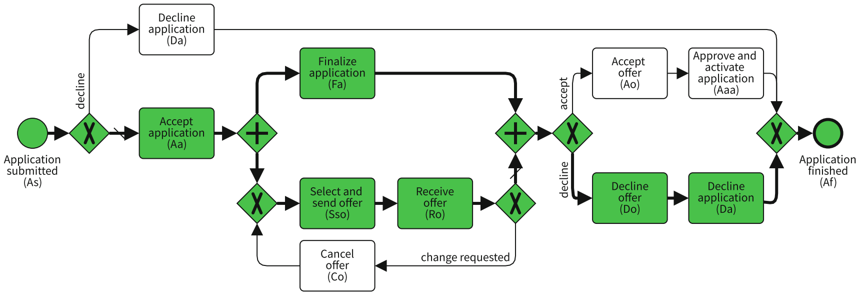


Fig. 4. Loan application process model with highlighted path corresponding to the fitting trace $\langle As, Aa, Fa, Sso, Ro, Do, Da, Af\rangle$ of case $A5634$ from the event log of Table 1.

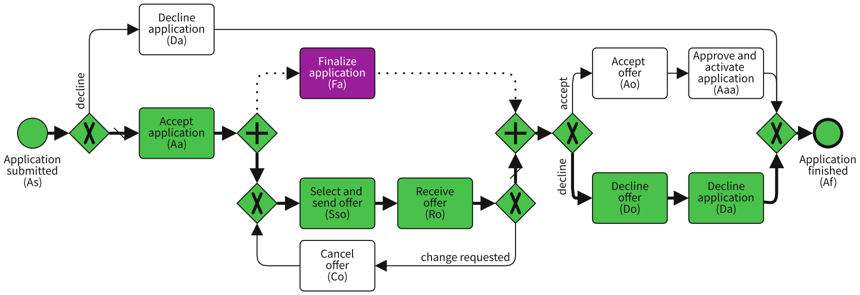


Fig. 5. Loan application process model with highlighted path corresponding to a trace $\langle As, Aa, Sso, Ro, Do, Da, Af\rangle$, which does not include an event to signal that the application has been finalised (Fa). In magenta, we show that the task (Fa) has not been observed, but it is required to reach the final state of the process model.

Precision is the counterpart of fitness. It can be calculated by looking at the fraction of the model behaviour that is covered in the log.

$$precision = \frac{|L \cap M|}{|M|} \tag{2}$$

We see that precision shares the numerator in the fraction with fitness from (1). This implies that if we have a log and a model with no shared behaviour, fitness is zero, and by definition also precision is zero. However, the denominator is replaced with the amount of modelled behaviour.

In summary, for the two main metrics reported above, algorithms that can assess the relation between log and model need to be considered. In the next section, we describe the three main algorithmic perspectives to accomplish this task. For an extensive analysis of metrics to assess the relation between observed and modelled behaviour, including metrics like *generalization* or *simplicity*, the reader is referred to [14]. Intuitively, generalization complements precision by quantifying the amount of behaviour that is modelled in a process model, but not observed in an event log. In practice, an event log cannot be expected to be complete, i.e., to contain all possible process behaviour (e.g., all possible interleavings of concurrent activities or all possible numbers of iterations of repetitive behaviour). Hence, a process model is typically assumed to generalize to some extent, i.e., not to show perfect precision, and generalization measure aim to quantify this amount of imprecision. Simplicity, in turn, refers to the structure and complexity of the model. Intuitively, simplicity measures induce some preference for process models that behave similarly in terms of the other dimensions, with the argument being that simple models are generally to be preferred.

2.2 Rule Checking

The basic idea of rule-based conformance checking is to exploit rules that are satisfied by all the execution sequences of a process model as the basis for analysis. Such rules define a set of constraints that are imposed by the process model. The verification of these constraints with respect to the traces of an event log, therefore, enables the identification of conformance issues.

Considering the running example of our loan application process as depicted in Fig. 3, rules derived from the process model include:

R1: An application can be accepted (Aa) at most once.
R2: An accepted application (Aa), that must have been submitted (As) earlier, and eventually an offer needs to be selected and sent (Sso) for it.
R3: An application must never be finalised (Fa), if the respective offer has been declined (Do) already.
R4: An offer is either accepted (Ao) or declined (Do), but cannot be both accepted and declined.

A careful inspection of each one of the rules above would reveal that they are different in nature: rule R1 is an example of *cardinality rule*, which defines an upper and lower bound for the number of executions of an activity. Rule R2 contains a *precedence rule*, which establishes that the execution of a certain activity is preceded by at least on execution of another activity. Rule R3 establishes an *ordering rule*, whereas rule R4 represents an *exclusiveness rule*. Tables 2 and 3 show examples of cardinality and exclusiveness rules, respectively, for the running example and two log traces.

Table 2. Precedence rules derived for the process model of the running example and their satisfaction (✓) and violation (✗) by the exemplary log trace $\langle As, Sso, Fa, Ro, Co, Ro, Aaa, Af\rangle$. Each non-empty cell refers to a precedence rule. For instance, the activity to finalize the application (Fa) is preceded by the submission of the application (As) and the acceptance of the application (Aa). Yet, only the former rule is satisfied, whereas the latter one is violated in the given trace.

	As	*Da*	*Aa*	*Fa*	*Sso*	*Ro*	*Co*	*Ao*	*Aaa*	*Do*	*Af*
As											
Da	✓										
Aa	✓										
Fa	✓		✗								
Sso	✓		✗								
Ro	✓		✗		✓						
Co	✓		✗		✓	✓					
Ao	✓		✓	✓	✓	✓					
Aaa	✓		✗	✓	✓	✓		✗			
Do	✓		✓	✓	✓	✓					
Af	✓										

By assessing to what extent the traces of a log satisfy the rules derived from a process model, rule-based conformance checking focuses on the fitness dimension, i.e., the ability of the model to explain the recorded behaviour. Traces are fitting, if they satisfy the rules, or non-fitting if that is not the case. Let R_M be a predefined set of rules. Fitness can be defined according[2] to R_M:

$$\text{fitness}(L, M) = \frac{|\{r \in R_M \mid r \text{ is satisfied by all } t \in L\}|}{|R_M|} \tag{3}$$

As the reader may already have grasped, the dimension of precision is not targeted by rule-checking.

[2] Notice that this makes fitness to depend on a particular set of rules, which is a limitation of the rule-based fitness checking.

Table 3. Exclusiveness rules derived for the process model of the running example and their satisfaction (✓) and violation (✗) by the exemplary log trace $\langle As, Aa, Sso, Ro, Fa, Ao, Do, Da, Af \rangle$. Again, each non-empty cell denotes a rule, i.e., the absence of the execution of two activities for the same case. For instance, the acceptance of an offer (Ao) must not be executed for cases for which the application is declined (Da). Yet, in the given trace, respective events for both activities can be found, so that the rule is marked as being violated.

	As	*Da*	*Aa*	*Fa*	*Sso*	*Ro*	*Co*	*Ao*	*Aaa*	*Do*	*Af*
As	✓										
Da		✓						✗	✓		
Aa			✓								
Fa				✓							
Sso											
Ro											
Co											
Ao		✗						✓		✗	
Aaa		✓							✓	✓	
Do								✗	✓	✓	
Af											✓

2.3 Token Replay

Intuitively, this technique replays each trace of the event log in the process model by executing tasks according to the order of the respective events. By observing the states[3] of the process model during the replay, one can determine whether, and to what extent, the trace indeed corresponds to a valid execution sequence of the process model.

In essence, token replay postulates that each trace in the event log corresponds to a valid execution sequence of the process model. This is verified by step-wise executing tasks of the process model, according to the order of the respective events in the trace. During this replay, we may observe two cases that hint at non-conformance (see Fig. 6):

(i) the execution of a task requires the consumption of a token on the incoming arc, but the arc is not assigned any token in the current state, i.e., a token is *missing* during replay;
(ii) the execution of a task produces a token at an outgoing arc, but this token is not consumed eventually, i.e., a token is *remaining* after replay.

[3] A state of a BPMN model is a distribution of tokens over the control flow arcs. A task is enabled in a state if its incoming control flow arc is assigned a token by the respective distribution. If it executes, this token is *consumed*, i.e., no longer assigned to the arc. Moreover, a token is *produced* on the outgoing control flow arc of the task.

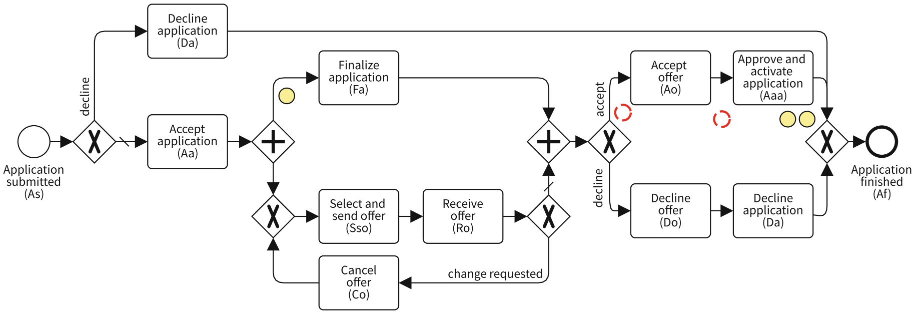


Fig. 6. State reached after replaying the full trace $\langle As, Aa, Sso, Ro, Ao, Aaa, Aaa\rangle$. One can see that there are three remaining tokens (denoted by yellow background), and two missing tokens (denoted by dashed red lines). (Color figure online)

By exploring whether the replay of a trace yields missing or remaining tokens, replay-based conformance checking mainly focuses on the fitness dimension. That is, the ability of the model to explain the recorded behaviour is the primary concern. Traces are fitting if their replay does not yield any missing or remaining tokens, and non-fitting otherwise:

$$\text{fitness}(L, M) = \frac{1}{2}\left(1 - \frac{\sum_{t\in L} missing(t, M)}{\sum_{t\in L} consumed(t, M)}\right) + \frac{1}{2}\left(1 - \frac{\sum_{t\in L} remaining(t, M)}{\sum_{t\in L} produced(t, M)}\right) \qquad (4)$$

In contrast to rule checking, precision can be estimated using token replay [34], but unfortunately, the corresponding technique strongly relies on the assumption that traces are fitting; if they are not, then the estimation of precision through token replay can be significantly degraded [2].

2.4 Alignments

Alignments take a symmetric view on the relation between modelled and recorded behaviour. Specifically, they can be seen as an evolution of token replay. Instead of establishing a link between a trace and sequences of task executions in the model through replay, alignments directly connect a trace with an execution sequence of the model.

An alignment connects a trace of the event log with an execution sequence of the process model. It is represented by a two-row matrix, where the first row consists of activities as their execution is signalled by the events of the trace and a special symbol $\gg$ (jointly denoted by e_i below), and the second row consists of the activities that are captured by task executions of an execution sequence of the process model and a special symbol $\gg$ (jointly denoted by a_i):

log trace	e_1	e_2	$\ldots$	e_n
execution sequence	a_1	a_2	$\ldots$	a_m

Each column in this matrix, a pair (e_i, a_i), is a *move* of the alignment, meaning that an alignment can also be understood as a sequence of moves. There are different types of such moves, each encoding a different situation that can be encountered when comparing modelled and recorded behaviour. We consider three types of moves:

– *Synchronous move*: A step in which the event of the trace and the task in the execution sequence correspond to each other. Synchronous moves denote the expected situation that the recorded events in the trace are in line with the tasks of an execution sequence of the process model. In the above model, a synchronous move means that it holds $e_i = a_i$ and $e_i \neq \gg$ (and thus $a_i \neq \gg$).

– *Model move*: When a task should have been executed according to the model, but there is no related event in the trace, we refer to this situation as a model move. As such, the move represents a deviation between the trace and the execution sequence of the process model in the sense that the execution of an activity has been skipped. In the above model, a model move is denoted by a pair (e_i, a_i) with $e_i = \gg$ and $a_i \neq \gg$.

– *Log move*: When an event in the trace indicates that an activity has been executed, even though it should not have been executed according to the model, the alignment contains a log move . Being the counterpart of a model move, a log move also represents a deviation in the sense of a superfluous execution of an activity. A log move is denoted by a pair (e_i, a_i) with $e_i \neq \gg$ and $a_i = \gg$.

Alignments are constructed only from these three types of moves (see an in-depth explanation on this in [14]). For instance, let us use the running example (see Fig. 3) and the trace $\langle As, Aa, Sso, Ro, Ao, Aaa, Aaa\rangle$. A possible alignment with this trace is:

log trace	As	Aa	Sso	Ro	$\gg$	Ao	Aaa	Aaa	$\gg$
execution sequence	As	Aa	Sso	Ro	Fa	Ao	Aaa	$\gg$	Af

This alignment comprises six synchronous moves, one log move, $(Aaa, \gg)$, and two model moves, $(\gg, Fa)$ and $(\gg, Af)$. The log move $(Aaa, \gg)$ indicates that the application had been approved and activated, even though this was not expected in the current state of processing (as this had just been done). The model move $(\gg, Fa)$ is the situation of the process model requiring that the application be finalised, which has not been done according to the trace. Furthermore, one can easily extract the original trace by projecting away the special symbol for skipping from the top row. Applying the projection to the bottom row yields the execution sequence of the model ($\langle As, Aa, Sso, Ro, Fa, Ao, Aaa, Af\rangle$).

In general, *optimal alignments*, i.e., alignments with a minimal number of model or log moves, are preferred. The alignment shown above is optimal since there is no other alignment with least number of deviations. Computing (optimal) alignments is a hot research topic, which has been addressed in many papers in the last years [1,9,19,31,39,44,45,56–58,60,66,68]. In this paper, however, we will refrain from describing the state-of-the-art methods for alignment computation, and refer the interested reader to the aforementioned papers, or to [14].

Moreover, the optimality of alignments may also be generalized in terms of a cost function that assigns costs to particular model moves or log moves, thereby enabling the categorization of deviations in terms of their severity. Then, an alignment is optimal, if the sum of costs assigned to all its moves is minimal. Setting the cost for all model moves and log moves to one, and for synchronous moves to zero, yields the aforementioned notion of optimality, i.e., alignments with a minimal number of model or log moves.

Remarkably, alignments provide a simple means to quantify fitness. Again, this may be done based on the level of an individual trace or the event log as a whole. However, the aggregated cost of log moves and model moves may be a misleading measure, though, as it is not normalised. A common approach, therefore, is to normalise this cost by dividing it by the worst-case cost of a aligning the trace with the given model. Under a uniform assignment of costs to log and model moves, such a worst-case cost originates from an alignment in which each event of the trace T_i relates to a log move, whereas all task executions of a sequence σ of the model relate to a model move and σ is as short as possible. Since the cost induced by the model moves of an execution sequence depends on its length, the shortest possible execution sequence leading from the initial state to a final state in the model is considered for this purpose.

Realising the above idea, we obtain two ratios that denote the relative share of non-fitness in the alignments of a trace or an event log, respectively. Let M be a model and L an event log. Then, we denote by $\mathit{cost}(t, M)$ the cost of an optimal alignment of a trace $t \in L$ with respect to the model. Furthermore, let $\mathit{cost}(t, \langle\rangle)$ and $\mathit{cost}(\langle\rangle, x)$ be the costs of aligning a trace t with an empty execution sequence, or some execution sequence $x \in M$ of the model with an empty trace, respectively. Then, fitness based on alignments is quantified for a trace or an event log:

$$\text{fitness}(L, M) = 1 - \left(\frac{\sum_{t \in L} \mathit{cost}(t, M)}{\sum_{t \in L} \left(\mathit{cost}(t, \langle\rangle)\right) + |L| \times \min_{x \in M} \mathit{cost}(\langle\rangle, x)} \right) \tag{5}$$

A simple precision metric based on alignments is grounded in the general idea of *escaping edges* [34]. To give the intuition, we assume that (i) the event log fits the process model; and (ii) that the process model is deterministic. The former means that we simply exclude non-fitting traces, for which the optimal alignment contains log moves or model moves, from the assessment of the precision of the model. The latter refers to a process model not being able to reach a state, in which two tasks that capture the same activity of the process are enabled. The model of our running example (see Fig. 3) is deterministic.

For the activity of each event of a trace of the event log, we can determine a state of the process model right before the respective task would be executed. Under the above assumptions, this state is uniquely characterised. What is relevant when assessing precision, is the number of tasks enabled in this state of the process model. Let M be a process model and L an event log, with $t \in L$ as a trace and, overloading notation, $e \in t$ as one of the events of the trace. Then, by $\mathit{enabled}_M(e)$, we denote the number of tasks and, due to determinism of the process model also the number of activities that can be executed in the state right before executing the task corresponding to e.

Similarly, we consider all traces of the log that also contain events related to the activity of event e, say a, and have the same prefix, i.e., events that indicate that the same sequence of activities has been executed before an event signalling the execution of activity a. Then, we determine the number of activities for which events signal the execution directly after this prefix, i.e., the set of activities that have been executed in the same context as the activity a as indicated by event e. Let this number of activities be denoted by $\mathit{enabled}_L(e)$, which, under the above assumptions, is necessarily less than or equal to $\mathit{enabled}_M(e)$. Then, the ratio of both numbers captures the amount of 'escaping edges' that represent modelled behaviour that has not been recorded. As such, precision of log L and M is quantified as follows:

$$\text{precision}(L, M) = \frac{\sum_{t \in L, e \in t} \mathit{enabled}_L(e)}{\sum_{t \in L, e \in t} \mathit{enabled}_M(e)} \qquad (6)$$

In summary, alignments are crucial to have accurate insights on the fitness and precision. However, as already acknowledged, they are hard to compute in general. In the remaining of this section, we briefly revise the challenge of computing alignments, together with some alternatives that have been proposed in recent years.

Computing Alignments. Computing an optimal alignment for an arbitrary combination of a process model and an event log is a far from trivial task. In terms of complexity, the task is as complex as reachability in Petri nets which, for general Petri nets, is undecidable. Nonetheless, several techniques exist to compute alignments. The best-known technique uses the $A^\star$ algorithm to find the shortest part in the reachability graph of the so-called synchronous product net [64]. This synchronous product is a combination of the process model and a Petri-net representation of the trace. Figure 7 shows an example of a synchronous product net for the running example. The algorithm associates costs to every transition in the synchronous product and uses these costs to find the shortest path from the initial marking to the final marking by expanding a minimal portion of the search space [67,68].

When synchronous products become too large to handle for a monolithic algorithm decomposition approaches can be used [29] to decompose the construction of an alignment into smaller problems which can be combined into a full optimal alignment. If optimality is not a requirement, sub-optimal alignments can be identified with a variety of techniques [51,52,60,62].

Another approach is the use of so-called satisfiability solvers [10]. The alignment problem is encoded as a SAT problem by translating the synchronous product to a set of boolean formulas. Because of this, the solution is limited

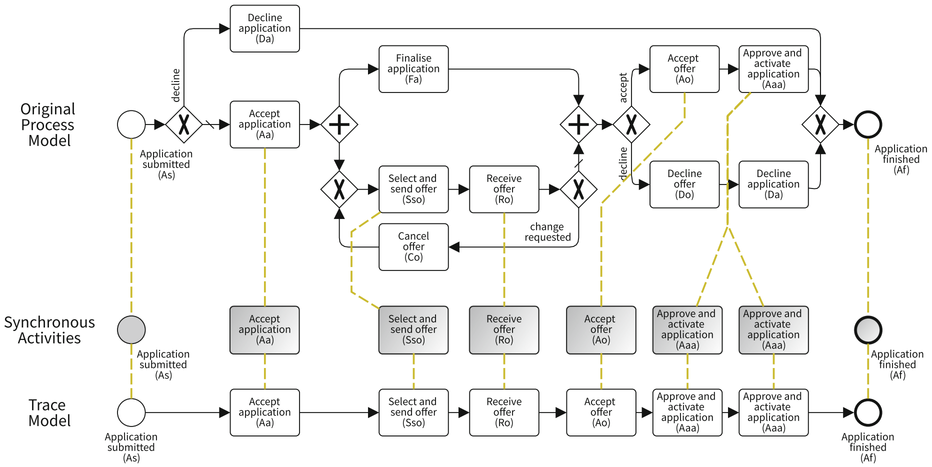

Original Process Model
Decline application (Da)
decline
Accept application (Aa)
Finalise application (Fa)
Select and send offer (Sso)
Receive offer (Ro)
Cancel offer (Co)
change requested
accept
decline
Accept offer (Ao)
Approve and activate application (Aaa)
Decline offer (Do)
Decline application (Da)
Application submitted (As)
Application finished (Af)
Synchronous Activities
Application submitted (As)
Accept application (Aa)
Select and send offer (Sso)
Receive offer (Ro)
Accept offer (Ao)
Approve and activate application (Aaa)
Approve and activate application (Aaa)
Application finished (Af)
Trace Model
Application submitted (As)
Accept application (Aa)
Select and send offer (Sso)
Receive offer (Ro)
Accept offer (Ao)
Approve and activate application (Aaa)
Approve and activate application (Aaa)
Application finished (Af)

Fig. 7. The synchronous product model for the running example and trace $T_1 = \langle As, Aa, Sso, Ro, Ao, Aaa, Aaa\rangle$.

to safe Petri nets. While strictly a limitation, this is hardly a problem as most process modelling languages found in industry belong to this class of models. A third approach for computing alignments, which is bound by the same limitation, uses job-shop schedulers to find the optimal set of moves [19].

Finally, symbolic techniques exist to compute alignments [43]. These techniques have the upside that they can compute alignments for large sets of traces at once, rather than trace by trace as all techniques above do. However, the downside is that they rely on the state space of the process model to be known. In models with many parallel constructs, this state space may be prohibitively large. An approach using an implicit representation of the state space by means of a *Binary Decision Diagram* was presented recently which alleviates the aforementioned explosion [9].

3 Relating Observed and Modelled Behaviour: Advanced Techniques

In the previous sections, the focus of conformance checking was very much on control flow, i.e. the ordering of activities in the event log in relation to the specified order in which activities should be executed according to the process model. However, real-life processes are not only about activities. Instead, processes are executed by people within an organization to reach a certain business goal. This goal is expressed by data in the process and the process model serves as a guide to reach the goal as efficiently or as precisely as possible.

Consider, for example, the event log in Table 1. Next to the case identifier and the activity, we also see other data such as the amount of the application, the corresponding offer id sent to the customer and whether or not this offer is signed. Not shown in this log is the identity of the employee who executed each activity, but it is not hard to imagine that companies have many employees of different roles and with different authorisation levels.

When doing conformance checking, it is important to consider all these elements and for this, more advanced conformance checking techniques, based on alignments, exist.

Data-Aware Alignments. Data plays a pivotal role in processes. Decisions are typically based on data that is provided at the start of a process or generated by any of the activities in the process. In our example of Table 1, the amount columns shows both types. Event e_{13} refers to an application being submitted by a customer, requesting a loan of 2000 euro. Event e_{37} subsequently shows that the bank offers the customer a loan for 1500 euro. In this process, the activity "Select and send offer" should not be executed with an amount higher than the requested amount. For application A5634 this is correct, but application A5636 shows a violation of this rule as the requested amount is only 200 euro, while the offered amount is 500.

To identify such data issues, several approaches exist. In [20,21] the authors first align the control flow using any of the techniques described above and then they check for deviations on the data level. This work is extended in [32] providing more control over the result and, especially, adopting a balanced view of control-flow and rules referring to the data perspective. Recently an approach that uses SMT solvers brings a fresh air to compute data-aware alignments [25].

Resource-Aware Alignments. Consider, for sake of argument, that the offered amount in our example can be higher than the requested amount, but only if the activity "select and send offer" is executed by a manager. In that case, the resource has a higher authorization level to actually deviate from the customer's request. However, if this happens, the final activity "Approve and activate application" also needs to be executed by a manager and this should not be the same person (four-eyes principle).

The relation between the roles and resource identities across different activities makes checking this more complex than data-aware alignments. The authors of [3] consider the resource perspective by looking at the various data operations in an event log and checking if these operations are performed by authorized resources.

Integrated Approaches. The techniques presented above share a common feature that they first align the control flow and then use the control-flow alignment to check data and resource rules. An important downside of this approach is that certain deviations may not be detected. Consider, for example, a manager who decides to login to the bank's system and read the application of his neighbour. As no activity is performed, the event log would not show any events and, when a data-access log is checked in isolation, the manager has the authority to read application data, hence no data-access violation is found. However, the manager read data outside of the context of a process, i.e. there was no business-goal associated to the read action.

To comprehensively check the conformance of an event log from the viewpoint of the control flow, the data-access and resource authorization, a more recent approach has been developed by Mozafari et al. [5]. In their paper, the combination of an event log, a data access log and a resource model is used to construct a large synchronous product. This synchronous product is subsequently used to find optimal alignments with respect to deviations in all three perspectives combined without favouring one over the other. These deviations include, for example, spurious data access and authorization problems where otherwise authorized users access data outside of the context of the case they are working on.

Compliance Checking. The focus of conformance checking so far has been on the situation where end-to-end process models are available. However, in many companies, such process models do not exist. Instead, each process is only governed by a set of compliance rules, i.e. all activities can be performed, as long as these rules are not broken. Rule engines, as discussed earlier in the introduction, can typically raise flags when as soon as business rules are violated. However, a rule engine typically only recognizes the moment when a rule is violated. Conformance checking using alignments can also be used to identify that specific business rules are not yet fulfilled, but no violation occurred yet.

The work of Ramezani et al. [54] shows how typical compliance rules from the accountancy and control domain can be translated into small Petri nets which in turn can be aligned with event logs to identify violations against these rules as log- or model-moves.

Realtime (or Righttime) Conformance Checking. So far, conformance checking was discussed as a technique to identify deviations after processes have been concluded. However, in some cases, it may be interesting to detect deviations during the execution of a process [12,13,70]. Such techniques are often referred to as streaming techniques, i.e., data is being processed as it comes in and a realtime dashboard provides insights into the current conformance level of an entire process. This is particularly useful in environments where employees have a great deal of flexibility in executing activities within a process but where specific conditions have to be met at the end.

Conformance Checking Without Process Models. Finally, a specific type of conformance checking exists which does not rely on a traditional notion of a process model. Instead, the event log itself is used as a representative of both the correct and incorrect behaviour and deviations are detected between the mainstream behaviour prevalent in the event log and the 'outlier' cases [36,37]. Specifically, this approach employs recurrent neural networks (RNNs) that are trained for next activity prediction moving through a trace forward or backward. These predictions can be seen as an approximation of a process model against which the alignments of traces are computed.

4 Applications of Conformance Checking

So far, we discussed essential techniques for conformance checking along with their generalization and extension to scenarios beyond the traditional, retrospective analysis of control-flow information. Next, we turn the focus to the broader field of applications of conformance checking.

We first note that an understanding of the link between the recorded and modelled behaviour of a process serves as a foundation for various model-based techniques for the analysis of qualitative and quantitative process properties. The importance of conformance checking for such analysis is detailed in Sect. 4.1, taking techniques for the analysis of performance characteristics and decision points as examples.

A second important observation relates to the fact that deviations between recorded and modelled behaviour, as revealed by conformance checking, can potentially be attributed to quality issues in the event log or the process model. Both, a log and a model, denote representations of the process at hand, which may be incomplete, outdated, imprecise, or simply wrong. This gives rise to a generalized notion of conformance checking, which aims at a separation of deviations that are due to quality issues in the event log or the process model. As discussed in Sect. 4.2, this generalized view on conformance checking enables us to describe common techniques for process mining as part of a unified framework.

4.1 The Case of Model-Based Process Analysis

Process models serve as the starting point for a plethora of process analysis techniques. Such analysis may be classified along various dimensions. That is, the point in time addressed by the analysis distinguishes retrospective, predictive, or even prescriptive analysis of a business process. The granularity of the analysis may be defined to be on individual instances of a process or a set thereof, thereby integrating potential interactions between different instances of a process. Moreover, analysis based on a process model may incorporate diverse process perspective, starting with the traditional view only on the control-flow of the process, through the data produced and consumed during its execution, the impact of such data on the control-flow, the integration of events produced by the environment in which the process is executed, the utilization of resources, and the definition of organizational responsibilities, to name just a few examples.

Regardless of the specific type of model-based process analysis, conformance checking provides a means to ensure that the models provide reasonable representation of the actual behaviour. Considering the behaviour as recorded in an event log as a representation of actual process execution, despite all potential issues related to data quality, such as accuracy and completeness of an event log, conformance checking establishes trust into the analysis results obtained from the models. In the following paragraphs, we reflect on this application of conformance checking for three types of model-based analysis techniques.

Performance Analysis. Performance properties are an important aspect of process analysis in various domains. Here, specific measures include information on the time needed by a process instance from start to end, also known as cycle time or sojourn time, which is captured in terms of simple statistics, such as the average or maximal sojourn time, or complete distributions. Moreover, understanding how much time is needed to reach a certain milestone in the execution of a process is valuable information for operational process management, e.g., related to the scheduling of resources.

To enable the respective analysis, a process model is enriched with performance information. Common notions include simple annotations such as the average execution time per task. Yet, one may also consider more elaborated annotations, such as the distributions of not only the execution time per task, but also the wait time between the execution of subsequent tasks. Based on these annotations, analytical techniques or simulation are used to compute performance measures.

Given an event log, conformance checking that links the events of traces to the tasks in a process model helps to extract such performance annotations. For instance, once an alignment is computed, the synchronous steps indicate for which temporal information attached to events needs to be incorporated for the annotations for specific tasks. Note that this is particularly beneficial, once a model contains several tasks with the same label, i.e., representing the same activity of the process. In that case, an alignment separates the events that shall serve as the basis for the performance annotation of the different tasks based on the behavioural context in which they are executed. For instance, the model for a loan application process in Fig. 3 contains two tasks for declining an application (Da). This way, the respective activity may be executed in different contexts, once directly after the submission of the application and once towards the end of the process, after an offer has been declined. Consequence, both tasks may be have different performance characteristics. Alignments help to incorporate these differences by separating the events that are linked to either task.

However, conformance checking may not only employed to extract performance annotations from an event log, but also enables their validation. For instance, performance annotations may have been defined manually, based on expectations. Then, temporal information of the event log may be utilized to validate these annotations, where, again, conformance checking indicates which events shall be considered for which of the tasks in the process model.

Decision Point Analysis. Decision point analysis aims at insights on the conditions that govern decision points in a process. In process modelling, it is a common abstraction to neglect such conditions and simply assume that a non-deterministic choice is taken, as the conditions may not be relevant for some control-flow-oriented analysis. However, this abstraction may also be problematic, as it hides how the context of process execution influences the control-flow, e.g., that certain activities are executed solely for certain types of cases. Such insights are particularly relevant also for performance analysis as discussed above, since the conditions at a decision point may induce highly skewed distributions. In our running example, Fig. 3, there is a first decision point directly after the submission of an application, which may lead to an immediate rejection and, hence, completion of process execution. Understanding the properties of cases that govern this decision will, therefore, be very beneficial for any analysis of performance characteristics.

To understand the conditions at decision points of a process, decision mining may employed. It takes traces of an event log, including the data attached to the events or the trace as a whole, as observations for particular decision outcomes. Then, a classification problem is derived, with the different outcomes being the classes, and common techniques for supervised classification enable the construction of a classifier. Assuming that the obtained classifier can be interpreted, e.g., is represented as a decision tree, the conditions for a decision point can be extracted and added to a process model.

In this context, conformance checking, again, helps to prepare a process model for analysis, as well as to validate existing annotations. In the former case, alignments that link events to tasks help to prepare the data needed for decision mining. Through an alignment, the data available at a specific decision point is characterised and may be used as input to the classification algorithm. The later case, the decision points in a process model have already been annotated with the respective conditions. Then, conformance checking reveals if these conditions are matched with the behaviour recorded in the event log, either by constructing an alignment solely based on control-flow information and checking the conditions at decision points separately, or by integrating the conditions directly in the computation of multi-perspective alignments as discussed in Sect. 3.

4.2 A General View on Conformance Checking

An event log and a process model both denote representations of an abstract entity, the actual process as it is implemented in an organization. From this view point, illustrated in Fig. 8, it becomes clear that any deviation detected between these representations may potentially be attributed to the way that the representations capture the actual process, i.e., the log and the model may show quality issues. For instance, logging mechanisms may be faulty and the integration of event data from different systems may be imprecise. Similarly, models may have been created based on an incomplete understanding of the process and may be biased towards the expected rather than the actual behaviour. Moreover, in many application contexts, processes are subject to change and evolve over time. Hence, process models created at some time point become outdated. Event logs that span a large time period, in turn, may contain information about

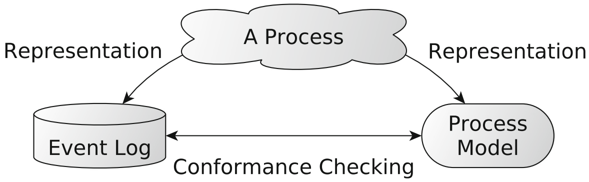


Fig. 8. Both, an event log and a process model, are representations of a process.

different versions of a process, so that the log in its entirety appears to describe a process that was actually never implemented as such at any specific point in time.

From the above observation, it follows that a deviation between an event log and a process model may be interpreted as an issue to fix in either of the representations. That is, one of the representations is assumed to be correct, i.e., it is assumed to truthfully denote the actual process, whereas the other representation is updated with the goal to resolve the deviation. Specifically, techniques to repair a process model based on the event log and techniques to repair an event log based on the process model have been proposed in the literature, as discussed next.

Model Repair. Assuming that an event log constitutes a correct representation of a process' behaviour, deviations detected between the log and a process model are a starting point for model repair. To this end, existing techniques are mostly based on alignments computed between a trace of an event log and the process model. The reason being that alignments clearly separate behaviour that is only observed in the process model (i.e., a move in model) and behaviour that is present only in the trace (i.e., a move in log).

Intuitively, a move in model captures the situation that the execution of an activity is defined to be mandatory, while this execution is optional according to the supposedly correct event log. Therefore, a simple repair strategy is to relax the control-flow defined by the model and explicitly enable the continuation of a process instance without executing the respective activity. Note though that different syntactical changes may be considered to realize this change. For instance, in a BPMN process model, one may insert a decision point before the task to determine whether it is executed, whereas a similar effect may also be achieved by changing the semantics of existing routing constructs, such as transforming a parallel split into an inclusive choice.

A move in log, on the other hand, hints at a supposedly correct activity execution that is without counterpart in the model. A repair strategy, therefore, is to insert a corresponding task into the process model. The location for this insertion is also determined based on the conformance checking result. That is, the alignment up to the respective move in log induces a state in the process model. The task needs to be inserted, such that it is activated in this state and such that before and after its execution, all tasks that have been activated in the original state are still activated. In practice, such a repair operation may not only be conducted on the level of individual model in log steps, but for sequences thereof. In this case, a model fragment to capture the behaviour of this sequence is discovered and inserted into the original model.

As an example, consider the following alignment for the process model introduced earlier (Fig. 3).

log trace	As	Aa	Sso	Ro	$\gg$	Ao	Aaa	Aaa	$\gg$
execution sequence	As	Aa	Sso	Ro	Fa	Ao	Aaa	$\gg$	Af

From the move in model $(\gg, Fa)$, one may derive a change in the process model that enables skipping of the respective task Fa in the process model. The move in log $(Aaa, \gg)$, in turn, suggests a change in the model that supports an additional execution of the activity to approve and activate an application. Yet, we note that this activity execution directly succeeds a synchronous move for a task referring to same activity. Hence, instead of adding a new task in the process model, it may be more desirable to generalise the process model and insert a loop around the existing task Aaa, so that it may be executed multiple times in an execution sequence of the model.

Log Repair. The idea of repairing a process representation based on the results of conformance checking may also be applied to event logs. Given an alignment of a trace and a process model, the actual changes to apply to the trace are derived from the types of the respective alignment steps. Under the assumption that the model is a correct representation of the process, a move in model would lead to the insertion of an event into the trace at the position of the alignment step. An event that is part of a move in log, in turn, would be deleted from the trace.

In practice, the insertion or deletion of events of a trace may be problematic. For instance, the creation of artificial events raises the question of how to define the values of an events' attributes, from generic ones such as an events' timestamp to domain-specific attributes (e.g., the state of a business object). Against this background, log repair may not focus on alignment steps in isolation, but aim at identifying high-level changes. An example would be the presence of two alignment steps, a move in model and a move in log, both related to the execution of the same activity. Instead of deleting and inserting an event, moving the event from the position of the move in log step to the position of the move in model step would enable repair without the need to generate an artificial event.

Taking up the aforementioned example, based on the alignment, log repair may suggest that the second event linked to the approval and activation of the application (Aaa) is erroneous (e.g., the activity execution was recorded twice due to a faulty logging mechanism) and, thus, shall be removed from the trace.

At the same time, it may suggest to insert events for the activities to finalise the application (*Fa*) and finish the application (*Af*), for instance, assuming that these steps are manually recorded, so that some incompleteness of the trace is to be expected.

Generalized Conformance Checking. Both, model repair and log repair consider one process representation to be correct, which may therefore serve as a ground truth. In the general case, however, quality issue may be present in both representations. As a consequence, some of the conformance issues detected between a model and a log may stem from the model not adequately capturing the process, some of them may originate from low quality of the event log, while some are also inherent deviations that need to be analysed.

To balance the different reasons of conformance issues, it was suggested to incorporate a notion of *trust* in the process model, denoted by $\tau_M \in [0, 1]$, as well as the event log, denoted by $\tau_L \in [0, 1]$ [48]. These trust values capture the assumed correctness of either representation and may reflect how the representation has been derived. For instance, a process model created as part of a first brainstorming session may be less trustworthy in terms of correctness and completeness compared to a model created as a part of a rigorous process management initiative. Similarly, an event log created by a process-oriented information systems can, in general, be expected to be more trustworthy than a manual documentation of activity executions by a diverse group of process stakeholders.

Once a trust level has been specified for both, the model and the log, conformance checking may be phrased as an optimization problem that incorporates model and log repair. To this end, the following notions need to be defined: A function δ_{L^2} to measure the distance of two event logs; a function δ_{M^2} to measure the distance of two models; and a function $\delta_{L,M}$ to measure the distance of an event log and a process model, such as alignment-based fitness or a combination of fitness and precision. Given an event log L and a process model M, generalized conformance checking [48] is then defined as the identification of some adapted log L^* and adapted model M^*, such that:

$$\begin{aligned} L^*, M^* = \operatorname{argmin}_{L', M'} &\left(\delta_{L^2}(L, L'), \delta_{M^2}(M, M'), \delta_{L,M}(L', M')\right) \\ \text{subject to } &\delta_{L^2}(L, L') \leq 1 - \tau_L \text{ and } \delta_{M^2}(M, M') \leq 1 - \tau_M. \end{aligned}$$

Intuitively, the above problem formulation considers that the given model M and log L may require to be adapted, if they are not fully trustworthy. However, the trust values induce a bound for the distance between any adapted model and log, and the original model and log, respectively, as illustrated in Fig. 9. Within the space set by these bounds, the distances between the adapted and original model, between the adapted and original logs, and between the adapted model and the adapted log shall be minimised. Here, a specific instantiation may require the minimisation of a linear combination of the three distances.

Generalized conformance checking unifies various tasks in the field of process mining [14,48]. Table 4 highlights how specific tasks can be seen as instances

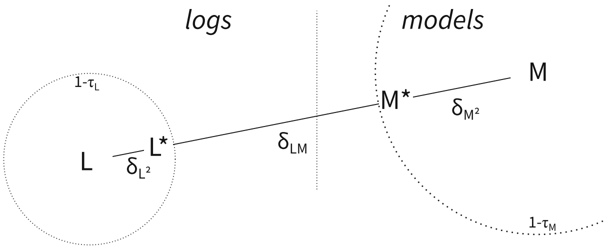


Fig. 9. The problem of generalized conformance checking (from [14]).

Table 4. Overview of process mining tasks listed as instances of the generalised conformance checking problem according to [14,48].

Process mining task	Log Trust	Model Trust
Classical Process Discovery finds a model that best fits to the entire event log, e.g., the alpha algorithm [65]	$\tau_L = 1$	$\tau_M = 0$
Heuristic Process Discovery algorithms apply preprocessing to the event log by discarding infrequent patterns [26,79]	$0 < \tau_L < 1$	$\tau_M = 0$
Model Repair fixes deficient models due to, e.g., a change in the system that is reflected in the log. For example [24]	$\tau_L = 1$	$0 < \tau_M < 1$
Conformance Checking tries to find misalignments between event log and model. Example works include [50,53,64]	$\tau_L = 1$	$\tau_M = 1$
Log Repair modifies the log such that it better conforms to the given trusted model [47,74]	$0 < \tau_L < 1$	$\tau_M = 1$
"Happy Path" Simulation is complementary to heuristic process discovery. It is a theoretical use case where we do not trust infrequent parts of the model [33]	$\tau_L = 0$	$0 < \tau_M < 1$
Process Simulation is complementary to process discovery, where we are given an untrustworthy empty log and a fully trustworthy model	$\tau_L = 0$	$\tau_M = 1$
Garbage In, Garbage Out. When both the model and the log are untrustworthy, the best log and model tuple that fits them is any pair of model and log that fits each other, including an empty log and an empty model	$\tau_L = 0$	$\tau_M = 0$
Generalised Conformance Checking answers the question where the model would best be adopted, and where the log would best be adopted for a better overall fit. This goes beyond alignments, as the latter only detect the misalignments without specifying which side is to "blame"	$0 < \tau_L < 1$	$0 < \tau_M < 1$

of the generalized conformance checking problem, depending on the trust into an event log or a process model. Specifically, tasks such as classical process discovery or process simulation fit into this picture when assuming that there is no trust into the model or the log, which can be interpreted as the setting that the respective artifacts are not available.

5 Further Reading

Conformance checking has evolved significantly in the last decade, enabling the industrial adoption and commercial software offerings. As it has been shown in Sect. 3, techniques beyond control flow are already been proposed in the last years, since considering other perspectives brings significant value and triggers adoption.

Still, the core of the techniques developed are still focusing on the algorithmic aspects of the computation of conformance artefacts for the control flow perspective. We now review further work on the three dimensions considered in this chapter, thereby providing pointers for further reading.

Rule Checking. The idea of rule-based conformance checking is to rely on constraints which are then checked for the traces of an event log. The idea of rule-based conformance checking has been brought forward in [76]. It employs constraints derived from the (causal) behavioural profile of a process model [75, 77], which are sets of binary relations over activities derived from the order of potential occurrences of tasks in the execution sequences of the model.

This general idea, however, is not limited to a specific set of rules. Rather, other notions of constraints can be used in the very same manner, including transition adjacency relations [80] and the rules of the 4C spectrum [42]. Such sets of binary rules to capture behaviour are inherently limited in their expressiveness, though, as already for relatively simple classes of models, an exponentially growing number of rules would be needed to capture the complete behaviour [40]. Rule-based conformance checking, therefore, lends itself to scenarios, where only certain constraints need to be checked rather than the complete behaviour as specified by a process model.

While the results of rule checking enable insights on deviant traces, they may also be used for aggregated conformance measures. For instance, fitness measures may be derived based on the numbers of satisfied and violated rules [76]. Also, filtering of rule violations and discovery of associations between them may provide further insights into context of non-conformance [76,78].

Finally, conformance checking based on rules has the advantage that it can be lifted to online scenarios in a straight-forward manner. To this end, rules can be translated to queries over streams of events, see [17,78], which enables the use of algorithms and systems developed for complex event processing [16].

Token Replay. Techniques for token replay were first introduced in [49]. Alternative techniques were presented in [72], and later adapted to an online scenario in [71]. Recently new heuristics have been recently proposed that make token replay a fast alternative to alignments [7,8].

Alignments. The seminal work in [1] proposed the notion of alignment and developed a technique based on A* to compute optimal alignments for a particular class of process models. Improvements of this approach have been presented recently in different papers [66,68]. The approach represents the state-of-the-art technique for computing alignments, and can be adapted (at the expense of increasing significantly the memory footprint) to provide all optimal alignments. Alternatives to A* have appeared in the last years: in the approach presented in [19], the alignment problem is mapped as an *automated planning* instance. Automata-based techniques have also appeared [31,44]. The techniques in [44] (recently extended in [45]) rely on state-space exploration and determination of the automata corresponding to both the event log and the process model, whilst the technique in [31] is based on computing several subsets of activities and projecting the alignment instances accordingly. We also highlight the recent approach that is grounded on the use of relaxation labelling combined with A*, to provide a light alternative to compute alignments [39].

The work in [57] presented the notion of *approximate* alignment to alleviate the computational demands by proposing a recursive paradigm on the basis of the structural theory of Petri nets. In spite of resource efficiency, the solution is not guaranteed to be executable. Alternatively, the technique in [59] presents a framework to reduce a process model and the event log accordingly, with the goal of alleviating the computation of alignments. The obtained alignment, called *macro-alignment* since some of the positions are high-level elements, is expanded based on the information gathered during the initial reduction. Techniques using local search have recently been also proposed very recently [61].

Against this background, the process mining community has focused on divide-and-conquering the problem of computing alignments, as a valid alternative to this problem with the aim of alleviating its complexity without degrading the quality of the solutions found. We turn now our focus to decompositional approaches to compute alignments, which are more related to the research of this paper.

Decompositional techniques have been presented [35,63,73] that, instead of computing optimal alignments, they focus on the *crucial problem* of whether a given trace fits or not a process model. These techniques vertically decompose the process model into pieces satisfying certain conditions (so only *valid* decompositions [63], which satisfy restrictive conditions on the labels and connections forming a decomposition, guarantee the derivation of a real alignment). Later on, the notion of *recomposition* has been proposed on top of decompositional techniques, in order to obtain optimal alignments whenever possible by iterating the decompositional methods when the required conditions do not hold [29]. In contrast to the aforementioned vertical decomposition techniques, this approach does not require this last consolidation step of partial solutions, and therefore can be a fast alternative to these methods at the expense of loosing the guarantee of optimality.

There has been related work also on the use of partial order representations of process models for computing alignments. In [13], unfoldings were used to capture all possible transition relations of a model so that they can be used for online conformance checking. In contrast, unfoldings were used recently in a series of papers [38,60] to speed-up significantly the computation of alignments.

We believe these approaches, specially the last two, can be easily integrated in our framework.

Also, the work of [45] can also be considered a decompositional approach, since it proposes decomposing the model into sequential elements (*S-components*) so that the state-space explosion of having concurrent activities is significantly alleviated. We believe that this work is quite compatible with the framework suggested in this paper, since the model restrictions assumed in [45] are satisfied by the partial models arising from our horizontal decomposition.

Finally, the MapReduce distributed programming model has already been considered for process mining. For instance, Evermann applies it to process discovery [23], whilst [15] applies it for monitoring declarative business processes. Recently, MapReduce techniques has been proposed to offer a horizontal decompositonal alternative to computing alignments [62].

6 Milestones and Challenges

Conformance checking is nowadays a mature field, demonstrated by its presence in some of the process mining commercial tools and process mining use cases. In spite of this, the available support for its adoption is far from complete. One example is the metrics available: whilst fitness or precision are considered well evaluated through current techniques, accurate generalization metrics that additionally can be evaluated efficiently are yet to come [41,69,72].

Alignments are a central pillar of current techniques for conformance checking. However, the complexity requirements of the state-of-the-art techniques hamper their application for large instances (see Sect. 2.4). Actually, process mining is facing the following paradox: whilst there exist techniques to discover process models arbitrarily large [4,30], most of the existing alignment computation techniques will not be able to handle such models. Alternative approaches, like the decomposition or structural techniques only alleviate the problem, at the expense of losing the guarantee of important properties like optimality. Also, when incorporating other dimensions like data or resources, so that a multi-perspective for conformance checking is enabled, the complexity of the problem increases significantly, making it difficult to be applied for real-life problems; we envision new contributions also for multi-perspective conformance checking in the near future that can overcome this limitation.

Beyond computational or algorithmic challenges, there are other equally important challenges, more oriented towards considering the understanding of conformance checking results. One of them is the visualization of deviations. In industrial scenarios, thousand of deviations can easily pop up when assessing conformance, and it is not so easy to rank the importance of each one with a criteria that really impacts the business of the organization. For instance, looking at Fig. 1, one can see the list of violations at the bottom of the figure, ordered by the percentage of the cases where these deviations occur. This may not necessarily be the most interesting ranking from a business perspective.

We now provide a list of particular challenges with the aim of triggering future research in the field. The list is by no means complete.

Representing Uncertainty and Preventing Bias. As mentioned above, conformance checking deals with the comparison of recorded behaviour against specified behaviour, typically represented as an event log and process model. Based on this comparison, conclusions can be drawn with respect to the recorded behaviour as well as the underlying process which produced the recorded behaviour.

This distinction becomes only irrelevant when the recorded behaviour contains all the process behaviour of interest. In all other situations, where the observed behaviour is only a sample of the complete process behaviour, a source of variation is introduced by the sample. Sampling variation will cause the outcome of the conformance checking activity, which is only an estimate of the true value, to vary over different samples. Initial work on this direction has been recently proposed [6,28].

Information on the accuracy of a specific conformance estimate is important for a practitioner to make informed decisions. Unfortunately, representing uncertainty is typically ignored by existing conformance checking techniques and remains an important open challenge.

A second related challenge is that practitioners not only want an idea about the estimate's accuracy, but also want some guarantee that the estimate is unbiased. Various sources of errors exist which could lead to biased results, which receive too little attention in the existing work on conformance checking.

Some of the most relevant sources of errors are coverage error and construct validity. Coverage error occurs when the recorded behaviour is not a representative sample of the underlying process under analysis. This could be caused e.g. by non-random sampling or an incorrect definition of the underlying process. Construct validity refers to the question whether the conformance technique actually measures what it claims to be measuring.

This issue of estimate bias raises at least three important challenges. Firstly, there is a need for further development of conformance techniques which produce unbiased estimates, as recent research empirically challenged the claim that existing estimators are unbiased [27]. Secondly, with respect to construct validity, more attention should be given to making explicit what a measure actually represents. In particular the concept of generalisation suffers from an ambiguous and unclear definition, while other conformance measures are so complex that it is no longer clear what is measured and how it behaves. Thirdly, illustrating that a conformance estimator is unbiased should become a fundamental methodological part of any paper introducing and reviewing conformance checking techniques.

Computational Feasibility. As with many data analysis tasks, computation feasibility is a challenge. In the context of conformance checking, different elements contribute to this. One element lies in the current approach itself. As we highlighted before, alignment-based approaches are the state-of-the-art techniques to conformance checking due to its robustness and detailed diagnosis on deviations at the event level. However, it is also a computationally intensive operation that can take a long time to execute and can even be unfeasible for industrial-sized processes.

Further, computational feasibility is challenged by the persistently growing size of event logs. In the industry, huge quantities of events are recorded. For example, Boeing jet engines can produce ten terabytes of operational information every thirty minutes and Walmart is logging one million customer transactions per hour. In these contexts, operational efficiency is typically of paramount importance and is ensured by having predefined operational protocols and guidelines. Consequently, aside from being capable of dealing with complex and large underlying processes, conformance checking techniques should also support large amounts of data.

Responding to these computational challenges, techniques that are tailored for the emergence of large event logs and processes are created. For example, it is often not possible to store all the event data produced by large processes due to the limitation of storage capacity. This has motivated techniques that allow conformance checking to be performed in an online setting to data streams that are continuously producing event data related to ongoing cases. While a solution for one challenge, this response in itself holds additional challenges.

Online Conformance Checking. Online conformance checking analyzes event streams to assess their conformance with respect to a given reference model (the reader is referred to [11] The key aspect of this problem is that events must be analyzed immediately after they are generated (without storing them). The key benefit of this technique is to be able to detect deviations immediately, thus giving time to the process manager to shift the trace back to the reference behaviour. More generally, the main benefit is the reduction in latency among the BPM lifecycle phases.

Event streams represent a specific type of data streams, where each data point is an event (as in standard event logs). General data stream mining techniques have been studied in the past and several stream operations models have been defined, including: insert-only streams; insert-delete streams; and additive streams. Respectively, events are only inserted; deleted; or "incremented" (this holds typically for numerical variables). Typically, event streams are assumed to be insert-only streams, where events are just added to the stream.

Since event streams are generally assumed to be unbounded and events are supposed to arrive at unpredictable pace, several constraints are imposed on the analysis. Specifically, once an element is received, it should be processed immediately: either it is analyzed or it is discarded. In case it is analyzed, it cannot be explicitly inspected again at a later stage: since the stream is unbounded, it is impossible to store it and its events have to be stored in an aggregated (or summarized) fashion. Additionally, the time scale plays a fundamental role in online conformance checking: a recent deviation is more important than older ones, as the process manager can immediately enact proper countermeasures.

Several problems are still open, for example how to find good conformance measures, which operate in efficient (i.e., constant) time. Other relevant and unsolved problems are handling streams where the arrival time of events does not coincide with the execution time (thus, events need to be "sorted" afterwards), or understanding when a process instance is really terminated (even if the termination state of the model has been reached).

Desired Properties of Conformance Measures. The objective of conformance checking is to provide insights on how well a model describes given event data or how well given event data describes the model. This is represented both quantitatively - for measuring conformance - and qualitatively - for providing diagnostic information. We discuss properties and challenges of measures and diagnostics information in conformance checking.

Similar to machine learning, conformance measures are used to assess how well a model describes the event data: A model should have a high fitness or recall to the log (be able to replay all observed traces) and a high precision to the log (show little additional behaviour). Models with high fitness and precision distinguish themselves further in terms of generalization (their ability to replay likely, but so far not observed traces of the process that generates the log) and simplicity (being structurally simple). In this sense, we use conformance measures to compare two different models M1 and M2 with each other in their ability to describe a given log L (in terms of fitness and precision), describe the unknown process P behind the log (in terms of generalization), and be easily understandable (through a simple model structure). A model M1 scoring higher than a model M2 in a measure is considered to be the "better" model. For most event logs, the quality measures define a pareto front: a model scoring better on one measure scores worse on another measure leading to a set of "best" models for which no model can be found scoring better on any measure without scoring worse on other measures. With these properties, conformance measures have two main applications: helping a user decide which among a set of possible models is a preferred description of the event data, evaluating and benchmarking algorithms in process mining.

As it has been recently suggested [55], establishing certain axioms is a safe way to be able to determine the boundaries of a certain technique for determining a particular quality dimension. These axioms are expected to clarify important aspects such as logical consistency, robustness, confidence, to name a few.

7 Conclusions

This chapter provided an overview on conformance checking, aiming at covering the basic techniques, pinpointing what are the natural applications of the field and looking into the future by listing challenges that we believe will be crucial to overcome in the years to come. The chapter may be seen as a gentle introduction to the reference book in the field, where most of the topics are extensively developed [14].

Acknowledgements. This work has been supported by MCIN/AEI funds under grant PID2020-112581GB-C21.

References

1. Adriansyah, A.: Aligning observed and modeled behavior. Ph.D. thesis, Technische Universiteit Eindhoven (2014)
2. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Measuring precision of modeled behavior. Inf. Syst. E-Bus. Manag. **13**(1), 37–67 (2015)
3. Alizadeh, M., Lu, X., Fahland, D., Zannone, N., van der Aalst, W.M.P.: Linking data and process perspectives for conformance analysis. Comput. Secur. **73**, 172–193 (2018)
4. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowl. Inf. Syst. **59**(2), 251–284 (2018). https://doi.org/10.1007/s10115-018-1214-x
5. Mozafari Mehr, A.S., de Carvalho, R.M., van Dongen, B.: Detecting privacy, data and control-flow deviations in business processes. In: Nurcan, S., Korthaus, A. (eds.) CAiSE 2021. LNBIP, vol. 424, pp. 82–91. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79108-7_10
6. Bauer, M., van der Aa, H., Weidlich, M.: Sampling and approximation techniques for efficient process conformance checking. Inf. Syst. **104**, 101666 (2022)
7. Berti, A., van der Aalst, W.M.P.: Reviving token-based replay: Increasing speed while improving diagnostics. In: van der Aalst, W.M.P., Bergenthum, R., Carmona, J. (eds.) Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2019 Satellite Event of the Conferences: 40th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2019 and 19th International Conference on Application of Concurrency to System Design ACSD 2019, ATAED@Petri Nets/ACSD 2019, Aachen, Germany, 25 June 2019, vol. 2371 of CEUR Workshop Proceedings, pp. 87–103. CEUR-WS.org (2019)
8. Berti, A., van der Aalst, W.M.P.: A novel token-based replay technique to speed up conformance checking and process enhancement. Trans. Petri Nets Other Model. Concurr. **15**, 1–26 (2021)
9. Bloemen, V., van de Pol, J., van der Aalst, W.M.P.: Symbolically aligning observed and modelled behaviour. In: 18th International Conference on Application of Concurrency to System Design, ACSD 2018, Bratislava, Slovakia, 25–29 June 2018, pp. 50–59 (2018)
10. Boltenhagen, M., Chatain, T., Carmona, J.: Optimized SAT encoding of conformance checking artefacts. Computing **103**(1), 29–50 (2020). https://doi.org/10.1007/s00607-020-00831-8
11. Burattin, A.: Streaming process mining. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
12. Burattin, A., Carmona, J.: A framework for online conformance checking. In: Teniente, E., Weidlich, M. (eds.) BPM 2017. LNBIP, vol. 308, pp. 165–177. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74030-0_12

13. Burattin, A., van Zelst, S.J., Armas-Cervantes, A., van Dongen, B.F., Carmona, J.: Online conformance checking using behavioural patterns. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 250–267. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_15
14. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99414-7
15. Chesani, F., Ciampolini, A., Loreti, D., Mello, P.: Map reduce autoscaling over the cloud with process mining monitoring. In: Helfert, M., Ferguson, D., Méndez Muñoz, V., Cardoso, J. (eds.) CLOSER 2016. CCIS, vol. 740, pp. 109–130. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62594-2_6
16. Cugola, G., Margara, A.: Processing flows of information: from data stream to complex event processing. ACM Comput. Surv. **44**(3), 15:1–15:62 (2012)
17. Daum, M., Götz, M., Domaschka, J.: Integrating CEP and BPM: how CEP realizes functional requirements of BPM applications (industry article). In: Bry, F., Paschke, A., Eugster, P.Th., Fetzer, C., Behrend, A. (eds.) Proceedings of the Sixth ACM International Conference on Distributed Event-Based Systems, DEBS 2012, Berlin, Germany, 16–20 July 2012, pp. 157–166. ACM (2012)
18. de Leoni, M.: Foundations of process enhancement. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
19. de Leoni, M., Marrella, A.: Aligning real process executions and prescriptive process models through automated planning. Expert Syst. Appl. **82**, 162–183 (2017)
20. de Leoni, M., van der Aalst, W.M.P.: Aligning Event logs and process models for multi-perspective conformance checking: an approach based on integer linear programming. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 113–129. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40176-3_10
21. de Leoni, M., van der Aalst, W.M.P., van Dongen, B.F.: Data- and resource-aware conformance checking of business processes. In: Abramowicz, W., Kriksciuniene, D., Sakalauskas, V. (eds.) BIS 2012. LNBIP, vol. 117, pp. 48–59. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30359-3_5
22. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Process-Aware Information Systems: Bridging People and Software Through Process Technology. Wiley, Hoboken (2005)
23. Evermann, J.: Scalable process discovery using map-reduce. IEEE Trans. Serv. Comput. **9**(3), 469–481 (2016)
24. Fahland, D., van der Aalst, W.M.P.: Model repair - aligning process models to reality. Inf. Syst. **47**, 220–243 (2015)
25. Felli, P., Gianola, A., Montali, M., Rivkin, A., Winkler, S.: CoCoMoT: conformance checking of multi-perspective processes via SMT. In: Polyvyanyy, A., Wynn, M.T., Van Looy, A., Reichert, M. (eds.) BPM 2021. LNCS, vol. 12875, pp. 217–234. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85469-0_15
26. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining – adaptive process simplification based on multi-perspective metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_24
27. Janssenswillen, G., Depaire, B.: Towards confirmatory process discovery: making assertions about the underlying system. Bus. Inf. Syst. Eng. **61**, 1–16 (2019)

28. Kabierski, M., Lam Nguyen, H., Grunske, L., Weidlich, M.: Sampling what matters: relevance-guided sampling of event logs. In: Di Ciccio, C., Di Francescomarino, C., Soffer, P. (eds.) 3rd International Conference on Process Mining, ICPM 2021, Eindhoven, Netherlands, 31 - November 4, 2021, pp. 64–71. IEEE (2021)
29. Lee, W.L.J., Verbeek, H.M.W., Munoz-Gama, J., van der Aalst, W.M.P., Sepúlveda, M.: Recomposing conformance: closing the circle on decomposed alignment-based conformance checking in process mining. Inf. Sci. **466**, 55–91 (2018)
30. Leemans, S.: Robust process mining with guarantees. Ph.D. thesis, Eindhoven University of Technology (2017)
31. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery and conformance checking. Softw. Syst. Model. **17**(2), 599–631 (2018)
32. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. Computing, **98**(4), 407–437 (2016)
33. Marquard, M., Shahzad, M., Slaats, T.: Web-based modelling and collaborative simulation of declarative processes. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 209–225. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_15
34. Muñoz-Gama, J., Carmona, J.: A fresh look at precision in process conformance. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 211–226. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15618-2_16
35. Munoz-Gama, J., Carmona, J., van der Aalst, W.M.P.: Single-entry single-exit decomposed conformance checking. Inf. Syst. **46**, 102–122 (2014)
36. Nolle. T.: Process learning for autonomous process anomaly correction. Ph.D. thesis, Technical University of Darmstadt, Germany (2020)
37. Nolle, T., Seeliger, A., Thoma, N., Mühlhäuser, M.: DeepAlign: alignment-based process anomaly correction using recurrent neural networks. In: Dustdar, S., Yu, E., Salinesi, C., Rieu, D., Pant, V. (eds.) CAiSE 2020. LNCS, vol. 12127, pp. 319–333. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49435-3_20
38. Padró, L., Carmona, J.: Approximate computation of alignments of business processes through relaxation labelling. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNCS, vol. 11675, pp. 250–267. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26619-6_17
39. Padró, L., Carmona, J.: Computation of alignments of business processes through relaxation labeling and local optimal search. Inf. Syst. **104**, 101703 (2022)
40. Polyvyanyy, A., Armas-Cervantes, A., Dumas, M., Garcia-Banuelos, L.: On the expressive power of behavioral profiles. Formal Aspects Comput. **28**(4), 597–613 (2016). https://doi.org/10.1007/s00165-016-0372-4
41. Polyvyanyy, A., Moffat, A., Garcia-Banuelos. L.: Bootstrapping generalization of process models discovered from event data (2021)
42. Polyvyanyy, A., Weidlich, M., Conforti, R., La Rosa, M., ter Hofstede, A.H.M.: The 4C spectrum of fundamental behavioral relations for concurrent systems. In: Ciardo, G., Kindler, E. (eds.) PETRI NETS 2014. LNCS, vol. 8489, pp. 210–232. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07734-5_12
43. Reißner, D., Armas-Cervantes, A., Conforti, R., Dumas, M., Fahland, D., La Rosa, M.: Scalable alignment of process models and event logs: an approach based on automata and s-components. Inf. Syst. **94**, 101561 (2020)
44. Reißner, D., Conforti, R., Dumas, M., La Rosa, M., Armas-Cervantes, A.: Scalable conformance checking of business processes. In: OTM CoopIS, Rhodes, pp. 607–627 (2017)

45. Reißner, D., Armas-Cervantes, A., Conforti, R., Dumas, M., Fahland, D., La Rosa, M.: Scalable alignment of process models and event logs: an approach based on automata and s-components. Inf. Syst. **94**, 101561 (2020)
46. Roehm, H., Oehlerking, J., Woehrle, M., Althoff, M.: Model conformance for cyber-physical systems: a survey. Trans. Cyber Phys. Syst. **3**(3), 30:1–30:26 (2019)
47. Rogge-Solti, A., Mans, R.S., van der Aalst, W.M.P., Weske, M.: Improving documentation by repairing event logs. In: Grabis, J., Kirikova, M., Zdravkovic, J., Stirna, J. (eds.) PoEM 2013. LNBIP, vol. 165, pp. 129–144. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41641-5_10
48. Rogge-Solti, A., Senderovich, A., Weidlich, M., Mendling, J., Gal, A.: In log and model we trust? A generalized conformance checking framework. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 179–196. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_11
49. Rozinat, A.: Process mining conformance and extension. Ph.D. thesis, Technische Universiteit Eindhoven (2010)
50. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)
51. Fani Sani, M., Garza Gonzalez, J.J., van Zelst, S.J., van der Aalst, W.M.P.: Conformance checking approximation using simulation. In: van Dongen, B.F., Montali, M., Wynn, M.T. (eds.) 2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, 4–9 October 2020, pp. 105–112. IEEE (2020)
52. Fani Sani, M., van Zelst, S.J., van der Aalst, W.M.P.: Conformance checking approximation using subset selection and edit distance. In: Dustdar, S., Yu, E., Salinesi, C., Rieu, D., Pant, V. (eds.) CAiSE 2020. LNCS, vol. 12127, pp. 234–251. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49435-3_15
53. Senderovich, A., Weidlich, M., Yedidsion, L., Gal, A., Mandelbaum, A., Kadish, S., Bunnell, C.A.: Conformance checking and performance improvement in scheduled processes: a queueing-network perspective. Inf. Syst. **62**, 185–206 (2016)
54. Taghiabadi, E.R., Gromov, V., Fahland, D., van der Aalst, W.M.P.: Compliance checking of data-aware and resource-aware compliance requirements. In: Meersman, R., Panetto, H., Dillon, T., Missikoff, M., Liu, L., Pastor, O., Cuzzocrea, A., Sellis, T. (eds.) OTM 2014. LNCS, vol. 8841, pp. 237–257. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45563-0_14
55. Tax, N., Lu, X., Sidorova, N., Fahland, D., van der Aalst, W.M.P.: The imprecisions of precision measures in process mining. Inf. Process. Lett. **135**, 1–8 (2018)
56. Taymouri, F., Carmona, J.: Model and event log reductions to boost the computation of alignments. In: Ceravolo, P., Guetl, C., Rinderle-Ma, S. (eds.) SIMPDA 2016. LNBIP, vol. 307, pp. 1–21. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74161-1_1
57. Taymouri, F., Carmona, J.: A recursive paradigm for aligning observed behavior of large structured process models. In: 14th International Conference of Business Process Management (BPM), Rio de Janeiro, Brazil, 18–22 September 2016
58. Taymouri, F., Carmona, J.: An evolutionary technique to approximate multiple optimal alignments. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 215–232. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_13
59. Taymouri, F., Carmona, J.: Model and event log reductions to boost the computation of alignments. In: Ceravolo, P., Guetl, C., Rinderle-Ma, S. (eds.) SIMPDA 2016. LNBIP, vol. 307, pp. 1–21. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74161-1_1

60. Taymouri, F., Carmona, J.: Structural computation of alignments of business processes over partial orders. In: 19th International Conference on Application of Concurrency to System Design, ACSD 2019, Aachen, Germany, 23–28 June 2019, pp. 73–81 (2019)
61. Taymouri, F., Carmona, J.: Computing alignments of well-formed process models using local search. ACM Trans. Softw. Eng. Methodol. **29**(3), 15:1–15:41 (2020)
62. Valencia-Parra, Á., Varela-Vaca, Á, J., Teresa Gómez López, M., Carmona, J., Bergenthum, R.: Empowering conformance checking using big data through horizontal decomposition. Inf. Syst. **99**, 101731 (2021)
63. van der Aalst, W.M.P.: Decomposing petri nets for process mining: a generic approach. Distrib. Parallel Databases **31**(4), 471–507 (2013)
64. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Min. Knowl. Discov. **2**(2), 182–192 (2012)
65. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
66. van Dongen, B., Carmona, J., Chatain, T., Taymouri, F.: Aligning modeled and observed behavior: a compromise between computation complexity and quality. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 94–109. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_7
67. van Dongen, B.F.: Efficiently computing alignments. In: Daniel, F., Sheng, Q., Motahari, H. (eds.) BPM 2018. LNBIP, vol. 342, pp. 44–55. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11641-5_4
68. Dongen, B.F.: Efficiently computing alignments. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 197–214. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_12
69. van Dongen, B.F., Carmona, J., Chatain, T.: A unified approach for measuring precision and generalization based on anti-alignments. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 39–56. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_3
70. van Zelst, S.J., Bolt, A., Hassani, M., van Dongen, B.F., van der Aalst, W.M.P.: Online conformance checking: relating event streams to process models using prefix-alignments. Int. J. Data Sci. Anal. **8(**3), 269–284 (2019)
71. vanden Broucke, S.K.L.M., Munoz-Gama, J., Carmona, J., Baesens, B., Vanthienen, J.: Event-based real-time decomposed conformance analysis. In: Proceedings on the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, 27–31 October 2014, pp. 345–363 (2014)
72. vanden Broucke, S.K.L.M., De Weerdt, J., Vanthienen, J., Baesens, B.: Determining process model precision and generalization with weighted artificial negative events. IEEE Trans. Knowl. Data Eng. **26**(8), 1877–1889 (2014)
73. Verbeek, H.M.W., van der Aalst, W.M.P.: Merging alignments for decomposed replay. In: Kordon, F., Moldt, D. (eds.) PETRI NETS 2016. LNCS, vol. 9698, pp. 219–239. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39086-4_14
74. Wang, J., Song, S., Lin, X., Zhu, X., Pei, J.: Cleaning structured event logs: a graph repair approach. In: Gehrke, J., Lehner, W., Shim, K., Cha, S.K., Lohman, G.M. (eds.) 31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, 13–17 April 2015, pp. 30–41. IEEE Computer Society (2015)

75. Weidlich, M., Mendling, J., Weske, M.: Efficient consistency measurement based on behavioral profiles of process models. IEEE Trans. Softw. Eng. **37**(3), 410–429 (2011)
76. Weidlich, M., Polyvyanyy, A., Desai, N., Mendling, J., Weske, M.: Process compliance analysis based on behavioural profiles. Inf. Syst. **36**(7), 1009–1025 (2011)
77. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Causal behavioural profiles - efficient computation, applications, and evaluation. Fundam. Informaticae **113**(3–4), 399–435 (2011)
78. Weidlich, M., Ziekow, H., Mendling, J., Günther, O., Weske, M., Desai, N.: Event-based monitoring of process execution violations. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 182–198. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23059-2_16
79. (Ton) Weijters, A.J.M.M., van der Aalst, W.M.P., Alves De Medeiros, A.K.: Process mining with the heuristics miner-algorithm. Technical Report 166, Technische Universiteit Eindhoven (2006)
80. Zha, H., Wang, J., Wen, L., Wang, C., Sun, J.: A workflow net similarity measure based on transition adjacency relations. Comput. Ind. **61**(5), 463–471 (2010)

Data Preprocessing

Foundations of Process Event Data

Jochen De Weerdt[1](✉) and Moe Thandar Wynn[2]

[1] KU Leuven, Leuven, Belgium
jochen.deweerdt@kuleuven.be
[2] Queensland University of Technology, Brisbane, Australia

Abstract. Process event data is a fundamental building block for process mining as event logs portray the execution trails of business processes from which knowledge and insights can be extracted. In this Chapter, we discuss the core structure of event logs, in particular the three main requirements in the form of the presence of case IDs, activity labels, and timestamps. Moreover, we introduce fundamental concepts of event log processing and preparation, including data sources, extraction, correlation and abstraction techniques. The chapter is concluded with an imperative section on data quality, arguably the most important determinant of process mining project success.

1 Introduction

This chapter is devoted to a core building block of process mining, namely event data or event logs. The particularities of event logs in comparison to traditional attribute-value data sets used for non-process mining data science and analytics applications, make that dedicated analysis techniques become worthwhile. To put it more concretely, classical data science analyses, e.g. learning a decision tree or running a clustering algorithm, when straightforwardly applied to an event log, will not give you workable results. This is because events in an event log, which can be considered as the observations (rows) in our dataset, are related to each other in terms of time and by means of an overarching case dimension, which, when not taken into account via dedicated analysis techniques, results in useless or biased results. In this chapter, we will first explain and exemplify the fundamental structure of event logs. In addition, we will discuss the most common sources from which event logs can be obtained. Furthermore, we will dive into the data preprocessing pipeline, bringing in the perspectives of event extraction, correlation and abstraction. Finally, given the uphill battle in many organizations in terms of data availability and especially data quality, we close the chapter with a discussion of this theme.

2 The Fundamental Structure of Event Logs

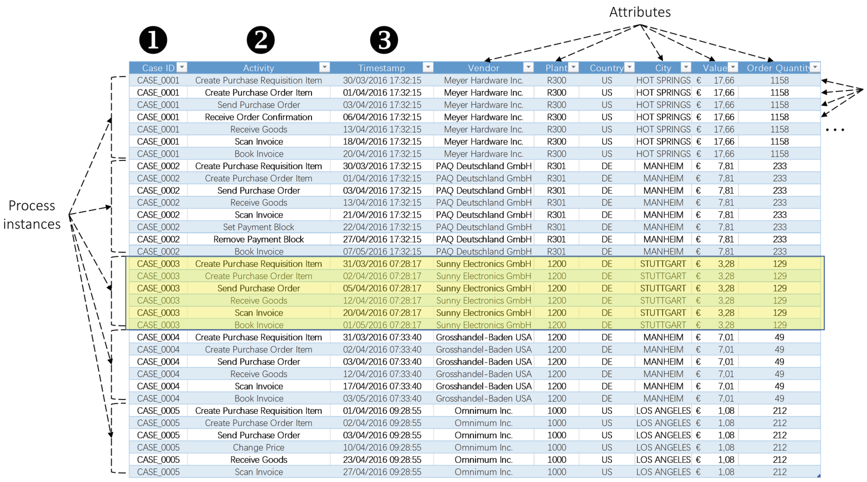
We refer to [3] for the conceptual definition of an event log. Here, we will complement the definition with a more practical view on the essential event log data requirements, an exploration on additional data attributes, an analysis of event types, as well as the link to the XES storage standard.

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 193–211, 2022.
https://doi.org/10.1007/978-3-031-08848-3_6

2.1 Essential Event Log Data Requirements

Figure 1 illustrates an excerpt of an example event log related to a fictitious Purchase-to-Pay (P2P) process. This small excerpt can help to understand the three essential data requirements for event logs to be analysis-ready for process mining technique application. First, each event should be linked to a case or process instance, typically by using a *Case ID*. This is “Requirement 1”. In the simple example of Fig. 1, each case or process instance will refer to one procurement of a product or service by an organization with one of its suppliers. Events will be collected for every process instance and will pertain to activities or steps executed within the different stages of the P2P process (e.g. requisitioning, invoicing, reception of goods, etc.).

We thus argue that the presence of a Case ID is an essential requirement for an event log. However, it should be pointed out that Case IDs are not always straightforwardly available. This problem has been addressed in both process mining literature, as well as in practice, and is often referred to as *event correlation*. This topic is addressed in Sect. 3. There also exists research on the direct application of process mining techniques on event data without Case IDs (e.g. [27]), however, this is a rather niche application. Nevertheless, it is important to point out that, in contrast to static event logs, an increasing number of process mining techniques are developed for streams of events. In such event streams, the notion of a CaseID is often even more complicated.



Case ID	Activity	Timestamp	Vendor	Plant	Country	City	Value	Order Quantit
CASE_0001	Create Purchase Requisition Item	30/03/2016 17:32:15	Meyer Hardware Inc.	R300	US	HOT SPRINGS	€ 17,66	1158
CASE_0001	Create Purchase Order Item	01/04/2016 17:32:15	Meyer Hardware Inc.	R300	US	HOT SPRINGS	€ 17,66	1158
CASE_0001	Send Purchase Order	03/04/2016 17:32:15	Meyer Hardware Inc.	R300	US	HOT SPRINGS	€ 17,66	1158
CASE_0001	Receive Order Confirmation	06/04/2016 17:32:15	Meyer Hardware Inc.	R300	US	HOT SPRINGS	€ 17,66	1158
CASE_0001	Receive Goods	13/04/2016 17:32:15	Meyer Hardware Inc.	R300	US	HOT SPRINGS	€ 17,66	1158
CASE_0001	Scan Invoice	18/04/2016 17:32:15	Meyer Hardware Inc.	R300	US	HOT SPRINGS	€ 17,66	1158
CASE_0001	Book Invoice	20/04/2016 17:32:15	Meyer Hardware Inc.	R300	US	HOT SPRINGS	€ 17,66	1158
CASE_0002	Create Purchase Requisition Item	30/03/2016 17:32:15	PAQ Deutschland GmbH	R301	DE	MANHEIM	€ 7,81	233
CASE_0002	Create Purchase Order Item	01/04/2016 17:32:15	PAQ Deutschland GmbH	R301	DE	MANHEIM	€ 7,81	233
CASE_0002	Send Purchase Order	03/04/2016 17:32:15	PAQ Deutschland GmbH	R301	DE	MANHEIM	€ 7,81	233
CASE_0002	Receive Goods	13/04/2016 17:32:15	PAQ Deutschland GmbH	R301	DE	MANHEIM	€ 7,81	233
CASE_0002	Scan Invoice	21/04/2016 17:32:15	PAQ Deutschland GmbH	R301	DE	MANHEIM	€ 7,81	233
CASE_0002	Set Payment Block	22/04/2016 17:32:15	PAQ Deutschland GmbH	R301	DE	MANHEIM	€ 7,81	233
CASE_0002	Remove Payment Block	27/04/2016 17:32:15	PAQ Deutschland GmbH	R301	DE	MANHEIM	€ 7,81	233
CASE_0002	Book Invoice	07/05/2016 17:32:15	PAQ Deutschland GmbH	R301	DE	MANHEIM	€ 7,81	233
CASE_0003	Create Purchase Requisition Item	31/03/2016 07:28:17	Sunny Electronics GmbH	1200	DE	STUTTGART	€ 3,28	129
CASE_0003	Create Purchase Order Item	02/04/2016 07:28:17	Sunny Electronics GmbH	1200	DE	STUTTGART	€ 3,28	129
CASE_0003	Send Purchase Order	05/04/2016 07:28:17	Sunny Electronics GmbH	1200	DE	STUTTGART	€ 3,28	129
CASE_0003	Receive Goods	12/04/2016 07:28:17	Sunny Electronics GmbH	1200	DE	STUTTGART	€ 3,28	129
CASE_0003	Scan Invoice	20/04/2016 07:28:17	Sunny Electronics GmbH	1200	DE	STUTTGART	€ 3,28	129
CASE_0003	Book Invoice	01/05/2016 07:28:17	Sunny Electronics GmbH	1200	DE	STUTTGART	€ 3,28	129
CASE_0004	Create Purchase Requisition Item	31/03/2016 07:33:40	Grosshandel-Baden USA	1200	DE	MANHEIM	€ 7,01	49
CASE_0004	Create Purchase Order Item	02/04/2016 07:33:40	Grosshandel-Baden USA	1200	DE	MANHEIM	€ 7,01	49
CASE_0004	Send Purchase Order	03/04/2016 07:33:40	Grosshandel-Baden USA	1200	DE	MANHEIM	€ 7,01	49
CASE_0004	Receive Goods	12/04/2016 07:33:40	Grosshandel-Baden USA	1200	DE	MANHEIM	€ 7,01	49
CASE_0004	Scan Invoice	17/04/2016 07:33:40	Grosshandel-Baden USA	1200	DE	MANHEIM	€ 7,01	49
CASE_0004	Book Invoice	03/05/2016 07:33:40	Grosshandel-Baden USA	1200	DE	MANHEIM	€ 7,01	49
CASE_0005	Create Purchase Requisition Item	01/04/2016 09:28:55	Omnimum Inc.	1000	US	LOS ANGELES	€ 1,08	212
CASE_0005	Create Purchase Order Item	02/04/2016 09:28:55	Omnimum Inc.	1000	US	LOS ANGELES	€ 1,08	212
CASE_0005	Send Purchase Order	03/04/2016 09:28:55	Omnimum Inc.	1000	US	LOS ANGELES	€ 1,08	212
CASE_0005	Change Price	10/04/2016 09:28:55	Omnimum Inc.	1000	US	LOS ANGELES	€ 1,08	212
CASE_0005	Receive Goods	23/04/2016 09:28:55	Omnimum Inc.	1000	US	LOS ANGELES	€ 1,08	212
CASE_0005	Scan Invoice	27/04/2016 09:28:55	Omnimum Inc.	1000	US	LOS ANGELES	€ 1,08	212

Fig. 1. Example event log from a fictitious P2P process, illustrating the three essential requirements: presence of a case ID, activity label, and timestamp per event.

The second key requirement (“Requirement 2”) for event log data is the fact that each event should correspond to an activity executed within the process. More specifically, an assumption is made that there exists a restricted set of labels, reflecting the activities in the business process, to which each event is mapped. In Fig. 1, this is shown in the second column. Given that activity labels are simple strings, there is a lot of freedom to tailor the activity label for the right analysis viewpoint. However, oftentimes, natural log data is stored at lower levels of granularity than desired for analysis purposes. Typically, one would prefer that the granularity level of activities is such that they can be understood and interpreted by business experts. Nonetheless, a lot of event data exists for which the granularity level is much lower. In Sect. 3, we discuss the task of bringing lower level events to a better granularity level, which is referred to as *event abstraction*.

Finally, the last requirement ("Requirement 3") entails that there exists an ordering of the events pertaining to a case. As such, each case logically consists of a sequence of events. Most often, this ordering will be derived from a timestamp attribute. However, this is not strictly mandatory, given that the order could also be derived from the order in which events are recorded in a database or table, insofar this order in which events occurred matched with their factual execution order within a process.

It should be pointed out that, while a Case ID, Activity and Timestamp column are essential requirements in order to be able to conduct process mining analyses, their definition might not always be as clear cut as is the case for the illustrative example. For instance, for many real-life datasets, different choices can be made in terms of using one single or multiple columns to create the activity label, and as such provide a different perspective on the process. A similar effect can also occur for Case IDs, where for instance, with an example from a clinical pathway perspective, the use of a patient ID instead of an admission ID as case identifier, can yield a very different analysis.

2.2 Additional Data Attributes

In addition to the mandatory elements of a Case ID, Activity, and Timestamp, event logs will usually contain several or often many additional attributes (columns). In Fig. 1, the event log contains additional attributes including Vendor, Plan, Country, City, Value and Order Quantity. In our example, the values for these attributes remain constant within a single case, and accordingly can be considered as process instance-level attributes. However, this is not mandatory, as attributes can pertain to events or activities, and might be updated throughout the execution of a process instance. For instance, an item number or item type that is recorded when a purchase order item is created is an example of such an event-level attribute.

Additional data attributes can have many purposes, but typically the following three uses are most important. Foremost, these additional data attributes can help to filter cases and events in order to obtain a more focused analysis viewpoint or perform comparative analysis between subsets of process instances. Secondly, these additional data attributes might contain valuable context information, and can therefore be exploited to gain better insights into the process. For instance, a textual comment field in an incident management process could contain essential information regarding the problem at hand, which in turn might impact routing choices, timing, resource allocation, etc. Finally, the availability of additional data attributes, especially information on resources, costs, etc. opens up possibilities for the application of process mining techniques that go beyond process discovery and conformance checking. For instance, organizational mining techniques were developed to focus on resources employed within the process [53]. Moreover, these additional data attributes also play a fundamental role in decision mining [18,47] (see [17]) and predictive process monitoring [19] (see [20]).

2.3 Storing Event Data

Event data is intrinsically simple attribute value data, easily visualized in a two-dimensional table. Nonetheless, unstructured data formats including Excel-files or plain text files, without any form of underlying schema, fail to serve as a proper storage format. This is mainly due to the complex interactions between events, cases, and their attributes. This observation drove the development of the eXtensible Event Stream (XES) standard [1], an IEEE Standards Association-approved language to transport, store, and exchange event data. Its metadata structure is represented in Fig. 2. XES uses the W3C XML Schema definition language, guaranteeing interoperability between various systems. An IEEE XES instance corresponds to a file-based event log or formatted event stream that can be used to transfer event data in a unified manner. In IEEE XES, events are considered as an observed atomic granule of activity. Next to events, IEEE XES specifies the concept of a log, a trace, and an attribute component. Event and/or trace classifiers are used to assign an *identity* to traces and events. The standard does not define a specific set of attributes for events, traces or logs. However, it does allow for *extensions*. An extension can be used to define a set of attributes for events, traces and/or logs. For instance, a common set of attributes can be defined for event logs within a particular application domain. An overview of currently available standard extensions is available on the XES website[1].

2.4 Event Types

To conclude the section on the fundamental structure of event logs, it is important to point to the concept of event types or lifecycle transitions of activities. When sourcing events from many process-aware information systems, events oftentimes relate to the transactional lifecycle that activities undergo. One example of such a transactional lifecycle model is shown in Fig. 3a. This is the transition lifecycle model of the BPMN 2.0 standard[2]. Such a transactional lifecycle model describes the states and state transitions which an activity might take in its execution. Also in IEEE XES, a *lifecycle* extension has been approved, which specifies a default activity lifecycle[3]. This state machine is shown in Fig. 3b.

[1] http://www.xes-standard.org/.

[2] https://www.omg.org/spec/BPMN/2.0/.

[3] http://www.xes-standard.org/.

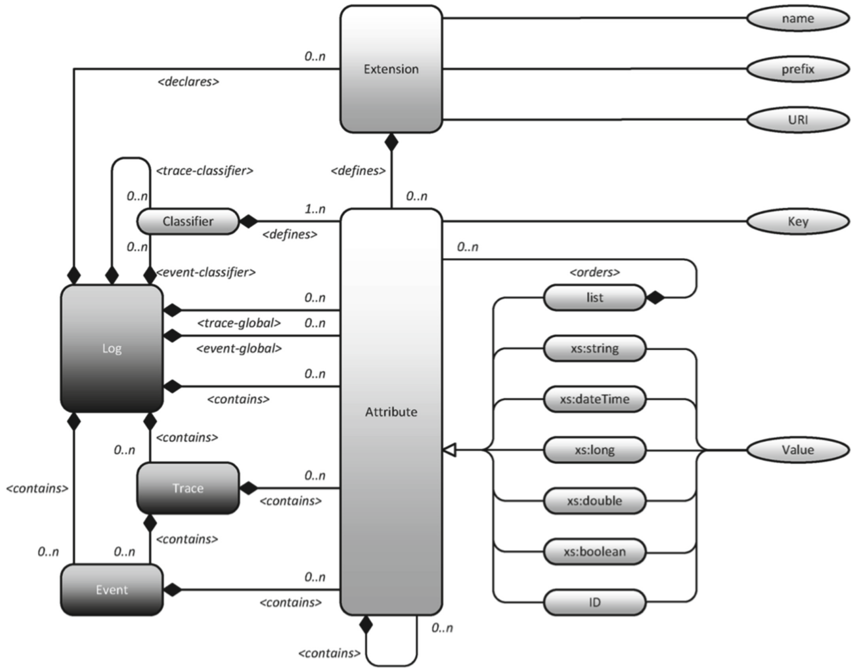
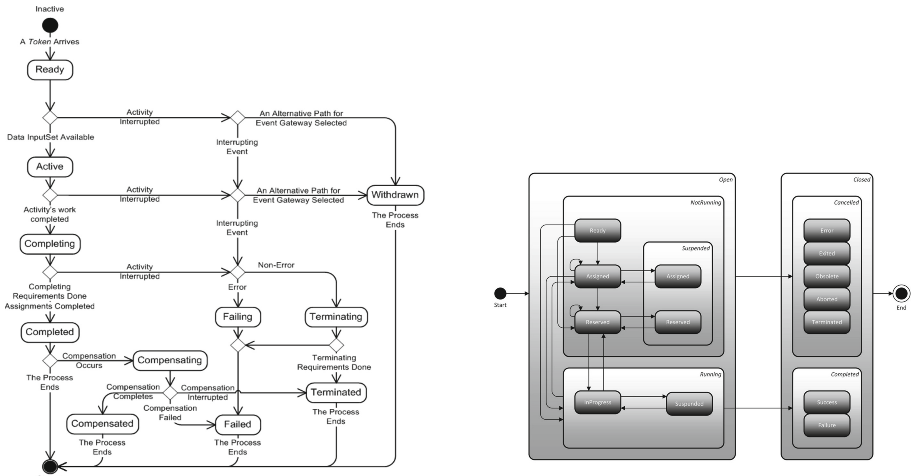


Fig. 2. The IEEE XES metadata structure



(a) The lifecycle of an activity as defined in BPMN 2.0

(b) State machine illustrating the most typical transitions in an activity's lifecycle, according to XES

Fig. 3. Two different activity life cycle models

When retrieving data from process-aware information system, especially from Business Process Management Systems (BPMS) [43], a large collection of event types might be readily available. This is oftentimes not the case in other environments, for instance for web data. In case there are no defined event types, one typically assumes that an event pertaining to the execution of an activity reflects the completion of the activity. In this case, every activity execution is represented by a single event. However, having only a single event per activity execution does not allow to make a distinction between waiting time and execution time of activities. As such, for more fine-grained performance analysis, one would typically prefer two events per activity execution, indicating its start and completion time.

3 Event Log Preprocessing

Data preprocessing is a fundamental part of any data science project. While not as attractive compared to model building or deployment, the preprocessing stage of a project is often most time and effort consuming. Estimates indicate that 80% of resources in typical data science projects is devoted to data preprocessing. One model illustrating the typical data analytics process is depicted in Fig. 4. This model, originally introduced in [25] as the Knowledge Discovery in Databases (KDD) process, reflects the main stages in the execution of a data analytics process. It should be pointed out that this model is an oversimplification of reality, given the frequent and unpredictable iterations that most often occur, rendering the management and completion of a typical data science project usually much more difficult. One notable complexity is the preprocessing of data, usually consisting of data selection, data cleaning, and data transformation.

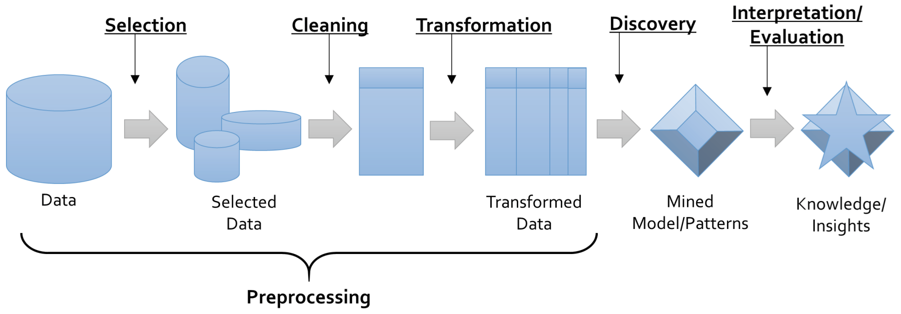


Fig. 4. A representation of the typical stages in a data analytics project [25]

In this part, we want to zoom in on a couple of aspects related to the different stages of a process mining-based analytics project. Most importantly, we want to elaborate on event log data sources, as well as the differences in terms of pipelines between classical data analytics projects and process mining projects.

3.1 Event Log Data Sources

Event data is rapidly becoming an almost untameable beast, given the widespread and drastic increase in availability of such kind of data. In application domains ranging from typical service sector companies including banks and insurers, over manufacturing, to healthcare and education. At system level, we identify the following categorization of most common and important sources for event data:

- **BPMS:** On a scale of most to least process-aware systems, BPMSs most likely rank on top. As such, without exception, event data obtained from these systems is readily available for process mining analysis. Very little or even no data integration is required, and logging is usually executed at the ideal level of granularity.
- **Case management and ticketing sytems:** In line with BPMSs, also case management and ticketing systems natively log timestamped data that is directly useful for process mining. Oftentimes, logs from case management and ticketing systems relate to status changes, so some additional preprocessing might be required to disentangle the true units of work or activity labels.
- **ERP/CRM:** Given their widespread adoption, these enterprise information systems are probably the most important source of event data for modern businesses and organizations. An ERP (Enterprise Resource Planning) system can be seen as a suite of integrated applications for supporting and managing the core business processes. CRM (Customer Relationship Management) systems on the other hand have a dedicated focus on managing all interactions and relationships with customers. By design, ERP systems use shared databases to store relevant business data. As such, and although sometimes a bit more arduous than expected, event log data can be sourced from ERP and CRM systems.
- **Operational databases:** Next to ERP and CRM systems, companies might employ other operational databases supporting their business processes. If these databases have some functionality to store historical data, they can often also serve as a valuable event data source.
- **Project management software:** Applications including popular Hive, Trello, ZOHO, and JIRA support many organizations with managing projects according to a scrum, agile, lean or other fancy project management methodology. When you take an interest into process mining analysis of project management and execution, these systems can provide valuable event data.
- **Data warehouses and data lakes:** Next to operational systems including ERP and CRM, many organizations have a dedicated stack of Business Intelligence (BI) systems and technologies in place. Classical data warehouses are oftentimes a goldmine for process miners. Their hype alternative, allowing for more flexible and unstructured data storage by shifting from schema-on-write to a schema-on-read data management, are referred to as data lakes.
- **Web data:** Website and apps data are another unmistakably important source of event data. From online shopping, gaming, investing, trading, media consumption, to social interaction, online platforms are the main driver of modern B2C business models. With the strong uptake in customer centricity

for business value and competitive advantage creation, customer-centric process mining analysis has strong potential. As such, in addition to CRM data, process mining has a strong interest into event data produced on these online platforms. Please note that, in many cases, including for instance learning environments such as MOOCs, a default standard for web-based platforms to store data is JSON (JavaScript Object Notation).

- **Internet of Things (IoT):** Finally, IoT systems also contain a high potential as source for event data. Sensors and actuators have been deployed widely for all kinds of purposes. Although the granularity gap between typical IoT data (sensor readings) and event data is sometimes challenging to bridge, IoT is becoming a hugely important source of even data in areas such as security, manufacturing, healthcare, and transportation.

It is pointed out that this is not a comprehensive list of all possible event log data sources. In an online survey with 289 participants spanning the roles of practitioners, researchers, software vendors, and end-users, SAP ECC (R/3), SAP S/4 HANA, and Salesforce are selected as the top three most analyzed source systems for process mining analysis [57].

3.2 A Comparison with Classical Analytics Data Preprocessing

While sourcing appropriate data is always the first step in any data preprocessing exercise, it seems reasonable to state that in many situations, analysts could rely on a vast amount of event data sources. This is in line with classical analytics tasks, for which a growth in available data has been observed as well. However, in comparison to classical data preprocessing stages within an analytics process, starker differences exist at the level of cleaning and transforming data.

With respect to data cleaning, where in classical setups, problems including missing values and outliers are a main focus, data cleaning of event logs has received much less scientific and practical attention. A more detailed discussion on data quality for process mining can be found below in Sect. 5. Other differences between a process mining project process and a classical data analytics process are even more notable.

First, at the selection stage, a typical procedure within classical data analytics is to, early-on in the process, divide obtained data into training and test data. Especially when considering predictive analytics, it is of crucial importance to evaluate the true predictive power of learned models by means of independent test data that was not used for training the model. This procedure is rarely seen in process mining, with the exception of some works on predictive process monitoring. One could claim that this is due to the more unsupervised nature of process discovery algorithm, nonetheless, the difference remains striking.

Another essential data preprocessing step for classical data analytics projects relates to transforming the features space such that more valuable features are provided to algorithms for training models. Feature transformation includes techniques such as normalization, grouping and binning. Moreover, advanced feature engineering is also an important but often neglected step to improve model performance. Feature engineering aims at crafting new features based on the original data. The typical data format of event logs, consisting of events pertaining to cases, make that the "rows" in event log are intrinsically correlated. This invalidates the assumption of data being independent and identically distributed (IID). This is a central assumption underpinning about every machine learning technique. However, for process mining, when considering events as the observation level, they are by definition not IID. As such, a large majority of techniques addressing data cleaning and feature transformation including advanced feature engineering, remain purposeless when applied to event data.

When making an assessment of one of the most recently introduced process mining methodologies, i.e. PM^2 [56], four event data preprocessing tasks are defined: (1) creating views, (2) filtering logs, (3) enriching logs, and (4) aggregating events. All these tasks are tailored to the process mining context, and have no immediate corresponding task in a classical data analytics pipeline. For instance, in CRISP-DM [52], data preparation includes selection, cleaning, construction, integration and formatting of data. Several process mining case studies such as the one presented in [6] adapted CRISP-DM to work with healthcare datasets.

In the next Section, we will dive deeper into the problem of event log preparation, which is often extensive and demanding, especially when data for process mining cannot be sourced from process-aware information systems.

4 Event Log Preparation

While possibly not perfectly disjoint, event log preparation often includes three types of techniques: extraction, correlation and abstraction [21]. Figure 5 illustrates the relationship between these types of techniques and fundamental process mining concepts.

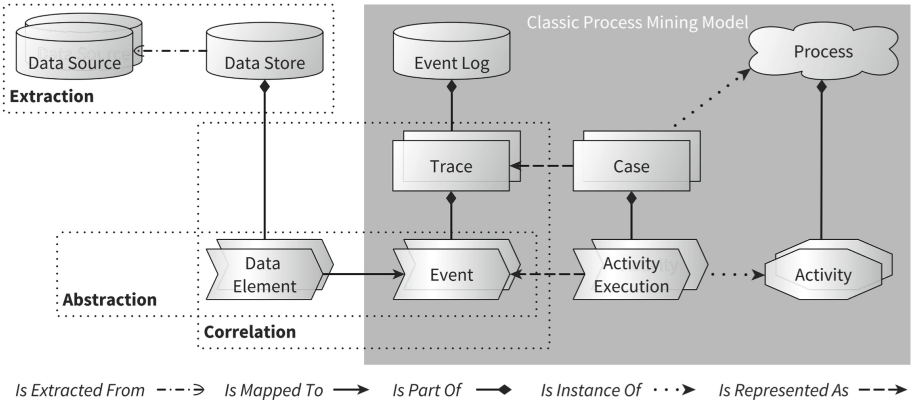


Fig. 5. Event log preparation techniques (extraction, correlation, and abstraction) and their relationship to key process mining concepts [21].

In what follows, we will provide a summary overview of reported tools and techniques for abstraction, correlation and abstraction of event data.

4.1 Extraction of Event Data

Extraction refers to obtaining event data from source systems, most often databases underlying a variety of information systems. Generally, data stored in such databases is not recorded with a process perspective in mind, and therefore will not automatically reflect essential concepts such as events and traces. Accordingly, identification of relevant event data is a primordial challenge. It often requires strong domain knowledge, and despite standardization efforts, often remains prone to ad-hoc solutions.

Two perspectives should be separated when investigating solutions for event data extraction. On the one hand, there is commercial process mining software, where vendors have adopted a clear strategic focus to address the challenges that come with extraction of event logs. Accordingly, a majority of commercial process mining tools comes with software solutions (connectors) that have been developed to allow tapping into all kinds of source systems and databases. Such connectors define how to extract relevant event data from particular source systems and which additional transformations should be applied. As such, these tools promise the holy grail of automating data extraction, a problem addressed in the academic community for over a decade.

One of the first tools stemming from scientific research was the ProM Import Framework [31]. Already in these early days, the idea of an extensible plug-in architecture allowing to develop adapters to hook into a large variety of systems was proposed and partially implemented. With the uptake of XES, XESame was developed as a more flexible successor to the ProM Import Framework. Other researchers have focused on extraction from ERP systems, e.g. the EVS Model Builder [33] and XTract [41], or other operational systems, e.g. Eventifier [46].

Another important stream of research within the realm of event extraction addresses object or artifact centricity. Many source systems, including popular ERP systems, store data at the logical level of objects instead of providing a true process perspective. Oftentimes, assumptions in terms of a desired perspective (definition of case id and activity) are required in order to flatten an object-centered database into a “flat” event log. One noteworthy scientific initiative in this context is ontology-based data access (ODBA) for event log extraction [13, 14]. The approach is based on an ontological view of the domain of interest and linking it as such to a database schema and has been implemented in the Onprom tool. Finally, the recently introduced OCEL standard[4] is another relevant piece of work, putting forward a general standard to interchange object-centric event data with multiple case notions.

The XES survey also uncovered the top tools that are currently being used by the process mining community for the preparing of event logs [57]. There is also ongoing work by the IEEE Task force on reinventing the IEEE XES standard

[4] http://ocel-standard.org/.

to address several identified data related challenges in the XES survey [57], in particular, to capture the semantics of event data and to support complex data structures.

4.2 Correlation of Event Data

Mapping event data extracted from source systems and databases to cases (instances of the business process under investigation) is denoted as correlation. In cases where event data is obtained but Case IDs are missing, a non-trivial process can be started to automatically or semi-automatically generate Case IDs. In a scientific context, several solutions have been proposed, most of them being focused on using additional event data attributes [12,15,42,44,48], sometimes aided by a conceptual model [9,40] or even a process model [8,37].

In practical situations, the problem of correlating event data is probably more related to a variety of non-integrated data sources, which all capture or support part of a business process. As such, an integration of these different sources should be achieved. Hereto, especially when an organizational data warehousing architecture is present, Extract-Transform-Load (ETL) processing would be a default technology to resort to. ETL tools are perfectly equipped to derive and deploy matching schemes to integrate data from non-integrated data sources. Nevertheless, an ETL-approach leading to a data consolidation integration pattern is not the sole option. Increasingly, companies start to focus on the introduction of data virtualization layers in order to realize a more federation-oriented data integration. Data federation can prevent the creation of yet another duplicated database or data store, but instead provides flexible querying and analysis tools for information from multiple source systems as if all data resides within a single integrated database.

4.3 Abstraction of Event Data

Next to extraction and correlation, abstraction is considered as the third prong of the process mining event data preparation trident. In many real-world scenarios, event data is stored at much more fine-grained granularity levels compared to a business-understandable process activity level. As such, abstraction techniques can be considered as mapping techniques that can translate one or more lower-level events into higher-level events pertaining to business process activities. For a detailed taxonomy of event abstraction, we refer the interested reader to [59].

IoT. One particular field of application in which event abstraction is becoming a crucial factor for success is IoT business processes [34]. In IoT, a wide variety of sensors and actuators record contextual observations of a physical environment. These sensor readings or measurements give rise to low-level events, which are intrinsically useful to derive activity-level events from. For instance, in [51], a technique for mapping location-based sensor data to process activities was proposed using so-called *interactions*. Another prominent work in this area is

[23], which relies on clustering of segmented continuous sensor data to derive higher-level activities.

Clustering. Given that event abstraction is a largely unsupervised learning problem in most cases (i.e. unless domain knowledge is used, there is no natural target available), a pretty intuitive way to map lower-level events to coarse-grained events is using clustering. The earliest proposed event abstraction techniques took this perspective, i.e. by clustering sets or sequences of lower-level events, abstraction into higher-level events can be obtained. For instance, in [32], coherent subsequences of events are learned via trace segmentation to create coarse-granular events. Also in [29,45], clustering techniques have been put forward for event abstraction.

Pattern-Based Approaches. Another frequently used paradigm to perform abstraction is pattern matching. The work by Bose and van der Aalst [11] can be considered as origination of pattern-based abstraction. Repeated local subsequence patterns, e.g. maximal repeats or tandem arrays are discovered and used as a basis for the creation of coarse-granular activities. In [38], a more advanced technique is proposed based on mining local process models.

Supervised Learning. Despite the unsupervised nature of the problem, abstraction techniques will often leverage additional domain knowledge, a process model, or other information to turn the problem into a more supervised approach. The technique in [7] relies on a predefined process model, an approach also followed by [26]. Other approaches expect supervision in the form of a set of annotated traces in which fine-granular event sets are matched with a higher-level activity [55], or in the form of timing information, e.g. for sessionization as in [36]. Another example of event abstraction from the healthcare domains was presented in [35], in which they rely on multi-level semantic abstraction using a combination of ontologies and dynamic programming. Also active learning is a promising pathway, bringing the expert in the learning loop to solve the supervision problem.

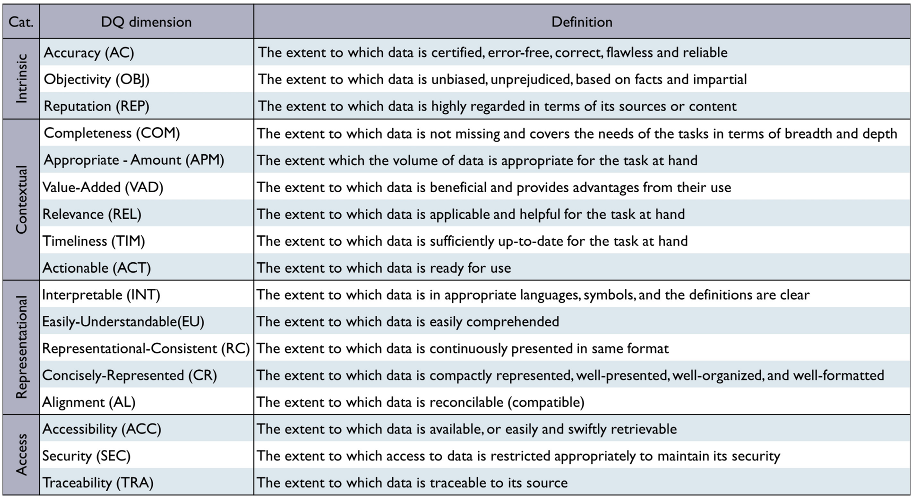
5 Process Mining Data Quality Considerations

"Garbage in, garbage out." It is by far the most mentioned quote in data science and far beyond. But it appears that the more the quote is used, the more relevant it becomes. In process mining, while the problem has been acknowledged in both scientific literature and in practice [57], there is still a need for further research into the development of a comprehensive framework to address the problem of bad quality data leading to incorrect analysis results [58]. We also need to have a better understanding of the root-causes of such data quality issues [5,24].

5.1 Data Quality Dimensions

Some typical data quality dimensions are shown in Fig. 6 [39]. Although there are some similarities between the data quality challenges encountered for event data and traditional data sets for data mining, a key distinguishing factor is our need for detailed correlated event data in their raw form, to capture the true behavior of processes.

In [10], four broad data quality dimensions are identified for event logs: missing data, incorrect data, imprecise data and irrelevant data. Among these four dimensions, incorrect data (where a data item is not recorded correctly) and imprecise data (where a recorded value is too coarse to be useful) for key event attributes such as activity labels and timestamps could have significant consequences for all forms of process mining techniques.



Cat.	DQ dimension	Definition
Intrinsic	Accuracy (AC)	The extent to which data is certified, error-free, correct, flawless and reliable
	Objectivity (OBJ)	The extent to which data is unbiased, unprejudiced, based on facts and impartial
	Reputation (REP)	The extent to which data is highly regarded in terms of its sources or content
Contextual	Completeness (COM)	The extent to which data is not missing and covers the needs of the tasks in terms of breadth and depth
	Appropriate - Amount (APM)	The extent which the volume of data is appropriate for the task at hand
	Value-Added (VAD)	The extent to which data is beneficial and provides advantages from their use
	Relevance (REL)	The extent to which data is applicable and helpful for the task at hand
	Timeliness (TIM)	The extent to which data is sufficiently up-to-date for the task at hand
	Actionable (ACT)	The extent to which data is ready for use
Representational	Interpretable (INT)	The extent to which data is in appropriate languages, symbols, and the definitions are clear
	Easily-Understandable(EU)	The extent to which data is easily comprehended
	Representational-Consistent (RC)	The extent to which data is continuously presented in same format
	Concisely-Represented (CR)	The extent to which data is compactly represented, well-presented, well-organized, and well-formatted
	Alignment (AL)	The extent to which data is reconcilable (compatible)
Access	Accessibility (ACC)	The extent to which data is available, or easily and swiftly retrievable
	Security (SEC)	The extent to which access to data is restricted appropriately to maintain its security
	Traceability (TRA)	The extent to which data is traceable to its source

Fig. 6. An overview of some of the most common data quality dimensions, taken from [39].

5.2 Detection and Repair

The process mining manifesto [2] categorizes the quality of event data from one star to five stars; while most real-life event logs are found to be in-between these two extremes of the scale with many quality issues [58]. Some advocate for repairing or fixing the erroneous data, while others argue that the data should be left alone as it is meant to reflect reality. Regardless of your personal view, it is unavoidable that these data quality issues are dealt with in one way or another. As a process mining professional, it is imperative that we measure the quality of an event log respective to the type of process mining analysis being considered [58]. The data pre-processing task is recognized to be one of the most time-consuming aspects of a process mining study with many spending 60–80% of their efforts while some spending up to 90% of their total efforts on this step [57].

Suriadi et al. [54] identified eleven event log imperfection patterns based on their experience with over 20 Australian industry data sets. The eleven patterns include form-based event capture, inadvertent time travel, unanchored event, scattered event, elusive case, scattered case, collateral event, polluted label, distorted label, synonymous labels and homonymous labels. These event log patterns have been used as a starting point for detection and repair of quality issues in event logs.

There is a growing body of work focusing on the detection and repair of data quality issues associated with activity labels, timestamps, and event orderings. In [49], crowdsourcing and gamification approaches are being proposed to solicit domain expert knowledge for the detection and repair of activity labels while [50] proposes an automated context-aware approach to detecting synonymous and polluted activity labels in an event log. In [28], the authors described a framework to detect timestamp quality issues in an event log and proposed measures to quantify the extent of these data quality issues as a way to measure the quality of an overall event log. In [16], an approach to automatically repairing same-timestamp errors in an event log is presented. In [22], an interactive approach to detect and repair event order imperfections in an event log is presented.

5.3 Quality-Informed Process Mining

Although data quality issues are well-acknowledged in the process mining community by now, most of the existing process mining algorithms do not explicitly take the potential presence of data quality issues. A notable exception is the removal of infrequent behaviors or noises from discovered process models. The algorithms also typically treat an event log as the "whole truth" without considering the potential effects of data-preprocessing on the reliability of the results [58]. This could lead to misleading or inaccurate conclusions about the process under investigation. In [30], the authors proposed a range of quality annotations at event, trace and log levels to keep track of the data quality issues founded in an event log and also to record the extent of repairs are made to the event log as a result. Such metadata about data quality can assist in undertaking quality-informed process mining. One such algorithm is presented as the 'Quality-Informed visual Miner'plug-in' which demonstrates the use of these data quality annotations for conformance checking and performance analysis purposes.

Alternatively, it is possible to determine whether certain data attributes are of high-quality (i.e., fit-for-purpose) before incorporating them into an event log and then into the process mining analysis. In the Process Mining in Practice book[5], checklists are provided to detect a range of data quality issues and suggestions are provided on how to potentially correct them. The quality issues covered

[5] https://fluxicon.com/book/.

include formatting errors, missing data (event, attribute values, case IDs, activities, timestamps, attribute history, timestamps for activity repetition) as well as zero timestamps, wrong timestamps, same timestamps for multiple activities and different timestamp granularity. In [4], a data-quality informed approach is proposed where data attributes from a relational database are evaluated on their quality across a range of data quality measures before generating an event log.

References

1. IEEE Standard for eXtensible Event Stream (XES) for achieving interoperability in event logs and event streams. IEEE Std 1849–2016, pp. 1–50 (2016). https://doi.org/10.1109/IEEESTD.2016.7740858
2. van der Aalst, W., et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28108-2_19
3. Aalst, W.: Process mining: a 360 degrees overview. In: van der Aalst,W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 3–34. Springer, Cham (2022)
4. Andrews, R., van Dun, C.G.J., Wynn, M.T., Kratsch, W., Röglinger, M., ter Hofstede, A.H.M.: Quality-informed semi-automated event log generation for process mining. Decis. Support Syst. **132**, 113265 (2020). https://doi.org/10.1016/j.dss.2020.113265
5. Andrews, R., Emamjome, F., ter Hofstede, A.H.M., Reijers, H.A.: An expert lens on data quality in process mining. In: van Dongen, B.F., Montali, M., Wynn, M.T. (eds.) 2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, 4–9 October 2020, pp. 49–56. IEEE (2020). https://doi.org/10.1109/ICPM49681.2020.00018
6. Andrews, R., Wynn, M.T., Vallmuur, K., Ter Hofstede, A.H., Bosley, E., Elcock, M., Rashford, S.: Leveraging data quality to better prepare for process mining: an approach illustrated through analysing road trauma pre-hospital retrieval and transport processes in Queensland. Int. J. Environ. Res. Public Health **16**(7), 1138 (2019)
7. Baier, T., Mendling, J., Weske, M.: Bridging abstraction layers in process mining. Inf. Syst. **46**, 123–139 (2014)
8. Bayomie, D., Helal, I.M.A., Awad, A., Ezat, E., ElBastawissi, A.: Deducing case ids for unlabeled event logs. In: Reichert, M., Reijers, H.A. (eds.) BPM 2015. LNBIP, vol. 256, pp. 242–254. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42887-1_20
9. Beheshti, S.-M.-R., Benatallah, B., Motahari-Nezhad, H.R., Sakr, S.: A query language for analyzing business processes execution. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 281–297. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23059-2_22
10. Bose, J.C., Mans, R., van der Aalst, W.M.P.: Wanna improve process mining results - it's high time we consider data quality issues seriously. In: IEEE Symposium on Computational Intelligence and Data Mining. pp. 127–134. IEEE (2013). https://doi.org/10.1109/CIDM.2013.6597227
11. Jagadeesh Chandra Bose, R.P., van der Aalst, W.M.P.: Abstractions in process mining: a taxonomy of patterns. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 159–175. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03848-8_12

12. 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)
13. Calvanese, D., Kalayci, T.E., Montali, M., Tinella, S.: Ontology-based data access for extracting event logs from legacy data: the onprom tool and methodology. In: Abramowicz, W. (ed.) BIS 2017. LNBIP, vol. 288, pp. 220–236. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59336-4_16
14. Calvanese, D., Montali, M., Syamsiyah, A., van der Aalst, W.M.P.: Ontology-driven extraction of event logs from relational databases. In: Reichert, M., Reijers, H.A. (eds.) BPM 2015. LNBIP, vol. 256, pp. 140–153. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42887-1_12
15. Cheng, L., Van Dongen, B.F., Van Der Aalst, W.M.: Efficient event correlation over distributed systems. In: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 1–10. IEEE (2017)
16. Conforti, R., Rosa, M.L., ter Hofstede, A.H.M., Augusto, A.: Automatic repair of same-timestamp errors in business process event logs. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) Business Process Management - 18th International Conference, BPM 2020, Seville, Spain, September 13–18, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12168, pp. 327–345. Springer (2020). https://doi.org/10.1007/978-3-030-58666-9_19
17. de Leoni, M.: Foundations of Process Enhancement. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 243–273. Springer, Cham (2022)
18. De Smedt, J., Hasić, F., vanden Broucke, S.K., Vanthienen, J.: Holistic discovery of decision models from process execution data. Knowl.-Based Syst. **183**, 104866 (2019)
19. Di Francescomarino, C., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. IEEE Trans. Serv. Comput. **12**(6), 896–909 (2016)
20. Di Francescomarino, C., Ghidini, C.: Predictive process monitoring. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 320–346. Springer, Cham (2022)
21. Diba, K., Batoulis, K., Weidlich, M., Weske, M.: Extraction, correlation, and abstraction of event data for process mining. WIREs Data Mining Knowl. Discov. **10**(3), e1346 (2020). https://doi.org/10.1002/widm.1346
22. Dixit, P.M., et al.: Detection and interactive repair of event ordering imperfection in process logs. In: Krogstie, J., Reijers, H.A. (eds.) Advanced Information Systems Engineering - 30th International Conference, CAiSE 2018, Tallinn, Estonia, 11–15 June 2018, LNCS, vol. 10816, pp. 274–290. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-91563-0_17
23. van Eck, M.L., Sidorova, N., van der Aalst, W.M.: Enabling process mining on sensor data from smart products. In: 2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS), pp. 1–12. IEEE (2016)
24. Emamjome, F., Andrews, R., ter Hofstede, A.H.M., Reijers, H.A.: Signpost - a semiotics-based process mining methodology. In: Rowe, F., et al. (eds.) 28th European Conference on Information Systems - Liberty, Equality, and Fraternity in a Digitizing World, ECIS 2020, Marrakech, Morocco, 15–17 June 2020 (2020), https://aisel.aisnet.org/ecis2020_rip/50
25. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: From data mining to knowledge discovery in databases. AI Mag. **17**(3), 37 (1996)
26. Fazzinga, B., Flesca, S., Furfaro, F., Masciari, E., Pontieri, L.: Efficiently interpreting traces of low level events in business process logs. Inf. Syst. **73**, 1–24 (2018)

27. Ferreira, D.R., Gillblad, D.: Discovering process models from unlabelled event logs. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 143–158. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03848-8_11
28. Fischer, D.A., Goel, K., Andrews, R., van Dun, C.G.J., Wynn, M.T., Röglinger, M.: Enhancing event log quality: detecting and quantifying timestamp imperfections. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNCS, vol. 12168, pp. 309–326. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58666-9_18
29. Folino, F., Guarascio, M., Pontieri, L.: Mining multi-variant process models from low-level logs. In: Abramowicz, W. (ed.) BIS 2015. LNBIP, vol. 208, pp. 165–177. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19027-3_14
30. Goel, K., Leemans, S.J., Martin, N., Wynn, M.T.: Quality-informed process mining: a case for standardised data quality annotations. ACM Trans. Knowl. Discov. Data **16**, 1–47 (2022)
31. Günther, C.W., van der Aalst, W.M.: Mining activity clusters from low-level event logs. Beta, Research School for Operations Management and Logistics (2006)
32. Günther, C.W., Rozinat, A., van der Aalst, W.M.P.: Activity mining by global trace segmentation. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 128–139. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12186-9_13
33. Ingvaldsen, J.E., Gulla, J.A.: Preprocessing support for large scale process mining of SAP transactions. In: ter Hofstede, A., Benatallah, B., Paik, H.-Y. (eds.) BPM 2007. LNCS, vol. 4928, pp. 30–41. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78238-4_5
34. Janiesch, C., et al.: The internet of things meets business process management: a manifesto. IEEE Syst. Man Cybern. Mag. **6**(4), 34–44 (2020). https://doi.org/10.1109/MSMC.2020.3003135
35. Leonardi, G., Striani, M., Quaglini, S., Cavallini, A., Montani, S.: Leveraging semantic labels for multi-level abstraction in medical process mining and trace comparison. J. Biomed. Inform. **83**, 10–24 (2018)
36. de Leoni, M., Dündar, S.: Event-log abstraction using batch session identification and clustering. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing, pp. 36–44 (2020)
37. Mannhardt, F., de Leoni, M., Reijers, H.A.: Extending process logs with events from supplementary sources. In: Fournier, F., Mendling, J. (eds.) BPM 2014. LNBIP, vol. 202, pp. 235–247. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15895-2_21
38. Mannhardt, F., Tax, N.: Unsupervised event abstraction using pattern abstraction and local process models. arXiv preprint arXiv:1704.03520 (2017)
39. Moges, H.T., Dejaeger, K., Lemahieu, W., Baesens, B.: A multidimensional analysis of data quality for credit risk management: new insights and challenges. Inf. Manag. **50**(1), 43–58 (2013)
40. Motahari-Nezhad, H.R., Saint-Paul, R., Casati, F., Benatallah, B.: Event correlation for process discovery from web service interaction logs. VLDB J. **20**(3), 417–444 (2011)
41. Nooijen, E.H.J., van Dongen, B.F., Fahland, D.: Automatic discovery of data-centric and artifact-centric processes. In: La Rosa, M., Soffer, P. (eds.) BPM 2012. LNBIP, vol. 132, pp. 316–327. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36285-9_36

42. Pérez-Castillo, R., Weber, B., de Guzmán, I.G.-R., Piattini, M., Pinggera, J.: Assessing event correlation in non-process-aware information systems. Softw. Syst. Model. **13**(3), 1117–1139 (2012). https://doi.org/10.1007/s10270-012-0285-5
43. Pourmirza, S., Peters, S., Dijkman, R., Grefen, P.: BPMS-RA: a novel reference architecture for business process management systems. ACM Trans. Internet Technol. **19**(1), 1–23 (2019)
44. Reguieg, H., Benatallah, B., Nezhad, H.R.M., Toumani, F.: Event correlation analytics: scaling process mining using Mapreduce-aware event correlation discovery techniques. IEEE Trans. Serv. Comput. **8**(6), 847–860 (2015)
45. Rehse, J.-R., Fettke, P.: Clustering business process activities for identifying reference model components. In: Daniel, F., Sheng, Q.Z., Motahari, H. (eds.) BPM 2018. LNBIP, vol. 342, pp. 5–17. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11641-5_1
46. Rodrıguez, C., Engel, R., Kostoska, G., Daniel, F., Casati, F., Aimar, M.: Eventifier: extracting process execution logs from operational databases. Proc. Demonstr. Track BPM **940**, 17–22 (2012)
47. Rozinat, A., van der Aalst, W.M.P.: Decision mining in ProM. In: Dustdar, S., Fiadeiro, J., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 420–425. Springer, Heidelberg (2006). https://doi.org/10.1007/11841760_33
48. Rozsnyai, S., Slominski, A., Lakshmanan, G.T.: Discovering event correlation rules for semi-structured business processes. In: Proceedings of the 5th ACM International Conference on Distributed Event-Based System, pp. 75–86 (2011)
49. Sadeghianasl, S., ter Hofstede, A.H.M., Suriadi, S., Turkay, S.: Collaborative and interactive detection and repair of activity labels in process event logs. In: van Dongen, B.F., Montali, M., Wynn, M.T. (eds.) 2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, 4–9 October 2020, pp. 41–48. IEEE (2020). https://doi.org/10.1109/ICPM49681.2020.00017
50. Sadeghianasl, S., ter Hofstede, A.H.M., Wynn, M.T., Suriadi, S.: A contextual approach to detecting synonymous and polluted activity labels in process event logs. In: Panetto, H., Debruyne, C., Hepp, M., Lewis, D., Ardagna, C.A., Meersman, R. (eds.) On the Move to Meaningful Internet Systems: OTM 2019 Conferences - Confederated International Conferences: CoopIS, ODBASE, C&TC 2019, Rhodes, Greece, 21–25 October 2019, LNCS, vol. 11877, pp. 76–94. Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-33246-4_5
51. Senderovich, A., Rogge-Solti, A., Gal, A., Mendling, J., Mandelbaum, A.: The ROAD from sensor data to process instances via interaction mining. In: Nurcan, S., Soffer, P., Bajec, M., Eder, J. (eds.) CAiSE 2016. LNCS, vol. 9694, pp. 257–273. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39696-5_16
52. Shearer, C.: The CRISP-DM model: the new blueprint for data mining. J. Data Warehousing **5**(4), 13–22 (2000)
53. Song, M., Van der Aalst, W.M.: Towards comprehensive support for organizational mining. Decisi. Support Syst. **46**(1), 300–317 (2008)
54. Suriadi, S., Andrews, R., ter Hofstede, A.H.M., Wynn, M.T.: Event log imperfection patterns for process mining: towards a systematic approach to cleaning event logs. Inf. Syst. **64**, 132–150 (2017). https://doi.org/10.1016/j.is.2016.07.011
55. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.: Mining process model descriptions of daily life through event abstraction. In: Bi, Y., Kapoor, S., Bhatia, R. (eds.) IntelliSys 2016. SCI, vol. 751, pp. 83–104. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-69266-1_5

56. van Eck, M.L., Lu, X., Leemans, S.J.J., van der Aalst, W.M.P.: PM^2: a process mining project methodology. In: Zdravkovic, J., Kirikova, M., Johannesson, P. (eds.) CAiSE 2015. LNCS, vol. 9097, pp. 297–313. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19069-3_19

57. Wynn, M.T., et al.: Rethinking the input for process mining: Insights from the XES survey and workshop. In: International Conference on Process Mining: Workshop Proceedings. LNBIP, Springer, Cham (2021). https://doi.org/10.1007/978-3-030-98581-3_1

58. Wynn, M.T., Sadiq, S.: Responsible process mining - a data quality perspective. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNCS, vol. 11675, pp. 10–15. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26619-6_2

59. van Zelst, S.J., Mannhardt, F., de Leoni, M., Koschmider, A.: Event abstraction in process mining: literature review and taxonomy. Granular Comput. **6**, 719–736 (2020)

A Practitioner's View on Process Mining Adoption, Event Log Engineering and Data Challenges

Rafael Accorsi[1](✉) and Julian Lebherz[2]

[1] Accenture Switzerland, Zurich, Switzerland
rafael.accorsi@accenture.com
[2] A.P. Møller-Mærsk, Copenhagen, Denmark
julian.lebherz@maersk.com

Abstract. Process mining is, today, an essential analytical instrument for data-driven process improvement and steering. While practical literature on how to derive value from process mining exists, less attention haas been paid to how it is being used in different industries, the effort involved in creating an event log and what are the best practices in doing so. Taking a practitioner's view on process mining, we report on process mining adoption and illustrate the challenges of log contruction by means of the order to cash (i.e. sales) process in an SAP system. By doing so, we collect a set of best practices regarding the data selection, extraction, transformation and data model engineering, which proved themselves handy in large-scale process mining projects.

Keywords: Process mining adoption · Event log engineering · SAP · Order to cash

1 Introduction

Process mining is, today, an essential analytical instrument for data-driven process improvement and steering [8,10,21]. It helps to understand how a specific process contributes to the whole value chain, to identify different types of operational debts and to quantify improvement opportunities and, eventually, to measure the impact of transformation projects. Put another way, it is the instrument by means of which the business process management (BPM) lifecycle, as in [11, p. 21], can be effectively brought to life.

However, it was not always this way. Considering the state of the Process Mining discipline as of 2013 [2], the majority of work was still very academia-focused. Use-cases and pilots ran within research projects or by pioneering process mining technology providers, which at that time were spin-offs founded by PhD researchers in the area, substantiated the power of process mining. The practical evidence for the suitability of process mining as a scalable instrument for process improvement identification was missing though.

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 212–240, 2022.
https://doi.org/10.1007/978-3-031-08848-3_7

There were two main reasons for this. The first reason was the lack of market (and methodology) maturity. In fact, stakeholders could not clearly distinguish between process mining and business intelligence, and providers/consultants could not clearly articulate (and/or substantiate) its advantages. The second reason was the fact that process mining, as well as any other data analytical instrument, requires a specific data model. This is, for process mining, an *event log*, of which the assembly requires a wide range of skills beyond pure data staging and aggregation. Experience in pulling together an event log for complex processes and hetorogeneous systems was lacking.

Put another way, while "academic" process mining work mostly starts with a given log L, "practical" process mining work starts with a set of systems (or tables) and aims at creating the log file L for subsequent analysis. Admittedly, the latter is easier said than done. Depending on the complexity of the source data model and process to be discovered, up to 80% of a project timespan is used for data preprocessing and log creation, leaving 20% for the real process analytical work [9]. While reviewing the existing literature, we have seen a focus on use-cases [14,23,25], on general approaches to (and techniques for) process analytics [5] and strategies and frameworks for creating event logs for process mining [4,17]. Recently, also data quality is receiving more attention [3]. However, we could not find previous work addressing all these elements *and* a hands-on data preprocessing example and corresponding best-practices.

Given this scenario and our practioner's view, the goal of this chapter is threefold:

1. Report on the process mining adoption in different industries, as well as on the drivers for process mining usage. We will illustrate different application scenarios and drivers with practical cases.
2. Elaborate on a real-world example focusing on the event log construction for the Order to Cash process (OTC) as seen on an SAP system.
3. Summarize the best-practices and the experience we have acquired by conducting process mining projects.

Below, we will explicitly take up a practical view of process mining. We thus refrain from formalizations and will introduce the necessary technical concepts – especially in the context of SAP – in an on-demand basis including only the necessary aspects. While focusing on SAP for the hands-on example, the methodology we elaborate on can be equally applied to other processes within SAP, or other ERP systems, such as Oracle, Navision and Salesforce. It is also agnostic to any data transformation approach and platform, and process mining technology, thereby decoupling data transformation from the specific analytical tool one intends to employ.

By focusing on data preprocessing, we deliberately leave out various other – equally relevant – phases of a process mining project. See [27] for a process mining project methodology. For example, although we explain the different angles that make out a process mining project scope in Sect. 4.1, we will not cover the *scoping phase* in detail (e.g. deciding which process or legal entities to be analyzed). We also skip the *data maturity assessment phase*, whose goal is to ensure that the system's data provides a basis for process mining. This is typically required for less known, highly customized or legacy systems, not as much for standard ERP systems and their common satellite applications. We also do not cover *analytical and improvement phases* with methods and methodologies, e.g. to derive insights from process mining and calculate a business case for change. The improvement perspective is extensively covered in [26].

The reminder of this paper is laid out as follows. Section 2 reports on the process mining adoption in different industries and drivers for process mining. Section 3 introduces the SAP O2C process and corresponding data foundation. Section 4 elaborates on how to construct a simple log file for SAP O2C. It does so by cutting through the complexities of data extraction, transformation and data model engineering in a general manner, and on the specific context of SAP O2C. Section 5 summarizes the best-practices in creating an event log. Section 6 takes stock and provides an outlook on the upcoming challenges for data preprocessing.

2 Process Mining Adoption

Process Mining is widely used in a multitude of industries and businesses to create transparency on the key processes. This section firstly provides an overview on where process mining is being used and, subsequently, elaborates on the drivers for firms to deploy process mining as a basis for process understanding, monitoring and improvement. Although we illustrate, by means of real-world cases, how process mining has contributed to processes improvements in those industries, this section will not deep-dive into the specific case studies. For this, we refer to [14], a database with example process mining applications, and to [23], a book compiling a series of industry use-cases for process mining.

2.1 Business Usage

We have seen Process Mining being used in several industries and processes. Still, their adoption focus differs depending on the underlying industry type and its characteristics. To better differentiate industry adoption in the different industry segments and map the corresponding processes to the industries, we split businesses in three types, namely (a) “Financial Products” (e.g. banks and insurance companies); (b) “Industrial Products” (e.g. pharmaceuticals and manufacturing); and (c) “Services” (e.g. telecommunication, healthcare, retail and government).

Overall, Financial and Industrial Products are, to-date, the segments with the highest process mining penetration [10]. That is not to say that process mining is not being successfully adopted in Services: healthcare [19, 24], telecommunication providers [23, Chap. 13 and 20] and municipalities [15] already today highly profit from process mining. However, according to technology providers and market research reports [12], they make around 15% of the installed process mining base. Below we provide examples of how process mining is being adopted in the main industry segments, focusing on the driving factors in Sect. 2.2.

Financial Products. These are predominantly banks, e.g. retail, corporate and investment banking, and different types of insurance companies, e.g. health, life, composite and reinsurance. In banking, we have observed the focus on two processes: (a) loan and mortage services and (b) account opening, in particular the KYC process (know-your-client), closely related to the anti-money laundry prevention mechanisms. Focusing on the former, the main focus is on unleashing operational efficiency by means of identifying automation potentials or redesigning the process completely. For example, we have applied Process Mining to assess the loan process of a large bank based out of the Benelux region. In doing so, we have understood that around 70% of the applications were rejected (by the bank) or canceled (by the applicant), which is well-above the industry benchmark for this type of process and region. More importantly, rejections and cancellation happened at the activity "Final Application Check", which was the penultimate process step before completion. Put another way, the applications ran (at least) ten process steps, including an "Initial Application Check" (second process step), to be rejected or withdrawn at nearly the end of the process. This insight has paved the way to reengineer the process by creating a more thorough initial application check and eventually reducing effort by 19 full-time equivalents (FTEs) per year.

Moving on to insurance – irrespective of its kind –, the focus is on two areas: first, claim management and processing, and second, back and front office functions, such as master data changes and lead management. Because of its sheer volume[1] and business relevance, the primary focus is on claim management's efficiency and effectiveness, specifically the level of fully automated claim processing and adherence to service level agreements (SLA), that is, the time elapsed between the submission and settlement of a claim. In a Swiss-based health insurance company with around 15 million claims per year, process mining first helped measuring the full automation rate over the year, namely 74% (target being 80%). Second, it shed transparency on the root-cause for manual work: a large bulk of claims were detoured to manual inspection just to set a final approval sign. While this activity took less than 10 s processing time, it delayed the process by a median 1.8 days (waiting time in work baskets) and reduced the automation level by 8.2%. By refining the rule-set for claims that *really* required the approval step, the automation immediately raised to 82.2%. As a side-effect, this has improved the SLA adherence by 8%.

Industrial Products. This type of industry is predominantly characterized by the manufacture of different types of products, such as cars, electronics, power plants or chemicals. Producing businesses, when transforming their operation towards bottom-line savings or top-line improvement, mainly focus on the so-called *operational support functions* including procurement, sales and general accounting, and *supply-chain* and *production*.

Because the operation of such industries is usually based upon a traditional, in terms of data structure widely-understood ERP system, such as SAP or Oracle, this industry can be seen as the forerunner for the deployment of process mining "in the large". The main targets for process mining are procurement – "procure to pay" (P2P) or "source to pay" (S2P) – or sales – "order to cash" (O2C) or "lead to cash" (L2C). We address the sales process in the context of SAP in detail in Sect. 3. In fact, these two core processes – procurement and sales – often deliver a number of quick-wins for rapid process improvements, both in terms of cost-savings and increased revenue.

[1] In Switzerland, for example, the largest health insurance companies receive on average around 1.5 million claims per month. In Germany, this can be up to 17 million claims per month.

As an example, we have analyzed the procurement process of a mid-sized company manufacturing laser-cutting machines, focusing the analysis on three main European legal entities. With process mining we identified cyclic payment runs for invoices (each fourth working day). By overlaying the payment cycles with the payment terms associated to those invoices, we have identified a negative offset. That is, discounts associated with paying an invoice within a specific period were not taken into account whilst prioritizing the payment runs. Over one year and considering only the three entities in scope, this amounted to EUR $.83$ million unrealised discounts.

Turning to production, a very popular analysis regards the interplay between the front-office (in charge of taking leads and orders) and the production plants. In other words, the interplay between the sales and the production process. In this setting, we have used process mining to analyze the impact of late change order requests (coming from the front-office executives) to four production plants for a global fragrance and flavor producer. Late requests led to changes in the production planning, requiring, depending on the situation, a reschedule of production or stock transfers for products to ensure production. The former created idle production times worth 40.7 FTE per year. By preventing order changes in the so-called "frozen zone," i.e. orders already scheduled for production, the company was able to reduce the idle time by 47% and ensure a more reliable customer service.

2.2 Drivers for Process Mining Deployment

The adoption of process mining as a technique for process understanding, monitoring and improvement is fueled by some characteristics of the leading industry segments. In this section we revisit some of these drivers and how they contribute to process mining adoption.[2]

System Homogeneity. Firms in the Industrial Products space are usually based upon one core ERP system, most predominantly Dynamics, Oracle, Navision and SAP, covering the main processes, with satellite systems for specific tasks, e.g. invoice processing with Basware or customs processing with SAP GTS. Because the underlying tables, data structures and operations for "standard" ERP systems are well-known by experts, the preparation of data towards proces mining becomes easier. Generally, the more homogeneous the system landscape, the easier it is to implement and use process mining, be it by collecting and transforming the data, or by connecting directly to a process mining tool which performs the data transformation. The downside of system homogeneity is that, because of system's maturity, one oftentimes finds less low-hanging fruits in terms of process improvements.

Transaction Volume. Some processes are executed once a month (e.g. the consolidation of financial statements in general accounting), others millions of times a day (e.g. cab

[2] Note that these drivers are independent from each other. For example, while insurance companies's technical ecosystems are highly heterogeneous (thereby making the application of process mining more intricate), they profit from scale, i.e. number of claims, and existing solid data foundation necessary to run the business in the actuarial space.

hailing rides at Uber). Both processes can undoubtedly lead to enterprise performance improvements when analyzed with process mining. However, the higher the number of transactions one has at hand, the higher the (at least potential) impact that can be achieved, and consequently the higher the return on investment (ROI) for process mining and improvement exercises. Just imagine one can identify, on average, USD .5 cost-savings per claim with 15 million claims processes a year. In practice, this inevitably turns into a scoping question when analyzing processes: what is the "minimal" transaction volume to qualify for process mining? There is no magic formula for this, as processes are subject to different cycles and seasonality. So even the same process (e.g. procurement) in the same industry (e.g. manufacturing) might considerably differ from company to company depending on what is produced (e.g. power plants vs. chips). Our recommendation is to start with the end in mind and delineate the scope based on the business questions to be answered, operational debts to be bridged and process improvement ambition. See Sect. 4.1 for the different scoping elements.

Process Drivenness. Some industries – and more specifically, companies in those industries, or even functions in specific companies – exhibit a high maturity level in terms of "process-drivenness" and, correspondingly, digitalization of processes. That is, processes are captured in a structured manner (e.g. by means of BPMN) and the underlying system landscape and data models responsible for the process execution exist (e.g. ER diagrams to capture the relationship of entity sets stored in a database or ADL specifications for architecture description). Other companies (or some of their functions), be it because of their business model or niche of operation, are less "process-driven." For example, in banks the Loan and Credit functions in banks are highly process-driven, while the lead management in Asset and Wealth Management is less so. In fact, for the latter, technically speaking each process execution is a legitimate variant. Clearly, the higher the process-drivenness and volume of transactions, the better the chances for being able to run process mining. The downside is that, because plenty of thinking has been spent on process design and implementation, the quick-wins in terms of improvement potential could already have been harvested by previous initiatives, irrespective of data-driven or not.

Existing Data Foundation. Irrespective of all the aforementioned drivers, some companies have largely invested in building a cross-functional data foundation as part of their data strategy [21], either in the sense of a data mart (department-wide for the provision of some form of business intelligence) or a data warehouse/data lake (enterprise-wide for large data analytics), the latter being the focus of current projects tackling the transformation towards data-driven decision making. Process mining profits substantially from an existing data foundation outside of (and combining the different) core systems. The reasons are threefold: first it avoids dedicated bulk data extractions, which are usually time consuming and require additional effort from IT or base teams; second, because the platforms on which they are deployed (e.g. SnowFlake or Teradata) offer a transformation layer allowing the (automatable, periodic) data transformation, thereby avoiding the setup of an additional transformation platform/layer; and third, they enforce some data homogeneity when standardizing data staging, for example, by making sure that timestamps are recorded to the precision of miliseconds.

Overall, these four key drivers put together factors favoring the use of process mining. Of course, transparency and analytics on their own do not lead to bottom-line savings or outperforming top-line. That is, process mining should be embedded in a broader context aiming at continuous improvement, and the identification and elimination of operational debts, measuring the impact of changes and recalibrating the performance goals according to a well-understood and well-established KPI framework [26]. The business process management lifecycle [11] provides a basis for data-driven process improvement based on process mining, in particular, and process analytics, in general [5].

3 Real-World Example: Order-to-Cash on SAP Systems

In order to make the approach and considerations presented hereafter tangible and easily related to actual use cases relevant for both industry and academia, we introduce an exemplary Order-to-Cash (O2C) process run on an SAP ERP system. O2C is not only prevalent across all three industry types as laid out in Sect. 2.1, but also very much relatable to anyone running a business or even just buying goods online. The twist is to simply look at this buying process through the vendor's eyes, i.e. the firm selling for example electronics through a web shop. Irrespective of the firm's business type, region or size, the main process steps of any O2C process are fairly similar. Hence, it makes a perfect running example to showcase event data preprocessing in a real-world scenario.

Many large organizations run their core business processes on ERP software solutions from Oracle or SAP, imposing a minimum level of standardization on process steps and their sequential flow. Since they are, however, designed to fit many different industries and business models, the predefined guardrails are not very strict, allowing for significant variation even in otherwise well-defined business processes like O2C. And while some of the companies even go to the length of modifying the underlying data structures in order to tailor the systems to their very needs, most modifications do not interfere with the core O2C process flow, but rather add complementary information. Paired with the fact that the adoption of SAP-based O2C process mining is far ahead of their Oracle-based counterparts, an SAP ERP has been chosen to exemplify event data preprocessing for O2C.

An end-to-end O2C process encompasses steps from the initial entry of a sales order and its items, all the way to the actual receipt of payment or another financial record clearing the open balance (e.g. a credit note). In practice we have encountered O2C process analyses with well over 100 different process steps, however, in favor of reducing this complexity to a manageable, but representative set of events, the process flow is exemplarily represented by nine individual steps (or events).

The events as depicted in the first swim lane of Fig. 1 have been selected in order to (a) capture at least one instance of each event archetype[3], while both (b) reducing the number of events substantially, but also (c) retain major milestones of a typical O2C process. We correlate the *Business Flow*, the underlying *Document Flow* as well as the corresponding *Data Flow* as follows.

[3] Technical event archetypes (immutable vs. mutable direct timestamp vs. log entry timestamp) further defined in Sect. 4.

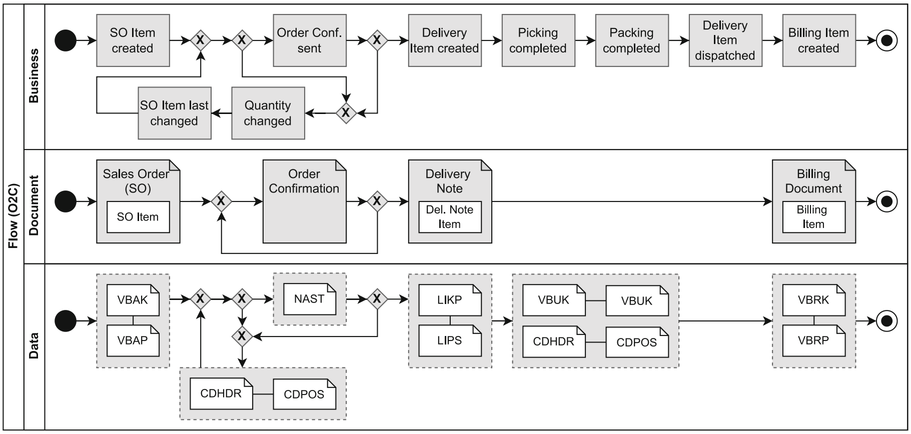


Fig. 1. SAP O2C process description across the different flow types

Business Flow. The process starts with the creation of a sales order (SO) with at least one item (`SO Item created`), after which a confirmation can be sent to the customer (`Order Conf. sent`). As a next step the corresponding delivery document is created including details for all items (`Delivery Item created`), after which the warehouse operations (`Picking completed`, then `Packing completed`) follow, illustrating the application of O2C for sales of physical goods in store. The goods are eventually sent to the customer (`Delivery Item dispatched`) and a corresponding billing document including respective items gets created (`Billing Item created`), which typically interfaces with the financial accounting part of the process. In favor of simplicity, this part is omitted (i.e. all financial postings, such as the settlement of billing documents).

In this given example, we include changes to the quantity ordered (`Quantity changed`) which can be triggered at any stage before the creation of the delivery note. This change event can be seen as a template and hence applied to a variety of other change attributes (e.g. price or requested delivery date). After each change, the corresponding marker on the sales order gets updated as well (`SO Item last changed.`).

Document Flow. The second perspective focuses on the business documents and their flow, as if actual paper documents would be processed. It starts with the sales order (`SO` and `SO Item`), after which the customer is sent a confirmation (`Order Confirmation`). A delivery document (`DD` and `DD Item`) is created and dispatched before the billing document (`BD` and `BD Item`) opens a balance for the respective customer.

Data Flow. Next, we focus on the main corresponding data structures holding information about the events and/or documents. For SAP-based O2C processes, sales orders and their items are stored in a pair of tables distinguishing sales order header information (table `VBAK`) from their item level information (`VBAP`). The data recorded in these tables include their creation date, as well as the date it was last changed.

Order confirmations are persisted in a log table (NAST) comprising nearly all outgoing messages, while delivery document information, including creation and dispatch, can be found in another table pair (headers: LIKP; items: LIPS). Picking and packing is traced through changelogs (headers: CDHDR; items: CDPOS) on sales document status (headers: VBUK; items: VBUP) and billing documents are stored in a separate table pair (headers: VBRK; items: VBRP). Similar to picking and packing, all change events – including quantity changes – are tracked in a change audit log (headers: CDHDR; items: CDPOS).[4]

Limitations. Finally, it is important to point out that the presented O2C process constitutes a radical oversimplification. While the individual events are indeed representative, the process flow and set of events should be treated solely as an excerpt for demonstration purposes. Not only will real-life O2C processes be significantly more complex, system customizations and other modifications to the SAP O2C standard configuration are likely to require additional attention.

4 Event Log Engineering in Practice

Data preparation for process mining in the form of event log engineering encompasses three main steps, namely:

1. *Data selection and extraction*
2. *Data transformation*
3. *Data-model engineering and fine-tuning*

This section addresses these three steps from two perspectives: first from a broad perspective by touching upon key aspects to be considered; and second, in a zoom into the specific setting introduced in Sect. 3. The following is not meant to be a complete cookbook for process mining preprocessing. Instead, it focuses on the predominant, recurring aspects and challenges – some specific to process mining, some applicable to a wider spectrum of data analysis initiatives.

4.1 Data Selection and Extraction

From a general standpoint, this step focuses on answering the following questions:

1. Which data is to be extracted and when?
2. Which attributes are necessary for the analysis?
3. Is data sensitive?
4. Is data readily available or archived?
5. Will the data size to be extracted be overwhelming?

[4] Table names in the SAP ECC data model are typically four to six character abbreviations of the context or document they capture. Because of the geographical origin of SAP, namely Germany, the abbreviations often stem from German. For example, VBAK and VBAP stand for, respectively, “Verkaufsbeleg: Kopfdaten” and “Verkaufsbeleg: Positionen,” where “Verkaufsbeleg” means “sales order”.

The answers to these questions can be clustered under the labels "scoping" and "sourcing." The *scoping phase* defines four analytical angles: *processual angle* (i.e. the subject of analysis), its *regional angle* (i.e. a specific country or set of legal entities), the *time angle* (i.e. time span of transactions to analyse), and the *analytical angle* (i.e. the "why" behind the analysis).

Once the scope is set, the *sourcing phase* establishes a mapping between the process steps and their attributes in a transaction and the events in the source systems, tables and objects. The overarching goal is to identify where – if at all – the necessary events are digitally represented and which attributes are natively available. In some situations, both events and attributes need to be derived by combining different characteristics. For example, the definition of an "automated event" in an SAP system depends on various factors, including user type and reference transaction. Hence capturing and interpreting them correctly is essential for the credibility of process mining (see Sect. 4.2 for details).

The final step in the sourcing phase is the "physical" data extraction from the relevant system and corresponding data objects to a destination outside the system. Assuming that all the data is based on a single ERP landscape, this usually happens by querying the corresponding tables and applying selection criteria to filter out, e.g., the transactions falling in the current time and regional angles. This could be either done by means of an ETL tool connecting to the system, by creating a dedicated extraction script (e.g. specialized ABAP code for SAP, or DART, SAP's embedded extraction tool), or by backing up the relevant tables and fields from the system (see Sect. 5 for best practices on data extraction). In companies with large data volumes or analyses considering a wide time angle (say, 10+ years), data extraction might need to consider so-called "archived transactional data". Whilst archived data can be seamlessly brought back to life, in practical settings, not the entire transaction is archived bur rather its main attributes, for storage capacity reasons. This might restrict the analytical angle for archived transactions.

When extracting transactional and associated master and change data, two aspects are important: first, *data size*; and second, *data protection*. For the former, to estimate the final size of extraction and, simultaneously test the extraction method, one usually extracts, say, one month of data. By extrapolating this to the final time angle, one approximates the final number of cases and events to be dealt with, and consequently the size of final extraction. For the latter, the advent of the General Data Protection Regulation (GDPR) specifically, and increased awareness for data protection generally, puts additional requirements to data extraction and processing. Here, two strategies comes handy. First, *data minimization*, that is extracting only the information strictly needed to cover the analytical angle. For example, if an analysis aims to measure the level of automation in a particular process, one can solely extract the user type, not necessarily the user name or ID. Second, for the necessary but sensitive fields, *data obfuscation* techniques generate – during the extraction – an irreversible value for a particular field. In practice, the most common method is by hashing the values for the sensitive fields. This is typically applied to personally identifiable information, such as user IDs and customer names. Security and privacy have been an important topic in business process management and process mining [1,20], the widespread adoption of process analytics and mining paired with stricter legislation created a sense of urgency which is translating in cutting-edge, scalable data-protection approaches, such as [18,22].

Data Selection and Extraction in the SAP O2C Scenario. In the following, we apply the general considerations discussed before to our SAP O2C running example. As a reminder, the scoping phase defines the rationale (“why”) and derives the object of study (“what”) using business terminology, while the sourcing phase translates this scope into technical delimitations and specifications, guiding the actual extraction of data. The exemplary scenario presented hereafter is fictitious, though resembling essential experiences and learnings from real process mining initiatives.

Scoping Phase. While the initial trigger for starting scoping discussions for process mining can originate from IT/analytics departments or solution vendors during pre-sales, we choose to exemplify an arguably more value-driven context. The Global Head of Order Management aims to optimize the firmwide order management process and has been introduced to the general concept of process mining, which seems to be a perfect fit. She initiates a pilot project to evaluate the suitability of the approach, drive process transparency and distill tangible process improvement levers. During scoping discussions with process mining experts, three hypotheses are agreed to become the predominant analysis directions for moving ahead in an orchestrated manner:

1. *Quantity changes*. The number, magnitude and time-wise distribution of quantity changes in sales order items is, while being a driver for additional manual effort and downstream ripple effects, concentrated around recurring patterns (e.g. customers, regions, product groups).
2. *SLA adherence*. Transparency on Service Level Agreement (SLA) performance with regards to sending order confirmations a minimum number of days before their dispatch can greatly improve both the adherence to and eventually the perceived value from such agreements (e.g. with key customers).
3. *Process conformance*. Data driven sensing of process flows violating the designed and desired process model informs process owners, operational staff running the process, as well as governance bodies (e.g. internal audit), about needs for additional training, additional guardrails or even process re-engineering.

While the first hypothesis looks at options to streamline the process, the second one bears potential to create additional value for customers. Lastly, hypothesis number three looks at more medium to long term objectives around process robustness and clarity of flow, which many times is a precursor for automation.

After rallying around the rationale for employing process mining, the scope (i.e. object of study) is being defined. As a largely business-driven exercise, process mining experts typically need to act in a (technical) counterbalance role, since the larger the business scope is set, the more complex all steps of the resulting process mining exercise will be. Hence, it must be the joint goal to aim at the smallest possible scope, while still retaining enough to be representative with regards to all shortlisted hypotheses.

The first delimitation is made with regards to the underlying business process. In the context of our SAP O2C example, all three hypotheses are related to the O2C process, more notably even, the non-financial part of O2C (sometimes referred to as order management). As the next level of detail, a minimum set of process steps or events is selected in line with the hypotheses (as described in the ‘Business Flow’ swim lane in Fig. 1). The second scoping task identifies corresponding business objects to be traced.

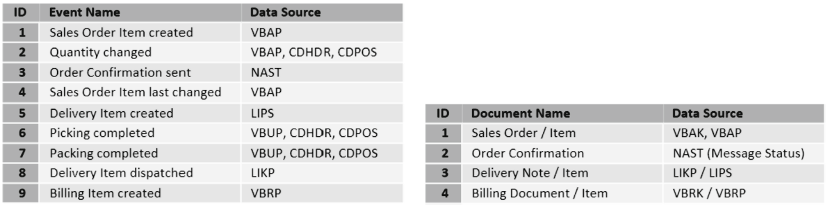
Please refer to the 'Document Flow' swim lane in Fig. 1. The third delimitation challenges whether all organizational units (e.g. legal entities, regions or segments) and transaction types (e.g. consignment vs. standard sales) need to be included to retain validity and significance of analytical results. Oftentimes, the project participants are highly acquainted with one specific part of the business, making it a natural choice to ensure the right expertise is available when validating results later. With regards to transaction types, high volume types are typically scoped in when looking at efficiency hypotheses. In our example we focus on 'standard sales from stock' only, while the fictitious firm operates as one legal entity with one sales organization and one warehouse. The fourth delimitation looks at the timeframe to be analyzed. Depending on the underlying data volume, it might become necessary to further restrict the timeframe in scope later during the sourcing phase. In order to capture seasonality, it is generally recommendable to cover one full calendar or fiscal year. In our example we restrict the analysis to data from 2020. The fifth and last business-driven scoping discussion typically presents the biggest challenge. Here, one aims to delimit the number of different data points associated with each business object. For example, each SO item has more than 400 individual attributes in any given SAP system. Some of them are collocated, others require multiple data linkages, but most importantly, many do not naturally indicate whether they might become useful context around process execution during the downstream analysis. While the default reaction of business favors retaining everything, the resulting spike in technical effort and complexity renders this extreme as inadvisable, sometimes even infeasible. In our simplified example we assume the process mining experts are seasoned enough to guide the team toward a narrow selection with necessary attributes only. Such a selection does typically not exceed 40 attributes in case of the SO item example.

Sourcing Phase. With the scope being clearly defined from a business perspective, the first step in the sourcing phase is to translate all delimitations into technical terms, i.e. a selection of source systems, data sources (e.g. tables or log files) within these systems, corresponding parameters to filter data records and last but not least the selection of required attributes within the data sources.

In our SAP O2C scenario, we focus on one source system only (exemplarily 'P42', an SAP R/3 ERP, even though the characteristics described largely apply to SAP S/4 instances as well). Since SAP ERP systems are capable of multi-tenancy, it is important to select the correct tenant in addition, which in case of the running example falls on the only active tenant configured in the productive ERP instance (i.e. 'P42/010').

Next, the process steps and traced documents (please refer to the 'Business Flow' and 'Document Flow' swims lane in Fig. 1) are translated into their respective data sources. Often, this exercise with its required deep expertise is indicative to whether multiple data scope refinements and, hence, data extractions will become necessary, thereby prolonging the project timeline. These translations applied to the running example are shown in the tables in Fig. 2.

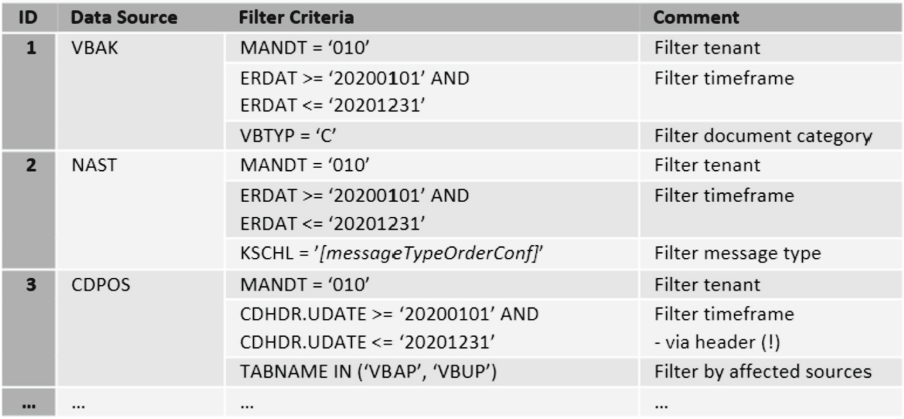
In order to restrict the extraction data volume for each data source, the delimitations on organizational scope and transaction type, as well as timeframe are translated into row filter criteria. Figure 3 shows exemplary filtering criteria. While the tenant filter represents an example for restricting the organizational scope, a timeframe filter



ID	Event Name	Data Source
1	Sales Order Item created	VBAP
2	Quantity changed	VBAP, CDHDR, CDPOS
3	Order Confirmation sent	NAST
4	Sales Order Item last changed	VBAP
5	Delivery Item created	LIPS
6	Picking completed	VBUP, CDHDR, CDPOS
7	Packing completed	VBUP, CDHDR, CDPOS
8	Delivery Item dispatched	LIKP
9	Billing Item created	VBRP

ID	Document Name	Data Source
1	Sales Order / Item	VBAK, VBAP
2	Order Confirmation	NAST (Message Status)
3	Delivery Note / Item	LIKP / LIPS
4	Billing Document / Item	VBRK / VBRP

Fig. 2. Data sources for the SAP O2C process

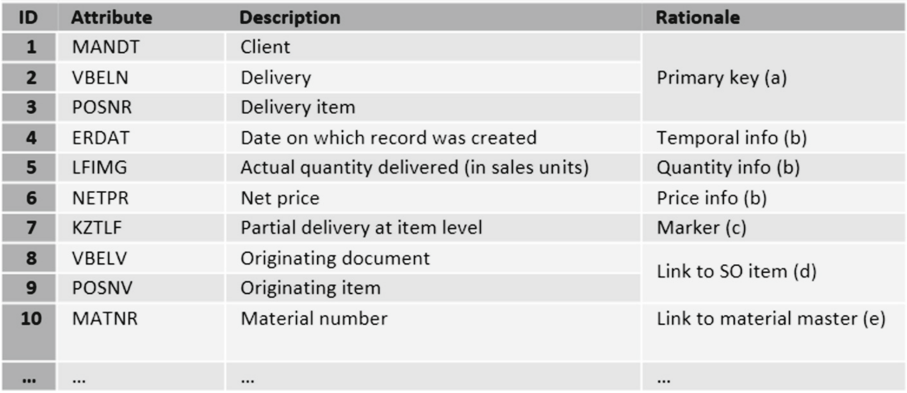


ID	Data Source	Filter Criteria	Comment
1	VBAK	MANDT = ‘010’	Filter tenant
		ERDAT >= ‘20200101’ AND ERDAT <= ‘20201231’	Filter timeframe
		VBTYP = ‘C’	Filter document category
2	NAST	MANDT = ‘010’	Filter tenant
		ERDAT >= ‘20200101’ AND ERDAT <= ‘20201231’	Filter timeframe
		KSCHL = ’*[messageTypeOrderConf]*’	Filter message type
3	CDPOS	MANDT = ‘010’	Filter tenant
		CDHDR.UDATE >= ‘20200101’ AND CDHDR.UDATE <= ‘20201231’	Filter timeframe - via header (!)
		TABNAME IN (‘VBAP’, ‘VBUP’)	Filter by affected sources
...	...	...	...

Fig. 3. Filtering criteria when extracting data for SAP O2C process

is also applied to each data source. As shown for the data source CDPOS, timeframe restrictions sometimes require linkage to another data source, like in this case its header information in CDHRD. Setting fixed timeframe boundaries will, however, lead to cut-off artifacts in the resulting analysis. If a sales order is registered on December 31 2020, the corresponding order confirmation will likely be created outside the selection window and thereby cut from the extraction. Preventing such artifacts would require substantial pre-extraction analysis and sophisticated extraction mechanisms catering to dependencies between data sources. This is typically deemed impractical, and analysts would rather deal with the resulting artifacts during analysis. Lastly, transaction type filters are exemplified through the document category for sales orders, the configured message type for order confirmations, which needs to be looked up in the system itself, and the list of tables affected by logged changes.

Equipped with a clear selection of data sources and filter criteria (i.e. restricting data records), selection criteria (i.e. restricting attributes/columns) are next. While data sources like sales order item or delivery item tables have over 350 columns, only a small number of them is required to evaluate specific hypotheses. Typically, practitioners hone in on (a) the identifying primary key, (b) temporal, quantity, price, cost, volume, and weight information, (c) markers indicating a state of the object, (d) links to other relevant objects, and (e) links to supplementary information. Figure 4 showcases attribute



ID	Attribute	Description	Rationale
1	MANDT	Client	Primary key (a)
2	VBELN	Delivery	
3	POSNR	Delivery item	
4	ERDAT	Date on which record was created	Temporal info (b)
5	LFIMG	Actual quantity delivered (in sales units)	Quantity info (b)
6	NETPR	Net price	Price info (b)
7	KZTLF	Partial delivery at item level	Marker (c)
8	VBELV	Originating document	Link to SO item (d)
9	POSNV	Originating item	
10	MATNR	Material number	Link to material master (e)
...	...	...	...

Fig. 4. Attribute selection for delivery items in SAP

selection for delivery items (system table `LIPS`). During this screening process it is natural to come across additional supplementary information not yet covered in the data source selection. In such cases, and if their usefulness gets validated, they need to go through the same delimitation procedure as other data sources.

The final step of the technical translation is the screening of a complete, resulting attribute selection for sensitive data. Some data types are prohibited to be transferred across country borders (even if it is solely for analysis), others fall into categories requiring additional safeguarding, pseudonymization or even anonymization. While the process act of obfuscation is part of the extraction itself, it is recommended to identify all attributes requiring special attention upfront. In the SAP O2C scenario, these could entail usernames and details from customer master data.

Lastly, the actual extraction is configured and run accordingly. As all major considerations regarding the extraction of large volumes of data from ERP systems have already been presented in the general section of this chapter, our running example assumes a proven one-time extraction mechanism is used. Such setups have been utilized extensively by external auditors, however, are usually limited to selective one-time extracts, storing the payload in individual files locally on the SAP application server. Since no sensitive data has been identified in the data scope of our SAP O2C example, no obfuscation mechanisms need to be configured.

4.2 Data Transformation

After the extraction, data is typically loaded onto a preprocessing platform to generate the target data model, i.e. the log file and ancillary tables (see Sect. 4.3 for details). This platform can be a database management system (such as a Microsoft SQL Database Server) or part of the ETL tool applied during the extraction. Depending on the process mining technology applied, transformation might also happen inside the tool, such as in UiPath Process Mining and Celonis.

Data transformation is the most important preprocessing step in the journey towards process analytics. This is in particular relevant because an error in reconstructing the end-to-end process may cascade to a completely flawed process mining exercise, delivering misleading results, creating negative experience and, in the worst case, discrediting the whole approach. Consequently, a lot of attention needs to be put into mapping the correct process.

Specifically, we want to call out the following key aspects, namely:

1. Case Identifier consistency,
2. Timestamp quality, and
3. Amount handling.

Serving as unique transaction identifier, the *Case Identifier* (CaseID for short) is a primary point of concern when transforming data in order to achieve an end-to-end representation of the process. When considering one single system, such as SAP, the transaction identifier is typically given by the key document number being tracked (e.g. sales order number), to which other related documents refer. When the transaction spans different systems, the CaseID might – or might as well *not* – be consistent across them.

Assuming that CaseIDs are not consistent, two situations might occur: (1) there exists a mapping between the systems, that is, one can precisely link the transactions, even though the CaseID used in the systems for the same transaction differ; or (2) there is no link between the systems, or this link is not persistent (e.g. being deleted after 24 h). In (2), transactions can only be approximated by relating timestamps and transactional attributes on both systems, the so-called *linkage criteria*. That is, assume that transactions are passed on from an Application A to an Application B. The linkage criteria will initially define a time range (e.g. from 1 to 10 s) within which a transaction ending on the Application A is connected with the transaction that commences on Application B. Ideally, the matching of timestamps on both ends will create an one-to-one linking of the transactions. The resultant CaseID could be the concatenation of the CaseIDs on Systems A and B, e.g. CaseID$_A$–CaseID$_B$. However, in practice a linkage criteria based solely on time ranges can lead to one-to-many relationships between the transactions on both systems, for instance when the cadence of transactions is high. By refining the linkage criteria with non-temporal matching attributes (e.g. the vendor and/or material), one gains precision and reduces uncertainty. Still, in some settings it is impossible to achieve a perfect mapping across systems. In these situations we recommend adding a case attribute that flags those transactions which perfectly match and those which do not.

Timestamps are essencial in process mining, as they mainly charaterize the partial ordering $\prec$ in which the events are sequenced by the underlying process mining algorithms. A typical problem happening in particular when analyzing automated process steps in sequence, but also in other contexts, is the precision of timestamps. Automated process steps happen in range of miliseconds, and this is the precision with which timestamps need to be captured, otherwise process steps will have the same timestamp. In practical settings, this leads to an extreme high number of process variants, as the process mining engines will pick events with the same timestamp in a random order and artificially create variants. When precise timestamps are not available, hardwiring the process ordering by means of a dedicated field in the final log might come handy.

Tools will use this field to enforce the ordering, avoiding unnecessary variants. Another solution is to subsume all the sequenced steps into one, assuming that their ordering is not relevant for the analytical angle.

Another aspect commonly overlooked in analysis is the fact that the timestamps might be captured in a different timezones, especially when the regional angle spans various continents. (Summer and winter time shifts shall not be forgotten, too.) As an example, suppose one is looking for outlier transactions in which invoices were settled outside of the standard European working hours for a particular company, e.g. between 7PM and 6AM. Completely legitimate settlements happening the US might be then considered illegitimate if one does not normalize the timezones. This can be either done by adding a supplementary timestamp field to the log (denoting the time according to the reference point, e.g. CET, defined in the analytical angle), or by adding an attribute field for the timezone offset according to the reference point (e.g. +2).

A final consideration regarding the timestamps is the fact that typical ERP systems, as well as most of the legacy systems, do not capture the begin *and* end timestamps of events. As events are therefore "atomic", it is impossible to measure the actual duration of an process step and, correspondly, to quantify the working time per step. In fact, the lead times between events in a discovered process maps encompass both processing and waiting times. To address this, approaches for so-called "effort mining" are being developed and tested in practical settings [28]. They using statistical methods to estimate the duration of tasks, thereby allowing the quantification of working time and productivity, as well as benchmarking.

Considering *amounts*, two frequent issues are: (1) *amount duplication*, and (2) *unharmonized currencies*. The duplication happens when loops exist in the process. For example, suppose the event "Issue Invoice" happens, with an associated event attribute "Invoice Amount." Furthermore, suppose that, because of a loop, this event happens twice in some cases. Naively adding up the amounts associated with the event "Issue Invoice" will include duplications because of the multiple occurrences of the event within a case. Similarly, amount corrections happening in a case must be taken into account when assessing the final amount related to a case. Ideally, to avoid dupliations and other errors associated with amounts one should parse, the execution traces create an ancillary case attribute table recording the amounts per case (see Sect. 4.3 for details on the data model), thereby avoiding calculations on the specific process mining tool.

`Unharmonized currencies` typically happen when the analytical angle spans different countries, e.g. Denmark and Brazil. As above, naively adding up amounts without taking into account the different local currencies will lead to a wrong financial assessment of the process. Therefore, as a preprocessing step, currencies shall be harmonized to a reference currency set during the analytical angle, such as USD or EUR. This will be the *reporting currency*. The basis for such a harmonization might be system tables capturing the currency conversion rates history (e.g. `TCURR` on SAP), or dedicated APIs from which the historical foreign exchange can be retrieved (e.g. Fixer.io). For flexibility, the resulting data model stores both the amounts in local and reporting currencies, optionally also the conversion rate.

Data Transformation in the SAP O2C Scenario. In the following, we apply the general considerations discussed before to our SAP O2C running example. While first focusing on unit harmonization (i.e. timestamps and prices), the second part will describe different archetypes of events and exemplarily discuss data transformation steps to generate event log entries.

Unit Harmonization. As characterized above, there are several types of attributes that can occur in different base units across the data sources, sometimes even within one source system. Starting with the timestamps, SAP typically stores date (`DATS`) and time (`TIMS`) data in the configured time zone in the SAP installation. Some timestamps are, however, persisted in the time zone of the individual user interacting with the system (e.g. in SAP Warehouse Management, short WM). Luckily, all attributes relevant to our example are based on the same time zone and therefore, no adjustment for different time zones needs to be made. Since we analyze a full year of data, the switch between summer and winter time can – depending on the SAP system configuration – still require adjustments and a decision to treat one of them as dominant.

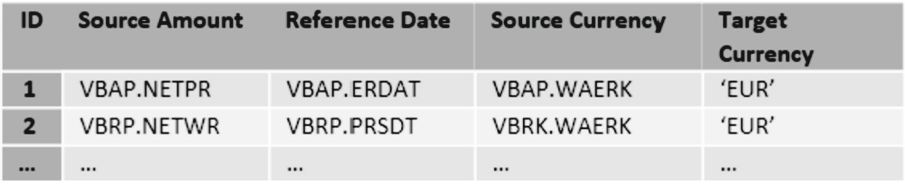
Before applying adjustments, we prepare our data sources by combining separated date and time information into timestamps (e.g. in `VBAP`: `ERDAT` & `ERZET` > `tsCreation`). If any data source contains multiple separated timestamps, each pair will result in an additional attribute. Moving to the actual adjustment and taking CET as our dominant base time zone, we adjust all timestamps in summertime by subtracting one hour. For traceability and testing purposes the adjusted timestamp shall be added as an additional column (`tsCreationCET`). Once a project matures into an operational monitoring solution, such steps are typically collapsed to reduce overall data volume.

Another major category for unit harmonization is currency denominated attributes. Within SAP, some data sources provide figures in multiple currencies (often document, local and reporting currency), other hold the transaction or document currency only. In these cases, and especially when firms engage in international business relations, respective metrics need to be harmonized before being compared or aggregated.

There is a substantial level of semantics captured in the way SAP ERP systems convert currencies.[5] However, in the context of our SAP O2C process at hand, we assume a currency conversion mechanism is available in the data transformation environment. Some of the currency attributes which need to be harmonized are static in terms of source and target currency (e.g. all records converted from USD to EUR), others need to dynamically capture the source currency per each individual record (e.g. sales order item price from document currency to EUR). Exemplarily, Fig. 5 shows the input to such a dynamic currency conversion function, whose output is then stored in an additional attribute.

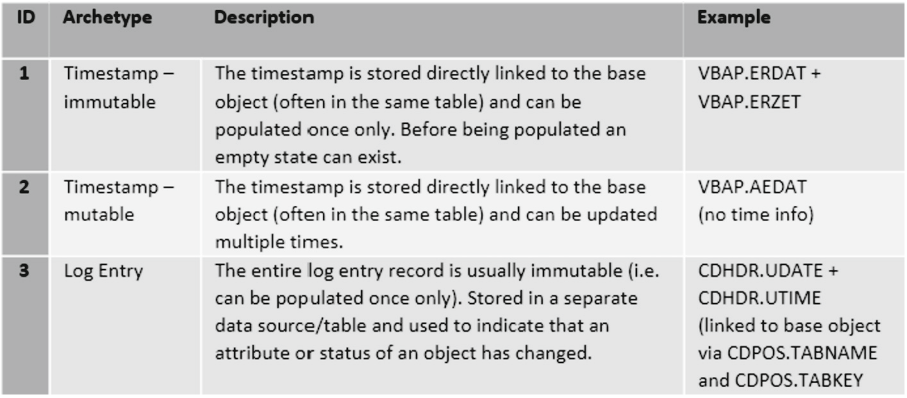
Event Data Transformation. When transforming data into an event log capturing all relevant events as defined in Sect. 4.1, different event data archetypes should be distinguished. These types inform corresponding transformation recipes and while they need to be tailored to individual events, their core structure remains largely intact. Figure 6

[5] It goes beyond the scope of this running example to explain the inner workings of currency conversion in SAP which is based on the tables `TCURF`, `TCURN`, `TCURR`, `TCURV`, and `TCURX`.



ID	Source Amount	Reference Date	Source Currency	Target Currency
1	VBAP.NETPR	VBAP.ERDAT	VBAP.WAERK	‘EUR’
2	VBRP.NETWR	VBRP.PRSDT	VBRK.WAERK	‘EUR’
...	...	...	...	...

Fig. 5. Exemplary currency conversion.



ID	Archetype	Description	Example
1	Timestamp – immutable	The timestamp is stored directly linked to the base object (often in the same table) and can be populated once only. Before being populated an empty state can exist.	VBAP.ERDAT + VBAP.ERZET
2	Timestamp – mutable	The timestamp is stored directly linked to the base object (often in the same table) and can be updated multiple times.	VBAP.AEDAT (no time info)
3	Log Entry	The entire log entry record is usually immutable (i.e. can be populated once only). Stored in a separate data source/table and used to indicate that an attribute or status of an object has changed.	CDHDR.UDATE + CDHDR.UTIME (linked to base object via CDPOS.TABNAME and CDPOS.TABKEY

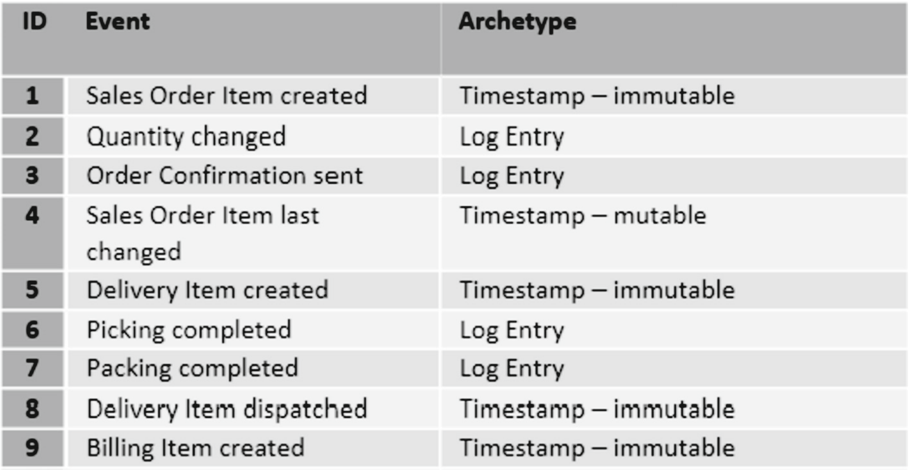
Fig. 6. Types of data archetypes.

delimits these three archetypes and Fig. 7 maps them to the events which are part of our SAP O2C running example.

Below, we detail the event `Sales Order Item created` to exemplify the immutable timestamp transformation archetype, the event `Sales Order Item last changed` to exemplify the mutable timestamp transformation archetype, and both events `Order Confirmation sent` as well as `Picking completed` to exemplify the log entry transformation archetype. For simplicity reasons we limit the transformations to the three basic elements for process mining: (a) the object ID/case ID candidate, (b) the event name, and (c) the timestamp.

Timestamp – immutable. In order to extract event records for the event type `Sales Order Item created` an object ID (caseID candidate) is crafted by concatenating `VBAP.MANDT`, `VBAP.VBELN` and `VBAP.POSNR`, the primary key of the respective data source table. In a preparation step we have already generated the corresponding timestamp `VBAP.tsCreated` from `VBAP.ERDAT` and `VBAP.ERZET`. Many event types can be extracted in this manner.

Timestamp – mutable. To extract event records for the event type `Sales Order Item last changed` we use the same object ID as for the immutable event. In a preparation step, we have also generated a corresponding timestamp `VBAP.tsLastChanged` from `VBAP.AEDAT` and `23:59:59`, a dummy time to fill the missing precision in this timestamp. It is very important to clearly document usage



ID	Event	Archetype
1	Sales Order Item created	Timestamp – immutable
2	Quantity changed	Log Entry
3	Order Confirmation sent	Log Entry
4	Sales Order Item last changed	Timestamp – mutable
5	Delivery Item created	Timestamp – immutable
6	Picking completed	Log Entry
7	Packing completed	Log Entry
8	Delivery Item dispatched	Timestamp – immutable
9	Billing Item created	Timestamp – immutable

Fig. 7. Mapping archetypes to the events.

of such dummy times, since they can lead to undesired analysis results due to misinterpretation of the event sequence. In general, such mutable event types are more valuable for operational process mining analyses, with shortened refresh cycles, and thus a greater chance of the data still being current at the time of analysis. We included it in the SAP O2C running example for completeness only.

Log entry. As the first example of the log entry archetype, the event records for `Order Confirmation sent` are retrieved. Assuming the data source `NAST` has already been filtered to solely include order confirmation message types, it is linked to VBAP based on the client (`MANDT`) and its object key (`OBJKY`) referencing to the header primary key of `VBAP` (`MANDT`, `VBELN`). The same concatenated object ID is used as for the immutable event. And since the two sources are linked already, tsProcessed as derived from `NAST.DATVR` and `NAST.UHRVR` is used as the event timestamp.

The second example derives the event `Picking completed` from SAP's change documentation. Assuming the data source `VBUP` has already been filtered to solely include the item status information of standard sales order items, we also restrict the change logs based on the affected table (`CDPOS.TABNAME` = `VBUP`), on the affected field (`CDPOS.FNAME` = `PKSTA`), and on the change type (`CDPOS.CHNGIND` = `U`) to retain value updates only. Thereafter, `VBUP` is linked to `CDPOS` based on `MANDT` and `CDPOS.TABKEY` referencing the primary key of `VBUP`. Next, change log header information (`CDHDR`) is linked based on `MANDT` and `CHANGENR`. Lastly, we can extract the object ID from VBUP analogously to the immutable timestamp example, and the prepared timestamp `tsUpdated`, derived from `CDHDR.UDATE` and `CDHRD.UTIME`. Many changelog structures for other event types work similar, even outside the SAP ecosystem.

The exemplarily described recipes can be applied beyond the events listed as part of our simplified SAP O2C process analysis. It is rarely a blind application, however, rather a tailoring exercise. Sometimes the name of the resulting events – mostly in the log entry archetype – is even meant to be dynamically derived from attributes on a record-by-record basis. This becomes particularly useful when analyzing workflow systems with potentially hundreds of different events, since all of them can be extracted with one transformation recipe.

4.3 Data Model Engineering

Generally speaking, the transformation creates an *event log* for the process in scope, as defined in the processual and regional angles. It further contains the necessary events and attributes needed to respond to the analytical angle happening in the time span prescribed by the time angle. This section focuses on considerations at building a data model fit for scalable process mining analytics.

The simplest target data model for an event log file is a table in which the columns capture the attributes and the rows capture the events. Although some tools still build upon a single event log table as their input format and although this format might be handy for small exercises, producing a single event log has several adverse practical implications, namely:

1. *Log file generation.* Even in narrow-scoped analysis, an event log may quickly have .5 million transactions (cases) with around 100 associated attributes. Changes in the transformation logic – a frequent step to appropriately capture the business logic or fine-tune for system customizations – lead to a full reprocessing of the whole event log. This procedure can, depending on the transformation platform, take hours (or even days) to complete.
2. *Lower analytical performance.* When analyzing a process, e.g. by applying a filter for a drill-down or computing a KPI, a large event log packed on a single table might have an adverse impact on the user experience in the tool. Specifically, the response time can be very high, making it hard to interactively produce insights.
3. *Unnecessary reduncancy.* A single event log does not distinguish between case (or transaction) attributes, such as legal entity or material, and event attributes, such as user name. Therefore, depending on how the transformation handles case attributes, they might be replicated throughout all the events within a case, or have empty values, which is a suboptimal use of the data model.
4. *Scalability and size.* While an event log might have a stable size during a proof-of-value, in process mining implementations tackling continuous process monitoring and improvement, the event log steadily grows as new cases are appendend to the existing model. Depending on the cadence of transactions (and number of events in those transaction), this can easily result in an average monthly increase well over 100 million lines. This has an implication on the disk space necessary for storage, as well as how the tools might be able to load and process this log.

Practical process mining thus calls out the need for more efficient and scalable logs. There are two complementary strategies for engineering event logs. The more general strategy focuses on splitting the log into at least two tables: the so called *event table* containing the events and their attributes, and the *transaction table* containing the case attributes. The key linking these two tables is the CaseID. This strategy can be further refined, depending on the needs of the analysis. For example, another usual structures seen in practice is the *change table* capturing updates in the main documents (e.g. `Quantity changed` in Sect. 3) and the *property table* capturing derived precomputed transaction attributes easing the analysis (e.g. the number of events in a particular transaction or precision of the linkage criteria, as of Sect. 4.2).

The more specific strategy takes the scope and its different angles into account, as well as who is eventually consuming the analysis. Specifically, when the *regional angle* comprises multiple geographies (e.g. five hubs of a Global Business Services (GBS) topology), it is wise to create one data model – irrespective of its layout – for each hub. While this does not prevent having a global analysis, benchmarks and knowledge transfer of best-practices, it by default ensures controllability and need-to-know policies, i.e. that hubs focus on their area of concern. The *analytical angle* is also a strong driver for event log engineering. For example, an SAP O2C analysis might focus on improving client servicing and lead management. In this case, the focus is on transactions against external customers, and *not* on intercompany or intracompany transactions[6]. Therefore, the data model for this analysis can be built to comprise only the relevant transactions.

Generally, narrowing the event log according to the scope reduces the risk of adding noise to the analysis, and the risk of misinterpretation. This is because it requires the clear-cut specification (and transparent communication) of the filtering criteria used during log engineering and data transformation. It also reinforces that there is no "one size fits all", standard target log file and set of events and attributes to be reconstructed.

Data Model Engineering in the SAP O2C Scenario. In the following, we apply the general considerations discussed before to our SAP O2C running example. Starting with the selection of a common process instance identifier or case ID suitable for the analytical angle at hand, we define a dedicated data table for information on each process instance (i.e. the case table). Lastly, contextual data is added in a scalable way and linked to the core data model.

Case Identifier. When transforming source data into event records as described in Sect. 4.2 the resulting object identifier (object ID) is typically referencing the underlying business object or document. Exemplarily, the events derived from sales order items (`VBAP`), e.g. `Sales Order Item created`, will have a concatenation of the table's primary key fields as its unique object ID reference. However, events derived from other data sources, like `Deliver Item created` will correspondingly have an object ID composed of the primary key fields of `LIPS` assigned. This results in a need to relate these objects and documents involved in our O2C process, in order to retrieve original process flow end-to-end.

We look at the document flow in Fig. 1 and use the link attributes we preserved in Sect. 4.1 to derive an object graph in accordance with the relationships between the

[6] Intercompany transactions are between two or more related internal legal entities in the same enterprise; intracompany transactions are between two or more entities within the same legal entity.

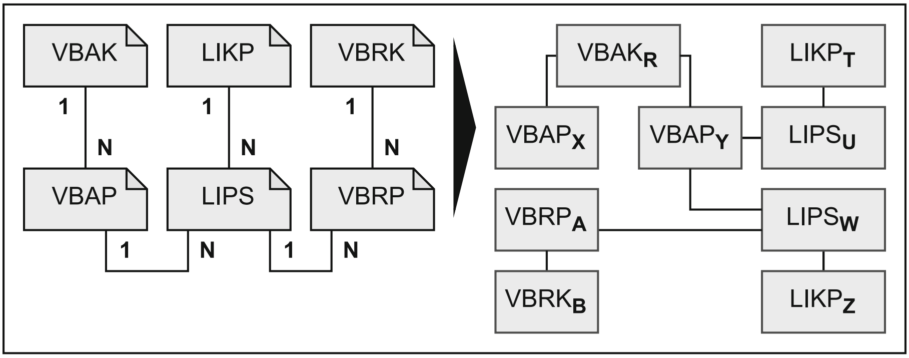


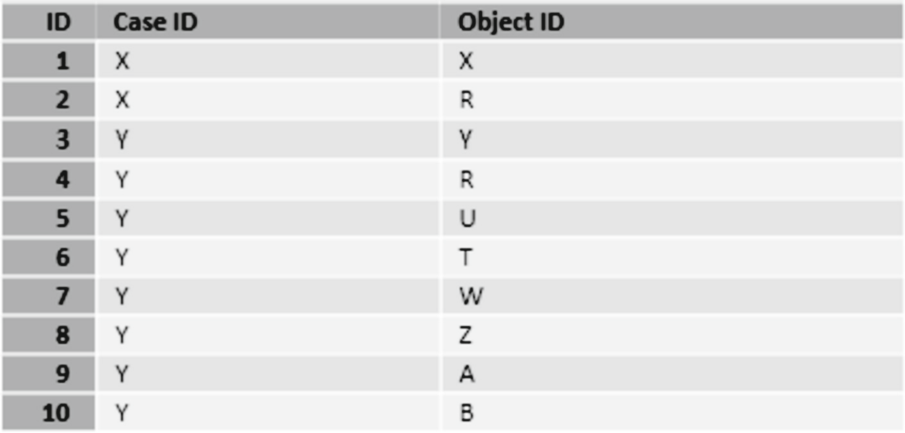
Fig. 8. Relationship model relating to the case identifier.

corresponding data sources. The only exception is the data source NAST, whose corresponding events have already been linked to the respective sales order item object ID during event data transformation as described in Sect. 4.2. Such direct links are typically used when the business object or document has very few additional attributes of relevance – link in the exemplary case – the order confirmation. Please refer to Fig. 8 for the resulting relationship model and exemplary graph.

As some of the relationships between the business object data sources are one-to-many (in some scenarios even many to many), the resulting graph/forest can become quite complex. Considering the example in Fig. 8, the part of the graph in which two sales order items exist (X and Y), with both belonging to the same sales order (R). While X has no link to any delivery item yet, Y references two distinct delivery items (U and W) with two different headers (T and Z). This means the sales order item was likely split into two deliveries. Now, one of the deliveries (W) is already billed with a billing document item (A) and header (B).

For most process analyses it is advisable to define one of the object ID types as the case identifier (caseID). Based on the underlying analytical angle and hypotheses, we select the sales order item as the identifying document type and create a mapping table, which lists all reachable objects within the forest as a function of the caseID (see Fig. 9). As a rule of thumb, when traversing the forest, the very same relationship shall not be traversed in both ways, i.e. after connecting X to R, we do not proceed to connect Y to the same set of reachable objects since it would take the same relation (VBAP $\rightleftharpoons$ VBAK) that connects X to R in a backward direction. This approach prevents linking objects and thereby their associated events erroneously. The combination of our mapping table with the event record table from Sect. 4.2 results in a final event log table, which can already be used for process mining.

Case Table. Most process mining analysis are moving beyond the pure event traces quite quickly, resulting in the need to add contextual information. The most straightforward option is to create an additional table with exactly one record per process instance (i.e. per active caseID) and adding so-called case-level information to it. Since we have



ID	Case ID	Object ID
1	X	X
2	X	R
3	Y	Y
4	Y	R
5	Y	U
6	Y	T
7	Y	W
8	Y	Z
9	Y	A
10	Y	B

Fig. 9. Mapping table listing reachable objects.

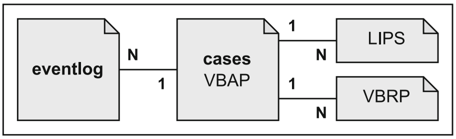


Fig. 10. Connecting additional contextual data sources to the case table.

defined the sales order item as our case ID type, we can simply add the attributes preserved in Sect. 4.1 as case attributes (e.g. net price, material number).

Contextual Data. When using process mining in real business scenarios, the thirst for contextual information does not stop at the case table. Applied to our running example, we can assume that just because we selected the sales order item as caseID type does not mean additional information on the delivery document items and billing items is irrelevant. One approach can be to add such information in the event log with the trade-off being an extremely detrimental impact on data volume. In practice, we rather opt to introduce additional tables, often one per objectID type (except the one selected as case ID type). Illustrated in Fig. 10, we connect two additional contextual data sources to the cases table. In order to establish the link from these object tables to the case table, the previously generated mapping table (caseID $\rightleftharpoons$ objectID) can be re-used.

In practice even more advanced data models, such as the one described above do not support the testing of every hypothesis project stakeholders come up with. Sometimes, hypothesis-specific “helper” tables are created and linked into the process mining data model. In such cases, it is advisable to challenge the business value from such modifications before triggering substantial data model modifications.

5 Best Practices

In this section we take stock on the above sections and distill best practices from our experience of rolling out process mining "in the large". Clearly this is a non-exhaustive list; its intent is to elude on the most relevant and recurring topics.

Data Selection and Extraction. This is the basis for process mining, and if not structured well, hiccups here can undermine the entire analytical effort. Four best-practices in this area:

BP1 Explicitly formulate the four analytical angles and confirm it with all stakeholders.
BP2 Find a sweet-spot between data minimization and extraction efficiency.
BP3 Estimate the final size (and time) of extraction.
BP4 Extract data from a QA environment or existing staging platform.

By following (BP1) one ensures common knowledge as to the analytical objectives and avoid getting lost in details. Turning to (BP2), as mentioned in Sect. 4.1, data extraction technically boils down to some form of `select`-statement on terabyte-sized tables. Data minimization criteria (formulated as `where`-constraints) add constraints to such a statement, slowing down the extraction. Therefore it is important find the sweet spot between minimization and efficiency. One way to do so is to follow (BP3) and carry out a probe extraction with a drastically reduced scope and extrapolate the values to the full scope range. Finally, because a data extraction might have an impact on the performance of the system, (BP4) recommends the extraction of data from a QA (test) environment or staging platform, as opposed to a productive environment. Of course, for this, the extraction environment must fully cover the scope.

Data Transformation. When transforming data towards an event log, events are discovered according to the business logic and system specific configurations. In doing so, the precision is essential for the analytical correctness. Five best-practices to emphasize in this area:

BP5 Modularize event discovery.
BP6 Harmonize timestamp format, currencies and other units.
BP7 Take system customizations into account.
BP8 Do not ignore the business logic and context.
BP9 Meaningful event naming convention

With (BP5) one creates modules to discover the different types of events (e.g. `SO Item created`). In doing so, adjustments in those events (e.g. naming convention or discovery logic) can be done locally without requiring the generation of a whole event log. With (BP6) one avoids misinterpretation of results and a sound basis for analysis. Besides those harmonization efforts, ERP systems are highly customized to a particular business and operational mode. This can be at the level of fields in a table (e.g. a field capturing a specific company flag for a completed delivery) or the way attributes add up for an attribute (e.g. what is an automated vs. a manual step). Therefore, (BP7) recommends to take those customizations into account when transforming data. One approach is to carefully resuse and validate existing transformation scripts.[7] Building on that, different attributes carry aspects on the business logic, e.g. document types associated to sales orders indicating external or internal sales. In (BP8) we recommend to take this into account when generating the event log by creating different events or transaction identifiers. Finally, by (BP9) one facilitates the understanding of process maps. For example, instead of naming an event `SO Item qty chg.`, use `SO Item qty incr.`, already indicating how the change impacted the sales order quantity field. This creates more meaningful logs and a more effective basis for analysis. However, if used exaggeratedly, this leads to an inflation of distinct events, making any analysis a complex undertaking.

Data Model Engineering. The best-practices regarding the target data model have an impact on the scalability and ease of analysis. We emphasize the following:

BP10 Add sanity checks.
BP11 Modularize logs according to the analytical scope.
BP12 Split the attributes according to attribute types.
BP13 Consider ancillary analytics, e.g. machine learning, prediction and simulation.

In (BP10) we recommend the use of sanity check tests indicating, for example, the overall number of cases reconstructed or a summary of fields including `NULL` or empty values. This helps in the quality assurance phase, e.g. by matching the number of expected transactions with the number of cases or by detecting and tracing transformation bugs. By (BP11), one ensures that event logs fit their analytical angle but, at the same time, separate business concerns. Regardless of modularization, by (BP12) one separates the characteristics of events and those of transactions. This eliminates redundant data and is less error-prone during analysis. Finally, process data has been increasingly used as a subject of more advanced analytics. The data model required for such analytics differs substantially from a plain event log. In (BP13) we recommend to take this into account when deciding on the necessary attributes and their aggregation level. In some situations, it is worth creating a separate table allowing, e.g., regression or time series analytics.

6 Outlook

During the past ten years, with process mining finally finding its way from academia into market leading organizations, a lot of progress has been made in both simplifying the approach, including data preparation, for business use, as well as extending associated functionalities and proving to generate tangible value in a growing number of industries. With spreading awareness and substantial increases in capital allocation from venture firms, this development has only accelerated in the more recent past. From our perspective as practitioners, we expect the following to be some of the most substantial improvements:

[7] Also ready-to-use connectors provided in some process mining technologies offer an "aid" but not a "replacement" for tailored scripts.

1. *Architected for Analytics*. With many large-scale tech modernization programs under way, corporations replace more and more legacy systems with modern architectures. Most of these solutions are already designed with modularity and an analysis angle in mind (e.g. with data provisioning APIs), leading to a substantial reduction of effort for data extraction and transformation.
2. *Maturing Connector Landscape*. Two factors will contribute to a maturing landscape of data extraction and transformation connectors. Firstly, and focused on the data intake, the growing adoption of process mining across industries and therefore across a wide variety of (partially) standardized source systems will produce proven, configurable and scalable connector modules. Corresponding know-how will be gradually commoditized, easing access to and usage of these modules. Secondly, and more focused on the data transformation output, vendor agnostic bodies in the field are working on standardizing event log data formats beyond academia. Once adopted by enough players in the field, such standardized formats greatly improve compatibility of approaches across process mining vendors and will eventually even lead to data source systems offering "native" data outputs conforming to the process mining data standard.
3. *Data Model Advancements*. As broached in Sect. 4.3, substantial complexity stems from the requirement to select a caseID type during data transformation. End users, however, desire to have the flexibility of switching between such case perspectives interactively during analysis. With solution vendors reacting to these requirements, data transformation steps related to linking all events to one single case type will become redundant. This field of object-centric and ontology-based process mining [6] not only constitutes an active academic field, but the exciting opportunity for researchers to work very close to tangibly impact businesses across the globe.

Besides the overall maturing industry and the improvements listed above in particular, we anticipate some of the already prevalent challenges to intensify, while new ones emerge from macrotrends:

1. *Ubiquitous Data*. As more and more processes get digitized and data storage capacity is easily and economically scalable, it is no surprise to expect data volumes to continue to increase significantly. However, with the growing adoption of Internet of Things (IoT) and usage sensor data in general, an arguably new data category is added to the mix. Not only can such IoT data volumes exceed transactional data volumes manyfold, they usually come in less structured formats (e.g. simple text files) and are subject to less scrutiny with regards to data quality (i.e. the occasional outlier and malfunction of single sensors is expected). Also, the frequency in which the sensor state is persisted is normally independent of process transactions, but rather driven by the sensor's configuration. Correlating such IoT data to individual process flows presents a major challenge ahead [16].
2. *Simulation*. After embracing the end-to-end transparency added with process mining, organizations demand putting this insight to use beyond educational purposes and the occasional process improvement. The first step – for obvious automation candidates – was and still is to interface or integrate automation solutions. But as soon as more sophisticated questions are asked in the context of process mining

based process analytics (e.g. "What if we change A? Would there be another bottleneck?"), the lack of simulation capabilities becomes apparent. In isolation from process mining, there has been plenty of research [7] and tool support for simulation engines [13]. The challenge will be to seamlessly integrate simulation engines with process mining engines, without turning the corresponding configuration into a Customer Experience (CX) nightmare. It is expected that, as part of this convergence, additional requirements towards event log engineering emerge (e.g. statistical distribution information).

3. *Data Exchange Restrictions*. With regulations and restrictions around data sensitivity and data exchange tightening around the world, it becomes increasingly difficult to manage the compliance angle of holistic and often global analytics initiatives. This shift also leads to additional precaution whenever data is shared with third parties and especially when these are within the open domain. It will accordingly become more and more difficult for academia to work on relevant business scenarios with representative underlying data sets, which in turn results in slower and less targeted innovation in the field, including event log engineering.

In summary, the discipline of process mining and corresponding event log engineering is expected to thrive under the increased attention of academia, solution vendors, professional service firms and financiers. The most substantial impact, however, will continue to emanate from firms of all sizes adopting process mining to streamline operations and – at times – turn process excellence into their competitive advantage.

Acknowledgment. Rafael Accorsi would like to thank Nadja Walti, Peter Blank and Wolf-Dietrich Zabka for their valuable comments, suggestions and proof-reading.

References

1. Accorsi, R., Crampton, J., Huth, M., Rinderle-Ma, S.: Verifiably secure process-aware information systems. Dagstuhl Rep. **3**(8), 73–86 (2013)
2. Accorsi, R., Damiani, E., van der Aalst, W.: Unleashing operational process mining (Dagstuhl seminar 13481). Dagstuhl Rep. **3**(11), 154–192 (2014)
3. Andrews, R., Emamjome, F., ter Hofstede, A., Reijers, H.: Root-cause analysis of process-data quality problems. J. Bus. Anal. (2021)
4. Augusto, A., Mendling, J., Vidgof, M., Wurm, B.: The connection between process complexity of event sequences and models discovered by process mining. Inf. Sci. **598**, 196–215 (2021)
5. Beheshti, S.-M.-R., et al.: Process Analytics: Concepts and Techniques for Querying and Analyzing Process Data. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-25037-3
6. Bistarelli, S., Di Noia, T., Mongiello, M., Nocera, F.: Pronto: an ontology driven business process mining tool. Procedia Comput. Sci. **112**, 306–315 (2017). Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 21st International Conference, KES-20176-8, Marseille, France, September 2017
7. Camargo, M., Dumas, M., González-Rojas, O.: Automated discovery of business process simulation models from event logs. Decis. Support Syst. **134**, 113284 (2020)
8. Davenport, T.H., Spanyi, A.: What process mining is, and why companies should do it. HBR Digital Article, April 2019

9. De Weerdt, J., Wynn, M.T.: Foundations of process event data. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
10. Deloitte. Global process mining survey 2021 (2021). https://www2.deloitte.com/de/de/pages/finance/articles/global-process-mining-survey-2021.html
11. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-33143-5
12. Everest Group. Process mining state of the market report 2020, June 2020
13. Hammann, J.E., Markovitch, N.A.: Introduction to arena [simulation software]. In: Winter Simulation Conference Proceedings, pp. 519–523 (1995)
14. HSPI Process Mining Applications Database. https://www.hspi.it/2020/01/database-delle-applicazioni-di-process-mining-2020/. Accessed June 2021
15. Zuiver ICT. Process mining use-case. http://www.zuiverict.nl/Media/Default/Cases/Case_GemeenteHengelo(EN).pdf
16. Janiesch, C., et al.: The internet of things meets business process management: a manifesto. IEEE Syst. Man Cybern. Mag. **6**(4), 34–44 (2020)
17. Lu, X.: Using behavioral context in process mining: exploration, preprocessing and analysis of event data. Ph.D. thesis, Mathematics and Computer Science. Proefschrift (2018)
18. Mannhardt, F., Koschmider, A., Baracaldo, N., Weidlich, M., Michael, J.: Privacy-preserving process mining. Bus. Inf. Syst. Eng. **61**, 595–614 (2019)
19. Mans, R.S., van der Aalst, W.M.P., Vanwersch, R.J.B.: Process Mining in Healthcare - Evaluating and Exploiting Operational Healthcare Processes. SBPM, Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16071-9
20. Müller, G., Accorsi, R.: Why are business processes not secure? In: Fischlin, M., Katzenbeisser, S. (eds.) Number Theory and Cryptography. LNCS, vol. 8260, pp. 240–254. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42001-6_17
21. PricewaterhouseCoopers. Are your business processes a black box? (2020). https://www.pwc.at/en/are-your-business-processes-a-black-box.html
22. Rafiei, M., van der Aalst, W.M.P.: Towards quantifying privacy in process mining. In: Leemans, S., Leopold, H. (eds.) ICPM 2020. LNBIP, vol. 406, pp. 385–397. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72693-5_29
23. Reinkemeyer, L.: Process Mining in Action. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-40172-6
24. Rojas, E., Munoz-Gama, J., Sepúlveda, M., Capurro, D.: Process mining in healthcare: a literature review. J. Biomed. Inform. **61**, 224–236 (2016)
25. Stocker, T., Accorsi, R., Rother, T.: Computergestützte prozessauditierung mit process mining. HMD - Praxis Wirtschaftsinform. **292** (2013)
26. van der Linden, E.-J.: Successful Process Improvement. Tilia Cordata (2021)
27. van Eck, M.L., Lu, X., Leemans, S.J.J., van der Aalst, W.M.P.: PM^2: a process mining project methodology. In: Zdravkovic, J., Kirikova, M., Johannesson, P. (eds.) CAiSE 2015. LNCS, vol. 9097, pp. 297–313. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19069-3_19
28. Zabka, W.-D., Blank, P., Accorsi, R.: Has the pandemic impacted my workforce's productivity? Applying effort mining to identify productivity shifts during COVID-19 lockdown. Accepted for publication at the Business Process Management Forum (2021)

Process Enhancement and Monitoring

Foundations of Process Enhancement

Massimiliano de Leoni(✉)

Department of Mathematics, University of Padua, Padua, Italy
massimiliano.deleoni@unipd.it

Abstract. Process models are among the milestones for Business Process Management and Mining, and used to describe a business process or to prescribe how its instances should be carried out. It follows that they need to fulfill certain properties to be useful. If they aim to represent how the process is currently being executed, they need to be precise and recall the behavior observed in reality. If the goal is to ensure that the process is executed according to laws and regulations, its model should only allow the behavior that is valid from a domain viewpoint and provides some guarantee to ensure good performance level. Process enhancement is the type of Process Mining that aims at models that fulfill these properties, and the literature further splits it into two subfields: process extension and process improvement. *Process extension* aims to incorporate the process perspectives on data, decision, resources and time into the model: their inclusion in process models enable designers to fine-tune the model specifications, thus obtaining models with higher levels of precision. Process improvement passes through an "improved" process model. If the model contains portions of behavior that lead to unsatisfactory outcomes (high costs, low customer satisfactions, etc.) or that violate norms and regulations, one would like those portions to be disallowed by the model. In case some executions are observed in reality and are not allowed by the model, they should be incorporated into the model if they are observed to generally yield good performances. This chapter discusses these two types of process enhancement, and illustrates some basic and some advanced techniques to tackle it, highlighting the pros and cons, and the underlaying assumptions.

Keywords: Process improvement · Process extension · Decision discovery · Role discovery · Bottleneck analyses · Model repair

A process model is one of the main milestones for Business Process Management and Mining, and may be of two natures. A first nature of process models is descriptive: they are used by process analysts to engage process stakeholders (e.g., actors, managers, chief officers) into discussions on how the instances of the process have typically been executed, or how they should be. A second nature is prescriptive, and that is the case when the models are used as input for Process-aware Information System to automate processes and enforce how they must be carried out [10]. In both of scenarios, desirable models need to fulfill certain properties to be of fruitful use:

1. Models need to be precise and only allow legitimate behavior (high precision). This is especially relevant for models with a prescriptive nature: one wants to ensure that the information systems enforce how process instances must be executed, and also how they must not be.

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 243–273, 2022.
https://doi.org/10.1007/978-3-031-08848-3_8

2. Models should enable the executions that have been observed (high fitness), and are valid from a business viewpoint.
3. If certain executions are proven to lead to poor performance levels, they should be disallowed even if observed. Similarly, if certain executions have proven to reach good performance levels, they should be allowed.

This chapter introduces a number of techniques for process enhancement, which is the type of process mining that aims to create models that fulfill one or more of the properties mentioned above. Process enhancement starts with the provision of a process model for which these properties are relevant. This model can be mined from data, or designed by hand on the basis of process documentations, and/or stakeholder input. The literature proposes two types of process enhancement [25]: *process extension* and *process improvement*.

Process extension focuses on the first property (high precision) and aims to incorporate different perspectives. The model often only defines the control-flow perspective, which is certainly the process-model backbone, but it is insufficient to precisely encode the behavior that a model must explicitly allow or disallow. Processes manipulate, read and produce data (objects), their activities are performed by resources within due deadlines, and they take time to be carried out. In literature, these aspects are named perspectives: data, resource and time perspectives. Their inclusion in process models enable designers to fine-tune the model specifications, thus obtaining models with higher levels of precision.

Process improvement focuses on the other properties, and starts from the belief that, if a model has a prescriptive nature, process improvement passes through an "improved" process model. Improvement can be regarded as ensuring process models *(i)* to better reflect reality, and/or *(ii)* to only allow executions that are valid from a domain viewpoint and/or are correlated to better performances.

1 Process Extension: Basic Techniques

The extension of process models to incorporate multiple perspectives relies on the presence of attributes associated with the events. The definition of simplified event log introduced in [1] can be extended accordingly:

Definition 1 (Simplified Multi-Perspective Event Log). *Let $\mathcal{U}_{ev}$ be the universe of events. A simplified event log $L \subset 2^{\mathcal{U}_{ev}^*}$ is a set of traces, sequences of events, with the constraint that an event can only belong to one trace: $\forall \sigma', \sigma'' \in L.\ e \in \sigma', e \in \sigma'' \Rightarrow \sigma' = \sigma''$.*

Sections 1.2, 1.3 and 1.4 illustrates basic techniques to extend the models to incorporate the data, resource and time perspective, respectively.

1.1 Model-Aligned Event Logs

Several techniques for process enhancement requires that the event-log traces can be replayed on the process model (cf. [5]). This requires events to be univocally mapped

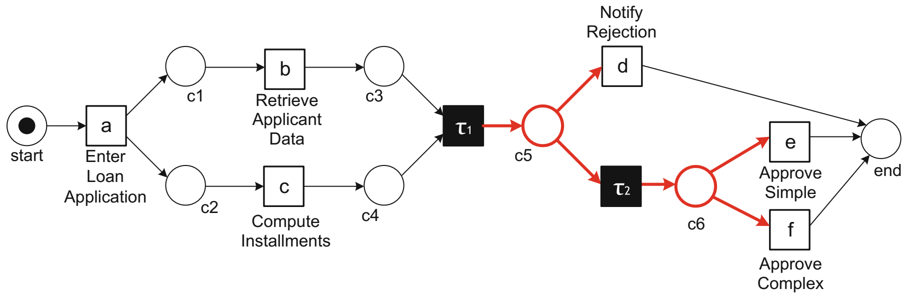


Fig. 1. The Petri net of the working example used in this chapter. The letters inside the transitions identify the transition names, while the script underneath indicates the transition label, namely the activity name. The thicker, red-coloured place and arcs identify a decision point, namely places with outgoing arcs to multiple transitions. (Color figure online)

onto process activities; in case of Petri-net models, events must be mapped onto transitions. However, multiple process activities (e.g., Petri-net transitions) can have the same label, and the choice of the activity to which to map each event is not necessarily local, but it depends on the entire sequence of activities that are executed. Furthermore, when processes are modelled via Petri nets, the model may include invisible transitions, namely transitions with no associated labels that, by definition, leave no trail in event logs. To further complicate the matter, log traces might not be compliant with the process model: certain activities might have been executed when not expected, or not executed when expected.

The situations above can be tackled by solving the following problem: given a log trace $\sigma_L = \langle e_1, \ldots, e_m \rangle$ and an accepting Petri net $AN = (N, M_{init}, M_{final})$ where $N = (P, T, F, l)$, we need to find the **model-aligned trace** $\sigma_P = \langle f_1, \ldots, f_n \rangle$ such that

1. the activity attribute of the events in σ_P is defined over the domain of transition names, namely for each $1 \leq i \leq n$, $act(f_i) \in T$;
2. σ_P is allowed by AN, namely $M_{init} \xrightarrow{\langle \#_{act}(f_1),\ldots,\#_{act}(f_m) \rangle} M_{final}$;
3. an event $f_i \in \sigma_P$ can be mapped to an event $e_j \in \sigma_L$ if the activity attribute of f_i takes on a value equal to the transition of the activity of e_j, namely $l(\#_{act}(f_i)) = \#_{act}(e_j)$.
4. σ_P is the best match to σ_L, namely that minimizes the number of events in σ_L and in σ_P that cannot be mapped.

The computation of a **closest model-aligned trace** can be achieved through alignments (cf. [5]), as explained through an following example: let us consider the log trace $\langle e_a, e_b, e_e \rangle$ where e_a, e_b, and e_e are respectively the events for activities *Enter Loan Application*, *Retrieve Applicant Data*, and *Approve Simple* the subscript indicates the event activity (e.g. $\#_{act}(e_a) = a$). The model is depicted in Fig. 1, where transitions τ_1 and τ_2 are invisible, and the label of each transition is shown under the respective transition. The alignment between the model and the trace is as follows

$$\gamma = \begin{array}{|c|c|c|c|c|c|} e_a & e_b & \gg & \gg & \gg & e_e \\ \hline a & b & c & \tau_1 & \tau_2 & e \end{array}$$

The top row and bottom row respectively identify the log component of the alignments (namely the events), and the process/model component (the Petri-net transitions). To create a model-aligned trace, we need to synthesize the sequence of events. For each synchronous move between an event e and a transition t, we create an event e' such that $\#_{act}(e') = t$, and for any other event attribute a, $\#_a(e') = \#_a(e)$. For each model move for a transition t, we create an event e_t such that only the activity attribute is populated $\#_{act}(e') = t$. These events are then ordered according to the order of the moves in the alignments. This means that, for the trace in question, the model-aligned trace is $\langle e'_a, e'_b, e'_c, e'_{\tau_1}, e'_{\tau_2}, e'_e \rangle$ where the subscript indicates the activity associated with the event, i.e. $\#_{act}(e'_x) = x$. Events e'_a, e'_b and e'_e are also populated with the additional attributes and values that are present for e_a, e_b, and e_e, respectively: for instance, for each attribute v of e_a different from act, $\#_v(e'_a) = \#_v(e_a)$.

Hereafter, the event log that originates from the information systems (i.e. with activity labels) is referred to as **event log**, while the event log defined over model transitions is named **model-aligned event log**. The events of model-aligned traces that stem from synchronous moves take on the attributes and their values from the mapped events of the real event log, including resource and timestamp. The events that come from model moves do not have any attribute but the activity. Note that, strictly speaking, a model-aligned trace is not a repaired trace as discussed in [5]: the activities of model-aligned traces are transitions names, where log traces refer to transition labels.

1.2 Data-Perspective Discovery

The data perspective focuses on how data objects are manipulated by the activities during the execution of process instances. The study of this perspective is of high relevance because the process-instance execution routing is affected by the characteristics of the specific process instance, such as the amount requested for a loan or the profile of the loan requestor, and also by the outcomes of previous steps in the process, such as the results of a verification activity. As an example, let us consider golden and silver profiles of potential loan requestors: a financial institute might decide to treat golden customers via a different procedure than that for other customers. Since the data perspective affects how decisions are made in the process, this perspective is also often referred to as **decision perspective**.

It is nowadays gaining momentum to represent this perspective in an integrated model, e.g., extending a BPMN model, or as a set of separate tables, also known as decision tables. This is also testified by the continous refinement of the Decision Model and Notation (DMN), a standard by the Object Management Group to describe and model decision tables [21].

Historically, the discovery of the data perspective is called *Decision Mining*, a name that was introduced by the seminal work by Rozinat et al. [23]. However, this work could not be applied on Petri nets containing invisible transitions or multiple transitions associated to the same label. This limitation has been lifted in [8] through the use of alignments and the construction of model-aligned tables.

Table 1. A fragment of a model-complaint event log for the model in Fig. 1. The gray events have been introduced as result of alignment model move for invisible transitions. Their case identifier is inherited from the other trace events.

CaseId	Activity	Loan	Loan	Age	Income	Verification	Instalment	Instalment	Resource
		Amount	Length				Amount	DivIncome	
1	a	400000	30	30	2500				Sue
1	b					TRUE			Max
1	c						1225	0.49	John
1	τ_1								
1	τ_2								
1	f								Jennifer
2	a	450000	30	30	3000				Mark
2	c						1380	0.46	Max
2	b					FALSE			John
2	τ_1								
2	d								Sue
3	a	10000	30	71	2500				Mark
3	c						25	0.01	Max
3	b					TRUE			John
3	τ_1								
3	d								Mark
4	a	400000	25	30	2700				Mark
4	b					TRUE			Max
4	c						1458	0.54	John
4	τ_1								
4	d								Sue
5	a	30000	3	30	2500				Sue
5	c						925	0.37	John
5	b					TRUE			Max
5	τ_1								
5	τ_2								
5	e								Anne

The simplest representation of the data perspective is to attach decision rules to decision points. When processes are modelled via Petri nets, decision points are places with arcs to multiple transitions (see, e.g., the red place with thick border and the outgoing arcs in Fig. 1). The rules explaining the choices are driven by additional process data, which are generated by activities/transitions preceding the split. In the remainder, this additional data is abstracted as a set of process attributes, where each attribute can take on a value within the respective attribute domain. For these Petri nets extended with data, a guard over these process attributes is attached to each transition, which can possibly be identically true. A transition is enabled if every incoming place has a token as for classical Petri nets, but also the associated guard needs to evaluatr true wrt. the current value assignment to process attributes. As for classical Petri net, a transition is enabled if every input place has a token; however, in this case, that is only a necessary

Table 2. The observation instances for the model in Fig. 1 and the event log in Table 1 to discover the guards at decision point for place $c5$. The last column indicates the class feature.

Loan Amount	Loan Length	Age	Income	Verification	Instalment Amount	Instalment DivIncome	Transition
400000	30	30	2500	TRUE	1225	0.49	τ_2
450000	30	30	3000	FALSE	1380	0.46	d
10000	30	71	2500	TRUE	25	0.01	d
400000	25	30	2700	TRUE	1458	0.54	d
400000	25	30	2700	TRUE	925	0.37	τ_2

condition: the guard also needs to evaluate true. Other process-modelling notations have equivalent constructs to represent this: for instance, BPMN models use XOR-split gateways, depicted as a diamond with a X symbol inside, and conditions are represented on the arcs going out the gateway.

The basic algorithm for guard discovery assumes that the decisions are mutually exclusive: when a process instance reaches a decision point, one and exactly one branch is enabled for any assignment of values to attributes.

In [8,23], the decision-mining problem is transformed into a classification problem: which transition is expected to fire (namely is enabled) for each valid assignment of values to process attributes. This problem can be tackled through decision-tree learning: decision trees have the remarkable advantage to explicitly indicate the classification criteria, namely which transition is enabled for each assignments of values to attributes.

The intuition can be given through an example related to a process modelled via the Petri net in Fig. 1. A corresponding model-aligned event log is represented conveniently in tabular form in Table 1. Loan lengths are measured in years, attribute *Income* is the monthly salary, *InstallmentAmount* is the amount of each monthly installment, and *InstalmentDivIncome* is the ratio between *InstallmentAmount* and *Income*. Last, *Verification* is a boolean process attribute to which a value is assigned as result of executing the activity *Retrieve Applicant Data* (Petri-net transition b): if the retrieval of applicant data confirms the information provided by applicants through the first activity, attribute *Verification* takes on a false value, and the loan request is going to be rejected. Let us focus on decision point $c5$, which is input place of transitions d or τ_2. It follows that d and τ_2 are mutually exclusive. We need to train a decision tree that define the conditions that discriminate when d or τ_2 is expected to occur, which are going to become the guards of d and τ_2. Table 2 shows the instances to be used to train the decision-tree model. The last column holds the class values of the learning instances, whereas the others columns refer to the independent variables. Since the model does not contain loops, exactly one token is produced in place $c5$, for each process instance. The first row refers to the case with identifier 1: in this case, the transition τ_2 is observed when the following values were assigned to the process variable through the execution of given activities (i.e. model transitions): $LoanAmount = 400000$, $LoanLength = 30$, $Age = 30$, $Income = 2500$, $InstalmentAmount = 1225$, $Verification = TRUE$ and $InstalmentDivIncome = 0.49$. The second row refers

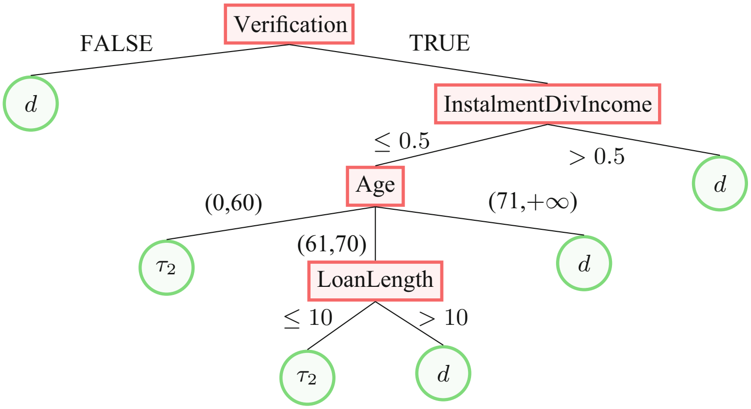


Fig. 2. A possible decision tree that is learned from the observation instances in Table 2.

to the second trace (id 2): in this case, d was observed with $LoanAmount = 450000$, $LoanLength = 30$, $Age = 30$, $InstalmentAmount = 1380$, $Income = 3000$, $Verification = FALSE$ and $InstalmentDivIncome = 0.46$. In case values are assigned to process attributes by different transitions, the latest observed value is considered. Figure 2 shows a possible tree that can be learned from the observation instances in Table 2. The guards can be extracted from the decision tree, traversing the paths from the room to leaves. The guard for any transition t is in the form of $expr_1 \vee \ldots \vee expr_n$ where $expr_i$ refers to the i-th path that leads to a leaf labeled as t, and is a conjunction of atoms $variable\ operator\ constant$ (e.g., $age \leq 60$ or $Verification = FALSE$) of the nodes and arcs part of the path.

As an example, the guard of transition d (activity *Notify Rejection*)is an expression with four subexpression: $expr^d_1 \vee \ldots \vee expr^d_4$. Sub-expression $expr^d_1$ refers to the path for the left-most node, which includes the root node $Verification$ and the edge associated with label $FALSE$, lead to expression $Verification = FALSE$; $expr^d_2$ refers to the second left-most node with label d:

$$Verification = TRUE \wedge InstalmentDivIncome \leq 0.5 \wedge 61 \leq Age \leq 70 \wedge LoanLength > 10$$

and etc. Considering the four paths in the decision tree, the guard for d is as follows:

$$\begin{array}{l} Verification = FALSE \\ \vee(Verification = TRUE \wedge InstalmentDivIncome \leq 0.5 \wedge 61 \leq Age \leq 70 \wedge LoanLength > 10) \\ \vee(Verification = TRUE \wedge InstalmentDivIncome \leq 0.5 \wedge Age \geq 70) \\ \vee(Verification = TRUE \wedge InstalmentDivIncome > 0.5) \end{array}$$

That is a disjunction of conjunction of terms related to paths from the root to the leaves labeled with d. One can similarly obtain the guard of τ_2, which is the disjunction of two expressions:

$$\begin{array}{l} (Verification = TRUE \wedge InstalmentDivIncome \leq 0.5 \wedge Age \leq 60) \\ \vee(Verification = TRUE \wedge InstalmentDivIncome \leq 0.5 \wedge 61 \leq Age \leq 70 \wedge LoanLength \leq 10) \end{array}$$

Missing Values

Let us consider again the model-aligned event log in Fig. 1, and suppose that the event for transition b in the trace with case identifier 1 comes from a model move. This means that, in the real event log before computing alignment, an event for b in the first trace was not observed. In such a case, the fact that transition b assigns a TRUE value to *Verification* is lacking. As a consequence, the first decision-tree training instance has a missing value for attribute *Verification*. Several techniques exist for the management of missing values of a given variable f, such as:

1. assigning the most common (if categorical) or the average value (if numerical) among those observed for f in the other training instances,
2. similarly as above but restricting to the instances with the same class value (i.e. the same transition in our setting),
3. creating one training instance for each of the n observed values - say value v - and assigning value v to the corresponding instance, with a weight that is equal to the number of instance with value v for f divided by the number of instance with some value for feature f.

Several implementations of decision-tree learning algorithms (e.g., of C4.5 [22]) are already equipped with the missing-value managements. However, it is important to think carefully about the meaning of missing values. It might be - as many schemes implicitly assume - that the value was produced but, for quality issues, was not recorded in the dataset. However, it could also mean that the transition did not produce that value, due to, e.g., some concept drift or impossibility to find a suitable value for the specific instance in question. When this is true, *the missing value conveys important information*, and the learning instance should carry the information that the value was missing via an additional boolean feature, instead of injecting random values. This additional feature can increase the discriminative power of the guards, differentiating the situations in which the information was provided from those in which the information was missing.

1.3 Organizational Mining

Organizational Mining focuses on the resources, which refer to anyone or anything involved in performing activities, such as a human process participant, a software system (e.g., a server) or an equipment (e.g., a production machine). The organization perspective, also referred to as *resource perspective*, aims to model how resources are grouped, ad how they interact to each other.

Among different goals, organizational mining aims at how resources collaborated to carry on individual process instances. Typically, the resource collaborations can be represented as *social networks*, which are graphs where nodes are the resources and arcs, direct or indirect, indicate some form of collaboration between pairs of resources [25,26]. Arcs can be also given weights, which is proportional to the frequency/intensity of the collaborations. One of the most studied social networks in Business Process Management relates to the hand-over of work between resources. A work hand-over between two resources a and b exists in a process instance p, if a has executed

	a	b	c	d	e	f
Anne					1	
Jennifer						1
John		0.4	0.6			
Mark	0.6			0.2		
Max		0.6	0.4			
Sue	0.4			0.4		

(a)

Role	Resources	Activities
R1	Anne, Jennifer	e, f
R2	John, Max	b, c
R3	Mark, Sue	a, d

(b)

Fig. 3. An example of application of role discovery for the process referring to the log in Fig. 1. Table (a) is the resource-activity matrix, where colors are used to define a reasonable grouping of rows, i.e. resources. When no value is depicted for a cell, it should be intended as zero. Table (b) details the discovered roles. Note that the role name cannot be automatically derived.

an activity for p, which is directly followed by a second activity that is executed by b. This implies that a has handed over the progression of the execution of p to b, which, in turn, can later hand it over to another resource. Among the different goals, organizational mining aims to discover these social networks, and later to analyze them. Social network analysis is very interesting in Organizational Mining because it can unveil relevant information about resources. Notably, it can discover *cliques* of resources that tend to work together, or critical resources that are less "replaceable". Less replaceable resources are characterized by a large degree of incoming and outgoing arcs, and the removal of the corresponding nodes in the graph may create longer paths between pairs of resources, or even yield disconnected components.

Space consideration prevents us from further discussing social-network analysis, and forces us to rather focus on analyzing the event logs to discover *roles*, groups of resources that work on the same activities. Clustering techniques are simple techniques to discover roles within organizations, especially under the assumption that a resource plays one single role. The starting point is to build a resource-activity matrix, such as that in Fig. 3(a). Rows refer to different resources and columns to different activities. The value for the row r and column a indicates the average number of times that r executes a in a process execution. For instance, Mark executes activity a 0.6 times per case, on average. Note that, if an activity is executed exactly once per process instance, the sum of the values of the cells of the corresponding column is one. A sum lower or higher than one indicates an activity to be optional or be involved in a loop.

In a resource-activity matrix, each row is a different resource and can be regarded as a vector with as many dimensions as the number of process activities: the value of the dimension for a given activity is equal to that of the corresponding cell in the matrix. Rows are thus vectors, points of a cartesian space, that can be clustered, e.g., via well-known clustering algorithms, such as K-Means or DBScan [19]). The row colors in Fig. 3(a) illustrates a reasonable clustering for the matrix in question. As an example, Mark and Sue belong to the same cluster and, hence, play the same role: their role allows them to perform activities *a* and *d*. The same is for John and Max, who can perform *b* and *c*. Anne and Jennifer form the third role that enables them to perform *e* and *f*. Note that it would be equally reasonable to split the cluster with Anne and Jennifer into two, although a simpler solution with fewer role is possibly preferable when equivalent.

1.4 Time Perspective

A process-instance execution can takes a considerable amount of time to be carried out. Depending on the domain, it might even take months or years to conclude: consider, e.g., a health-care process to follow up cancer diagnoses, or that to give monthly unemployment benefits, or even a process to reintegrate workers who have suffered physical issues that prevent them from going back to their original employment. It follows that process activities are not instantaneous as we have so far considered, but they take some time to be executed. In fact, certain activities require external inputs (e.g., the production of documents, the arrival of materials and other goods), and the availability of necessary machines and suitable human resources. If these requirements are not met at the moment when the activities are ready to be started, their execution is forcibly delayed. These delays can have a cascading effect on other activities that follow in the process.

Within the realm of Process Mining, the time perspective focuses on the timing of events that carry timestamp information. The time-perspective analysis can notably be used to discover process bottlenecks, and monitor the service levels: their analysis enables verifying whether executions are carried on within a reasonable amount of time (e.g., a complaint is addressed within the same day in which it is filed), or whether the temporal process constraint are fulfilled (e.g., the second shot of the COVID-19's Pzifer vaccine is given within 21 days from the first). The analysis of the perspectives on time and resource is also partly overlapping: thanks to the time information, process analysts can assess, for instance, whether resources are fairly, overly, or scantily utilized.

The verification of the satisfaction of time-related constraints is related to conformance checking (see [5]) As an example, Mannhardt and Blinde illustrates an interesting case study to check the conformance of the treatment of patients who suffered from Sepsis [18]. The conformance checking of time perspective is not covered in this chapter, which conversely focuses on extending and annotating a process model to unveil potential time-related issues, especially process' bottlenecks.

The presence or absence of bottlenecks can be related to *(i)* waiting time, namely the difference between the timestamp of the actual start of an activity instance and the earliest moment in which the instance could have started (cf. above discussion of delays caused by lack of resources), or *(ii)* service time, i.e. the duration of an activity-instance execution.

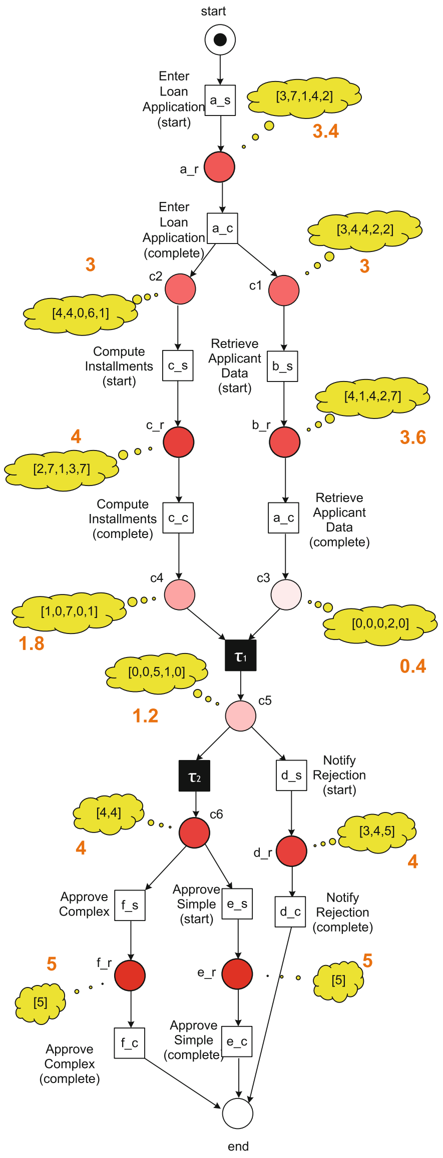
Several ways exists to analyse the service and waiting times of the activities of a process model, e.g. modelled via Petri nets. The performances at the different points of the model can be analyzed through *queue mining* [24]. For instance, queue mining can be employed to estimate how long a token typically remains unconsumed in a Petri-net place. This estimation is far from being easy because it requires to consider several factors: the average length of the token queues in places, the policy of consumptions of tokens (FIFO or according to some priorities), the relationships between places (e.g., connected to the same transition), etc. Queue mining considers the process model as a queuing network, whose characteristics are determined after analysing the information stored in event logs. A queuing network is used to determine the activity execution policies. When a queue network is created, several off-the-shelf techniques can be employed for its analysis.

Space limitation forces this chapter to only focus on a simple technique based on the Petri-net token-replay game: real-log traces are transformed in model-aligned traces that are replayed on the Petri-net model to collect waiting and service times. The transformation to model-aligned traces ensures that they are replayable on the model. However, the firings of Petri-net transitions are atomic by definition, and hence their execution take no time. This is clearly not realistic, and requires to explicitly model the starting and completion of activity instances are two separate Petri-net transitions. This explicit modelling can be simply explained through our working example of the process modelled as in Fig. 1.

Each visible transition is split into the sequence of two transitions that model the starting and completion of activity instances, yielding the Petri net in Fig. 4(a). For instance, activity *Enter Loan Application* is now represented through two transitions, named *a_s* and *a_c*, which respectively fire when instances of that activity starts or completes. We aim to play the token game: this means that transition *a_s* fires upon a start event for activity *Enter Loan Application*, and *a_c* upon a complete event for the same activity. This means that, when a token is present in the places named $a\_r, \dots, f\_r$, it indicates that the activity associated with transitions $a\_s, \dots, f\_s$ are being executed, respectively. Note that there is no need to split invisible transitions: they are necessary for modelling purposes, and do not represent an actual activity, and hence can be considered as instantaneous. As mentioned earlier, the real event log need to be translated into a model-aligned event log that can be directly replayed on the Petri net of the process model, and alignment techniques are used for this purpose. In the scenario in which events refer to either the starting or the completion of the activities, the two transitions that indicates the starting or completion of any activity *x* need both to be mapped to events for *x*, but the first to events related to the starting of *x* and the second to events related to the completion of *x*. For the model in Fig. 4(a), transition *a_s* is mapped to events related to the starting of *Enter Loan Application*, and *a_c* to events related to the completion of *Enter Loan Application*.

After computing the alignments with this mapping, it is possible to synthesize the model-aligned event log in Fig. 4(b). Gray rows refer to the firing of invisible transitions: in that case, the timestamp of the associated events is assigned to be equal to the earliest moment in which the transition could fired. Consider transition τ_1 and the first trace: τ_1 can fire when both transitions b_c and c_c have fired, b_c fires at time 11 and c_c at time 10 for the first trace, and consequently the earliest moment in which τ_1 can fire is at time 11.

Each trace of the model-aligned event log can be replayed on the Petri net. This allows computing the amount of time in which a token resides in a given place, i.e. the difference between the timestamp in which the token was consumed and the timestamp when it was produced. For example, consider the place a_r: tokens are produced in that place when transition a_s fires and are consumed when transition a_c fires. For trace



(a)

Case Identifier	Activity	Timestamp
1	a_s	1
1	a_c	4
1	b_s	7
1	b_c	11
1	c_s	8
1	c_c	10
1	τ_1	11
1	τ_2	11
1	f_s	15
1	f_c	20
2	a_s	2
2	a_c	9
2	c_s	13
2	c_c	20
2	b_s	13
2	b_c	14
2	τ_1	20
2	d_s	20
2	d_c	23
3	a_s	2
3	a_c	3
3	c_s	3
3	c_c	4
3	b_s	7
3	b_c	11
3	τ_1	11
3	d_s	16
3	d_c	20
4	a_s	20
4	a_c	24
4	b_s	26
4	b_c	28
4	c_s	30
4	c_c	33
4	τ_1	33
4	d_s	34
4	d_c	39
5	a_s	18
5	a_c	20
5	c_s	21
5	c_c	28
5	b_s	22
5	b_c	29
5	τ_1	29
5	τ_2	29
5	e_s	33
5	e_c	38

(b)

Fig. 4. An example of extending the process model with the time perspective. The left-hand side picture shows how the process model in Fig. 1 can be annotated with temporal information wrt. the model-aligned log shown in the right-hand side table. The gray lines in the table are the events related to invisible transitions.

with case identifier 1, a_c and a_s respectively fired at time 4 and 1, thus the difference is 3. The residence of each token in each place can be computed by replaying the model-aligned event logs: these timestamp differences are shown within the clouds associated to the different places in Fig. 4(a). The average per place can subsequently be computed, which is shown next to the respective cloud. One can red color each place with a color intensity that is proportional to the mean value of time: white is associated to an average of zero, and the color becomes closer and closer to dark red as the average is closer and closer to the largest observed value.

Considering place a_r again, the average time is 3.4 for the event log in Fig. 4(b). This indicates that the average duration of instances of activity *Enter Loan Application* is 3.4 time units (e.g., hours). Consider place $c1$: tokens are produced in the place after the completion of the same activity and consumed when transition b_s fires, namely when activity *Retrieve Applicant Data* starts. For the first trace, the amount of time a token is $c1$ is 3 time units, namely the timestamp of the event for b_s, which is seven for first trace, minus the timestamp of the event for a_c, i.e. 4. After collecting the times for each token in $c1$ for all traces (see the cloud connected to the place) and computing the average, one can conclude that the average time between the starting of activity instances of *Retrieve Application Data* and the completion of the corresponding instance of the preceding activity *Enter Loan Application*.

Dealing with Non-compliant Traces and Missing Timestamps

So far, we have assumed that *(i)* activity executions leave trails in log through both start and completion events, and *(ii)* every trace is compliant with the model. In particular, assumption *(ii)* means that the model-aligned traces only include additional events related to firing of invisible transitions. These assumptions do not always hold in reality: event logs often only contain the events related to the completions of activity instances, and some traces are not fully compliant with the model (cf. the Conformance Checking field discussed in [5]).

Assumption (i) *is not met.* In this case, one can employ a naïve approach that assumes that the next activity in the process starts as soon as the previous completes: in this case, the timestamp of the starting event is the same as the timestamp of the completion event of the activity that precedes. This is often unrealistic, as pictorially depicted in Fig. 5. In the timeline, *Completion of a* indicates the moment in which activity instance a completes and *Real start of a* is the actual moment in which a started, which has left no trail in the event log. Moment *Completion of the activity instance preceding a* is when the previous activity concluded. The time difference between *Completion of the activity instance preceding a* and *Real Start of a* corresponds to the waiting time of a. If this time difference is set to 0, no waiting time is assumed. A better estimation can be obtained if the event log contains information about the resource perspective: one can look at the completion event of a given activity instance a and consider the resource r that performed the instance: the starting timestamp of the activity instance is equal to the earliest moment after the completion of the activity instance that precedes a in which r has completed any activity instance and has become available [20].

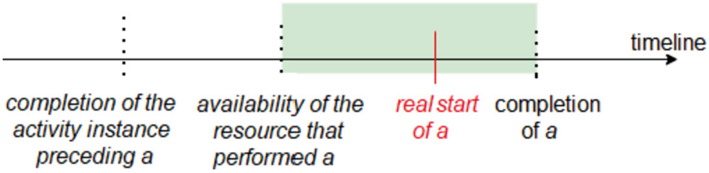

timeline
completion of the activity instance preceding a
availability of the resource that performed a
real start of a
completion of a

Fig. 5. Representation of the scenario when the timestamp of the start event of an activity instance a is not present in the event log and needs to be estimated. This timestamp is located between the timestamp when a completes and the earliest timestamp when the resource r that is going to perform a is available to start a. This time interval is represented through a green area, and the real start of a, which is unknown, is located within the area. In case the resource information is missing, we do not even have the earliest timestamp of availability of r: this introduces further uncertainty, because we can only rely on the timestamp of completion of the activity instance that precedes a in the trace.

This corresponds to the moment in figure labelled as *Availability of the resource that performed a*: this introduces some waiting time, namely the time difference between *Completion of the activity instance preceding a* and *Availability of the resource that performed a*, thus being more realistic. The latter case is still often unrealistic in practice [11]: *(a)* resources work on multiple processes and continuously switch from one to the other while event logs refer to one process, *(b)* take breaks during the working days (e.g., when tired), *(c)* carry on additional duties that lead no trail in the event logs (e.g., when answering the phone). Let us consider Fig. 5 again: the actual start is in a moment between when the resource has become available and when the activity instance has been completed. The choice of estimating different start moments leads to estimating different activity-instance durations. In [14], Fracca et al. proposes a technique to estimate the starting event where different activity-duration configurations are simulated, and the resulting simulated event log is compared with the real event logs to assess the similarity with respect to time-related aspects (activity-instance waiting times and process-instance durations): the more similar are the real and simulated event log is, the more realistic are the estimation of activity instance durations. The simulation of different activity-duration configurations requires a simulation model, which consists of a process model that is extended with additional information related to the simulation aspects, such as the inter-arrival time, the routing probabilities at the XOR gateways, the roles and the resource-activity allocation, potential work calendar, and more. The simulation model can be constructed by combining different process mining techniques, as also discussed in [14].

Assumption (ii) *is not met.* This can be clearly caused by not meeting the assumption *(i)*: the starting events are missing, yielding model moves for every Petri-net transition linked to the starting of activity instances. We consider the situation hereafter in which assumption *(i)* is met. In this case, the deviations are related to the activities that have not been performed in accordance to the process model. In this case, both the starting and completion events are missing. If the number of non-compliant traces is limited, these can be excluded from the analysis. Otherwise, the log traces are aligned to create model-aligned traces, without adding the timestamps to the events that come from model moves for visible transitions: in this case, statistics are computed for reliability by only considering pairs of subsequent events that have a timestamp associated.

2 Process Extension: Advanced Techniques

This section introduces some advanced techniques to overtake the limitations of the basic algorithms for decision mining and for role discovery: In particular, the basic algorithm for decision mining introduced in Sect. 1.2 is only able to discover with atoms of form *var-op-const* where *var* is a variable, *op* is a comparison operator and *const* is a constant (e.g. $Age \leq 60$ or $Verification = FALSE$), while the basic algorithm for role discovery in Sect. 1.3 assumes a resource to be able to play one single role, only. Sections 2.1 and 2.2 discussed some advanced techniques that aim to overcome these limitations.

2.1 Data-Perspective Discovery of Guards with Variable Comparison

Let us consider a variant of the event log in Table 1 where *InstalmentAmount* is present but attribute *InstalmentDivIncome* is missing. As mentioned above, the basic guard-discovery algorithm will be unable to discover guards that include an atom $^{InstalmentAmount}/_{Income} > 0.5$, or its negation. The work by de Leoni et al. [7] reports on an extension to the basic algorithm that can discover atoms of form *var-op-var*, such as $InstalmentAmount > 0.5 \cdot Income$.

The algorithm builds on some oracle that discovers invariants in a set of observation instances, such as the Daikon system [6]. Analogously to the basic algorithm, the algorithm is applied for each place p of the Petri Net modelling a process, and consists of five steps:

1. The basic algorithm in Sect. 1.2 observation instances Ψ for p (e.g., those in Table 2).
2. For each transition t_i with outgoing arcs to p, we extract the observation instances $\Psi_{t_i} \subset \Psi$ with class t_i (e.g., those in Table 2 with transition\class τ_2). In fact, $\Psi_{t_1}, \ldots, \Psi_{t_n}$ is a clustering of Ψ where $t_1, \ldots, t_n$ are the transitions with outgoing arcs to p.
3. A set V_{t_i} of invariants that hold in Ψ_{t_i} is computed (for instance, $^{InstalmentAmount}/_{Income} > 0.5$).
4. For each invariant $v \in V_{t_1} \cup \ldots \cup V_{t_n}$, one boolean feature f_v is added to every observation instance $\psi \in \Psi$, and it takes on a true or false value depending whether invariant v holds with the variable-to-value assignment defined in ψ. For instance, invariant $^{InstalmentAmount}/_{Income} > 0.5$ holds in the fourth instance in Table 1, and does not for the others (recall that attribute *InstalmentDivIncome* is assumed to not be present).

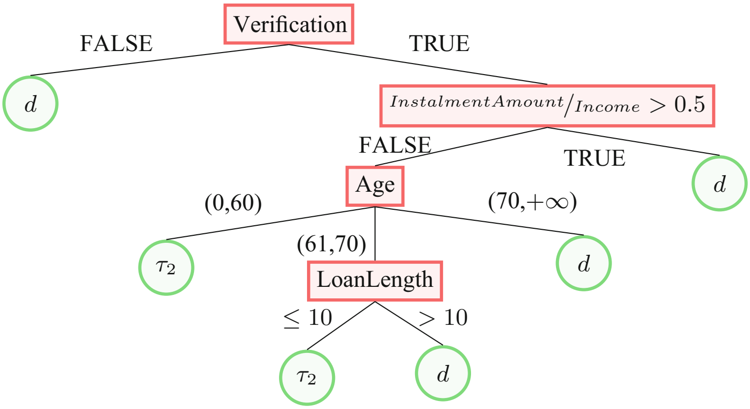


Fig. 6. The possible decision tree that is learned from the observation instances in Table 2 augmented with boolean features related to discovered invariants, such as $^{InstalmentAmount}/_{Income} > 0.5$.

5. A decision tree is trained using the set of augmented observation instances.

For the working example, such a decision tree as in Fig. 6 is learnt: the invariant is now able to discriminate between the instances of d and of τ_2.

2.2 Discovery Roles with Overlapping Resources

The basic organization-mining technique discussed in Sect. 1.3 relies on clustering, and thus assumes each resource to play exactly one role. In many settings, this assumption does not hold: resources can associated with multiple roles. Burattin et al. [4] lift this assumption, by clustering activities instead of resources: the clustering puts together the activities that require to be executed by resources playing the same role.[1] The starting point is a process model and its dependencies of form $a \rightarrow b$, i.e. activity b can follow a but a cannot follow b. Clustering is obtained by removing all the dependencies $a \rightarrow b$ for which the handover is larger than a given threshold τ^w:[2]

Definition 2 (Resource Handover for a Model Dependency). *Let $a \rightarrow b$ be the dependency between two activities a and b. Let L be an event log and $\mathcal{R}_{a \rightarrow b} = \biguplus_{\sigma \in L} \biguplus_{\langle e_i, e_j \rangle \in \sigma . \#_{act}(e_i) = a \wedge \#_{act}(e_j) = b} (\#_{res}(e_i), \#_{res}(e_j))$ be the multiset of pairs of resources in L where the first resource executes a and is immediately followed by the second resource executing b. Let $\mathcal{R}^a_{a \rightarrow b}$ and $\mathcal{R}^b_{a \rightarrow b}$ be the projection over the first and*

[1] The terminology and formalization used hereafter slightly different those in [4], to harmonize with the rest of the chapter.

[2] Given two multisets X and Y, the interection $X \cap Y$ returns a multiset that contains every element z present in X and Y with the lowest cardinality for z between that of X and of Y. Symbol $\uplus$ indicates the union of multisets: the cardinality of each element in the union of two multisets X and Y is equal to the sum of the cardinalities of the element in X and in Y. Given a sequence σ, a second sequence $\sigma' \in \sigma$ if σ' is a sub-sequence of σ.

second component of $\mathcal{R}^a_{a\to b}$, respectively. Let $\mathcal{R}^=_{a\to b}$ be the pairs with the same resource value on both components. The resource handover for dependency $a \to b$ for L is defined as follows:

$$w_{ab}(L) = 1 - \frac{|\mathcal{R}^a_{a\to b} \cap \mathcal{R}^b_{a\to b}| + |\mathcal{R}^=_{a\to b}|}{|\mathcal{R}^a_{a\to b}| + |\mathcal{R}^b_{a\to b}|}$$

The definition states that $w_{ab}(L)$ is closer and closer to zero if it is more and more frequent that two activities a and b are performed by the same resources. If activities belong to the same cluster, the resources that perform them can play the same role.

As an example, let us consider the dependency $a \to c$ for the model in Fig. 1 and the log in Table 1. It follows $\mathcal{R}_{a\to c} = [(Mark, Max)^2, (Sue, Anne)^1]$ where the superscript indicates the cardinality; hence, $\mathcal{R}^a_{a\to c} = [Mark^2, Sue^1]$ and $\mathcal{R}^c_{a\to c} = [Max^2, Anne^1]$. Therefore, the resource handover for the dependency is $w_{ac}(L) = 1$. Value 1 is obtained when the set of resources are totally disjoint, as the case is for a and c: the dependency is hence removed, making a and c belong to different clusters. Repeating the reasoning on dependency $a \to$, we obtain $w_{ab} = 1$, thus causing a and b to belong to different clusters.

Ultimately, this means that activity *a* is a cluster with only itself. However, Fig. 3(b) shows that activities a and d should belong to the same cluster, so as to add the performing resources to the same role. However, this cannot happen if we only look at the dependencies because there is no dependency $a \to d$, or vice versa. Therefore, after partitioning the activities, some clusters can be merged. This occurs if the so-called merging degree is larger than a given threshold τ^ρ:

Definition 3 (Merging Degree). *Let $A_1 = \{a_{1,1}, \ldots, a_{1,n}\}$ and $A_2 = \{a_{2,1}, \ldots, a_{2,n}\}$ be two activity clusters. Let L be an event log. For any set A of activities, let us denote the multiset of resource executing activities in A with $\mathcal{R}_A = \biguplus_{\sigma\in L} \biguplus_{e\in\sigma:\#_{act}(e)\in A} \#_{res}(e_i)$. The merging degree of A_1 and A_2 is defined as:*

$$\rho_{A_1,A_2}(L) = 2\frac{|\mathcal{R}_{A_1} \cap \mathcal{R}_{A_2}|}{|\mathcal{R}_{A_1}| + |\mathcal{R}_{A_2}|}$$

Similarly to Definition 2, this measures the amount of shared resources between those that execute two sets A_1 and A_2 of activities. If $\rho_{A_1,A_2}(L) > \tau^\rho$, A_1 and A_2 are merged.

In conclusion, the algorithm to discover roles where resources belong to multiple is as follows:

1. Select the resource-handover threshold τ^w and the merging threshold τ^ρ.
2. For each model dependency $x \to y$, compute the handover w_{xy}.
3. Remove every dependency $x \to y$, such as $w_{xy} < \tau^w$.
4. Cluster the activities according to the retained dependencies.
5. Merge two activity clusters A_1 and A_2, if the merging degree $\rho_{A_1,A_2} > \tau^\rho$. Note this step can be applied recursively to merged clusters until no further merging.
6. For each final cluster $\overline{A}$, a role is created that contains every resource that has performed an instance of any activity in $\overline{A}$. A further threshold can be defined to discard resources that seldom perform activities in $\overline{A}$.

It is worthwhile reflecting that the actual relevant values for the resource-handover threshold τ^w are limited to the set of handover values w_{xy} computed for each dependency $x \rightarrow y$. Given that the number of dependencies is finite and usually small, it is possible to extensively apply the role discovery setting τ^w iteratively to every value w_{xy} where $x \rightarrow y$. This enables process analysts to evaluate the different configuration and determine the most realistic role set, using business knowledge. Also, once a value is set for τ^w and the clusters are created, one can similarly reason for τ^ρ: the number of values to test is finite, i.e. the values $\rho_{A',A''}(L)$ for each pairs (A', A'') of clusters at step 4.

3 Process Improvement

Process analysts and certain stakeholders (e.g., CEOs) may oftentimes have partial or helicopter-like view on the organizations in which such process are executed. As a consequence, the process models that they have in mind (also known as "to-be" models) may not summarize how processes are *really* executed by resources. In these cases, such "to-be" models are of limited use. Improvement can be regarded as altering the model so that it reflects reality (i.e., improvement on fitness) while ensuring the other quality criteria remain within a certain reasonable range (i.e., precision, generalization and simplicity). The result is an "as-is" model that show how the process is *really* executed. Section 3.1 details how models can be improved on fitness.

However, if models are used to prescribe how processes ought to be executed, they should only represent the behavior with which the organization is satisfied. If the model contains portions of behavior that lead to unsatisfactory outcomes (high costs, low customer satisfactions, etc.) or that violate norms and regulations, one would like those portions to be disallowed by the model. Section 3.2 details how models can be improved to ensure no regulation violations and to incorporate behavior that has proven to yield good performance levels.

The classical problem of process discovery discussed in [2,3] and that of process improvement share some commonalities in that they both aim to come up with "as-is" models. The difference lays on the fact that the problem of process discovery is largely unsupervised (little or no knowledge is fed in), while process improvement is supervised: an original model is provided, which constitutes the initial "backbone" that is later altered to obtain a "as-is" model. It follows naturally that process improvement is generally able to produce better models because the original model encodes behavior that is deemed appropriate from a business viewpoint. This reasoning is especially valid when the original model is hand-designed by or in concert with process owners.

The remainder of this section will use the same working example that was used in Sects. 1 and 2, namely the process modelled in Fig. 1. However, hereafter we will differently assume that the activity names in the real event log coincides with the transition names $a, \ldots, f$, to keep the discussion simple. Also, since we discuss techniques for process improvement that only consider the control flow, traces will be considered as sequences of activities, which coincide with transition names for the considerations above (i.e., a simple formulation)

3.1 Model Repair to Reflect Reality

The problem of repairing a process model M to reflect the reality recorded in a log L can be formulated as finding a process model M' that is able to replay each trace in L and is the closest possible to M (i.e. with the minimum number of changes). Note that, if M can replay L, $M' = M$. This section focuses on the case in which M and M' are accepting Petri nets, and the goal is to repair models wrt. the control-flow perspective, thereby ignoring the other perspectives. This formulation suggests that model repair primarily aims at perfect fitness, generating a set of models with optimal fitness. Within this set, the final choice refers to any model that best balances simplicity, precision and generalization (cf. the conformance-checking problem discussed in [5]).

The assumption here is that the repaired model must be able to replay every trace in event log L. However, event logs may record executions that are outliers or refer to process instances that were still running at the moment of the extraction of the event log. Those traces should not be allowed by the repaired model M'. Hereafter, we however assume that every trace that should not be replayed by M' is already filtered out from L before applying the model-repair algorithm on L.

This section reports on the repair technique discussed in [13], whose basic intuition can be given via the following example. Let us consider again the model in Fig. 1 and the following event log $\overline{L} = [\langle a, g, w, b, c, d\rangle, \langle a, w, g, b, c, e\rangle]$ where g and w are the shortcut names of two new activities: e.g. *fix application* and *add witnesses* respectively. These two activities are not part of the model, and thus cannot be replayed on the model. It follows that the model needs to be executed to add some transitions labelled g and w to make the model compliant with $\overline{L}$. The model-repair algorithm needs to determine the point in which the two transitions should be included, namely which places are in the presets and postsets of these transitions. The technique discussed in [13] aims to address this question by aligning the original model M and each of the traces in $\overline{L}$. Optimal alignments for the traces in $\overline{L}$ wrt. the model in Fig. 1 are:

$$\gamma_1 = \begin{array}{|c|c|c|c|c|c|c|}\hline a & g & w & b & c & \gg & d \\ \hline a & \gg & \gg & b & c & \tau_1 & d \\ \hline [c1, c2] & & & [c3, c2] & [c3, c4] & [c5] & [end] \end{array}$$

$$\gamma_2 = \begin{array}{|c|c|c|c|c|c|c|c|}\hline a & w & g & b & c & \gg & \gg & e \\ \hline a & \gg & \gg & b & c & \tau_1 & \tau_2 & e \\ \hline [c1, c2] & & & [c3, c2] & [c3, c4] & [c5] & [c6] & [end] \end{array}$$

Here, the third alignment rows indicate the marking of the Petri net after each synchronous or model move. As usual, the model moves for τ_1 and τ_2 are not considered deviations, and hence do not need to be taken into account when repairing the model. In both of alignments, the actual deviations consist in a sequence of two log moves for activities g and w, namely related to log sub-traces $\langle g, w\rangle$ and $\langle w, g\rangle$. These sequences of two log moves (and the corresponding sub-traces) both occurred when the Petri-net model was at marking $[c1, c2]$. The model needs to be repaired so that the log-move sequences would be replaced in the respective alignments by sequences of synchronous moves.

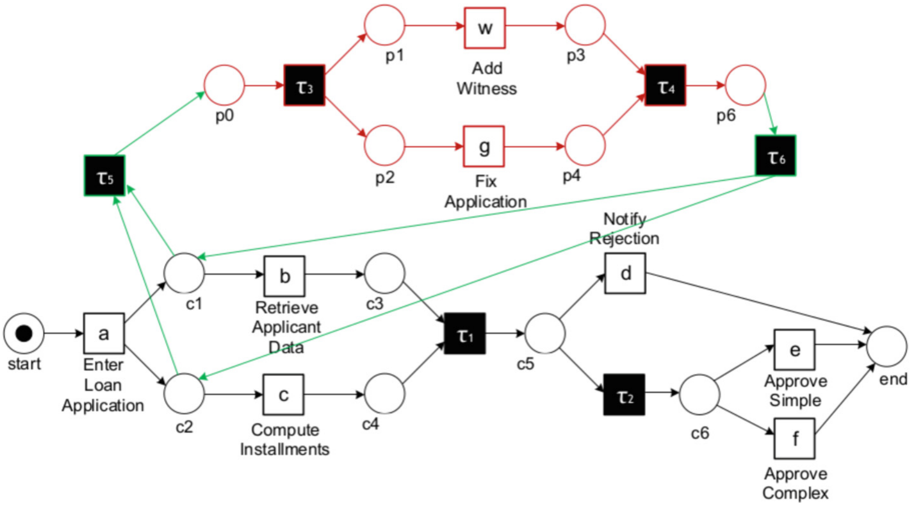


Fig. 7. The model in Fig. 1 repaired by adding the parts in red and green to allow for executions $\langle a, g, w, b, c, d\rangle$ and $\langle a, w, g, b, c, e\rangle$.

The two sub-traces $\langle g, w\rangle$ and $\langle w, g\rangle$ can in fact be regarded as an event log, which can be given as input to some process-discovery techniques to mine a model. If we employed the Inductive Miner, the model would be similar to the Petri net marked through a red border in Fig. 7: transitions *w* and *g* are modelled in a parallelism. Markings $[p0]$ and $[p6]$ are respectively the initial and final marking. Since $\langle g, w\rangle$ and $\langle w, g\rangle$ were observed at marking $[c1, c2]$, marking $[p0]$ needs to be reachable from $[c1, c2]$ without firing any transition: this is modelled via the invisible transition τ_5. Furthermore, when reaching marking $p6$, the execution should be able to reach back marking $[c1, c2]$, motivating the introduction of invisible transition τ_6.

The example above helps introduce the algorithm to repair an accepting Petri net AN with respect to a log L:

1. The traces in L are aligned, thus discovering sequences of log moves.
2. For each maximal sequence $\overline{\gamma}$ of log moves (e.g., the sequence of log moves for g and w in the above example), we determine the marking $m_{\overline{\gamma}}$ of AN when $\overline{\gamma}$ was observed (e.g., $[c1, c2]$ in the above example), and we synthesize the sequence $\sigma_{\overline{\gamma}}$ of activities involved in $\overline{\gamma}$ (e.g., $\langle g, w, \rangle$). The pair $(m_{\overline{\gamma}}, \sigma_{\overline{\gamma}})$ is added to a multiset P of pairs that consist of markings, and sequences of events observed in the log but not allowed by AN at those markings.
3. The pairs P discovered at previous point are grouped by marking so as to obtain a set P_L of pairs where each pair consists of *(i)* a markings m and a log $L = \biguplus_{((m',\sigma)\in P\ :\ m'=m)} \sigma$ that contains the sequences of events observed at m.
4. For each $(m, L_m) \in P_L$, an accepting Petri net $\overline{AN}$ is discovered using L_m as input event log. Petri net $\overline{AN}$ is merged with AN at marking m as discussed in the example above (cf. the green part in Fig. 7).

The algorithm above only considers the log moves, which are linked to sequences of activities that need to be allowed by the repaired model. Of course, the alignment may also point out sequences of model moves, namely sequences of activities that were expected in an observed process instance but not observed. The repaired model should make these expected, but unobserved sequences as optional. As an example, let us consider the model in Fig. 7, which has already been repaired wrt. the sequences of log moves in the alignments. Let us suppose the event log to contain traces related to applications that are desk-rejected because of their clear incorrectness: these correspond to traces consisting of two events $\langle a, d\rangle$. The corresponding alignment would then be as follows:

$$\gamma_1 = \begin{array}{|c|c|c|c|c|} a & \gg & \gg & \gg & d \\ \hline a & b & c & \tau_1 & d \\ \hline [c1, c2] & [c3, c2] & [c3, c4] & [c5] & [end] \end{array}$$

It contains a sequence of three model moves. The repaired model should be such that those three model moves are no more necessary. This can be easily tackled, and one can insert an invisible transition that consumes one token in $c1$ and one in $c2$, i.e. the places containing tokens before the first model move (i.e., before b), and produces a token in $c5$, the place containing one token after the last model move (i.e. after τ_1).

Advanced Repair for Higher Precision and Simplicity

The repairing algorithm discussed above is largely focusing on fitness, thereby overlooking the other dimensions. In fact, the procedure above can have a negative influence on precision and simplicity, because it may allow additional behavior, and increase the size of the model.

Higher model precision can be obtained by removing transitions that seldom appear. In a nutshell, the event-log traces are aligned with the model. For each transition t in the model, we count the number of occurrence of synchronous or model move for t in all the alignments. If this is smaller than a user-defined threshold, t is removed, along with every arc that goes in or comes out from t. The procedure can cause some places to have no more incoming or outgoing arcs: these places are removed, as well.

Model simplification can be achieved as an a-posteriori step, e.g., using the technique proposed by Fahland et al. [12], which aims to simply the model, while preserving the same behavior and well balancing generalization and precision [12]. However, simplification can partly be achieved during the repair, e.g. in case of structured loops [13]. Let us consider the model in Fig. 1 and an event log consisting of two traces $\sigma_1 = \langle a, b, c, r, b, c, d\rangle$ and $\sigma_2 = \langle a, b, c, r, c, b, e\rangle$ where r is the shortcut for a new activity *Ask for Additional Documents* to, e.g., enable a more thorough assessment. The alignments of the two traces are as follows:

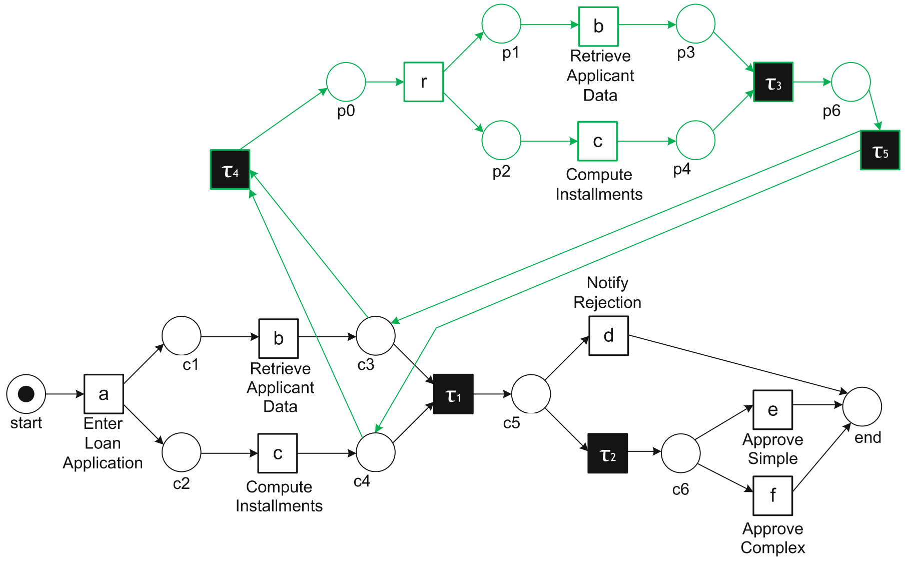


Fig. 8. Repair of the model in Fig. 1 to allow for $\langle a, b, c, r, b, c, d\rangle$ and $\langle a, b, c, r, c, b, e\rangle$, using the basic model-repair technique.

$$\gamma_1 = \begin{array}{|c|c|c|c|c|c|c|c|}
a & b & c & r & c & b & \gg & d \\ \hline
a & b & c & \gg & \gg & \gg & \tau_1 & d \\ \hline
[c1, c2] & [c3, c2] & [c3, c4] & & & & [c5] & [end]
\end{array}$$

$$\gamma_2 = \begin{array}{|c|c|c|c|c|c|c|c|c|}
a & b & c & r & c & b & \gg & \gg & e \\ \hline
a & b & c & \gg & \gg & \gg & \tau_1 & \tau_2 & e \\ \hline
[c1, c2] & [c3, c2] & [c3, c4] & & & & [c5] & [c6] & [end]
\end{array}$$

Using the repair technique discussed so far, we would obtain the model in Fig. 8, where the newly included part is shown in green. The model has multiple transitions of the same label (see b and c), which would be actually unnecessary if the technique could discover that the green part aims to model a structured loops of repeating b and c.

The basic repair algorithm can be extended to implement such structured loops as in the example above. We give an intuition on how the algorithm is extended via the above example: let us take $\sigma_1 = \langle a, b, c, r, b, c, d\rangle$), with the alignment γ_1 shown at page 22. The marking before the first log move is $[c3, c4]$, and the sequence of events that are associated with the maximal sequence of log moves is $\sigma_{\gamma_1} = \langle r, c, b\rangle$.

We search in the model to be repaired, namely the model in Fig. 1, for the smallest connected subnet that *(i)* ends with places $c3$ and $c4$, namely the places with a token at the marking before the first log move, and *(ii)* contains each transition t in $\sigma_{\gamma_1} = \langle r, c, b\rangle$, excluding r, which is not in the model to be repaired. This subnet corresponds to the gray area in Fig. 9.The trace fragment σ_{γ_1} is then projected on this subnet: the events related to transition of the fragment are the only retained, yielding a subtrace $\overline{\sigma_1} = \langle c, b\rangle$. We create an accepting Petri net $\overline{AN}$ from the fragment, using the marking

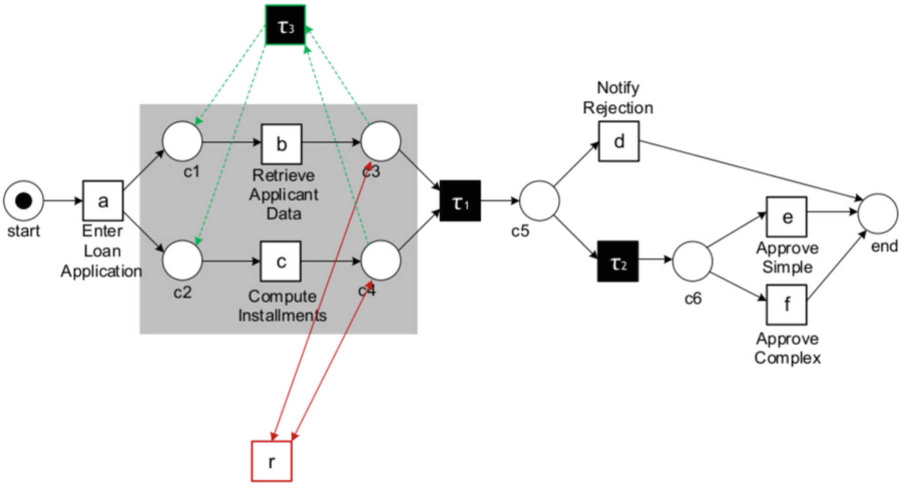


Fig. 9. Repair of the model in Fig. 1 to allow for $\langle a, b, c, r, b, c, d\rangle$ and $\langle a, b, c, r, c, b, e\rangle$, using the advanced model-repair algorithm that increases simplicity (cf. result of the basic algorithm in Fig. 8).

$[c3, c4]$ before the first log move as final marking, and the marking with one token in each place with no incoming arcs as initial marking. Since $\overline{\sigma_1}$ is replayable on $\overline{AN}$, the transition τ_3 can be introduced, which constructs a structured loop where transitions b and c can be repeated. Note that the algorithm above is applied on single log sequences of individual traces, and transition r in Fig. 9 has not been introduced yet. The algorithm needs to be iteratively applied to each sequence of events that come from the projection of the log component of each alignment.

In our example, after repairing the model by adding transition τ_3, the algorithm is applied on $\sigma_2 = \langle a, b, c, r, c, b, e\rangle$, but yields no changes. Indeed, after the first repair, the alignment of the model in Fig. 9 with σ_2 is now as follows:

$$\gamma_2 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} a & b & c & r & \gg & c & b & \gg & \gg & e \\ \hline a & b & c & \gg & \tau_3 & c & b & \tau_1 & \tau_2 & e \\ [c1, c2] & [c3, c2] & [c3, c4] & & [c1, c2] & [c3, c2] & [c3, c4] & [c5] & [c6] & [end] \end{array}$$

Recall that transition r is not yet part of the model. This will be added as final step, which consists in reapplying the basic repair algorithm on the same traces $\sigma_1 = \langle a, b, c, r, b, c, d\rangle$ and $\sigma_2 = \langle a, b, c, r, c, b, e\rangle$ and on the model in which invisible transition τ_3 is included.

This section has focused on repairing the model to reflect the reality observed in the event log. However, repairing the model can also be regarded as to ensure that the model is sound. In the domain of process model, model soundness implies several properties of which the most important is the absence of deadlocks or livelocks that prevent executions from being completed. Interesting approaches that focus on model repair for soundness are provided by Gambini et al. [15] and by Lohmann et al. [16, 17], which are not discussed here due to space limitations.

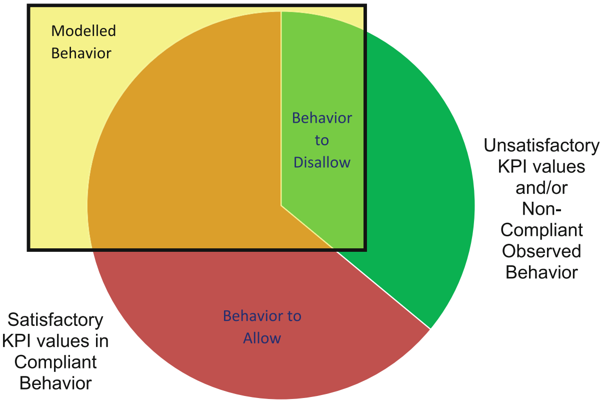

Modelled Behavior
Behavior to Disallow
Unsatisfactory KPI values and/or Non-Compliant Observed Behavior
Satisfactory KPI values in Compliant Behavior
Behavior to Allow

Fig. 10. The basic idea of KPI-driven Model Improvement: the observed behavior (i.e., in the event log) that is satisfactory and compliant with rules should be incorporated in the model, while the observed behavior that is not satisfactory or not compliant, should be not incorporated or disallowed in the model.

3.2 KPI-Driven Model Improvement

If the model is used to prescribe how the corresponding process should be carried on, one does not want to incorporate the whole behavior observed in the event log, but only that portion that has shown to usually lead to satisfactory values of a certain Key Performance Indicator (KPI) of interest. Furthermore, behavior can only be incorporated if it does not violate the protocols, regulations, and norms. The definition KPI of interest varies depending on the domain, needs to be customized, and may be numerical or defined over an enumeration of values, including boolean. Examples are execution costs, customer satisfaction, execution time, or whether or not the corresponding loan was eventually approved. Similarly, one wants to disallow the behavior allowed by the model that, unfortunately, typically yield unsatisfactory KPI values. Figure 10 graphically illustrates the idea. The rectangle shows the amount of behavior allowed by the model, while the pie shows the amount observed in the event log. The green pie slide is the portion of observed behavior that is associated with unsatisfactory KPI values or with violations of norms or protocols: the part in light green that intersects the modelled behavior should be disallowed from the model after repair. The red and the orange pie portions show the portion with satisfactory KPI values: the part in dark red is not allowed by the model but that should be incorporated because of being associated with executions characterized by satisfactory KPI values.

The remainder of this section focuses on a methodology to extend the model to allow the portion in dark red, which has been introduced by Dees et al. [9]. The starting point is an existing process model, here represented as an accepting Petri net, an event log, and the definition of a Key Performance Indicator (KPI). A KPI is a pair consisting of *(i)* a function that, given a trace, returns the KPI value, and *(ii)* the set of satisfactory KPI values:

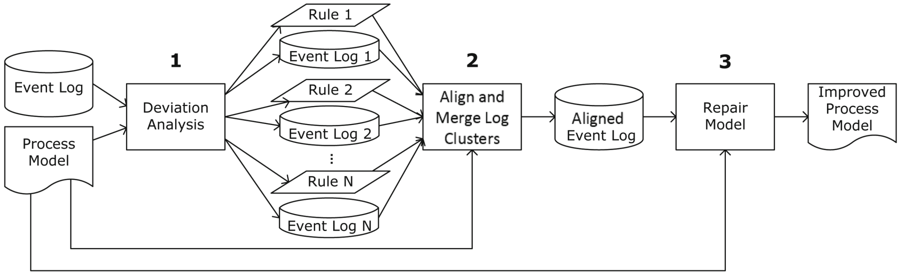


Fig. 11. The main steps of the methodology for KPI-driven Model Improvement (adapter from [9])

Definition 4 (Key Performance Indicator). *Let L be a simplified multi-perspective event log. Let $\mathcal{V}$ be the set of possible values for a key performance indicator. A* key performance indicator *is a pair (κ, K) consisting of a function $\kappa : L \rightarrow \mathcal{V}$ that assigns a KPI value $\kappa(\sigma)$ to each trace σ and of a set $K \subset \mathcal{V}$ that contains the KPI values that are satisfactory from a business viewpoint.*

Typically, the function κ in a KPI definition depends on the attributes present in the event log. However, this section remains general on how the KPI values of process executions (i.e., traces) are computed.

Partially Model-Aligned Traces

The technique described hereafter also relies on the concept of model-aligned event logs that has been introduced in Sect. 1.1. However, we extend the concept to allow for traces that are partially model-aligned. It is indeed possible to ignore individual moves: ignoring a model move means that the corresponding event is not added to the trace, and ignoring a log move means that the corresponding event is not removed. To clarify, let us consider a trace $\langle a, b, b, d \rangle$ and the model in Fig. 1. The alignment is as follows:

$$\gamma = \begin{array}{|c|c|c|c|c|c|} a & b & b & \gg & \gg & d \\ \hline a & b & \gg & c & \tau_1 & d \end{array}$$

A full model-alignment trace is $\langle a, b, c, \tau_1, d \rangle$. Ignoring log moves for b would generate $\langle a, b, b, c, \tau_1, d \rangle$, namely the new alignment would still generate the log move for b; ignoring model moves for c would generate $\langle a, b, \tau_1, d \rangle$, i.e. the model move for c is still present. It is possible to ignore multiple moves at the same time: in our example, repairing neither the model move for c nor the log move for b would produce $\langle a, b, b, \tau_1, d \rangle$. Note that, hereafter, we always to ignore all model moves for invisible transitions when model-aligning a trace, and we consider to fully model-align a trace even when we ignore model moves for invisible transitions.

The Methodology in a Nutshell

The methodology takes an event log and the original process model as input and returns an improved process model. It is composed by three main steps (cf. Fig. 11):

Step 1. Deviation Analysis. Deviations are detected and a set of rules is discovered that correlate deviations to a selected KPI. Rules are mutually exclusive, which enables to split the event-log traces into groups of traces, such that a trace belongs to at most one cluster (in fact, outlier traces are filtered out).

Step 2. Align and Merge Log Clusters. Traces in the different sublogs are partially model-aligned to only keep the deviations in the original trace that have a positive impact on the value of the KPI. All sublogs are then merged to obtain a single partially aligned-model event log.

Step 3. Repair Model. Finally the partially aligned-model event log is used as input to repair the model: the process model is modified in such a way that it can replay all the behavior of the partially aligned-model event log. In the partially aligned-model event log we have repaired all deviations corresponding to behavior that should not be incorporated in the model. In this way the repair-model technique will only modify the model to make the desired deviating behavior possible.

The remainder will elaborate on the sub-sets within steps 1 and 2, using the same case study as in Fig. 1. Step 3 does not require further details since it consists in applying any technique for model repair to reflect reality, such as the technique by Fahland et al. [13] discussed in Sect. 3.1.

Step 1. Deviation Analysis

The deviation-analysis step takes an event log L, an accepting Petri net AN, and a KPI definition (κ, K). The result is a decision tree that allows splitting L in so many sub-logs as the tree leaves. Each sub-log is associated with a different KPI value. Note that certain traces are considered outliers and filtered out, namely the union of the sub-logs does not necessarily coincide with L. To achieve this, the following sub-steps can be identified:

Step 1.1: Conformance Checking. The first step is checking conformance of the event log and the process model. This is done to determine all deviations that are observed between the log and the model. The result of conformance checking is an alignment for each log trace.

> *Example: let us consider the model in Fig. 1 and three non-compliant traces:* $\sigma_1 = \langle a, b, c, w_1, f \rangle$, $\sigma_2 = \langle a, b, c, w_2, f \rangle$ *and* $\sigma_3 = \langle a, b, c, w_3, f \rangle$ *where* $w1$, $w2$ *and* $w3$ *is a shortcut for the activities to ask for one, two or three witnesses, respectively. The alignments are of the following form where* w_X *respectively stands for* w_1, w_2 *and* w_3*:*

$$\gamma_2 = \begin{array}{|c|c|c|c|c|c|c|} a & b & c & w_X & \gg & \gg & f \\ \hline a & b & c & \gg & \tau_1 & \tau_2 & f \end{array}$$

The KPI is here boolean: **true** and **false** respectively indicate whether the approval process has finally led to a loan that is eventually repaid in full or only in part. The latter case is undesired because it requires the involvement of a credit-collection agency. For the three executions in the example, σ_1 refers to a loan paid back in full whereas σ_2 and σ_3 to loans paid back in part.

Step 1.2: Moves' Correlation to KPI Values. The number of model moves and log moves of activities is correlated with the chosen KPI. To model that the improved model should comply rules and regulations, the concepts of *disallowed activities* and *mandatory activities* has been introduced. The set G_D of disallowed activities include those that should never become part of the process model, whereas the set G_M of mandatory activities are those that cannot become optional or be removed from the model. In this step, we build a set of so-called *observation instances*, which are used to train a classification tree. Let T and l be the set of transitions and the labelling function of the labelled Petri net of AN. Let A be the activities of N, i.e. the Petri-net labels: $A = \cup_{t \in T} l(t)$. To keep it simple, we assume without losing generality that the log activities coincide with A, too. We build one observation instance for each trace $\sigma \in L$ with the following features:

- The number of model moves in the optimal alignment of σ for each allowed activity $a \in A \setminus G_D$.
- The number of log moves in the optimal alignment of σ for each non-mandatory activity $a \in A \setminus G_M$.
- The KPI value for σ, namely $\kappa(\sigma)$.

From the set of observation instance, we learn a decision tree, using the KPI value as target feature, and the number of log and model moves as independent features. If the domain of the KPI values is finite (e.g., satisfactory vs unsatisfactory), a classification tree is used; otherwise, we employ a regression tree.

> *Example (cont.): log moves for w_2 and w_3 are correlated with full repay, where log moves for w_1 are correlated with part repay. However, let us assume w_3 be within the set of disallowed activities (e.g., three witnesses require too much additional work). Thus, log moves for w_3 are not allowed as independent feature. The result could be such a decision tree as in Fig. 12: when there are log moves for w_2, the KPI is fulfilled: the loan is eventually repaid in full.*

Step 1.3: Splitting of the Event Log into Groups and Outlier Filtering. The classification tree can be seen as a clustering of the traces of an event log. Each leaf is a different cluster and the path from the root to the leaf provides a rule that characterizes the traces that belong to a certain group. For reliability, the wrongly-classified traces are removed from the groups, namely the traces classified to have KPI values that differ from the actual values. The wrongly-classified traces might potentially affect the repair-model phase, and allow behavior in the model that would not be linked to actual, satisfactory KPI values.

> *Example (cont.): The trace cluster associated with leaf* Part Repay *(left leaf) is $L_1 = [\langle a, b, c, w_1, f \rangle]$, whereas the cluster for leaf* Full Repay *is $L_2 = [\langle a, b, c, w_2, f \rangle]$. Note that trace $\sigma_3 = \langle a, b, c, w_1, f \rangle$ would also be in L_2, but would be wrongly classified and consequently filtered out. In fact, σ_3 is associated with a loan that is repaid in full but the decision tree in Fig. 12 would classify it as partly repaid: it does not indeed contain log moves for w_2.*

Step 2. Model-Align and Merge Log Clusters

Step 1 concluded with splitting L in n sublogs and filtering out those traces that are wrongly classified. Let $\{L_1, \ldots, L_n\}$ be the sublogs obtained via splitting. Each L_i refers to a different decision-tree leaf v_i, associated with a KPI value $\mathcal{C}(v_i)$.

Step 2.1: Conformance Checking of the Sublogs. Conformance Checking is done with the original process model and each log cluster. Note that Step 2.1 is a conceptual step: in practice, one does not need to recompute the alignments for the cluster logs as one can simply reuse the alignments obtained as result of Step 1.1.

Step 2.2: Model-Align of the Sublogs. This step is repeated for each cluster L_i, associated with a leaf v_i. If L_i is associated with an unsatisfactory KPI value (i.e. $\mathcal{C}(v_i) \notin K$), every deviation is repaired. Note that, even if the traces are fully model-aligned, they are kept in the log that is used for repairing the model at step 3. Those traces provide support to not remove behavior that is not observed: see discussion on achieving higher model precision in subsection *Advanced Repair for Higher Precision and Simplicity* within Sect. 3.1.

If L_i is associated with a satisfactory KPI value, every deviation is repaired, except those in the conditions in the path from the decision-tree root to the leaf v_i.

> *Example (cont.): Trace σ_1 is model-aligned in full because related to an unsatisfactory KPI value, yielding a partial model-aligned trace $\sigma_1^{\tau} = \langle a, b, c, f \rangle$. Trace σ_2 is related to satisfactory KPI values (see leaf Full Repay in the decision tree in Fig. 12), and associated to a tree path that indicates that the number of log moves for w_2 is larger*

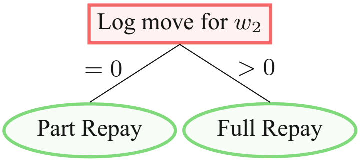


Fig. 12. A decision tree that correlates alignment moves to KPI values.

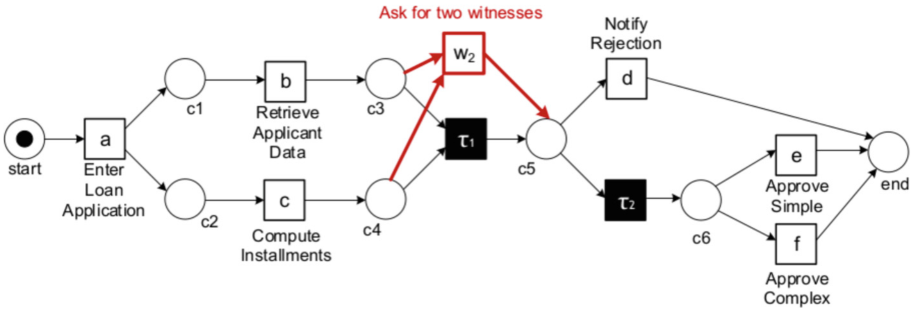


Fig. 13. The model repaired to increase the changes for loan to be repaid in full (the KPI). The change consists in introducing the activity *Ask for two witnesses*, which are shown to be beneficial for a better risk assessment.

than zero. This means that the log move for w_2 is ignored when model-aligning σ_2: thus, the partial model-aligned trace σ_2^{τ} coincides with the original trace σ_2.

Step 2.3: Merge the Sublogs. We merge all model-aligned sublogs into a single event log. This is a requirement to apply the next step, namely repairing the process model.

Example (cont.): This step generates the event log $\overline{L} = [\langle a, b, c, f \rangle, \langle a, b, c, w_2, f \rangle]$, which is used for model repair.

When the log is used with a model-repair technique (e.g., that in Sect. 3.1), the model in Fig. 2 is repaired as shown in Fig. 13: the transition w_2 is introduced.

Acknowledgement. Some of the ideas and techniques reported in this chapter is the result of author's collaborations with various researchers. While it is not possible to name them all, the author would like to give a special mention to Wil van der Aalst, Marcus Dees, Marlon Dumas, Felix Mannhardt, and Hajo Reijers (in strict alphabetical order).

References

1. van der Aalst, W.M.P.: Process mining: a 360 degrees overview. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
2. van der Aalst, W.M.P.: Foundations of process discovery. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
3. Augusto, A., Carmona, J., Verbeek, E.: Advanced process discovery techniques. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
4. Burattin, A., Sperduti, A., Veluscek, M.: Business models enhancement through discovery of roles. In: 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), pp. 103–110 (2013)
5. Carmona, J., van Dongen, B., Weidlich, M.: Conformance checking: foundations, milestones and challenges. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
6. Ernst, M.D., et al.: The daikon system for dynamic detection of likely invariants. Sci. Comput. Program. **69**(1), 35–45 (2007). Special issue on Experimental Software and Toolkits
7. de Leoni, M., Dumas, M., García-Bañuelos, L.: Discovering branching conditions from business process execution logs. In: Cortellessa, V., Varró, D. (eds.) FASE 2013. LNCS, vol. 7793, pp. 114–129. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37057-1_9
8. de Leoni, M., van der Aalst, W.M.P.: Data-aware process mining: discovering decisions in processes using alignments. In: SAC 2013, pp. 1454–1461. ACM (2013)
9. Dees, M., de Leoni, M., Mannhardt, F.: Enhancing process models to improve business performance: a methodology and case studies. In: Panetto, H., et al. (eds.) OTM 2017. LNCS, vol. 10573, pp. 232–251. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69462-7_15
10. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer (2018). https://doi.org/10.1007/978-3-662-56509-4

11. Estrada-Torres, B., Camargo, M., Dumas, M., García-Bañuelos, L., Mahdy, I., Yerokhin, M.: Discovering business process simulation models in the presence of multitasking and availability constraints. Data Knowl. Eng. **134**, 101897 (2021)
12. Fahland, D., van der Aalst, W.M.P.: Simplifying discovered process models in a controlled manner. Inf. Syst. **38**(4), 585–605 (2013)
13. Fahland, D., van der Aalst, W.M.P.: Model repair—aligning process models to reality. Inf. Syst. **47**, 220–243 (2015)
14. Fracca, C., de Leoni, M., Asnicar, F., Turco, A.: Estimating activity start timestamps in the presence of waiting times via process simulation. In: Proceedings of the 34th International Conference on Advanced Information Systems Engineering (CAiSE 2022), LNCS. Springer (2022)
15. Gambini, M., La Rosa, M., Migliorini, S., Ter Hofstede, A.H.M.: Automated error correction of business process models. In: Proceedings of the 9th International Conference on Business Process Management, BPM 2011, pp. 148–165, Springer, Heidelberg (2011)
16. Lohmann, N.: Correcting deadlocking service choreographies using a simulation-based graph edit distance. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 132–147. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85758-7_12
17. Lohmann, N., Fahland, D.: Where did i go wrong? In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 283–300. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_18
18. Mannhardt, F., Blinde, D.: Analyzing the trajectories of patients with sepsis using process mining. In: RADAR+EMISA 2017, volume 1859 of CEUR Workshop Proceedings, pp. 72–80. CEUR-WS.org (2017)
19. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
20. Nakatumba, J.: Resource-aware business process management: Analysis and Support. PhD thesis, Technische Universiteit Eindhoven (2013)
21. Object Management Group (OMG): Decision model and notation (DMN) v1.1 (2016)
22. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
23. Rozinat, A., van der Aalst, W.M.P.: Decision mining in ProM. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 420–425. Springer, Heidelberg (2006). https://doi.org/10.1007/11841760_33
24. Senderovich, A.: Queue mining. In: Sakr, S., Zomaya, A.Y. (eds.) Encyclopedia of Big Data Technologies. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-77525-8
25. van der Aalst, W.M.P.: Process Mining - Data Science in Action, 2nd Ed. Springer (2016). https://doi.org/10.1007/978-3-662-49851-4
26. van der Aalst, W.M.P., Reijers, H.A., Song, M.: Discovering social networks from event logs. Comput. Supp. Coop. Wor. **14**(6), 549–593 (2005)

Process Mining over Multiple Behavioral Dimensions with Event Knowledge Graphs

Dirk Fahland(✉)

Eindhoven University of Technology, Eindhoven, The Netherlands
d.fahland@tue.nl

Abstract. Classical process mining relies on the notion of a unique case identifier, which is used to partition event data into independent sequences of events. In this chapter, we study the shortcomings of this approach for event data over *multiple entities*. We introduce *event knowledge graphs* as data structure that allows to naturally model behavior over multiple entities as a network of events. We explore how to construct, query, and aggregate event knowledge graphs to get insights into complex behaviors. We will ultimately show that event knowledge graphs are a very versatile tool that opens the door to process mining analyses in multiple behavioral dimensions at once.

Keywords: Event knowledge graph · Process mining

1 Introduction—A Second Look at Processes

Process mining aims at analyzing processes from recorded event data. Thereby, the actual processes are rather complex and emerge from the interplay of multiple inter-related *entities*: the various *objects* handled by the process as well as the *organizational entities* that execute the process. We best explain this kind of interplay by an example.

1. Consider a retailer who took two *Orders* for multiple *Items* from the same customer: the customer first places Order $O1$ for 2 items X and 1 item Y, and shortly afterwards Order $O2$ for 1 item X and 1 item Y. The retailer promises to ship every order within 6 days.

The retailer handles both orders as explained next and illustrated in Fig. 1.

2. Items X are provided by supplier A while items Y are provided by supplier B. To save costs, *workers* of the retailer bundle the orders for the items and place two *Supplier Orders*, one at A for 3 items X and one at B for 1 item Y. Suppliers ensure to deliver their products within 3 days of placing the order.
3. Invoice $I2$ for Order $O2$ is created right after placing the supplier order at B.
4. When the retailer receives the Supplier Order from A, workers unpack three Items X one by one and store them in an *automated warehouse* until needed for shipment. At this point, workers also create *Invoice* $I1$ for $O1$.

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 274–319, 2022.
https://doi.org/10.1007/978-3-031-08848-3_9

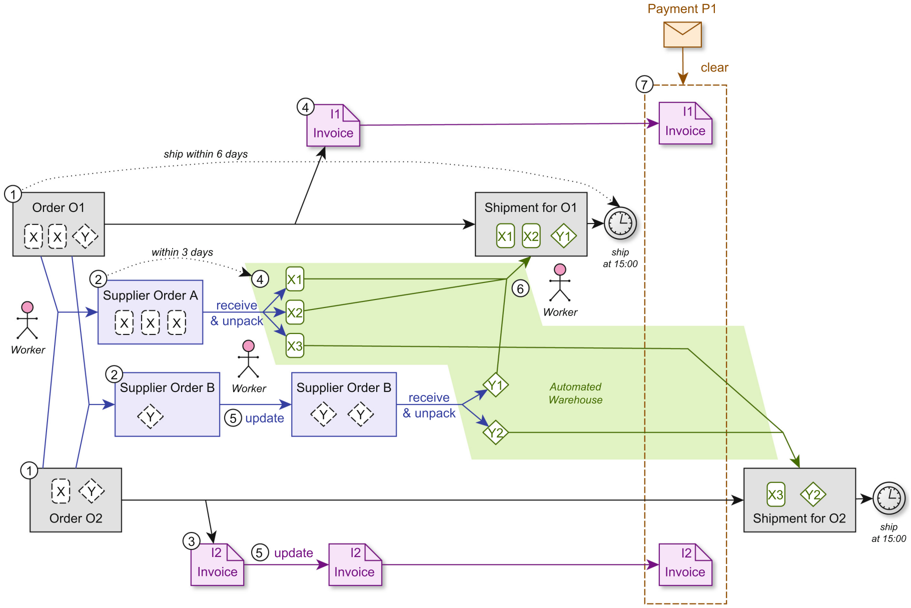


Fig. 1. Illustration of a multi-entity process: a retailer handles two orders for multiple items by placing and receiving supplier orders for specific items.

5. Around the time of receiving the supplier order from A, a *worker* notices they made a mistake: they ordered only one item Y from B while $O1$ and $O2$ both require one item Y *each*. The worker updates the Supplier Order $O2$ and invoice $I2$ accordingly.
6. When finally the supplier order from B is received, the items Y are unpacked. One item Y is stored in the warehouse while the other item Y is packed together with two items X taken from the *warehouse* into the shipment for $O1$. Packed shipments are picked up for delivery every day at 15:00.
7. The retailer has the policy that they only ship to a customer if there is at most one unpaid invoice. Thus, packing and shipment of $O2$ (another item X and the second Y) are delayed until *Payment* $P1$ is received which covers the amount for both invoices $I1$ and $I2$.

This process relies on 7 different types of entities. *Actors* (human workers) and *machines* (an automated warehouse) together handle 5 types of objects: *Orders*, *Supplier Orders*, *Items*, *Invoices*, *Payments*.

Challenges Due to Event Data over Multiple Entities. A process mining analysis of the above process execution relies on recorded event data. Each event has to record in its attributes at least (1) which *action* (or activity) has been executed (2) at which *time*. To construct an event log, classical process mining also expects each event to record (3) in which process execution, typically called

Table 1. Event table of events underlying the event log of Table 2.

EventID	Activity	Time	Actor	Order	Supplier Order	Order Details	Item	Invoice	Payment
e1	Create Order	01-05 09:05	R1	O1		2·X, 1·Y			
e2	Create Order	01-05 09:30	R1	O2		1·X, 1·Y			
e3	Place SO	01-05 11:25	R1		A	3·X			
e4	Place SO	02-05 11:55	R3		B	1·Y			
e5	Create Invoice	03-05 16:15	R3	O2				I2	
e6	Receive SO	00-01 10:00	R2		A		X1,X2,X3		
e7	Update SO	04-05 10:25	R1	O2	B	2·Y			
e8	Unpack	00-01 10:30	R2		A		X3		
e9	Update Invoice	04-05 10:50	R2					I2	
e10	Unpack	04-05 11:00	R2		A		X1		
e11	Unpack	04-05 11:15	R2		A		X2		
e18	Create Invoice	06-05 14:35	R3	O1				I1	
e19	Receive SO	07-05 10:10	R2		B		Y1,Y2		
e20	Unpack	07-05 10:45	R2		B		Y1		
e21	Unpack	07-05 11:00	R2		B		Y2		
e27	Pack Shipment	07-05 17:00	R4	O1			X1,X2,Y1		
e28	Ship	08-05 15:00	R4	O1					
e29	Receive Payment	09-05 08:30	R5						P1
e30	Clear Invoice	09-05 08:45	R5					I1,I2	P1
e33	Pack Shipment	09-05 11:45	R4	O2			X3,Y2		
e34	Ship	09-05 15:00	R4	O2					

case, the event occurred (see [13], Sect. 2). Table 1 shows the events related to the above example.

1. create *Orders* in e_1, e_2;
2. create *Supplier Orders* in e_3, e_4;
3. create *Invoice* $I2$ in e_5;
4. receive *Supplier Order* from A in e_6 and unpack *Items* X in e_8, e_{10}, e_{11}, and create *Invoice* $I1$ in e_{18};
5. update *Supplier Order* for B in e_7 and *Invoice* $I2$ in e_9;
6. receive *Supplier Order* for B in e_{19} and unpack *Items* Y in e_{20}, e_{21}, and pack and ship *Order* $O1$ in e_{27}, e_{28};
7. receive *Payment* $P1$ and clear *Invoices* $I1, I2$ in e_{29}, e_{30} and finally pack and ship *Order* $O2$ in e_{33}, e_{34}.

In contrast to classical event logs, Table 1 contains no typical case identifier attribute by which each event is related to one specific process execution. Instead, we see multiple sparsely filled attributes identifying *multiple entities* of various types: *Order* ($O1, O2$), *Supplier Order* (A, B), *Item* ($X1, X2, X3, Y1, Y2$), *Invoice* ($I1, I2$), and *Payment* ($P1$).

This makes it difficult to construct an *event log* which is the basis for process mining analysis. Recall that to obtain a classical event log we select one *case identifier* attribute. Then all events referring to the same case id and ordered by time form the *trace* of this case, that is, one process execution. In this way, classical event logs partition the recorded behavior into multiple process executions. Process mining techniques then identify frequent patterns shared by all process executions, or identify outliers and deviations of specific process executions.

However, what exactly *is* a process execution in our example? It is not just all events related the one particular entity. For instance, if we chose *Order* as case identifier, we would obtain traces $\langle e_1, e_{18}, e_{27}, e_{29} \rangle$ for $O1$ and $\langle e_2, e_5, e_7, e_{33}, e_{34} \rangle$ for $O2$. These traces do reveal that both orders were not shipped within 6 days as intended by the supplier. However, they do not allow us to understand the cause for this as they clearly do not describe the entire behavior shown in Fig. 1. We could try to group all events into traces using *multiple related case identifiers*. However, we will see in Sect. 3 that doing so introduces false behavioral information called *convergence* and *divergence* [41,45] in the resulting event log leading to false analysis results (see [1], Sect. 3)

False behavioral information arises when flatting Table 1 into sequential traces because we *cannot* partition the entities $O1, O2, A, B, X, Y, I1, I2, P1$ into disjoint sets, each belonging to one process execution that is independent of all others. Rather, the behavior itself is a larger “fabric” of multiple entities that are inter-related and inter-twined over time as shown in Fig. 1. This “fabric” is even more complex as individual *Actors* ($R1, \ldots, R5$) are specialized in specific activities across multiple different entities, e.g., $R2$ specializes receiving, updating, and unpacking *Supplier Orders* and handling *Items*. In the following, we explain how to analyze this very “fabric” of multiple inter-related entities as a whole from a simple event table over multiple entity identifiers such as Table 1.

A Graph-Based Approach. Our trick will be to slightly adapt the existing definitions for obtaining an event log from an event table: instead of constructing entire traces related to a single case identifier, we discuss in Sect. 3 a *local directly-follows* relation for each *individual* entity in the data. Each event can be part of multiple such directly-follows relations, depending on to how many entities it is correlated. We then use the model of *labeled property graphs* in Sect. 4 to create an *event knowledge graph* having events as nodes and the local directly-follows relations as edges between events. We obtain a graph similar to what is shown in Fig. 1, but with precise semantics for events and behavioral information.

A path of directly-follows edges over events related to the same entity is similar to a classical trace. However in an event knowledge graph, such paths meet whenever an event is related to more than one entity, where in an event log each trace is disjoint from all others. We explain in Sect. 5 how to interpret and analyze behavioral information in event knowledge graphs. We show how basic *querying* on event knowledge graphs gives insights into complex behavioral properties.

We show how *aggregation* on event knowledge graphs allows to construct multi-entity process models that better describe such processes.

We finally explore the versatility of event knowledge graphs beyond the control-flow perspective in Sect. 6. We show how event knowledge graphs naturally integrate the control-flow perspective and the *actor perspective*. Querying for specific structures in the event knowledge graph reveals complex patterns of *task instances* not visible in either perspective alone. Further, we show how event knowledge graphs allow us to take a *system-perspective* (or queueing perspective) to analyze emergent behavior and performance problems across multiple entities. We conclude in Sect. 7 with an outlook on the various applications areas of event knowledge graphs in process mining, and on open research challenges.

All concepts for constructing and analyzing event knowledge graphs presented in this chapter are implemented as Cypher queries on the graph database system Neo4j[1] at https://github.com/multi-dimensional-process-mining/event graph_tutorial [28].

2 Multi-entity Event Data

Before we discuss problems and solutions for analyzing event data over multiple entities, we first define what "event data over multiple entities" actually is.

2.1 Events

We assume all data to be given in a single event table. Data is recorded from a universe of values Val; timestamps $Val_{time} \subseteq Val$ are totally ordered by $\leq$.

Definition 1 (Event Table). *An* event table $T = (E, Attr, \#)$ *is a set* E *of events, a set* $Attr$ *of* attribute names *with* $act, time \in Attr$*. Partial function* $\# : E \times Attr \nrightarrow Val$ *assigns an event* $e \in E$ *and an attribute name* $a \in Attr$ *to a value* $\#_a(e) = v$*;* $\#_a(e) = \perp$ *if* a *is undefined for* e*.*

Each event $e \in E$ *records an activity and a timestamp, i.e.,* $\#_{act}(e) \neq \perp$ *and* $\#_{time}(e) \in Val_{time}$*.*

We write $e.a = v$ for $\#_a(e) = v$ as a shorthand. An event table specifically allows multi-valued attributes, e.g., sets of values $\#_a(e) = \{v_1, v_2, v_3\}$ or a list of values $\#_a(e) = \langle v_1, v_2, v_3, v_1 \rangle$.[2] Simplifying notation, we also may write $v \in e.a$ if $e.a = v$ or if $e.a = \langle \ldots, v, \ldots \rangle$.

An event table only defines $e.activity$ and $e.time$ attributes for each event. The special characteristic of event data over multiple entities is that it does not record a unique case identifier attribute, but identifiers of *multiple entity types*.

Definition 2 (Event table with entity types). *An* event table with entities types $T = (E, Attr, \#, ENT)$ *additionally designates one or more attributes* $\emptyset \neq ENT \subseteq Attr$ *as names of* entity types*.*

[1] neo4j.com.

[2] We assume the values in an event table to be consistent with some data model that is specified elsewhere. Our subsequent discussion does not rely on it.

A classical event log corresponds to an event table with a single entity type $\mathit{ENT} = \{\mathit{case}\}$. We can consider Table 1 is an event table with entity types $\mathit{ENT} = \{\mathit{Resource}, \mathit{Order}, \mathit{Supplier\ Order}, \mathit{Item}, \mathit{Invoice}, \mathit{Payment}\}$.

Event tables (Definition 1) are also called raw event logs and are – besides relational data – the most common form of input to process mining. The entity types of Definition 2 can be retrieved from an event table through schema recovery techniques [46]. Note that Definition 2 formalizes the object-centric event logs (OCEL) described in Sect. 3.4 of [1]; we here use the more general term "entity" instead of "object" as we will later study behavior over entities which are not tangible objects.

Event tables do not model the ordering of events with respect to a case or an entity which is needed for process mining. Before we study the ordering of events, we explain how events relate to entities.

2.2 Entities and Correlated Events

Each entity type $\mathit{ent} \in \mathit{ENT}$ is a column in the event table T. Each value in that column refers to a specific entity.

Definition 3 (Entities). *Let $T = (E, \mathit{Attr}, \#, \mathit{ENT})$ be an event table with entities. Let $\mathit{ent} \in \mathit{ENT}$ be an entity type. The* set of entities *in T of type ent is $\mathit{Entities}(\mathit{ent}, T) = \{n \mid \exists e \in E : n \in e.\mathit{ent}\}$.*

From Table 1 we identify 6 entity types with corresponding entities: (1) Order: $\{O1, O2\} = \mathit{Entities}(\mathit{Order}, T)$, (2) Supplier Order: A, B, (3) Item: $X1, X2, X3$ and $Y1, Y2$, (4) Invoice: $I1, I2$, (5) Payment: $P1$, (6) Resource: $R1 - R5$ (see Definition 3).

An event $e \in E$ which has a value $n = e.\mathit{ent}$ or $n \in e.\mathit{ent}$ is *correlated* to entity n.

Definition 4 (Correlation). *Let $T = (E, \mathit{Attr}, \#, \mathit{ENT})$ be an event table with entities. Let $n \in \mathit{Entities}(\mathit{ent}, T)$ be an entity of type $\mathit{ent} \in \mathit{ENT}$.*

Event e is correlated to *entity n, written $(e, n) \in \mathit{corr}_{\mathit{ent},T}$ iff $n = e.\mathit{ent} \vee n \in e.\mathit{ent}$. We write $\mathit{corr}(n, \mathit{ent}, T) = \{e \in E \mid (e, n) \in \mathit{corr}_{\mathit{ent},T}\}$ for the set of events correlated to entity $n \in \mathit{Entities}(\mathit{ent}, T)$.*

For example, for Table 1, event e_{30} is correlated to $I1$, $I2$, and $P1$, i.e., $(e_{30}, I1), (e_{30}, I2) \in \mathit{corr}_{\mathit{Invoice},T}$ and $(e_{30}, P1) \in \mathit{corr}_{\mathit{Payment},T}$. The events correlated to $I2$ are $\mathit{corr}(I2, \mathit{Invoice}, T) = \{e_5, e_9, e_{30}\}$. In case the entity identifiers used by different entity types are disjoint, e.g., there are not an Order $O3$ and an Item $O3$, we can omit entity types and just write $(e_{30}, I1), (e_{30}, I2), (e_{30}, P1) \in \mathit{corr}_T$ and $\mathit{corr}(I2, T) = \{e_5, e_9, e_{30}\}$.

Correlation lifts to a *set* N of entities by union: $\mathit{corr}(N, T) = \bigcup_{n \in N} \mathit{corr}(n, T)$. We will later use this to collect events of (transitively) related entities, which we discuss next.

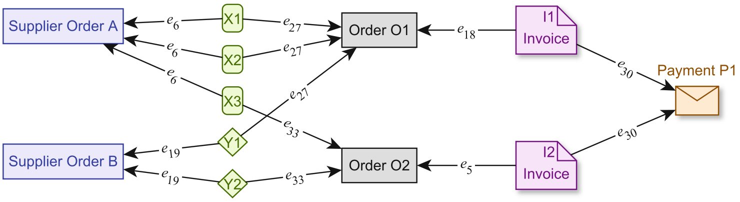


Fig. 2. Relations between entities derived from Table 1

2.3 Relations Between Entities

We now make a first important observation. Although our data only defines entity types explicitly, it *implicity* defines *relations between entity types*. A record e in Table 1 containing two identifiers n_1, n_2 of two different types implicitly relates n_1 and n_2. For example, event e_5 defines that $e_5.Order = O2$ is related to $e_5.Invoice = I2$ and event e_{18} defines that $e_{18}.Order = O1$ is related to $e_{18}.Invoice = I1$. We can write this as a relation $R_{(Invoice,Order)} = \{(O1, I1), (O2, I2)\}$.

Definition 5 (Relation). *Let $T = (E, Attr, \#, ENT)$ be an event table with entities. Let $ent_1, ent_2 \in ENT$ be two entity types. The* relation between ent_1 and ent_2 in T *is* $R_{(ent_1,ent_2)} = \{(e.ent_1, ent_2) \mid e.ent_1 \neq \perp, e.ent_2 \neq \perp\}$.

Note that Definition 5 does not impose the direction of a relation. Figure 2 visualizes the relations we can derive from Table 1.

Recall that in relational data modeling, each relation $R_{(ent_1,ent_2)}$ has a *cardinality* describing how many entities of type ent_1 are related to each entity of type ent_2, and vice versa. We can infer this cardinality from the tuples in $R_{(ent_1,ent_2)}$ if we assume that the data in the input event table is sufficiently complete. For example, for the relations in Fig. 2,

- $R_{(Invoice,Order)}$ is a 1-to-1 relation as each invoice is related to one order, and vice versa;
- $R_{(Invoice,Payment)}$ is an n-to-1 relation as both $I1$ and $I2$ are related to $P1$;
- $R_{(Item,Order)}$ is an n-to-1 relation as each order has multiple items but each item relates to exactly one order;
- $R_{(Item,Supplier\ Order)}$ is an n-to-1 relation.

Entities are also transitively related by concatenating or joining the relations on a shared entity typed (and then omitting this shared entity type). For example, $R_{(Order,Payment)} = R_{(Invoice,Order)} \bowtie R_{(Invoice,Payment)} = \{(O1, P1), (O2, P1)\}$ is an n-to-1 relation, and $R_{(Order,Supplier\ Order)} = R_{(Item,Order)} \bowtie R_{(Item,Supplier\ Order)} = \{(O1, A), (O1, B), (O2, A), (O2, B)\}$ is an n-to-m relation.

Entities, relations, and correlation of events can be automatically retrieved from event tables [46] and relational databases [41,43] through schema recovery techniques. However, we have to be aware that relations and their cardinalities recovered according to Definition 5 are a *static* view of the relations obtained by aggregating all observations over time while *a process updates relations dynamically*. For instance, *Order* $O1$ was not related to any *Item* until event e_{27}. Modeling such dynamics requires additional concepts as defined in XOC event logs [39,40]. We have to ignore this aspect in the remainder.

3 Shortcomings of Event Logs over Multi-entity Event Data

Having defined event data over multiple entities, we can now discuss ways of ordering events correlated to a case or an entity, which is the basis for process mining analysis. We first explain how transforming multi-entity data into a classical event log with a single case identifier (Sect. 3.1) introduces false behavioral information leading to false analysis results (Sect. 3.2). We then propose a different approach to ordering events with respect to individual entities (Sect. 3.3).

3.1 Classical Event Log Extraction

We cannot directly turn the event data in Table 1 into a classical event log, because we lack a clear case identifier column that is defined for all events. While *Actor* is an entity identifier defined for all events, it does not group events into the process executions described in Sect. 1. The standard procedure to extract a classical event log from such data is the following (see also Def. 5 of [1] and [13]).

Step 1. Determine relevant entities in the data. An event table with entity identifiers already defines the set of entities in the process (see Definition 3). For extracting an event log for a process execution, we only consider entities that are also handled "along" or "within" a process execution. Thus, we now focus on *Order*, *Supplier Order*, *Item*, *Invoice*, and *Payment* and exclude *Actor*.[3]

Step 2. Pick one entity as case identifier. As the process goal is to complete an order, entity *Order* is our best candidate for a case identifier. This identifier defines two cases: $O1$ and $O2$. However, as most events in Table 1 are not directly correlated to an *Order*, we cannot simply group events by attribute *Order*.

Step 3. Define the set of all entities related to a case. The classical idea is to "enlarge" the scope of the case. We include all entities which are (transitively) related to the case entities $O1$ and $O2$ via the relations we can identify in the event table (see Definition 5 and Fig. 2).

- Order $O1$ is related to Invoice $I1$, Payment $P1$, Items $X1, X2, Y1$, and Supplier Orders A, B, i.e., $\mathit{caseEntities}(O1) = \{O1, I1, P1, X1, X2, Y1, A, B\}$.
- Order $O2$ is related to Invoice $I2$, Payment $P1$, Items $X3, Y2$, Supplier Orders A, B, i.e., $\mathit{caseEntities}(O2) = \{O2, I2, P1, X3, Y2, A, B\}$.

[3] In later sections we will not have to make such a distinction and can consider behavior along any kind of entity.

Step 4. Construct a trace from events of all entities in a case. Each event e correlated to an entity $n \in \mathit{caseEntities}(O1)$ is now also considered as correlated to case $O1$: $\mathit{corr}^*(O1,T) = \mathit{corr}(\mathit{caseEntities}(O1),T)$. For example, for $O1$ we extract from Table 1:

- $\mathit{corr}(O1,T) = \{e_1, e_2, e_{18}\}$
- $\mathit{corr}(I1,T) = \{e_{18}, e_{30}\}$
- $\mathit{corr}(P1,T) = \{e_{29}, e_{30}\}$
- $\mathit{corr}(X1,T) = \{e_6, e_{10}, e_{27}\}$
- $\mathit{corr}(X2,T) = \{e_6, e_{11}, e_{27}\}$
- $\mathit{corr}(Y1,T) = \{e_{19}, e_{20}, e_{27}\}$
- $\mathit{corr}(A,T) = \{e_3, e_6, e_8, e_{10}, e_{11}\}$
- $\mathit{corr}(B,T) = \{e_4, e_7, e_{19}, e_{20}, e_{21}\}$

Taking their union yields $\mathit{corr}^*(O1,T) = \{e_1, e_3, e_4, e_6, e_{10}, e_{11}, e_{18}, e_{19}, e_{20}, e_{27}, e_{28}, e_{29}, e_{30}\}$. We store all events extracted for $O1$ in a new event table where we explicitly set the attribute *Case* to $O1$. In this way, we materialize that each $e_i \in \mathit{corr}^*(O1,T)$ is correlated to $O1$. We repeat this procedure for each case. Table 2 shows the extracted events for $O1$ and $O2$.

Note that this extraction approach can extract the same event *multiple times* for different cases but with a different value for the newly set *Case* attribute. For instance, e_3 and e_{30} are extracted both for $O1$ and for $O2$. This is due to the n-to-m relation between *Order* and *Supplier Order* and the n-to-1 relation between *Payment* and *Order*.

Ordering the extracted events by time in each case results in the *traces* from the viewpoint of $O1$ and from the viewpoint of $O2$ respectively as shown in Tab 2.

Event logs can be automatically extracted in this way from event tables with multiple entity identifiers [46]. Extraction from relational databases succeeds through SQL queries that extract and group events from different tables into traces [35]. These queries can be generated automatically using a variety of techniques [6,7,12,29,35,41]; see [2,13] for a detailed discussion.

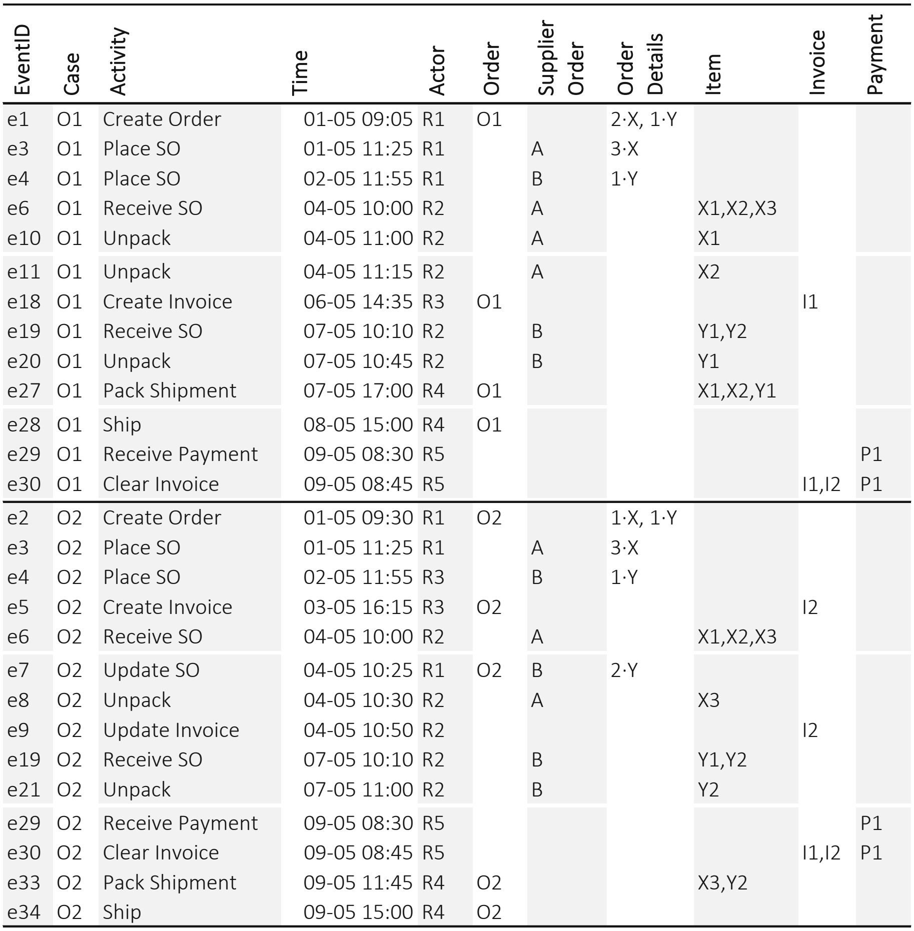
3.2 False Behavioral Information in Classical Event Logs

Note that the event log in Table 2 contains numerous *false* behavioral information. Some events were duplicated and occur in both traces, e.g., $e_3, e_4, e_6, e_{19}, e_{29}$, suggesting that in total four *Supplier Orders* were placed and received (while there were only two) and that two *Payments* were received (while there was only one). This is also known as *divergence* [41,41,45,52].

Further, the order of events in both traces gives false behavior information. For instance, in the trace for O2, *Update SO* (e_7) occurs after *Receive SO* (e_6) suggesting a supplier order was updated after it had been received (while this never happened for any Supplier Order). This is also known as *convergence* [41,45,52].

Where divergence falsifies frequencies of events, convergence falsifies the behavioral information in the directly-follows relation, which is the basis for

Table 2. Classical event log of order process with events extracted for case identifier *Order*.



EventID	Case	Activity	Time	Actor	Order	Supplier Order	Order Details	Item	Invoice	Payment
e1	O1	Create Order	01-05 09:05	R1	O1		2·X, 1·Y			
e3	O1	Place SO	01-05 11:25	R1		A	3·X			
e4	O1	Place SO	02-05 11:55	R1		B	1·Y			
e6	O1	Receive SO	04-05 10:00	R2		A		X1,X2,X3		
e10	O1	Unpack	04-05 11:00	R2		A		X1		
e11	O1	Unpack	04-05 11:15	R2		A		X2		
e18	O1	Create Invoice	06-05 14:35	R3	O1				I1	
e19	O1	Receive SO	07-05 10:10	R2		B		Y1,Y2		
e20	O1	Unpack	07-05 10:45	R2		B		Y1		
e27	O1	Pack Shipment	07-05 17:00	R4	O1			X1,X2,Y1		
e28	O1	Ship	08-05 15:00	R4	O1					
e29	O1	Receive Payment	09-05 08:30	R5						P1
e30	O1	Clear Invoice	09-05 08:45	R5					I1,I2	P1
e2	O2	Create Order	01-05 09:30	R1	O2		1·X, 1·Y			
e3	O2	Place SO	01-05 11:25	R1		A	3·X			
e4	O2	Place SO	02-05 11:55	R3		B	1·Y			
e5	O2	Create Invoice	03-05 16:15	R3	O2				I2	
e6	O2	Receive SO	04-05 10:00	R2		A		X1,X2,X3		
e7	O2	Update SO	04-05 10:25	R1	O2	B	2·Y			
e8	O2	Unpack	04-05 10:30	R2		A		X3		
e9	O2	Update Invoice	04-05 10:50	R2					I2	
e19	O2	Receive SO	07-05 10:10	R2		B		Y1,Y2		
e21	O2	Unpack	07-05 11:00	R2		B		Y2		
e29	O2	Receive Payment	09-05 08:30	R5						P1
e30	O2	Clear Invoice	09-05 08:45	R5					I1,I2	P1
e33	O2	Pack Shipment	09-05 11:45	R4	O2			X3,Y2		
e34	O2	Ship	09-05 15:00	R4	O2					

most process discovery techniques. As a result, also discovered process models are wrong. Figure 3 (left) shows the directly-follows graph (DFG) of the log in Table 2 and the corresponding process model discovered with the Inductive Miner (IM) annotated with the mean waiting times. Both models show false information suggesting that

- a *Supplier Order* was *Updated* after it was *Received* while this never happened;
- *rework* happened around receiving a *Supplier Order* and unpacking *Items* while each Supplier Order and each Item was touched only once;
- an *Invoice* can be *created* and *updated* in an arbitrary order while only one order was observed;

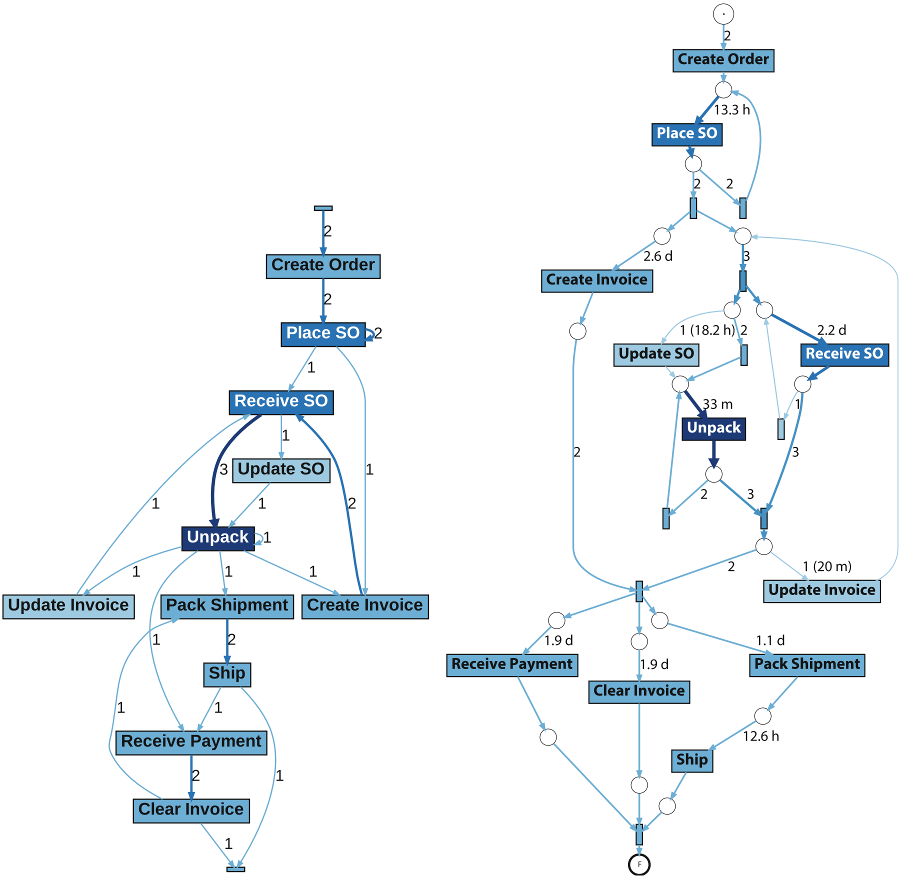


Fig. 3. Directly-follows graph of event log of Table 2 (left) and Inductive Miner model (right) show false dependencies.

The performance information in the IM model suggests that

- the mean time for receiving a *Supplier Order* after placement is 2.2d while A was received within 3d after placement (e_3-e_6) and B was received within 5d after placement (e_4-e_{19}) and within 3d after the last update (e_7-e_{19}).

This false behavioral information makes it impossible to properly locate deviating behaviors and causes for delays, e.g., the reasons why both orders were not shipped within 6 days.

3.3 Correct Behavioral Information: Local Directly-Follows

The reason why the event log in Table 2 contains false behavioral information is the following:

- Events that are (transitively) correlated to the global case identifier *Order* via a 1-to-m relationship are visible to multiple cases, and thus extracted multiple times, e.g. e_3.
- Extracting events from multiple different entities and ordering them by time from the perspective of the global case identifier *Order* constructs a temporal order between events that are actually unrelated, e.g., e_6 and e_7.

We can avoid both problems by simply *not* extracting all events towards a single case identifier, but keeping all events local to the entities they are *directly* correlated to. To analyze behavior, we only construct a temporal order between events that are related, e.g., correlated to the same entity.

In other words, instead of defining one global directly-follows relation for all events based on a global case identifier, we define a local directly-follows relation *per* entity [30, Def. 4.6].

Definition 6 (Directly-Follows (per Entity)). *Let* $T = (E, Attr, \#, ENT)$ *be an event table with entities. Let* $n \in Entities(ent, T)$ *be an entity of type* $ent \in ENT$.

Let $e_1, e_2 \in E$ *be two events;* e_2 directly follows e_1 from the perspective of n, *written* $e_1 \lessdot_{n,T} e_2$ *iff*

1. $(e_1, n), (e_2, n) \in corr_{ent,T}$ *(both are correlated to* n*),*
2. $e_1.time < e_2.time$ *(*e_1 *occurred before* e_2*),*
3. *and there is no other event* $(e', n) \in corr_{ent,T}$ *with* $e_1.time < e'.time < e_2.time$

For example, while e_7 directly follows e_6 globally for $O2$, they do not follow each other locally from the perspective of $O2$. Instead, from the perspective of $O2$, e_7 directly follows e_4, i.e., $e_4 \lessdot_{B,T} e_7$. Interestingly, also $e_2 \lessdot_{O2,T} e_7$ and $e_6 \lessdot_{R2,T} e_7$ hold. That means e_7 directly follows *three* different events as seen from three different perspectives: the Supplier Order B, the Order $O2$ and resource $R2$.

We cannot represent this information in a single table or a sequential event log. Extracting a *collection* of related sequential event logs from event tables [46] and relational databases [41] results in collection of directly-follows relations per entity-type. However, the behavioral information remains separated per entity type, hindering reasoning about the process as a whole [25]. We therefore turn to a graph-based data model.

4 Event Knowledge Graphs

Our primary aim is to model multiple local directly-follows relations (see Definition 6) over events correlated to multiple entities. To construct these relations, we also have to model entities, relations between entities, and correlations of entities to events (see Sect. 2). A *typed* graph data model such as *labeled property graphs* [48] allows to distinguish different types of nodes (events, entities) and relationships (directly-follows, correlated-to). We adopt labeled property graphs to construct a *knowledge graph* [33] of a process from event data, to augment this graph with further knowledge, and to even perform process mining analysis within a graph. Section 4.1 defines the generic data model of labeled property graphs which we use in Sect. 4.2 to define *event knowledge graphs* and "directly-follows" paths in an event knowledge graph. In Sect. 4.3 we discuss how to algorithmically construct an event knowledge graph from an event table.

4.1 Labeled Property Graphs

A labeled property graph is a graph where each node and each directed edge (called relationship) has a type, called *label*. Further, each node and each relationship can carry attribute-value pairs as properties. For the remainder, we fix a set λ_N of node labels, a set λ_R of relationship labels, and a set *Attr* of property names over a value domain *Val*.

Definition 7 (Labeled Property Graph). *A* labeled property graph *(LPG)* $G = (N, R, \lambda, \#)$ *is a graph with* nodes N*, and* relationships R *with the following properties:*

1. *Each node* $n \in N$ *carries a* label $\lambda(n) \in \Lambda_N$*.*
2. *Each relationship* $r \in R$ *carries a label* $\lambda(r) \in \Lambda_R$ *and defines a directed* edge $\overrightarrow{r} = (n_{source}, n_{target}) \in N \times N$ *between two nodes.*
3. *Any node* n *and relationship* r *can carry* properties *as attribute-value pairs via function* $\# : (N \cup R) \times Attr \nrightarrow Val$

We write $x.a = v$ for $\#(x, a) = v$ and $x.a =\perp$ if a is undefined for x. We write $N^\ell = \{n \in N \mid \lambda(n) = \ell\}$ and $R^\ell = \{r \in R \mid \lambda(r) = \ell\}$ for the nodes and relationships with label ℓ, respectively. We also write $(n_1, n_2) \in R^\ell$ if there exists $r \in R^\ell$ with $\overrightarrow{r} = (n_1, n_2)$.

Figure 4 shows an example of a labeled property graph, defining 5 nodes with label *Event*, 3 nodes with label *Entity*, 7 relationships with label *corr*, and 4 relationships with label *df*.

We here also provide some notation for standard operations on LPGs. Let $G_1 = (N_1, R_1, \lambda_1, \#^1)$ and $G_2 = (N_2, R_2, \lambda_2, \#^2)$ be two LPGs.

G_2 is a *sub-graph* of G_1, written $G_2 \subseteq G_1$, iff $N_2 \subseteq N_1, R_2 \subseteq R_1, \lambda_2 = \lambda_1|_{N_2 \cup R_2}, \#_2 = \#_1|_{N_2 \cup R_2}$. The *union* of G_1 and G_2 is $G_1 \cup G_2 = (N_1 \cup N_2, R_1 \cup R_2, \lambda_1 \cup \lambda_2, \#^1 \cup \#^2)$ under the assumption that $\lambda_1(x) = \lambda_2(x)$ and $\#^1_a(x) = \#^2_a(x)$ for all $a \in Attr$ for any $x \in (N_1 \cup R_1) \cap (N_2 \cup R_2)$. For a set $\mathbf{G} = \{G_1, \ldots, G_n\}$ of graphs, we write $\bigcup_{G \in \mathbf{G}} G = G_1 \cup \ldots \cup G_n$.

Labeled property graphs are a native data structure for knowledge graphs [33] and for a variety of *graph database systems* [48] that provide data management and query languages for reading and manipulating graphs [5].

4.2 Formal Definition of an Event Knowledge Graph

To precisely model event data in an LPG, we have to restrict ourselves to specific node labels for events and entities, and to specific relationship labels for correlation and directly-follows. Thereby, directly-follows relationships can only be

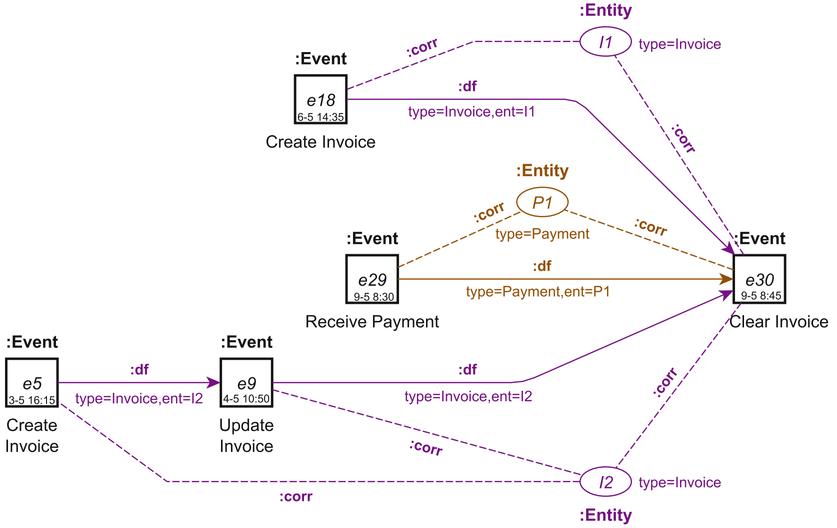


Fig. 4. Event knowledge graph of events $e_5, e_9, e_{18}, e_{29}, e_{30}$ of Table 2.

defined between events that are correlated to the same entity and directly follow each other from the viewpoint of that entity (Definition 6). This is formalized in the model proposed by Esser [25] which we here call *event knowledge graph*[4]

Definition 8 (Event Knowledge Graph). *An* event knowledge graph *(or just* graph*) is an LPG $G = (N, R, \lambda, \#)$ with node labels $\{Event, Entity\} \subseteq \Lambda_N$ and relationship labels $\{df, corr\} \subseteq \Lambda_R$ indicating "directly-follows" and "correlation" with the following properties.*

1. *Every event node $e \in N^{Event}$ records an activity name $e.act \neq \perp$ and a timestamp $e.time \neq \perp$.*
2. *Every entity node $n \in N^{Entity}$ has an entity type $n.type \neq \perp$.*
3. *Every correlation relationship $r \in R^{corr}, \overrightarrow{r} = (e, n)$ is defined from an event node to an entity node , $e \in N^{Event}, n \in N^{Entity}$; we write $n \in corr(e)$ and $e \in corr(n)$ as shorthand.*
4. *Any directly-follows relationship $df \in R^{df}, \overrightarrow{df} = (e_1, e_2)$ is defined between event nodes $e_1, e_2 \in N^{Event}$ and refers to a specific entity $df.ent = n \in N^{Entity}$ such that*
   (a) e_1 and e_2 are correlated to entity n: $(e_1, n), (e_2, n) \in R^{corr}$;
   (b) e_1 occurs before e_2: $e_1.time < e_2.time$; and
   (c) there is no other event $e' \in N^{Event}$ correlated to $n, (e', n) \in R^{corr}$ that occurs in between $e_1.time < e'.time < e_2.time$

[4] The initially chosen term "event graph" [25,38] which seems natural and shorter has previously been coined for a model for discrete event simulation [49]. At the same time, we will see that the proposed event *knowledge* graph model allows to capture more than just events.

We write $df.type = df.ent.type$ *and* $(e1, e2) \in R_n^{df}$.

Figure 4 shows an event knowledge graph for entities $I1, I2, P1$ of Table 2 and their correlated events. Each *df* relationship is defined between any two subsequent events correlated to the same entity. In the following, we omit the labels and use dashed edges for *corr* relationships, square nodes for *Event* nodes, and ellipses for *Entity* nodes.

A path along *df*-relationships corresponds to a trace in a classical event log. A *path* in a graph G is a sequence $\mathbf{r} = \langle r_1, \ldots, r_k \rangle \in R^*$ of consecutive relationships, i.e., the target node of $\overrightarrow{r_i} = (n_{i-1}, n_i)$ is the start node of $\overrightarrow{r_{i+1}} = (n_i, n_{i+1})$, $1 \leq i < k$.

Definition 9 (df-path). *Let* $G = (N, R, \lambda, \#)$ *be an graph.*

A path $\mathbf{r} = \langle r_1, \ldots, r_k \rangle \in (R^{df})^*$ *of df-relationships is a* directly-follows path (df-path) *iff all relationships are defined for the same entity, i.e., for all* $1 \leq i < k$, $r_i.ent = r_{i+1}.ent = n$; *we also say* $\mathbf{r}$ *is a df-path for entity* n.

$\mathbf{r}$ *is* maximal *iff there is no other df-relationship* $r \in R^{df}$ *so that* $\langle r, r_1, \ldots, r_k \rangle$ *or* $\langle r_1, \ldots, r_k, r \rangle$ *is also a df-path.*

For a path $\mathbf{r} = \langle r_1, \ldots, r_k \rangle \in (R^{df})^*, \overrightarrow{r_i} = (e_{i-1}, e_i)$ we write just the sequence of its nodes $\langle e_0, \ldots, e_k \rangle$ in case the correlated entity is clear. The graph in Fig. 4 defines three DF-paths: for $I1$: $\langle e_{18}, e_{30} \rangle$, for $I2$: $\langle e_5, e_9, e_{30} \rangle$, and for $P1$: $\langle e_{29}, e_{30} \rangle$.

Event knowledge graphs can be efficiently stored and queried using graph database systems [25]. This enables retrieving df-paths from graph databases using query languages, such as Cypher [25,33]. While the nodes and relationships of Definition 8 can also be encoded in RDF [11], the df-paths rely on attributes of relationships (Definition 9) which are not supported by RDF but by LPGs.

Alternative formalizations of Definition 8 define just a partial order over events [4,30,55,56] describing the local directly-follows relation wrt. various entities 6. Such a partial order view is equivalent to a family of df-paths [30, Cor. 4.9]. This equivalence allows to switch perspectives depending on the analysis task at hand.

4.3 Obtaining an Event Knowledge Graph from an Event Table

Event data is (currently) not recorded in the form of a graph, but for example in the form of an event table T with multiple entities (Definition 2). We obtain an event knowledge graph from an event table T in three steps.

1. Create an event node $e \in N^{Event}$ for each event record in the event table T.
2. *Infer entities and correlation relationships* from the event attributes: For each unique entity identifier found at some event e, create an entity node n and a *corr* relationship from e to n.
3. *Infer directly-follows relationships* between all events $e_1, \ldots, e_k$ with a *corr* relationship to the same entity node n.

We now explain and define each step along the running example of Table 1. We assume as input an event table $T = (E, Attr, \#^T, ENT)$ with multiple entities as stated in Definition 2. The central requirement is that each unique entity type $ent \in ENT \subseteq Attr$ is explicitly recorded as a dedicated attribute (column) of T, and that each value in column ent is an entity identifier.

Step 1: Create Event Nodes. We start by translating each event record in event table T into an event node in graph G.

Definition 10 (Event nodes from an event table). *Let $T = (E, Attr, \#^T, ENT)$ be an event table with entities. The* event nodes of T *are the graph $G_T^{Event} = (N^{Event}, \emptyset, \lambda, \#^G)$ with*

1. *$N^{Event} = E$, i.e., each event of T becomes an event node, and*
2. *$\#_a^G(e) = \#_a^T(e)$ for all $a \in Attr$, i.e., each event keeps all attributes from T as properties in G.*

The resulting graph G is a set of disconnected *Event* nodes only.

Step 2: Create Entity Nodes and Correlation Relationships. Each attribute of an event e in T that refers to an entity, e.g., $e.ent = \{n\}$, is now a property of the event node e in G. The basic idea is to "push out" this property: we make each unique value n an *Entity* node n and link e to n by a *corr* relationship. The following definition constructs a small graph $G^{corr}(n)$ that does exactly this. We then use graph union $G \cup \bigcup_n G^{corr}(n)$ to add them to G. The reason for doing so is that we can later calculate with various subgraphs.

Definition 11 (Entity and correlation inference). *Let $G = (N, R, \lambda, \#^G)$ be a graph and ENT be known entity types.*

Given a property name $ent \in ENT$, each property value $e.ent$ we find on an event node $e \in N^{Event}$ is an entity identifier of ent in G*: $Entities(ent, G) = \{n \mid \exists e \in N^{Event} : n \in e.ent\}$, see Definition 3.*

Let $n \in Entities(ent, G)$ be an identifier of type $ent \in ENT$. The entity and correlation inferred for n in G *is the graph $G^{corr}(n) = (N', R', \lambda' \#')$ with:*

1. *entity node $N'^{Entity} = \{n\}$ with $\#'_{type}(n) = ent$;*
2. *event nodes $N'^{Event} = \{e \in N^{Event} \mid n \in e.ent\}$ with $\#'(e) = \#(e)$ for each $e \in N'^{Event}$, i.e., each e is correlated to n, see Definition 4; and*
3. *correlation relationships $r_{e,n} \in R'^{corr}$, $\overrightarrow{r}_{e,n} = (e, n)$ iff $n \in e.ent$.*

We can infer entities and correlation on *any* event knowledge graph, not just the graph produced by Definition 10. This allows us to apply Definition 11 multiple times in any order. We can infer entities and correlation for an entity type ent by $G^{corr}(ent) = \bigcup_{n \in Entities(ent,G)} G^{corr}(n)$. We can add the inferred entities and correlation to graph G for all entity types ENT by graph union $G \cup \bigcup_{ent \in ENT} G^{corr}(ent)$. In the result, each value $n \in Entities(ent, T)$ becomes a new node n with $n.type = ent$. Correspondingly, each pair $(e, n) \in corr_{ent,T}$ becomes a new relationship of type $corr$ from e to n.

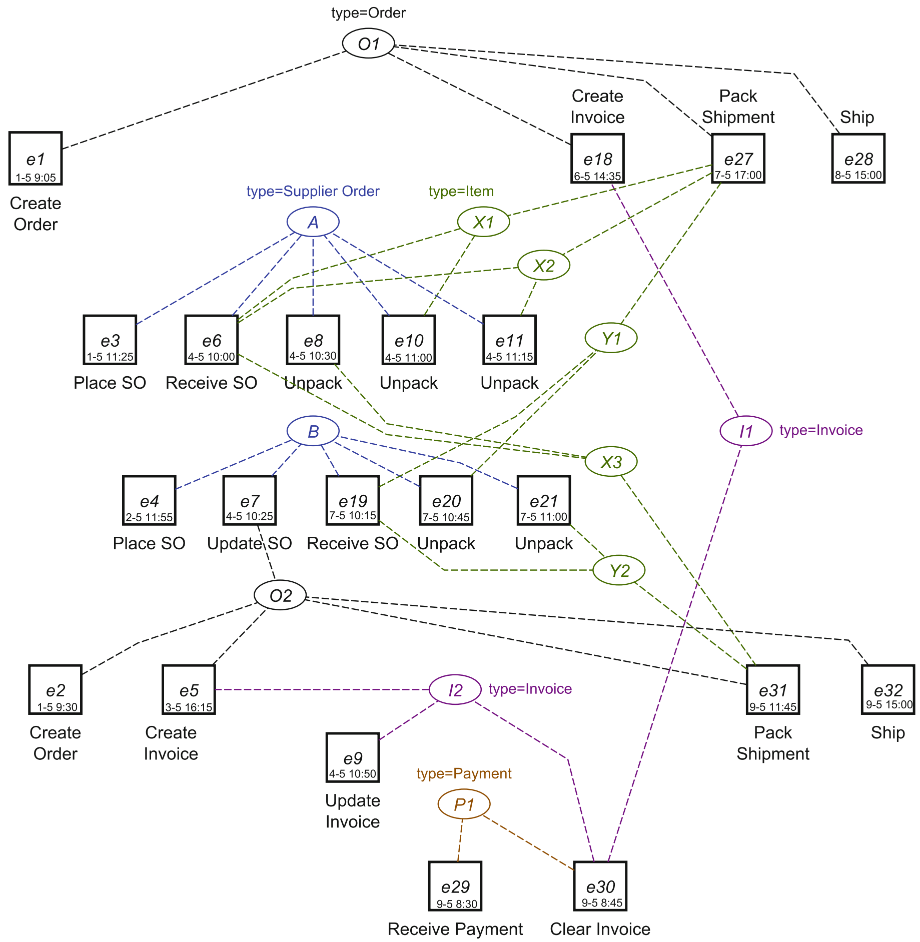


Fig. 5. Event graph of events of Table 2 without directly-follows relationships.

For example, applying Definition 10 on the event table of Table 2 results in the event nodes $e_1, \ldots, e_{11}, e_{18}, \ldots, e_{21}, e_{27}, \ldots, e_{32}$ shown in Fig. 5. Inferring entities and correlation for entity types *Order*, *Supplier Order*, *Item*, *Invoice*, and *Payment* adds the entity nodes and correlation edges shown in Fig. 5. In this graph we see that events $e_1, e_{18}, e_{27}, e_{28}$ are the events correlated to entity $O1$ of type *Order*. Moreover, event e_{18} is correlated to two entities *Order* $O1$ and *Invoice* $I1$; event e_{27} is correlated to four entities *Order* $O1$, *Item* $X1$, *Item* $X2$, and *Item* $Y1$.

Step 3: Infer Local Directly-Follows Relations. We now can infer the local directly-follows relation (Definition 6) and materialize it as *df*-relationships between event nodes. Again, the basic idea is simple: for each entity node n we retrieve all events $e_1, \ldots, e_n$ with a *corr*-relationship from e_i to n. We order

$e_1, \ldots, e_n$ by time and define a new *df*-relationship r from e_i to e_{i+1}; to remember for which entity r holds, we set $r.ent = n$.

As before, we do not add the *df*-relationships directly to G but construct a separate graph $G^{df}(n)$. We then add to G by graph union $G \cup \bigcup_n G^{df}(n)$ which later allows us to calculate with graphs.

Definition 12 (df inference). *Let $G = (N, R, \lambda, \#)$ be a graph. Let $n \in N^{Entity}$. Let $\langle e_0, \ldots, e_k \rangle$ be the sequence of events $\{e_0, \ldots, e_k\} = corr(n)$ correlated to n and sorted by time: $e_{i-1}.time < e_i.time, 1 \leq i \leq k$.*

The df-relationships inferred for *n in G is the graph $G^{df}(n) = (N'^{Event}, R'^{df}, \lambda', \#')$ with*

1. *event nodes $N'^{Event} = \{e_0, \ldots, e_k\}$, and*
2. *for each $1 \leq i \leq k$ one* df-*relationship $r_i \in R'^{df}$ with $\overrightarrow{r_i} = (e_{i-1}, e_i), \#'_{ent}(r_i) = n, \#'_{type}(r_i) = \#_{type}(n)$.*

We can only infer a df-relationship for entity n if $|corr(n)| > 1$. Thus, for df-inference to have any effect, we have to have inferred the entity n and correlation using Definition 11 and there are at least two events correlated to n. As for entity and correlation inference, we can add the inferred df-relationships to G by graph union $G \cup \bigcup_{n \in N^{Entity}} G^{df}(n)$.

For example, if we infer the df-relationships for each entity in the graph of Fig. 5 and add them to that graph, we obtain the graph shown in Fig. 6. Note that we only show the *corr* relationships to the first event of each entity for readability. This graph explicitly models the events, entities, correlation, and local directly-follows relations of all events in Table 2.

Complete Procedure. The following definition summarizes how to apply the above three definitions to obtain an event knowledge graph of an event table T.

Definition 13 (Event knowledge graph of an event table). *Let $T = (E, Attr, \#^T, ENT)$ be an event table with entities. The event table T defines the* graph $G = (N, R, \lambda, \#^G)$ of T *as follows:*

1. *Obtain the graph of event nodes G^{Event} of T (Definition 10).*
2. *Infer the entities and correlation for each entity type $ent \in ENT$ from G^{Event} (Definition 11), i.e., $G^{corr} = \bigcup_{ent \in ENT} G^{corr}(ent)$ which results in the intermediate graph $G^{Event} \cup G^{corr} = (N^{Event} \cup N^{Entity}, R^{corr}, \lambda, \#^G)$.*
3. *Infer the df-relationships $G^{df} = \bigcup_{n \in N^{Entity}} G^{df}(n)$ from $G^{Event} \cup G^{corr}$ (Definition 12) and return $G = G^{Event} \cup G^{corr} \cup G^{df}$.*

From Definition 10–13 follows that the df-relationships in graph G materialize the local directly-follows relation of event table T (Definition 6).

Lemma 1. *Let $G = (N, R, \lambda, \#^G)$ be the event knowledge graph of event table $T = (E, Attr, \#^T, ENT)$ with entities. For any entity $n \in Entities(ent, T)$, $ent \in ENT$ holds $e_1 \lessdot_{n,T} e_2$ (e_2 directly follows e_1 from the perspective of n) iff $(e_1, e_2) \in R^{df}_n$.*

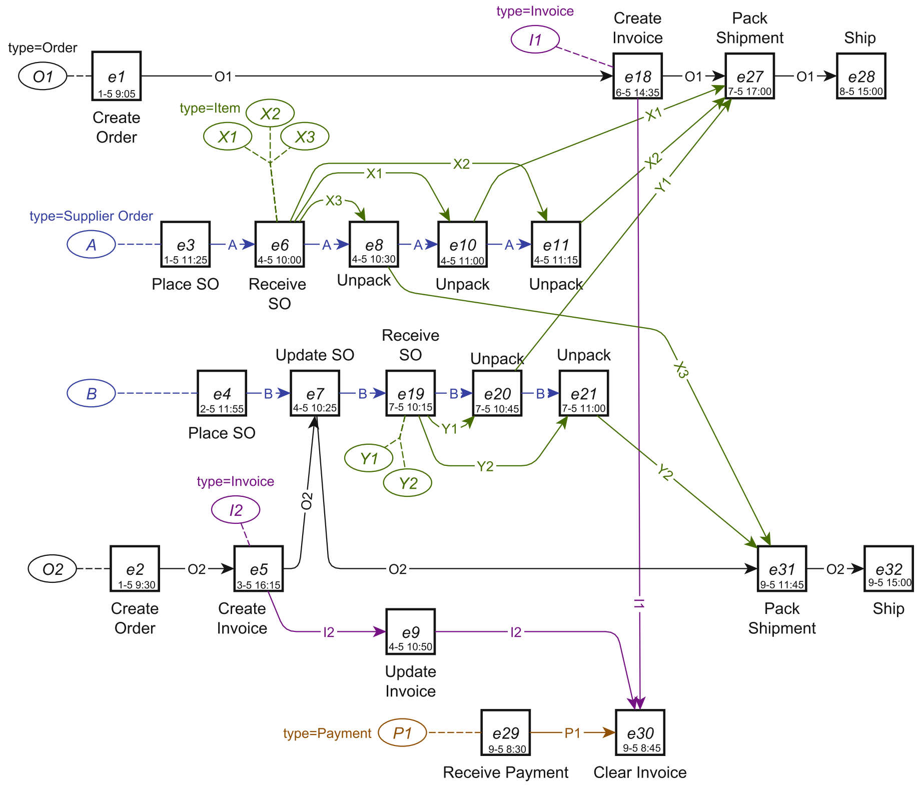


Fig. 6. Event graph of events of Table 2 after inferring directly-follows relationships.

4.4 Inferring Entity Interactions

The procedure of Definition 13 infers the local directly-follows relation for each entity in the graph. However, there are also important behavioral dependencies in the process *between* related entities, such as *Orders* and *Payments*, that are not visible in the graph of Fig. 6.

We know from Fig. 1 that shipping $O2$ has to wait until the invoice of $O1$ has been cleared by the related payment $P1$, but the graph of Fig. 6 suggests that e_{31} of $O2$ does not depend on e_{30} of $P1$ or any event of $O1$. This is because there is no entity correlated to both e_{31} and e_{30} or any event of $O1$.

Our analysis in Sect. 2.3 found that *Orders* are related to *Payments*. We can materialize this information in an event knowledge graph. We apply Definition 5 on all *Event* nodes to obtain relation $R_{(ent_1,ent_2)}$ between any two (interesting) entity types ent_1, ent_2. For each pair, $(n_1, n_2) \in R_{(ent_1,ent_2)}$ we add a new relationship with label *related* from entity node n_1 to entity node n_2. Figure 7 illustrates the result of this step for $(Order, Invoice)$ and $(Invoice, Payment)$. We can infer transitive relationships by materializing paths of *related*-relationships (ignoring their directions) as new *related*-relationships.

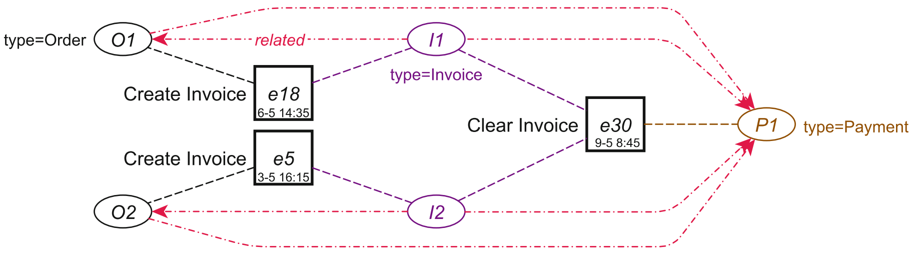


Fig. 7. Inferring relations between *Orders*, *Invoices*, and *Payments*.

For example, we materialize $\langle O1, I1, P1\rangle \in (R^{related})^*$ and $\langle O2, I2, P1\rangle \in (R^{related})^*$ as $(O1, P1), (O2, P1) \in R^{related}$ in Fig. 7. These steps obviously require domain knowledge to decide which potential relations to materialize, esp. when considering paths over n-to-1 and 1-to-n relationships [41].

We then can infer the behavior between two related entities by adapting entity and correlation inference (Definition 11) as follows [25]:

1. We *reify* the relation between two entity types ent_1 and ent_2 into a new *derived* entity type (ent_1, ent_2). That is, we make each pair $(n_1, n_2) \in R^{related}$ an entity node $(n_1, n_2) \in N^{Entity}$ with $(n_1, n_2).type = (ent_1, ent_2)$. For example, we create two entity nodes $(O1, P1), (O2, P1)$ of type *(Order,Payment)*. For traceability, we add a new relationship $d \in R^{derived}$ with label *derived* from entity (n_1, n_2) to n_1 and to n_2.
2. An event e is then correlated to a derived entity (n_1, n_2) iff e is correlated to n_1 or n_2 (or both). Formally, we add a new correlation relationship from e to (n_1, n_2) iff there is a correlation relationship $r \in R^{corr}$ from e to n_1 or n_2, i.e., $\overrightarrow{r} = (e, n_1)$ or $\overrightarrow{r} = (e, n_2)$.
3. Then we can treat any derived entity (n_1, n_2) just like any other entity and infer the df-relationships for (n_1, n_2), which results in a new path describing the interactions between n_1 and n_2.

Figure 8 shows the result of reifying the relation between *Order* and *Payment* entities of Fig. 7 into derived entities $(O1, P1)$ and $(O2, P1)$ of type $(Order, Payment)$ and inferring the df-relationships for this entity type. We now inferred df-paths from *Create Invoice* in $O1$ (e_{18}) via *Clear Invoice* in $P1$ (e_{30}) to *Pack Shipment* in $O2$ (e_{31}).[5]

Not all df-relationships for $(O1, P1)$ and for $(O2, P2)$ provide new information. For example in Fig. 8, $(e_2, e_5) \in R^{df}_{O2}$ and $(e_2, e_5) \in R^{df}_{(O2,P1)}$ run in parallel.

We say that a df-relationship $(e_1, e_2) \in R^{df}_{(n_1,n_2)}$ of a derived entity (n_1, n_2) *provides new information* if there is not already an existing df-relationship

[5] Our example here exploits that both orders of the same customer have invoices cleared by the same payment. For the more general case, we would have to include the customer in the data and infer the dependency via the customer entity.

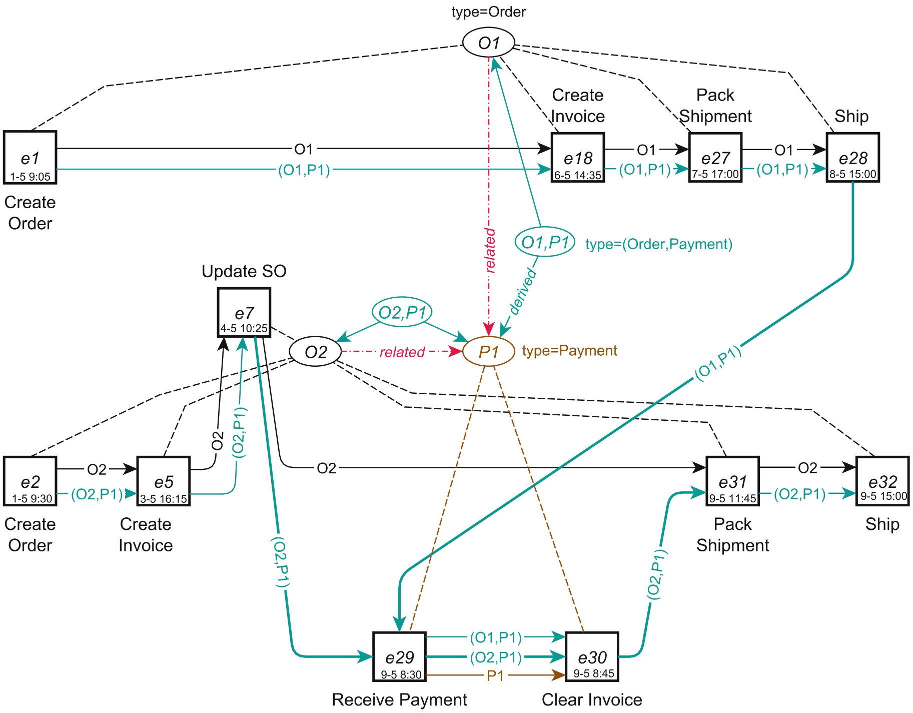


Fig. 8. Result of reifying the relation between *Order* and *Invoice* entities of Fig. 6 into a derived entity of type (*Order*, *Invoice*) and inferring the df-relationships for this entity type.

$(e_1, e_2) \in R^{df}_{n_1}$ or $(e_1, e_2) \in R^{df}_{n_2}$ for one of the original entities n_1 or n_2. Thus, a df-relationship (e_1, e_2) provides new information if it actually describes an interaction from n_1 to n_2 or vice versa. In Fig. 8, (e_7, e_{29}), (e_{28}, e_{29}), and (e_{30}, e_{31}) provide new information.

In principle we should keep only those *df*-relationships of a derived entity (n_1, n_2) that provide new information. However, we can best study the interaction between n_1 and n_2 when all *df*-relationships between n_1 and n_2 are part of a path related to (n_1, n_2). We therefore keep all *df*-relationships of (n_1, n_2) that either provide new information or are between two *df*-relationships of the *df*-path for (n_1, n_2) that do provide new information. In Fig. 8, for $(O2, P1)$, we keep (e_7, e_{29}) and (e_{30}, e_{31}) (provide new information) and also (e_{29}, e_{30}) (between df-relationships that provide new information); for $(O1, P1)$, we only keep (e_{28}, e_{29}).

The complete graph for Table 1 after inferring the *df*-relationships between *Order* and *Payment* entities is shown in Fig. 9.

4.5 Creating Event Knowledge Graphs from Real-Life Data

This method for constructing event knowledge graphs uses basic principles of information inference: (1) construct entities and correlation based on the presence of an entity identifier or a relation; and (2) derive a local directly-follows relation from the viewpoint of *each* entity. Our definitions assume the data to be accurate wrt. the real process, for instance, that entity identifiers and time stamps are recorded correctly and precise; otherwise further preprocessing is required [30,44,47].

All steps of the method can be implemented as a series of Cypher queries[6] to construct event knowledge graphs in a graph database for our running example [28] as well as for various real-life datasets comprising single and multiple event tables [24]; several event knowledge graphs of real-life processes are available [19–24]. A variant of event knowledge graphs, called *causal event graph* that only models events but not the entities, can be extracted automatically from relational databases [56].

In the following, we exploit the flexibility of LPGs that underly event knowledge graphs to infer and materialize further behavioral information, going beyond what event tables or event logs can describe.

5 Understanding Behavior over Multiple Entities

The event knowledge graph of Fig. 9 we obtain with the method of Sect. 4 explicitly models what we observed earlier in Sect. 1: the behavior of the different entities forms a complex *network* of synchronizing *df*-paths. This section first discusses how to interpret df-paths (Sect. 5.1) and how they synchronize (Sect. 5.2). We then discuss querying graphs through selection of entities and projection onto events in Sect. 5.3; we apply these operations to understand why the retailer of our example in Sect. 1 could not ship orders within the promised 6 days. We finally introduce aggregation in Sect. 5.4 which we use to discover basic process models directly within event knowledge graphs in Sect. 5.5.

5.1 How to Read Df-Paths in an Event Knowledge Graph

We discuss how to read *df*-paths over events based on running example of Fig. 6.

In a classical event log, each trace has a unique initial event and a unique final event indicating the start and completion of a process execution. A graph has multiple initial and final events – one per entity. Event e is *starting* or *ending* event if it has no incoming or outgoing *df*-relationship at all, e.g., $e_1, \ldots, e_4$, and e_{32}. Event e is *starting* or *ending* event for entity n if it has no incoming or outgoing *df*-relationship for n. For example, e_{11} is the ending event of the *df*-path for A but it still has an outgoing *df*-relationship for $X2$. Some events are starting/ending events for *multiple* *df*-paths or entities. For example, e_6 is the

[6] https://github.com/multi-dimensional-process-mining/eventgraph_tutorial.

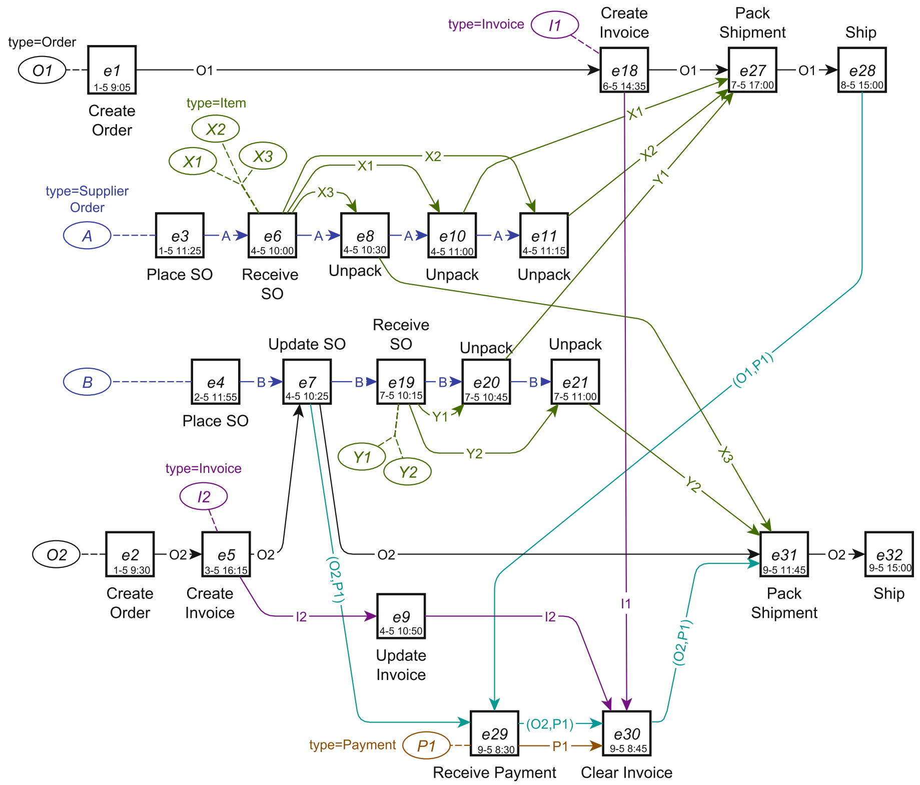


Fig. 9. Complete event knowledge graph of event table Table 1.

starting event for $X1, X2, X3$ and e_7 is the starting event for $Y1, Y2$ while e_{27} is the ending event for $X1, X2, Y1$ and e_{31} is the ending event for $X3, Y2$.

We call an event *intermediate* in a df-path of an entity n if it is not a starting or ending event in the df-path of n. For example, e_6 is an intermediate event of A.

In graph in Fig. 9 we see that the df-paths of entities of the same type are rather similar to each other.

- $O1$ and $O2$ both start with *Create Order* and end with *Ship* events with *Create Invoice* followed by *Pack Shipment* in between.
- A and B both start with *Place SO* (eventually) followed by *Receive SO*, ending with multiple *Unpack* events. Specifically
- Items $X1, \ldots, Y2$ start with *Receive SO* followed by *Unpack* and end with *Pack Shipment*
- $I1$ and $I2$ start with *Create Invoice* and end with *Clear Invoice*

Note that the graph no longer shows any directly-follows relation from *Receive SO* to *Update SO* that was falsely observed in Sect. 3. We can also analyze time differences between events on the df-path. For example, in Sect. 1 we stated that each Supplier Order is to be received within 3 days of placing the order.

- On the df-path of A, event e_6 (*Receive SO* for A, 4-5 10:15) is directly preceded by e_4 (*Place SO* for A, 1-5 11:25) which is within 3 days as required.
- On the df-path of B, event e_{19} (*Receive SO* for B, 7-5 10:15) is directly preceded by e_7 (*Update SO* for B, 4-5 10:25) which is within 3 days, but 6 days since e_4 (*Place SO* for B, 1-5 11:25). Thus, while the supplier delivered within the required 3 days since *Update SO*, the update itself introduced a 3-day delay.

Thus, the graph now shows temporal information and delays for individual entities correctly, in contrast to the classical event log of Sect. 3.

5.2 How to Read Synchronization in a Graph

Analyzing the df-paths for $O1$ and $O2$ also shows that none of the orders were shipped within 6 days: $e_{20}.time - e_1.time > 7\,days$ and $e_{32}.time - e_2.time > 8\,days$. As completing the orders depends on other entities, i.e., the items, we now analyze entity interactions through synchronization of df-paths.

A df-path $\mathbf{r} = \langle e_0, \ldots, e_k \rangle$ *goes through* an event e iff $e = e_i, 0 \leq i \leq k$. An event e is *local* to an entity n if there is only one df-path of entity n that goes through e, e.g., e_1, e_2, e_{32}. Two or more entities $n_1, \ldots, n_k$ *synchronize* in a *shared* event e if two or more df-paths of $n_1, \ldots, n_k$ go through e, e.g., e_7 synchronizes Supplier Order B and Order $O2$ whereas e_{19} synchronizes Supplier Order B and Items $Y1$ and $Y2$.

Reading Entity Creation and Updates. We now discuss different interpretations of entities $n_1, \ldots, n_k$ *synchronizing* in a shared event.

Event e *intermediately synchronizes* entities $n_1, \ldots, n_k$ when e is an intermediate event for $n_1, \ldots, n_k$. We can interpret an intermediate synchronization as an update or state change of one or more entities that requires the involvement of the other entities. For example, event e_7 intermediately synchronizes Order $O2$ and Supplier Order B to update B based on the information in $O2$; event e_8 updates both Supplier Order A and Item $X3$. Which entity changes state in e_8 is not visible in the graph of Fig. 9.

An event e that is intermediate for one entity n but a starting event for entities $n_1, \ldots, n_k$ can be interpreted as *entity n "created" or "initiated" entities $n_1, \ldots, n_k$*. For example, Supplier Order A created Items $X1, X2, X3$ in e_6, and Supplier Order B created $I2$ in e_5. Correspondingly, an event e that is intermediate for entity n and ending event for $n_1, \ldots, n_k$ is "closing" or "completing" entities $n_1, \ldots, n_k$. For example, Order $O1$ "completes" items $X1, X2, Y1$ in e_{27}.

An event e where multiple entities $n_1, \ldots, n_k$ of the same type synchronize is a *batching* event for $n_1, \ldots, n_k$ [36,42,55]. For example, e_{27} batches $X1, X2, Y1$, e_{30} batches $I1, I2$, and e_{31} batches $X3, Y2$.

However, we have to be careful with those interpretations as, both, the graph and the data from which it was created may be incomplete. Entities that are "created" or "closed" may continue to exist both prior and after the data recorded, e.g., all Items $X1, \ldots, Y2$ certainly exist prior to this process and after it, thus e_6 and e_{27} only show when these items entered the visibility or scope of our observations. Likewise, a starting event e for an entity n that is *not* an intermediate event for another entity n_2 does *not* describe how n was created. For example, $e_1, \ldots, e_4$ do not explain how $O1, O2, A, B$ were created. This is because our graph of Fig. 9 is incomplete as we did *not* (a) infer the *Resource* entity and the corresponding *df*-relationships from Table 1 and (b) we only recorded data in a limited time window. A helpful principle to check for incompleteness in distributed behavior is due to C.A. Petri [27]: most events happens due to a synchronous interaction of two or more entities, and most physical entities are never created from nothing and never disappear into nothing.

Reading Entity Interactions. Events and df-paths describe different modes of interaction. An event e where the df-paths of n_1 and n_2 synchronize is a *synchronous interaction.* A df-path for entity n describes an *asynchronous interaction* between n_1 and n_2 if n synchronizes both with n_1 and n_2 in different events. If the df-path for n has only 2 events $\langle e_1, e_2 \rangle$ then we can interpret entity n as *message* from n_1 to n_2. We can interpret an event e that is the ending event of entity n_1 and the starting event of entity n_2 as a *handover* from n_1 to n_2. In Fig. 9, e_7 is a synchronous interaction of $O2$ and B, the df-path of $Y1$ describes an asynchronous interaction from B to $O2$, and e_{28} is a handover from $O1$ to $(O1, P1)$.

If two entities n_1 and n_2 never synchronize in a shared event but there is at least one asynchronous interaction between n_1 and n_2, then n_1 and n_2 *interact asynchronously.* If all asynchronous interactions, i.e., df-paths, only go from n_1 to n_2, then the interaction is *one-directional*, and it is *bi-directional* otherwise. In Fig. 9, A and $O1$ interact asynchronously and one-directional (from A to $O1$ via $X1$), $O2$ and $P1$ interact asynchronously and bi-directional (via $(O2, P1)$).

n_1 and n_2 *interact indirectly* if for any two events e_1 of n_1 and e_2 of n_2 the shortest df-path from e_1 to e_2 involves df-relationships from multiple other entities. For example, $O1$ interacts indirectly with $O2$ via $(O1, P1)$ and $(O2, P2)$ (df-path $\langle e_{28}, e_{29}, e_{30}, e_{31} \rangle$).

Finally, n_1 and n_2 *do not interact* if there is no df-path from n_1 to n_2, or vice versa. For example, A and B do not interact. Note, however, that (indirect) interactions via other entities as well as non-interaction are subject to which entities have been included in the construction of the graph and which relations have been reified into derived entities.

Reading Event Dependencies and Delays. We observed in Sect. 5.1 that neither $O1$ nor $O2$ was shipped within 6 days as required in Sect. 1. We now want to analyze which entities, that synchronized with $O1$ and $O2$, delayed either order to be shipped on time.

Consider an event e that synchronizes the df-paths of multiple entities $n_1, \ldots, n_k$. Event e *directly depends on* any event e_i that directly precedes e via an incoming df-relationship $(e_i, e) \in R^{df}_{n_i}, 1 \leq i \leq k$ along entity n_i. We call $e.time - e_i.time$ the *delay* between e_i and e.

Suppose $e_1, \ldots, e_k$ are sorted on their delay to e. Event e_1 was the first event that directly preceded e, i.e., e could not have occurred earlier than e_1. The entity n_1, for which $(e_1, e) \in R^{df}_{n_1}$ was observed, was the first entity ready to synchronize in e. We can interpret that each later event $e_i, i > 1$ *delayed* the synchronization in e as entity n_i became ready to synchronize later than n_1 did, with e_k and n_k delaying e the most.

For example in Fig, 9, e_{31} (*Pack Shipment* for $O2$) depends on e_7, e_8, e_{21}, e_{30} along entities $O2$, $X3$, $Y2$, and $(O2, P1)$ with delays of 3 days, 3 days, 2 days, and 3 h, respectively. While $O2$ was first ready to synchronize in e_{31} after e_7 (*Update Order*); e_{31} was delayed most by e_{30} (*Clear Invoice* for $I1, I2$) along $(O2, P1)$.

For a given event e, we can build the set $delay^*(e)$ of transitive predecessors that delayed e the most, by first adding event e' that delayed e most, then adding event e'' that delayed e' most, etc. For example in Fig. 9, $delay^*(e_{32}) = \{e_{31}, e_{30}, e_{29}, e_{28}, e_{27}, e_{20}, e_{19}, e_7, e_5, e_2\}$.

Comprehending such subsets of events (and the dynamics they describe) is rather difficult. We use graph querying to reduce a graph to a subgraph of interesting events.

5.3 Basic Querying Operations

Similarly to classical event logs, we can also subset (or filter) event knowledge graphs for a more focused analysis. Recall that we have two basic operations to sub-setting classical event logs: selection (include only a subset of the cases with specific properties but keep all events in a case) and projection (keep all cases but keep only a subset of events with specific properties). The same operations can be applied on event knowledge graphs.

We *select* a subset of entities, but keep all event nodes correlated to the entities and all directly-follows relations between the events of these entities. Formally, given a graph G, we select entity nodes $N^{Entity}_{sel} \subseteq N^{Entity}$ from G by (1) removing all entity nodes $N^{Entity} \backslash N^{Entity}_{sel}$ and all adjacent *corr* relationships, then (2) removing all event nodes $e \in N^{Event}$ which no longer have any *corr* relationships (because none of their entities was selected) and the adjacent *df* relationships.

We *project* on a subset of events by keeping all entity nodes but only the selected event nodes; as this may interrupt df-paths (if an intermediate event gets removed) we have to recompute all df-relationships. Formally, given a graph G, we project onto event nodes $N^{Event}_{proj} \subseteq N^{Event}$ from G by (1) removing all *df*-relationships from G, (2) removing all event nodes $N^{Event} \setminus N^{Event}_{proj}$, and then (3) doing df-inference on the resulting graph (Definition 12).

The criteria by which we select events and entities can consider properties of events and entities but also relations to other event and entity nodes, and even

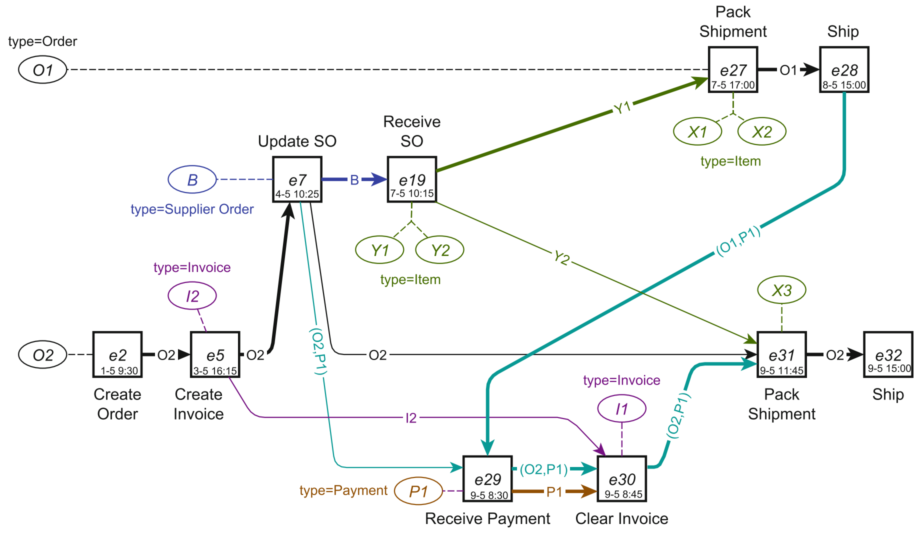


Fig. 10. Projection of Fig. 9 onto events that delayed most e_{28} and e_{32} and are not *Unpack* events. Bold df-relationships indicate which preceding event delayed an event the most.

more complex paths or sub-graphs. For example, to understand what caused delays in shipping order $O1$ (e_{28}) and $O2$ (e_{32}) while also removing unnecessary events, we can project the graph of Fig. 9 onto the events the (1) delayed either shipment the most (2) but without *Unpack* events. Formally, we project onto $(delay^*(e_{32}) \cup delay^*(e_{28})) \setminus \{e \in N^{Event} \mid e.act = Unpack\}$. Figure 10 shows the resulting graph. Note the new df-relationships $(e_5, e_{30}) \in R^{df}_{I2}$, $(e_{19}, e_{27}) \in R^{df}_{Y1}$, $(e_{19}, e_{31}) \in R^{df}_{Y2}$, obtained after doing df-inference over the remaining events.

In Fig. 10, we observe the following: *Pack Shipment* for $O1$ (e_{27}) was delayed by Item $Y1$ which was only ready for e_{27} after *Receive SO* (e_{19}). In turn, e_{19} was delayed by Supplier Order B with *Update SO* (e_7), which we already identified as cause for not receiving all items within 3 days in Sect. 5.1. *Pack Shipment* for $O2$ (e_{31}) was delayed by entity $(O2, P1)$, that means, by *Clear Invoice* (e_{30}) for the Payment $P1$ related to $O2$. *Receive Payment* for $P1$ (e_{29}) was delayed by $(O1, P1)$, that means, by *Ship* (e_{28}) for the related order $O1$.

Altogether, this allows us to pinpoint the bottlenecks in the process: *Update SO* delayed delivery of items $Y1, Y2$ needed for both $O1$ and $O2$, causing a delay in shipment for $O1$. The fact that the customer only paid and cleared both invoice $I1, I2$ after $O1$ was shipped delayed shipping $O2$ together with the retailer's policies.

5.4 Aggregating Events and Df-Relationships

Selection and projection allow to subset the data. Aggregation allows to materialize new nodes and relationships in the data. While the aggregation principle we explain here can be applied for many purposes, we specifically discuss it for

- aggregating sets of events into activities (or event classes), and
- aggregating df-relationships between events into corresponding relationships between activities.

The basic aggregation principle from sets of events to activities is formally identical to creating entity nodes from event properties as given in Definition 11.

- We select one event property that identifies a unique concept shared by many events, in this case the property *Activity*.
- For each value $c \in \{e.Activity \mid e \in N^{Event}\}$ of the *Activity* property that we find among the events in the graph, we create a new node c with label *Class* (representing the class of events with the same *Activity* property).
- We add an *observes* relationship from each event e to the *Class* node $c \in N^{Class}$ if $e.Activity = c$.
- We can also materialize how many events observe class $c \in N^{Class}$ in property $c.count$.

The yellow rounded rectangles in Fig. 11 represent the *Class* nodes of the events for Orders $O1$,$O2$ and Supplier Orders A,B. The dashed edges represent the *observes* relationship, e.g., e_2 and e_1 both observe *Create Order*.

We then can aggregate the df-relationships in a straight-forward way: for any two class nodes $c1$ and $c2$ we add a *df* relationship of type *ent* from c_1 to c_2 if there are corresponding events e_1 and e_2 that directly follow each other for *ent*, i.e., if $(e_1, c_1), (e_2, c_2) \in R^{observes}$ and $(e_1, e_2) \in R^{df}_{n}, n.type = ent$. We can also count how many df-relationships occur between events of c_1 and c_2 and add this as property to this relationship.

For example, in Fig. 11, we observe two df-relationships from *Create Order* to *Create Invoice* (e_1, e_{18}) and (e_2, e_5). Note, that this definition also creates self-loops around event classes, e.g., we observe three df-relationships from *Unpack* to *Unpack*. Also note that, as for events nodes, a class node can be part of *df*-relationships for multiple different entity types, e.g., *Update SO* is an activity that occurs for *Order* and *Supplier Order*.

5.5 Discovering Multi-entity Process Models

The aggregation operation of Sect. 5.4 essentially constructs a directly-follows graph. The key difference to the directly-follows graph of classical event logs is that each df-relationship between *Class* nodes is specific to one entity type. Thus, it respects the idea of the local directly-follows relation laid out in Definition 6. The resulting graph is a *multi-entity directly-follows graph*, also called *multi-viewpoint DFG* [4] or *artifact-centric model* [41].

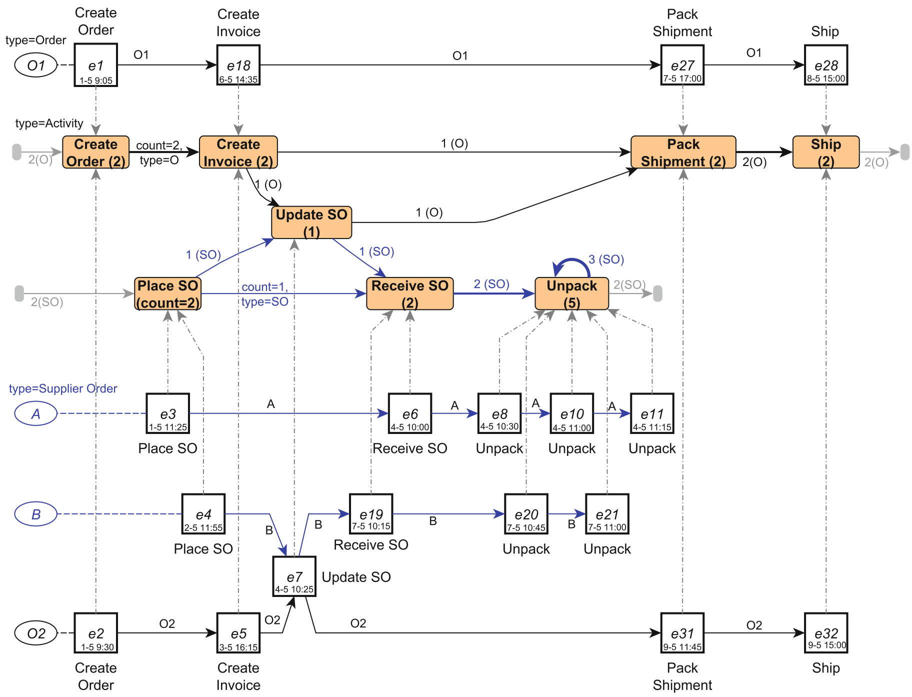


Fig. 11. Aggregating events to event classes and lifting the directly-follows relationships

Applying the event and df-aggregation of Sect. 5.4 to the graph of Fig. 9 results in the multi-entity DFG shown in Fig. 12. While the graph as a whole is rather complex, each edge is grounded in temporal relations of a specific entity type. Moreover, we can see that the behavior for each entity type is rather simple.

Event and df-aggregation can be implemented as simple, scalable queries[7] over standard graph databases, enabling efficient in-database process discovery [25,34]; the queries can be extended to filter based on frequencies or properties of the event knowledge graph [28].

An alternative representation of the multi-entity DFG is the proclet model [26] shown in Fig. 13. It is constructed by not creating a global *Class* node per unique *e.Activity* value in the data, but by creating a *Class* node per unique pair of activity name and entity type (*e.Activity*, *ent*). As a result, we see for example two *Create Invoice* nodes, one for *Order* and one for *Invoice*. Two class nodes of the same name are linked by a *cardinality* relationship that indicates how many entities are involved in an event of this class. For example, in every *Create Invoice* events, one *Order* and one *Invoice* is involved, while in every *Receive SO* event one *Supplier Order* and 2-3 *Items* are involved.

[7] https://github.com/multi-dimensional-process-mining/eventgraph_tutorial.

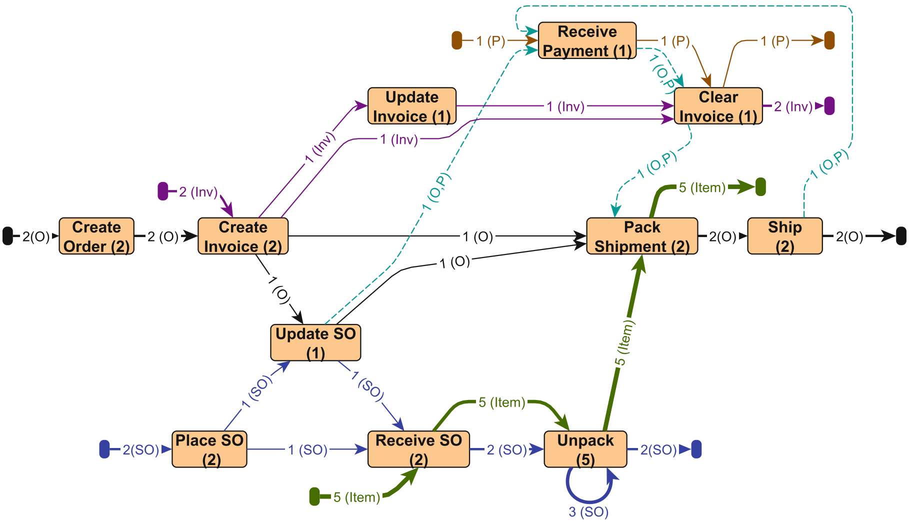


Fig. 12. Multi-Entity Directly-Follows-Graph of the running example obtained by aggregating the graph of Fig. 9

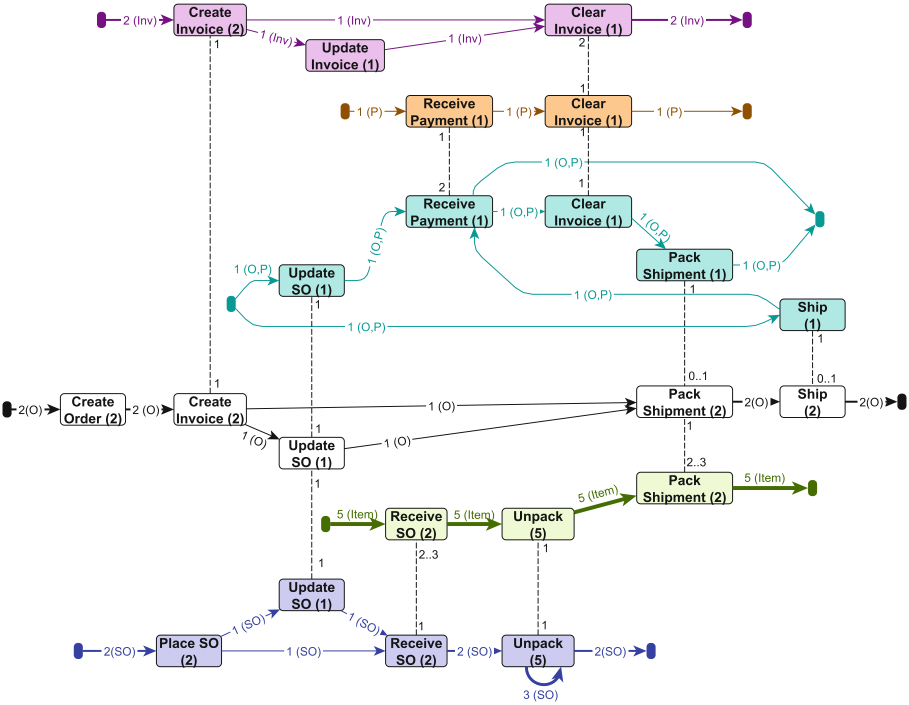


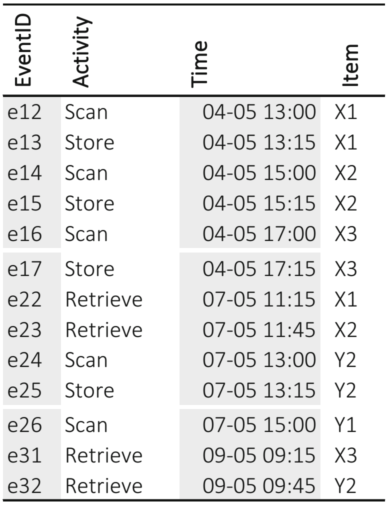
Fig. 13. Synchronous proclet model of the running example obtained by aggregating the graph of Fig. 9

6 Beyond Control-Flow: Multi-dimensional Process Analysis

So far, we analyzed the entities that are created and updated by the process based on the event data in Table 1. We now turn our attention to the *organizational entities* that actually make the process happen: the workers and supporting systems often called *resources*, and the work itself that is being carried out. Along the way, we showcase how flexible event knowledge graphs are. We integrate new events from a different data source in Sect. 6.1. We then enrich event knowledge graphs with df-paths over *activities* Sect. 6.2, which reveals *queues*. Enriching event knowledge graphs with df-paths over *workers* in Sect. 6.3 reveals patterns of how individual workers perform larger scale *tasks*. Finally, we show how to infer new information from (enriched) event knowledge graphs in Sect. 6.4.

6.1 Extending Event Knowledge Graphs with New Events

The process is supported by an automated warehouse (see Fig. 1). Figure 14 shows events of how the *Items* were handled by the warehouse. To analyze how the warehouse influenced the process, we have to combine these events with the events from Table 1. Luckily, we can avoid combining both tables into one joint event table and repeating the entire procedure of Sect. 4.3. We can simply *locally update* an existing graph with new events as follows. We choose to start from the graph of Fig. 6.



EventID	Activity	Time	Item
e12	Scan	04-05 13:00	X1
e13	Store	04-05 13:15	X1
e14	Scan	04-05 15:00	X2
e15	Store	04-05 15:15	X2
e16	Scan	04-05 17:00	X3
e17	Store	04-05 17:15	X3
e22	Retrieve	07-05 11:15	X1
e23	Retrieve	07-05 11:45	X2
e24	Scan	07-05 13:00	Y2
e25	Store	07-05 13:15	Y2
e26	Scan	07-05 15:00	Y1
e31	Retrieve	09-05 09:15	X3
e32	Retrieve	09-05 09:45	Y2

Fig. 14. Warehouse events

1. Import Fig. 14 into new event nodes (Definition 10). This results in new event nodes $e_{12}, \ldots, e_{17}, e_{22}, \ldots, e_{26}, e_{31}, e_{32}$.
2. Infer entities and correlation from the new event nodes (Definition 11). This results in the already existing entity nodes $X1, \ldots, Y2$.
3. For each entity node n inferred in step 2, remove every df-relationship $r \in R^{df}, r.ent = n$, and then infer the df-relationships for n (Definition 12) now including the new imported events.

The resulting graph is shown in Fig. 15. Note that we can obtained the original Fig. 6 again by selection of the original entities and projection onto the original events (see Sect. 5.3).

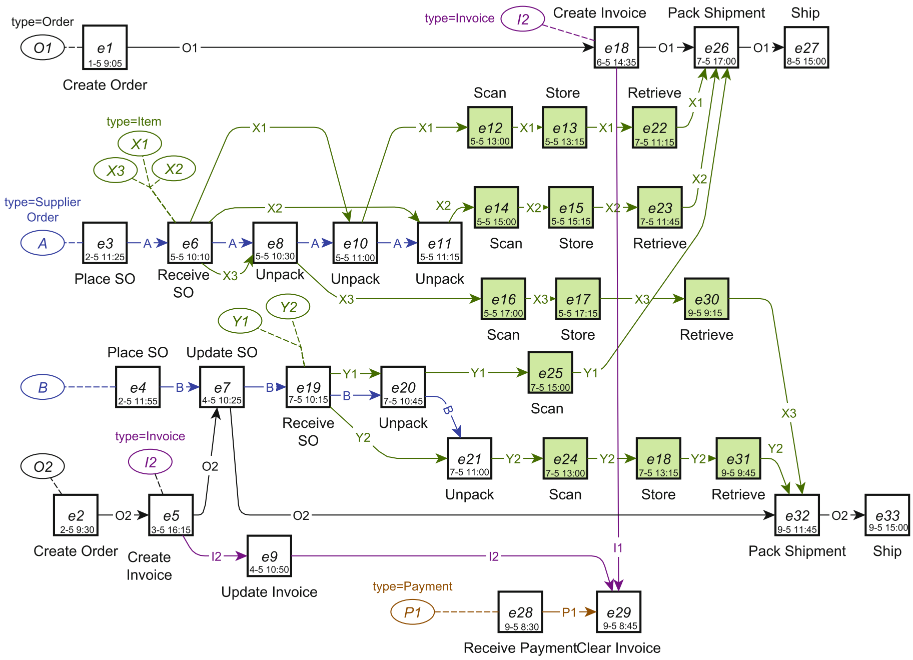


Fig. 15. Event graph after extending Fig. 6 with Fig. 14 (new events highlighted).

6.2 Adding Activities as Entities Reveals Queues

We defined entity inference in Definition 10 for the entity type attributes of the source event table. However, Definition 10 can be applied on *any* property of an event node.

For example, if we pick the *Activity* property as "entity identifier", we infer entities such as *Receive SO*, *Unpack*, *Scan*, *Store*, *Retrieve*, *Pack Shipment*. These are not entities handled by the process. No, these entities are the actual building blocks of the process. For example, each *Item* handled has to "pass through" each of these entities to be completely processed. We can visualize how other entities "pass through" activities by inferring in the graph of Fig. 15 the entity nodes for *Activity* and their df-paths[8]. Figure 16 shows the resulting graph (limited to a subset of events for readability).

We can see that the (red) *Activity* df-paths "go across" all the existing df-paths while the (green) *Item* df-paths traverse the different *Activity* df-paths largely "in parallel". Whenever an *Item* df-path synchronizes with an *Activity*

[8] Note that the *Entity* nodes identified by the activity property are *semantically different* from the *Class* nodes identified by the activity property that we obtained in Sect. 5.4. The *Class* nodes semantically aggregate the existing *df* relationships between events observed for other entities to *df* relationships between *Class* nodes. *Entity* nodes of the entity type *Activity* instead derive new df relationships in addition to existing df relationships for other entities.

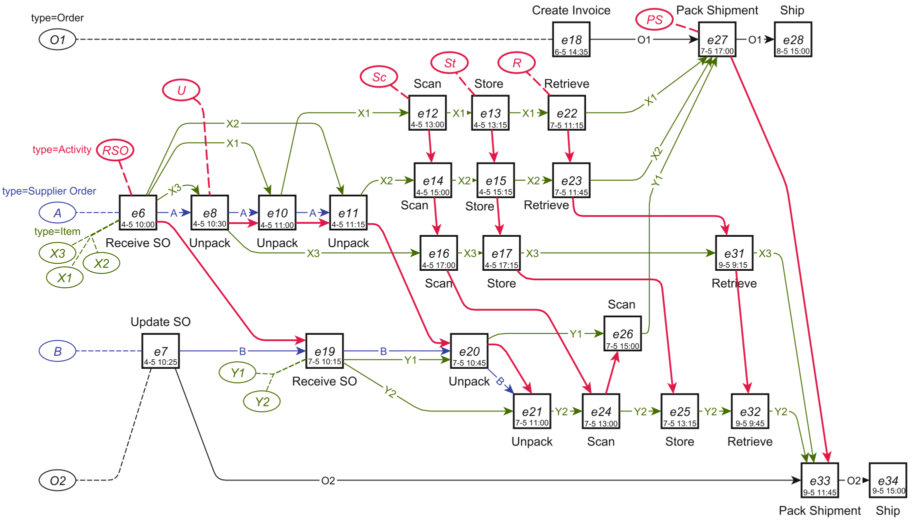


Fig. 16. Inferring *Activity* as entities in the graph of Fig. 15 reveals *Queues*. (Color figure online)

df-path in an event, the item is being worked on. Thus, we can interpret each *Activity* entity A as an abstract "work station" and its events as the work that is being performed there.

The space between two work stations A and B is a *queue* $A : B$, i.e., the space where *Items* after being worked on at A wait until being worked on at B. We can see in the graph in Fig. 16 that the *Items* do not always leave a queue in the same order they entered it: $X1$ entered *Unpack:Scan* after $X3$ (e_{10} follows e_8 in the df-path for *Unpack*) but leaves before $X3$ (e_{12} precedes e_{16} in the df-path for *Scan*).

We can better understand this behavior by changing the layout of the graph in Fig. 16. We select from Fig. 16 only *Item* and *Activity* entities. Setting the x-coordinate of each event by its time property and the y-coordinate by its *Activity* entity results in the graph in Fig. 17, which is called the *Performance Spectrum* [16].

The Performance Spectrum shows us that batching happens at *Receive SO* and *Pack Shipment* (diverging/converging *Item* df-paths), that *Scan:Store* and *Store:Retrieve* are being FIFO queues, that *Unpack:Scan* is *not* a FIFO queue, e.g., $X3$ is overtaken by $X1, X2$ and $Y1$ is overtaken by $Y2$.

We already identified in Sect. 5 reasons why Order $O2$ was not shipped within the 6 days promised by the retailed (see Sect. 1). We now can also clarify the reasons for $O1$. Figure 17 shows that although the second supplier order B with the required item $Y1$ was received on *7-5* (the 6th day of $O1$), order $O1$ was only packed *after* the *15:00* pick-up time. The non-FIFO handling in *Unpack:Scan* seems to be at fault. We observe

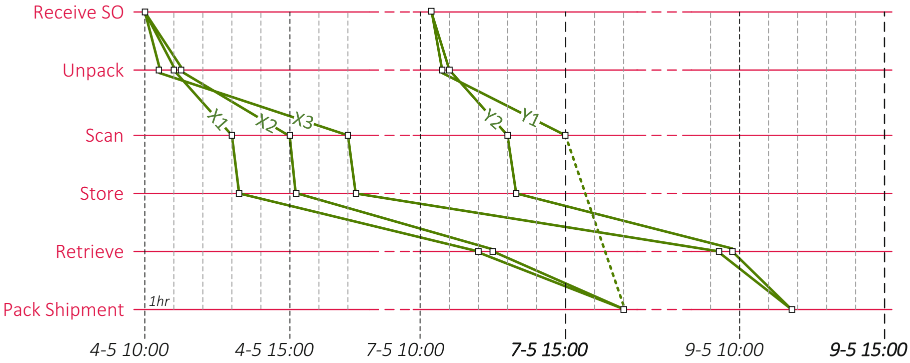


Fig. 17. Sub-graph of Fig. 16 for *Activity* entities and *Item* entities, with event coordinates defined by *Activity* (y-axis) and *time* (x-axis), results in the *Performance Spectrum.*

1. a consistent *2-h minimum waiting time* between two subsequent *Scan* activities (along its *Activity* df-path) causing $Y1$ to finish *Scan* after $Y2$ at *7-5 15:00*, and
2. a consistent *2-h minimum soujourn time* for the last *Item* reaching *Pack Shipment*, i.e., Pack Shipment for $X1, X2, Y1$ completes at *7-5 17:00.*

Thus, if *Unpack:Scan* had followed a strict FIFO policy, $Y1$ could have completed its *Scan* activity at *7-5 12:45*; the subsequent *Pack Shipment* event over $X1, X2, Y1$ could have completed at *7-5 14:45* just before the scheduled pick-up at *7-5 15:00.*

The Performance Spectrum reveals further, far more involved patterns of process performance over time than just batching and FIFO [16]. It is also implemented as a visual analytics tool over event data [15] and in combination with process models [54]. Mining performance patterns from it [36] allows to engineer so called inter-case features for improving the accuracy of remaining time prediction [37].

6.3 Adding Actors as Entities Reveals Complex Tasks

We found in Sect. 6.2 that *Activity* entities describe the abstract “work stations” where other entities are being worked on. Workers are performing this actual work. Often called “resources” in process management literature [18], we prefer the term *Actor* used in organizations research [32], as each actor follows its own behavior. To study actor behavior in the graph of Fig. 6, we only have to (1) infer the *Actor* entities from the event nodes (see Table 1), and (2) infer each actor’s df-path. Figure 18 shows the resulting graph.

We can see actors $R1, R2, R3$ working “intertwined” in the same part of the process. In contrast, $R4$ and $R5$ work more separated from the other actors. Also, the actor df-paths actors show very different characteristics. The df-path $\langle e_1, e_2, e_3, e_7 \rangle$ of $R1$ synchronizes with any other entity only in one event, and

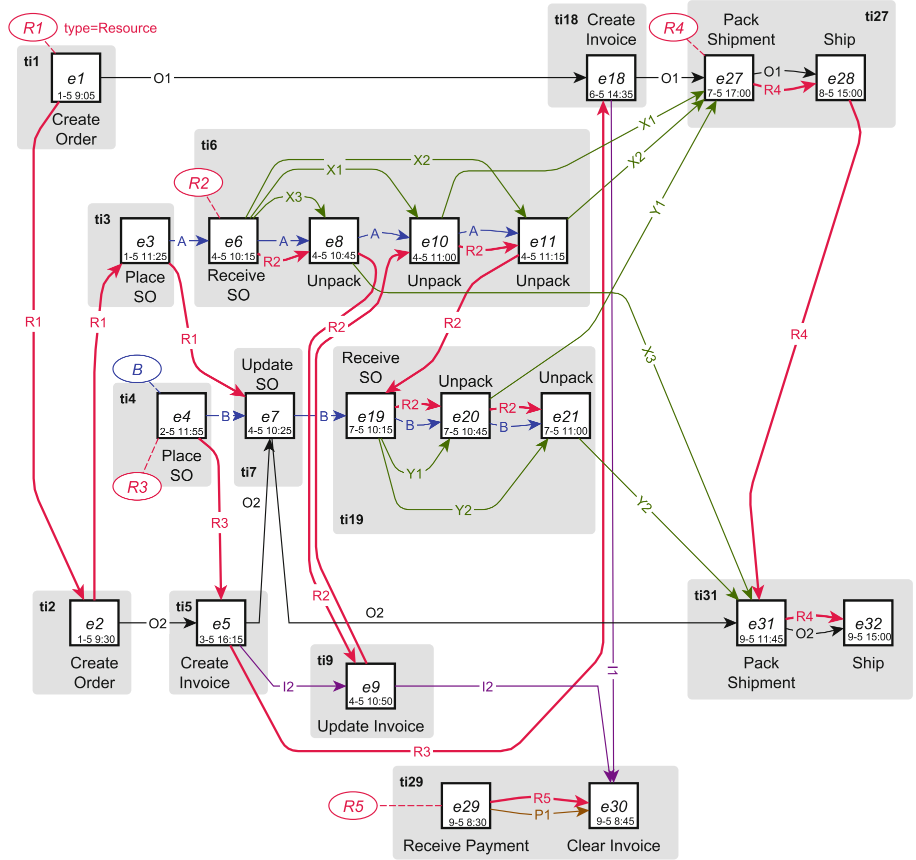


Fig. 18. Adding actors (*Resource*) entities to the event knowledge graph reveals task execution patterns

then moves on to the next entity $O1$, $O2$, A, B, always performing just a single activity on each. In contrast, the df-path of $R4$ synchronizes over multiple subsequent events with the same entity, i.e., e_{27}, e_{28} in $O1$ and e_{31}, e_{32} in $O2$, meaning $R4$ always performs a "unit of work" that consists of two subsequent activities. Such a larger unit of work of multiple related activities is called a *task* [32,38].

A *task instance* of an actor R working on an entity X materializes in an event knowledge graph as a specific subgraph over event nodes $e_1, \ldots, e_k$: (1) the df-paths of R and X both meet in e_1, (2) diverge in e_k, (3) synchronize in each event node $e_1, \ldots, e_k$, and (4) at least one of their df-paths has no other event in between $e_1, \ldots, e_k$ [38]. The grey rectangles highlighted in Fig. 18 shows several task instance. The task instances themselves and the way they are ordered in the graph reveal unique characteristics of performing work.

- Actor $R1$ only performs a series of singleton tasks ti_1, ti_2, ti_3, ti_4. The df-path of $R1$ describes that *Supplier Order A* has been placed only after both *Orders* $O1$ and $O2$ were created.
- Actor $R4$ performs two instances ti_{27} and ti_{31} of the same task (first *Pack Shipment* then *Ship*) directly after each other on two different *Orders*.
- Actor $R2$ also performs two instances ti_6 and ti_{19} of the same task (first *Receive* then repeatedly *Unpack*) directly after each other; however $R2$ interrupts ti_6 on A to perform ti_9 (*Update Invoice*) on $I2$.

Further, more complex types of task instances can be identified in event knowledge graphs [38]. The df-relationships between task instances also reveal patterns of how work is handed over between actors. For example $R1$ hands work over to $R2$ in all *Supplier Orders*, to $R3$ in all *Orders*, and to $R4$ in $O2$; $R2$ hands work over to $R4$ in all *Items* and to $R5$ in $I2$. Such patterns are studied in the area of routines research [32].

We clearly can see some undesirable behavior in how actors collaborate over the different entities.

- $R1$ created both *Orders* $O1$ and $O2$ but only placed *Supplier Order A*. Instead, $R3$ placed B and we cannot observe a handover from $R1$ to $R3$; this lack of collaboration may have led to $R3$ placing a wrongly *Supplier Order B* (with only one item Y). The *Update SO* by $R1$ remedies the problem but caused to the delay in delivering $Y1$ we identified in Sect. 5. The problem may have been avoided by $R1$ completing the "larger task" $\langle e_1, \ldots, e_4 \rangle$ alone.
- $R2$ is *interrupting* their work on unpacking *Supplier Order A* after $X3$ (e_8) to *Update Invoice* $I2$ before continuing on unpacking $X1$ (e_{10}). This "context switch" between handling *Items* and *Invoice* results in a longer delay between two subsequent *Unpack* events (30mins) than usual (15mins), which we can directly see in the Performance Spectrum in Fig. 17. The longer delay is a risk to packing shipments on time as we analyzed in Sect. 6.2. The risk could be reduced by ensuring that $R2$ is not interrupting their task; $R3$ could have updated the invoice instead.

The process model shown in Fig. 21, and further explained in Sect. 7, describes for each actor behavioral routines that could avoid undesirable behavior.

6.4 Inference in Event Knowledge Graphs with Multiple Layers

Our discussions so far focused on constructing, understanding, and finding patterns in graphs over *Entity* and *Event* nodes and the *df* and *corr* relationships. As the model of event knowledge graphs (Definition 8) is based on labeled property graphs (Definition 7), we can extend an event knowledge graph with further node and relationship types, to describe more knowledge about the process. We already did that in Sect. 5.4 when aggregating multiple *Event* nodes of the same activity to a new node with label *Class*. In the following we expand on this idea by an example. We do so in the style of a process mining analyst applying all the concepts of the previous sections as data processing operations. In fact all steps shown here can be realized through Cypher queries over a graph database.

Suppose we want to create a concise summary of how actors organize the work of handling *Supplier Orders*, based on the graph with actor df-paths shown in Fig. 18. The actors correlated to *Supplier Order* events are $R1, R2, R3$. We select entities A and B and $R1, R2, R3$ and then project onto events of a *Supplier Order* or between two *Supplier Order* events (to keep e_9). The resulting graph is shown in Fig. 19 as "Event Entity Layer".

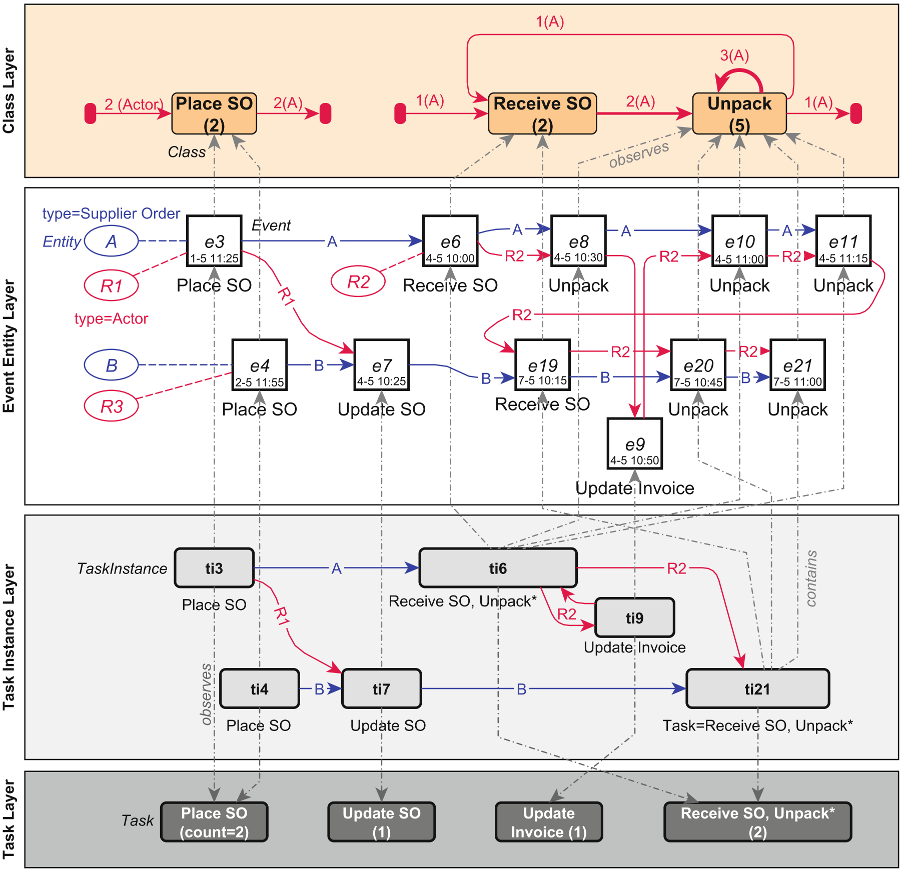


Fig. 19. An event knowledge graph extended with additional layers into a "process knowledge graph".

Next, we aggregate the event layer into a new "Task Instance Layer".

1. For each task instances, i.e., each subgraph of an *Actor* df-path and an *Supplier Order* df-path synchronizing on consecutive events as defined in Sect. 6.3,

we extend the graph with a new node with label *TaskInstance*, resulting in the nodes $ti_3, ti_4, ti_6, ti_7, ti_9, ti_{21}$ shown in the "Task Instance Layer" of Fig. 19.

2. We add a new *contains* relationship $(ti, e) \in R^{contains}$ from each *TaskInstance* node ti to each *Event* node e that is part of the task instance, e.g., $(ti_9, e_{19}), (ti_9, e_{20}), (ti_9, e_{21}) \in R^{contains}$ in Fig. 19. This connects the nodes in both layers.
3. Each $ti \in N^{TaskInstance}$ gets the property $ti.Task$ by concatenating the *Activity* values of the event nodes it contains along their df-path (abstracting repetitions with a Kleene star), e.g., $ti_6.Task = \langle Receive\ SO, Unpack^* \rangle$.
4. We then lift the df-relationships from *Event* nodes to *TaskInstance* nodes. For each df-relationship $(e, e') \in R^{df}$ between events e, e' contained in different task instances $ti \neq ti', (ti, e), (ti', e') \in R^{contains}$, we create a new df-relationship $(ti, ti') \in R^{df}$ between task instance nodes ti and ti' (and copy the properties of the df relationship).

The resulting "Task Instance Layer" in Fig. 19 represents the "Event Entity Layer" at the aggregation level of task executions instead of activity executions.

5. To understand which tasks are performed and how often, we aggregate *TaskInstance* nodes into *Task* nodes by their *Task* property (see Sect. 5.4).

The resulting "Task Layer" in Fig. 19 shows four tasks *Place SO* (performed twice in ti_3, ti_4), *Update SO* (performed once in ti_7), *Update Invoice* (performed once in ti_9), and $\langle Receive\ SO, Unpack^* \rangle$ (performed twice in ti_6, ti_{21}).

We now want to visualize the behavior all actors regarding the *frequent tasks* in handling *Supplier Orders*, e.g., tasks performed at least twice. The visualization shall be on the abstraction level of the activities performed by actors, i.e., a multi-entity DFG. To achieve this, we aggregate the "Event Entity Layer" into a "Class Layer" using the "Task Layer" as context.

1. Select from the "Event Entity Layer" only the *Actor* entities; this removes all df-relationships for A and B.
2. Project onto all event nodes $e \in N^{Event}$ having a path $\langle e, ti, t \rangle$ to a task node $t \in N^{Task}$ with $t.count \geq 2$, i.e., only events that are contained in a task instance ti of a frequently occurring task. This removes e_7 and e_9 from the graph in Fig. 19 and introduces $(e_8, e_{10}) \in R^{df}_{R2}$.
3. Aggregate the *Event* nodes to *Class* nodes by their *Activity* property and lift df-relationships from *Event* nodes to *Class* nodes (see Sect. 5.4).

The resulting multi-entity DFG forms a new "Class Layer" in the graph, that is connected to the "Event Entity Layer" by *observes* relationships, as shown in Fig. 19. The multi-entity DFG shows that $R1$ and $R2$ work on disjoint sets of activities, and that $R2$ indeed follows a cyclic, structured behavior. The paths from *Class* nodes *Receive SO* and *Unpack* to the *Task* nodes show that all activities belong to the same task, i.e., one cycle is one "unit of work".

The multi-entity DFG is a filtered DFG: it lacks df-relationships for *Supplier Orders* and it omits *Update SO*. Thus, the multi-entity DFG *does not fit* or *deviates* from the "Event Entity Layer". We can identify the deviations in multi-layered process knowledge graph in Fig. 19 similar to alignments [9]; see [8]. For instance, for df-relationship $(\mathit{Unpack}, \mathit{Unpack}) \in R^{df}$ in the "Class Layer", we see

- two corresponding "synchronous" df-relationship $(e_{10}, e_{11}), (e_{20}, e_{21}) \in R^{df}_{R2}$ with $(e_i, \mathit{Unpack}) \in R^{observes}$; and
- one corresponding "log-move" df-path $\langle e_8, e_9, e_{10} \rangle \in (R^{df}_{R2})^*$ with $(e_8, \mathit{Unpackt}), (e_9, \mathit{Update\ Invoice}), (e_{10}, \mathit{Unpack}) \in R^{observes}$, i.e., e_9 occurs in between *Unpack* and *Unpack*.

7 Conclusion and Outlook

The preceding sections studied different forms of process mining over multiple behavioral dimensions that are summarized in Fig. 20. We showed in Sect. 3 how classical process mining techniques fail when the assumption of a single entity handled by a single execution (bottom left quadrant in Fig. 20) is violated.

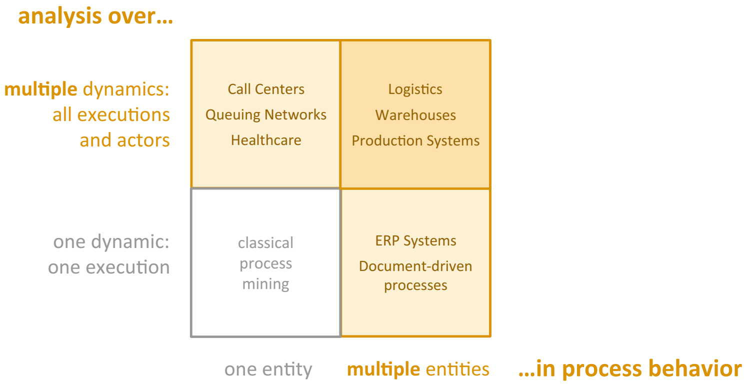


Fig. 20. Quadrants of process analysis over multiple behavioral dimensions

To overcome these assumptions, we introduced process mining with event knowledge graphs, that rests on three simple, but fundamental principles:

1. Explicitly represent every entity that an event is correlated to as a node. An entity thereby can be anything: a specific object, a person or actor, or even an abstract concept such as an activity.
2. Infer directly-follows relations over events per entity. This results in directly-follows paths forming complex, but meaningful structures that can be filtered for.
3. Aggregate any structure of interest formed by directly-follows paths into new nodes describing process-related concepts, explicitly linked to the structures that generate them. This allows to infer interactions between related entities (see Sect. 4.4), multi-entity process models (see Sect. 5.5), and task instances (see Sect. 6.3).

Applying these principles, we constructed event knowledge graphs from standard event data through simple concepts in Sect. 4.3. We showed in Sect. 5 how to analyze processes where each execution involves *multiple related entities*, such as ERP systems and document-driven processes (bottom right quadrant in Fig. 20). We showed in Sect. 6 how event knowledge graphs also allow to analyze *multiple dynamics* together. We added actor and queue behavior to study how entities pass through queues or actors perform tasks across multiple entities, which are dynamics studied in call centers or in healthcare (top left quadrant in Fig. 20). Note that, in Sect. 6 we always focused on a single entity processed in a queue or in a task. How to analyze the combination of *multiple dynamics* over *multiple entities* (top right quadrant in Fig. 20) is an open question.

Event knowledge graphs give rise to a number of novel research questions.

We have shown how to construct event knowledge graphs from event tables, even automatically [24,25]. We also need techniques to construct event knowledge graphs from relational database while preserving the existing entities and relations. Existing automated conversion techniques from relational to graph databases [50] only convert records into entity nodes, while event knowledge graphs require to construct event nodes.

The quality of a process mining analysis on event knowledge graphs relies on having identified the relevant structural relations (between entities) and behavioral or cause-effect relations (between events) (see Sect. 4.4). We need automated techniques to infer relevant relations that take the temporal semantics of the df-relationship into account. Promising first steps are techniques that explicitly allow to incorporate domain knowledge when inferring causal relationships from relational data [56], or use ontologies [6,7] for extraction. Specifically, dynamically changing relationships and changes of object properties [39,40] still need to be considered.

We have sketched the possibility of structuring a complex process mining analysis by adding analysis layers to the graph, but limited ourselves to simple selection, projection, and aggregation queries. Adequate query languages that also can handle process-relevant phenomena such as frequency, noise, performance in relation to multiple entities need to be considered. Also, more complex behavioral dynamics can be discovered. For example, enriching the event knowledge graph with the activity dimension to derive the performance spectrum (see Sect. 6.2) allows detecting subgraphs that indicate high workload (many events in a short interval) or a dynamic bottleneck (a short-term increase in waiting time) [51]. Aggregating these to “high-level events” and mining for cause-effect relations among them reveals how performance anomalies cascade through a process [51].

Finally, while we did discuss how to discover multi-entity directly-follows graphs through aggregation, true process discovery of models with precise semantics from event knowledge graphs still has to be addressed. In principle, such models can be discovered through principles of artifact-centric process mining [41,46]: First obtain a classical event log per entity type, e.g., by extracting the df-paths per entity type from the graph, and discover a classical process

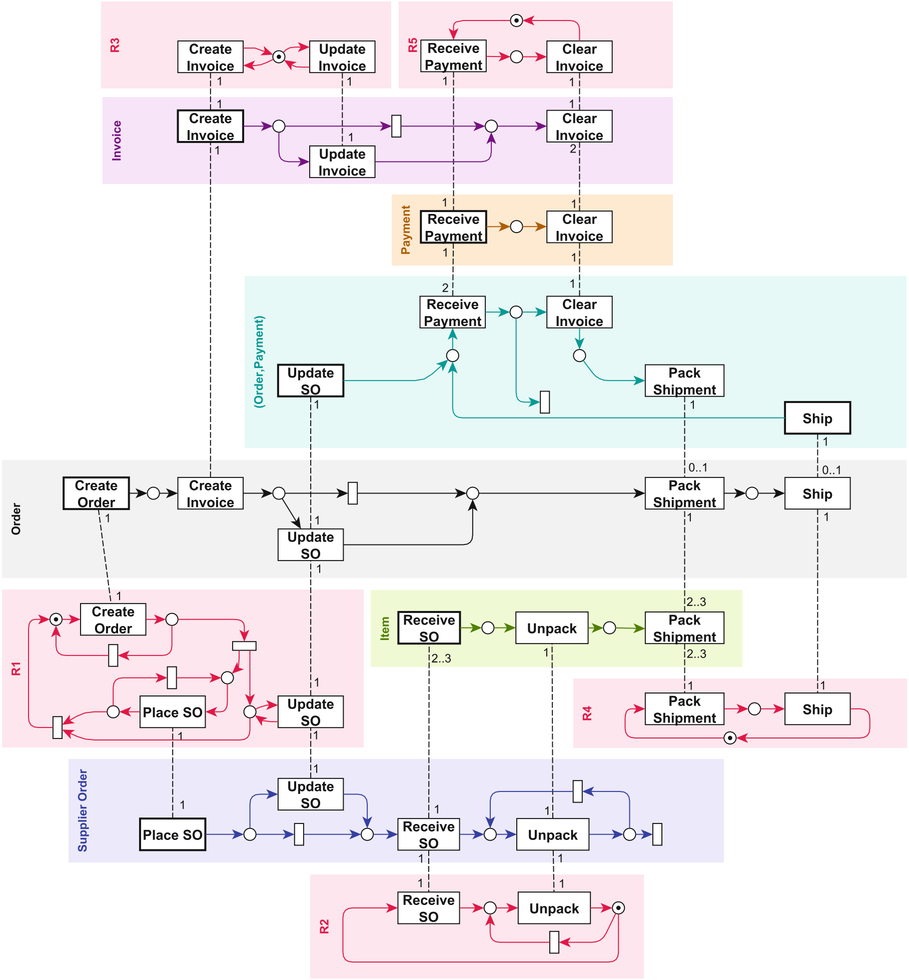


Fig. 21. Synchronous proclet model for the graph of Fig. 9 extended with proclets describing the *intended* (not the observed) behavior for all actors.

model per entity type. Then compose the models of the different entity types to express their synchronization.

Figure 21 shows a possible process model that could be obtained in this way for our example, using a multi-entity extension for Petri nets, called *synchronous proclets* [26]. Each proclet is a Petri net that describes the behavior of one entity type; bold-bordered initial transitions describe the creation of a new entity. The dashed *synchronization edges* describe which transitions occur together; the multiplicity annotations indicate how many entities of each type have to be involved. Note that the proclet model in Fig. 21 is a hybrid between discovered and manually created model. The proclets for *Order*, *Supplier Order*, *Invoice*, *Item*, *Payment*, and *(Order,Payment)* are each discovered from the entity type's df-paths of the graph in Fig. 9. The proclets for the *Actors* however are created manually[9], describing the intended routine for each actor based on the insights in Sect. 6.3. Bold-bordered initial transitions describe the creation of a new entity; note that the proclets for actors do not have an initial transition but an initial marking as actors are not created in the process. Dashed *synchronization edges* between transitions describe that the transitions have to occur together; the multiplicity annotations indicate how many entities of each type have to be involved. For instance, $R1$ creates 1 new *Order* in each occurrence of *Create Order*, but $R4$ always packs 2–3 *Items* into 1 *Shipment* in each occurrence of *Pack Shipment*.

An alternative formalization of this concept are *object-centric Petri nets* [53]. Object-centric Petri nets also first discover one Petri net per entity type, then annotate the places and arcs with entity identifiers, and then compose all entity nets along transitions for the same activity, resulting in a coloured Petri net model that is accessible for analysis [53] and measuring model quality [3]. However, synchronization by composition prevents explicitly modeling (and thus discovering) interactions between entities such as the relation from *Order* to *Payment* described by proclet *(Order,Payment)* in Fig. 21.

Though, while proclets can describe entity interactions, the behavior of entity interactions tends to be rather unstructured resulting in overly complex models [41]. Extensions of declarative models (see [10]) such as modular DCR graphs [14], that apply similar principles as synchronous proclets, could be more suitable. Alternatively, scenario-based models [31] that specify conditional partial orders of events over multiple entities could be applied. For instance, the conditional scenario in Fig. 22 specifies the interaction between *Orders* and *Payments* observed in the graph of Fig. 9.

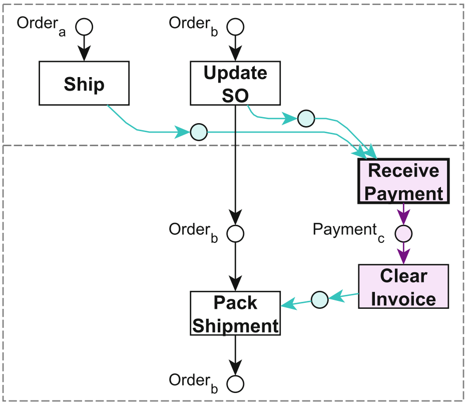


Fig. 22. Conditional scenario describing an interaction of 2 *Orders* and 1 *Payment*.

Altogether, event knowledge graphs give rise to entirely novel forms of process mining that support novel forms of process management [17].

[9] We created one proclet per actor as introducing a proclet for all actors would result in a very complex proclet as different actors follow very different behavior. Further, the manually created model conveniently avoids the issue of having to layout how $R2$ synchronizes both with *Supplier Order* and with *Invoice*.

References

1. van der Aalst, W.M.P.: Process mining: a 360 degrees overview. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 3–34. Springer, Cham (2022)
2. Accorsi, R., Lebherz, J.: A practitioner's view on process mining adoption, event log engineering and data challenges. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 212–240. Springer, Cham (2022)
3. Adams, J.N., van der Aalst, W.M.P.: Precision and fitness in object-centric process mining. In: ICPM 2021, pp. 128–135. IEEE (2021)
4. Berti, A., van der Aalst, W.: Extracting multiple viewpoint models from relational databases. In: Ceravolo, P., van Keulen, M., Gómez-López, M.T. (eds.) SIMPDA 2018-2019. LNBIP, vol. 379, pp. 24–51. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-46633-6_2
5. Bonifati, A., Fletcher, G.H.L., Voigt, H., Yakovets, N.: Querying Graphs. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, San Rafael (2018)
6. Calvanese, D., Kalayci, T.E., Montali, M., Santoso, A.: OBDA for log extraction in process mining. In: Ianni, G., et al. (eds.) Reasoning Web 2017. LNCS, vol. 10370, pp. 292–345. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61033-7_9
7. Calvanese, D., Kalayci, T.E., Montali, M., Tinella, S.: Ontology-based data access for extracting event logs from legacy data: the onprom tool and methodology. In: Abramowicz, W. (ed.) BIS 2017. LNBIP, vol. 288, pp. 220–236. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59336-4_16
8. Carmona, J., van Dongen, B., Weidlich, M.: Conformance checking: foundations, milestones and challenges. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 155–190. Springer, Cham (2022)
9. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99414-7
10. Di Ciccio, C., Montali, M.: Declarative process specifications: reasoning, discovery, monitoring. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 108–152. Springer, Cham (2022)
11. Cyganiak, R., Hyland-Wood, D., Lanthaler, M.: RDF 1.1 concepts and abstract syntax. W3C Proposed Recommendation (2014)
12. de Murillas, E.G.L., Reijers, H.A., van der Aalst, W.M.P.: Case notion discovery and recommendation: automated event log building on databases. Knowl. Inf. Syst. **62**(7), 2539–2575 (2019). https://doi.org/10.1007/s10115-019-01430-6
13. De Weerdt, J., Wynn, M.T.: Foundations of process event data. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 193–211. Springer, Cham (2022)
14. Debois, S., López, H.A., Slaats, T., Andaloussi, A.A., Hildebrandt, T.T.: Chain of events: modular process models for the law. In: Dongol, B., Troubitsyna, E. (eds.) IFM 2020. LNCS, vol. 12546, pp. 368–386. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63461-2_20
15. Denisov, V., Belkina, E., Fahland, D., van der Aalst, W.M.P.: The performance spectrum miner: visual analytics for fine-grained performance analysis of processes. In: BPM 2018 Demos. CEUR Workshop Proceedings, vol. 2196, pp. 96–100. CEUR-WS.org (2018)

16. Denisov, V., Fahland, D., van der Aalst, W.M.P.: Unbiased, fine-grained description of processes performance from event data. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 139–157. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_9
17. Dumas, M., et al.: Augmented business process management systems: a research manifesto. CoRR, abs/2201.12855 (2022)
18. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, 2nd edn. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-662-56509-4
19. Esser, S., Fahland, D.: Event Graph of BPI Challenge 2014. Dataset. https://doi.org/10.4121/14169494
20. Esser, S., Fahland, D.: Event Graph of BPI Challenge 2015. Dataset. https://doi.org/10.4121/14169569
21. Esser, S., Fahland, D.: Event Graph of BPI Challenge 2016. Dataset. https://doi.org/10.4121/14164220
22. Esser, S., Fahland, D.: Event Graph of BPI Challenge 2017. Dataset. https://doi.org/10.4121/14169584
23. Esser, S., Fahland, D.: Event Graph of BPI Challenge 2019. Dataset. https://doi.org/10.4121/14169614
24. Esser, S., Fahland, D.: Event Data and Queries for Multi-Dimensional Event Data in the Neo4j Graph Database, April 2021. https://doi.org/10.5281/zenodo.4708117
25. Esser, S., Fahland, D.: Multi-dimensional event data in graph databases. J. Data Semant. **10**(1–2), 109–141 (2021). https://doi.org/10.1007/s13740-021-0122-1
26. Fahland, D.: Describing behavior of processes with many-to-many interactions. In: Donatelli, S., Haar, S. (eds.) PETRI NETS 2019. LNCS, vol. 11522, pp. 3–24. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21571-2_1
27. Fahland, D.: Petri's understanding of nets. In: Reisig, W., Rozenberg, G. (eds.) Carl Adam Petri: Ideas, Personality, Impact, pp. 31–36. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-96154-5_5
28. Fahland, D.: multi-dimensional-process-mining/eventgraph_tutorial, April 2022
29. Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.M.P.: Behavioral conformance of artifact-centric process models. In: Abramowicz, W. (ed.) BIS 2011. LNBIP, vol. 87, pp. 37–49. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21863-7_4
30. Fahland, D., Denisov, V., van der Aalst, W.M.P.: Inferring unobserved events in systems with shared resources and queues. Fundam. Informaticae **183**(3–4), 203–242 (2021). https://doi.org/10.3233/FI-2021-2087
31. Fahland, D., Prüfer, R.: Data and abstraction for scenario-based modeling with petri nets. In: Haddad, S., Pomello, L. (eds.) PETRI NETS 2012. LNCS, vol. 7347, pp. 168–187. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31131-4_10
32. Goh, K., Pentland, B.: From actions to paths to patterning: toward a dynamic theory of patterning in routines. Acad. Manag. Ann. **62**, 1901–1929 (2019)
33. Hogan, A., et al.: Knowledge graphs. ACM Comput. Surv. **54**(4) (2021). https://doi.org/10.1145/3447772
34. Jalali, A.: Graph-based process mining. In: Leemans, S., Leopold, H. (eds.) ICPM 2020. LNBIP, vol. 406, pp. 273–285. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72693-5_21
35. Jans, M., Soffer, P.: From relational database to event log: decisions with quality impact. In: Teniente, E., Weidlich, M. (eds.) BPM 2017. LNBIP, vol. 308, pp. 588–599. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74030-0_46

36. Klijn, E.L., Fahland, D.: Performance mining for batch processing using the performance spectrum. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) BPM 2019. LNBIP, vol. 362, pp. 172–185. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37453-2_15
37. Klijn, E.L., Fahland, D.: Identifying and reducing errors in remaining time prediction due to inter-case dynamics. In: ICPM 2020, pp. 25–32. IEEE (2020). https://doi.org/10.1109/ICPM49681.2020.00015
38. Klijn, E.L., Mannhardt, F., Fahland, D.: Classifying and detecting task executions and routines in processes using event graphs. In: Polyvyanyy, A., Wynn, M.T., Van Looy, A., Reichert, M. (eds.) BPM 2021. LNBIP, vol. 427, pp. 212–229. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85440-9_13
39. Li, G., de Carvalho, R.M., van der Aalst, W.M.P.: Automatic discovery of object-centric behavioral constraint models. In: Abramowicz, W. (ed.) BIS 2017. LNBIP, vol. 288, pp. 43–58. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59336-4_4
40. Li, G., de Murillas, E.G.L., de Carvalho, R.M., van der Aalst, W.M.P.: Extracting object-centric event logs to support process mining on databases. In: Mendling, J., Mouratidis, H. (eds.) CAiSE 2018. LNBIP, vol. 317, pp. 182–199. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92901-9_16
41. Xixi, L., Nagelkerke, M., van de Wiel, D., Fahland, D.: Discovering interacting artifacts from ERP systems. IEEE Trans. Serv. Comput. **8**(6), 861–873 (2015)
42. Martin, N., Pufahl, L., Mannhardt, F.: Detection of batch activities from event logs. Inf. Syst. **95**, 101642 (2021)
43. Nooijen, E.H.J., van Dongen, B.F., Fahland, D.: Automatic discovery of data-centric and artifact-centric processes. In: La Rosa, M., Soffer, P. (eds.) BPM 2012. LNBIP, vol. 132, pp. 316–327. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36285-9_36
44. Pegoraro, M., Bakullari, B., Uysal, M.S., van der Aalst, W.M.P.: Probability estimation of uncertain process trace realizations. In: Munoz-Gama, J., Lu, X. (eds.) ICPM 2021. LNBIP, vol. 433, pp. 21–33. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-98581-3_2
45. Piessens, D.A.M.: Event log extraction from SAP ECC 6.0. Master's thesis, Eindhoven University of Technology (2011)
46. Popova, V., Fahland, D., Dumas, M.: Artifact lifecycle discovery. Int. J. Cooperative Inf. Syst. **24**(1), 1550001:1–1550001:44 (2015). https://doi.org/10.1142/S021884301550001X
47. Pourmirza, S., Dijkman, R.M., Grefen, P.: Correlation miner: mining business process models and event correlations without case identifiers. Int. J. Cooperative Inf. Syst. **26**(2):1742002:1–1742002:32 (2017)
48. Robinson, I., Webber, J., Eifrem, E.: Graph Databases. O'Reilly Media, Sebastopol (2013)
49. Schruben, L.: Simulation modeling with event graphs. Commun. ACM **26**(11), 957–963 (1983)
50. Stoica, R., Fletcher, G.H.L., Sequeda, J.F.: On directly mapping relational databases to property graphs. In: 13th Alberto Mendelzon International Workshop on Foundations of Data Management. CEUR Workshop Proceedings, vol. 2369. CEUR-WS.org (2019)
51. Toosinezhad, Z., Fahland, D., Köroglu, Ö., van der Aalst, W.M.P.: Detecting system-level behavior leading to dynamic bottlenecks. In: ICPM 2020, pp. 17–24. IEEE (2020). https://doi.org/10.1109/ICPM49681.2020.00014

52. Aalst, W.M.P.: Object-centric process mining: dealing with divergence and convergence in event data. In: Ölveczky, P.C., Salaün, G. (eds.) SEFM 2019. LNCS, vol. 11724, pp. 3–25. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30446-1_1
53. van der Wil, M.P.: Aalst and Alessandro Berti. Discovering object-centric petri nets. Fundam. Informaticae **175**(1–4), 1–40 (2020)
54. van der Aalst, W.M.P., Tacke Genannt Unterberg, D., Denisov, V., Fahland, D.: Visualizing token flows using interactive performance spectra. In: Janicki, R., Sidorova, N., Chatain, T. (eds.) PETRI NETS 2020. LNCS, vol. 12152, pp. 369–380. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51831-8_18
55. Waibel, P., Novak, C., Bala, S., Revoredo, K., Mendling, J.: Analysis of business process batching using causal event models. In: Leemans, S., Leopold, H. (eds.) ICPM 2020. LNBIP, vol. 406, pp. 17–29. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72693-5_2
56. Waibel, P., Pfahlsberger, L., Revoredo, K., Mendling, J.: Causal process mining from relational databases with domain knowledge (2022). https://doi.org/10.48550/ARXIV.2202.08314

Predictive Process Monitoring

Chiara Di Francescomarino and Chiara Ghidini(✉)

Fondazione Bruno Kessler, Trento, Italy
{dfmchiara,ghidini}@fbk.eu

1 Introduction

Predictive Process Monitoring [29] is a branch of process mining that aims at predicting the future of an ongoing (uncompleted) process execution. Typical examples of predictions of the future of an execution trace relate to the outcome of a process execution, to its completion time, or to the sequence of its future activities.

Being able to predict in advance the outcome of a process execution, the time that a process instance will require to complete, or the activities that will be executed next can be extremely valuable in several domains and scenarios, e.g., for production processes, allowing organizations to prevent undesired outcomes, issues and delays. Indeed, differently from the problem of monitoring business processes in a *reactive* way [28], i.e., so that the violation or the delay is identified only after its occurrence, predicting the violation or the issue before it occurs, would allow for supporting users and organizations in *preventing* it by taking the appropriate preventive countermeasures. Fueled also by the wave of technical developments in Data Science, Predictive Analytics, and data driven Artificial Intelligence, the development of predictive techniques tailored to the field of Process Mining has rapidly established itself both as a vibrant research topic and as an impactful functionality with a direct application in innovative organizational contexts and process mining tools, which often go hand in hand. Examples are the development of new Predictive Process Monitoring pipelines for specific organizations (such as hospitals) [3] and the investigation of explainable Predictive Process Monitoring techniques performed together with leading Process Mining companies such as myInvenio[1] [18] with the aim of incorporating the features within their Process Mining tools (see also [36]).

Predictive Process Monitoring approaches usually leverage past historical complete executions in order to provide predictions about the future of an ongoing (incomplete) case. They usually have two phases: a *training or learning phase*, in which a predictive model is learned from historical (complete) execution traces and a *runtime or prediction phase*, in which the predictive model is queried for predicting the future of an ongoing case.

[1] Recently acquired by IBM as part of the IBM Process Mining suite. See www.ibm.com/cloud/cloud-pak-for-business-automation/process-mining/.

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 320–346, 2022.
https://doi.org/10.1007/978-3-031-08848-3_10

The chapter is structured as follows:[2] after an introduction of a simple explanatory example (Sect. 2) and of the main dimensions characterizing the family of the Predictive Process Monitoring approaches for business processes (Sect. 3), the typical encodings and approaches used for the prediction of outcomes (Sect. 4), numeric values (Sect. 5), and sequences of activities - and related payloads - (Sect. 6), are described in the next three sections, respectively. Finally, Sect. 7 presents new relevant trends in the context of operational support techniques based on Machine Learning and Sect. 8 introduces the main available open source tools supporting Predictive Process Monitoring tasks. We assume as prerequisite for the next sections that the reader has some machine/deep learning knowledge, especially on classification and regression algorithms, as well as on recurrent neural networks. The interested reader can refer to [5,19,20].

2 Running Example

During the execution of a business process, process participants cooperate to satisfy certain business constraints. At any stage of the process enactment, decisions are taken aimed at achieving the satisfaction of these constraints. Being able to predict in advance certain aspects of a process execution allows organizations to take advantage or adapt to desirable future enfolding or to react and be able to prevent an undesirable scenario by taking the appropriate preventive countermeasures.

In this chapter we will illustrate the potential and characteristics of Predictive Process Monitoring by means of a running example in a healthcare scenario.[3] The example describes the process of a patient going to a hospital to perform a radiology exam and related medical checks. The process covers both the clinical aspects, such as the visit(s) and the radiology exam(s) and administrative issues, such as the admission to the radiology department, the computation of the medical bill and its payment. During the process execution, the doctor has to make decisions on whether further exams are required, and - if possible - issued. Depending on the examinations visits can precede and/or follow the radiology exam, which vary in range from Ultrasound, to X-ray, to Pet, MRI, Breast Imaging, and so on. The process typically starts with the admission, the execution of the medical activities (exams and visits) and the computation and paying of the bills. Different executions are nonetheless possible such as a payment in advance, before the visit.

In this scenario, historical information about past executions of the process, and in particular data related to the clinical history of other patients with similar characteristics, could be used to support the hospital predicting the unfolding of a certain execution. As an example, at a certain time during the process execution, one could predict whether a certain patient will require ultrasounds

[2] In this chapter we mostly focus on the main pipeline, omitting aspects mainly related to the preprocessing and evaluation phase.

[3] This example and its instantiations in the following sections are taken and inspired from the running example used in [25].

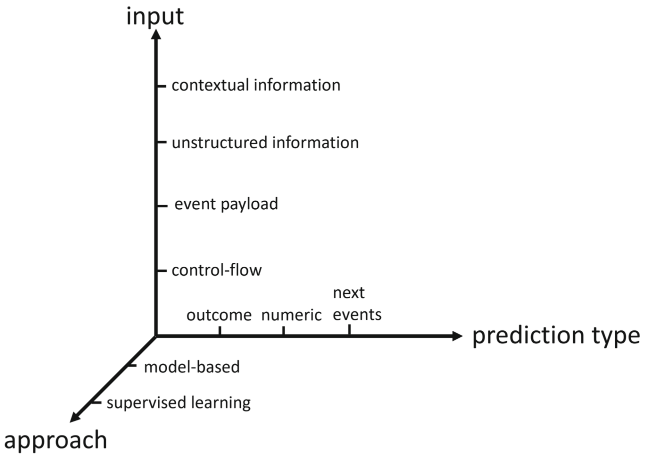


Fig. 1. Predictive Process Monitoring along three dimensions.

and/or at what time. This may be used by the hospital staff to improve or adapt the scheduling of their facilities.

3 The Family of Predictive Process Monitoring Approaches

Although Predictive business Process Monitoring is a relatively young field, it has been growing fast in the latest years, as it is also witnessed by recent surveys on the topic [13,31]. As depicted in Fig. 1, the literature on predictive business process monitoring can be roughly classified along three main dimensions:

- type of prediction (i.e., the type of predictions provided as output);
- type of adopted approach and technique;
- type of information exploited in order to get predictions (i.e., the type of information taken as input).

Concerning the type of prediction, according to the literature [13,46], we can classify the existing prediction types into three main big categories:

- predictions related to predefined categorical or boolean outcome values (*outcome-based predictions*);
- predictions related to measures of interest taking numeric or continuous values (*numeric value predictions*);
- predictions related to sequences of future activities and related data payloads (*next event predictions*).

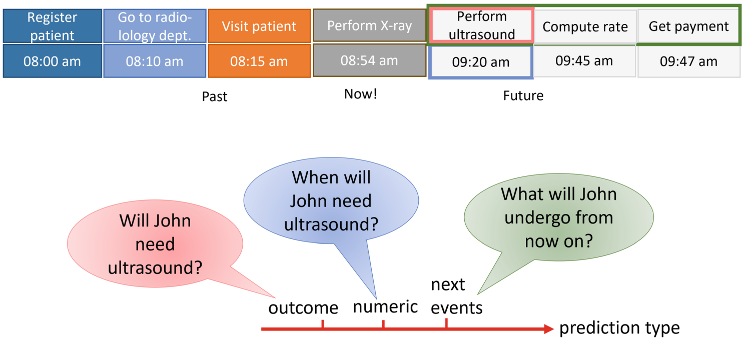


Fig. 2. Types of predictions.

Figure 2 shows an example of an execution trace describing the activities carried out by John. Let us assume that it is 8:54 a.m now. At 8:00 a.m. John has registered to the hospital to undergo some health checks, at 8:10 he was taken to the radiology department where he was visited at 8:15 and he is now having X-rays. Predictive Process Monitoring would allow us to answer different types of questions on the future of John. For instance, we could predict whether John will undergo an ultrasound scan in the future. The answer to this specific question will be a boolean value (e.g., it is true that John will undergo an ultrasound in the future). This is a typical example of an outcome-based prediction. However, this class of predictions also includes predictions assuming categorical values, that is, values that range in a limited and fixed number of possible options. Examples are the class of discount that will be applied to a customer at the end of his shopping, the class of risk of a given execution, or, in our scenario, the specific exam out of a number of options. Another typical question Predictive Process Monitoring could allow us to answer about John's future is, once we know that he will undergo an ultrasound, in how much time he is going to have it. The answer to this question is generally provided in terms of a numeric value (e.g., John is going to have an ultrasound exam in 26 min) and is an example of a numeric-value prediction. Typical examples in this settings are predictions related to the remaining time of an ongoing execution, predictions related to the duration or to the cost of an ongoing case. Finally, we could even predict what John is going to do from now on. The answer to this question is a sequence of future activities (e.g., John will undergo an ultrasound, will ask for his bill and will pay it). Typical examples of predictions falling under this category refer to the prediction of the sequence of the future activities (and of their data payloads) of a process case upon its completion.

Predictive Process Monitoring approaches are usually characterized by two phases. In a first phase, the *training or learning* phase (see the light blue part in Fig. 3), one or more models are built or enriched by leveraging the information contained in the execution log. In the second phase, the *runtime or prediction phase* (see the light green part in Fig. 3), the learned model(s) is(are) exploited

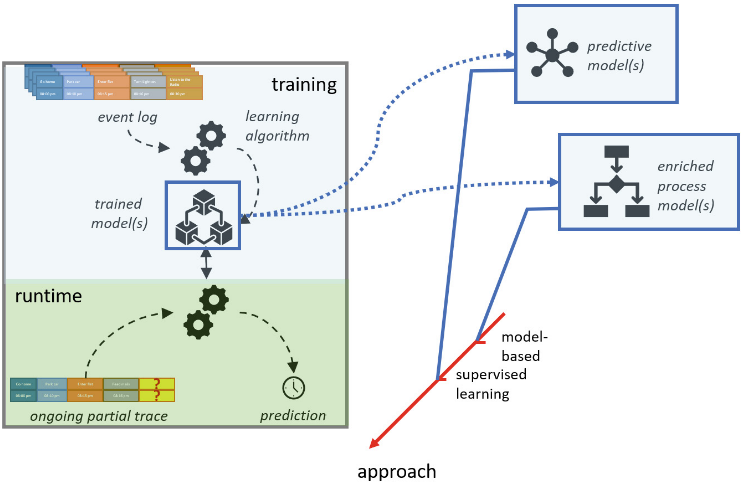


Fig. 3. Types of PPM approaches.

in order to get predictions related to an ongoing execution trace. We can identify two main groups of approaches dealing with the prediction problem:

- approaches relying on an explicit model (*model-based approaches*), e.g., annotated transition systems. The explicit model can either be discovered from the event log and then enriched with the information the log contains or directly be enriched, if an explicit model is already available. In model-based approaches, the model that is then leveraged at runtime in order to get predictions is an (enriched) model in which the process control flow is somehow made explicit (see the blue box in the middle on the right of Fig. 3).
- approaches leveraging machine learning and statistical techniques, e.g., classification and regression models, as well as neural networks. These approaches only rely on (implicit) predictive models built by encoding event log information in terms of features to be used as input for machine/deep learning techniques (see the blue box at the top on the right of Fig. 3).

Finally, we can identify four different types of information that can be used as input to the Predictive Process Monitoring approaches, e.g., for building a model annotated with execution information or for building the features to be used by machine learning approaches:

- information related to the control flow - i.e., the sequence of events. As depicted in the fourth row of Fig. 4, in the example of John's history this is the information related to the activities carried out by John (e.g., check in to the hospital, go to the radiology department, . . .).
- information related to the structured data payload associated to the events. This information usually include the timestamp of the events, but it can also include other types of data attributes. For instance, in John's history, besides

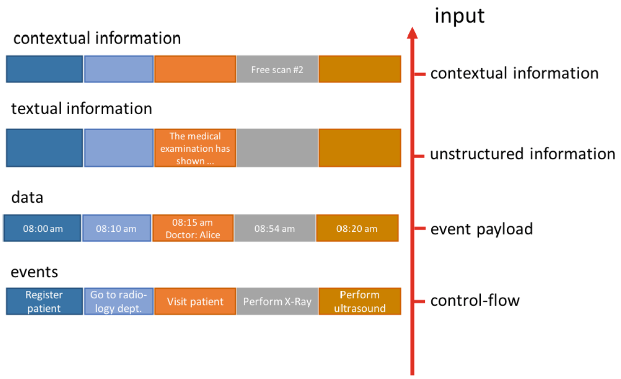


Fig. 4. Information used for making predictions.

the timestamp associated to each event, the data payload of the event `Visit patient` also includes the doctor who has visited John, i.e., *Alice* (see the third row in Fig. 4).

- information related to unstructured (textual) content, which can be available together with the event log. Indeed, it often happens that, together with the structured information related to the events and data payload, some unstructured information is also available. In John's example, for instance, the text of Alice's medical report is available together with the event `visit patient` (see the second row in Fig. 4) and could provide useful information on what John is going to do later on.
- information related to process context, such as workload or resource availability. In John's example, this kind of information could be related for instance to the availability of free ultrasound scan machines (first row in Fig. 4). Contextual information could provide useful information on what John is going to do later on and when. For example, the time required to John to perform an ultrasound could be related to the immediate availability of a scan equipment.

In several approaches, more than one of these types of information is used in order to learn from the past.

After reporting few more details on the model-based approaches and approaches leveraging machine learning in the next subsection, in the following sections, we will mainly focus on machine-learning approaches and on encodings taking into account event and data payload features. We will look in more detail at each of the three prediction type macro-categories, i.e., predicting outcomes, numeric values and sequences of activities (and related payloads), respectively.

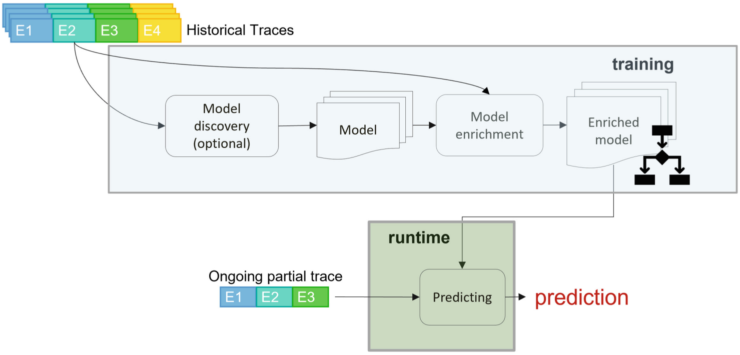


Fig. 5. Overview of the typical phases of model-based approaches.

σ_1 (Visit patient (VP) {*07:00*}, Register patient (RP) {*08:00*}, Compute rate (CR) {*11:00*}, Get payment (GP) {*12:00*}, Perform ultrasound (PU) {*18:00*})
σ_2 (Visit patient (VP) {*08:00*}, Compute rate (CR) {*10:00*}, Perform X-Ray (XR) {*13:00*}, Get payment (GP) {*16:00*})
σ_3 (Compute rate (CR) {*09:00*}, Visit patient (VP) {*13:00*}, Perform X-Ray (XR) {*14:00*}, Get payment (GP) {*15:00*})
σ_4 (Visit patient (VP) {*10:00*}, Register patient (RP) {*11:00*}, Compute rate (CR) {*14:00*}, Get payment (GP) {*15:00*}, Perform X-Ray (XR) {*17:00*})

Fig. 6. Running example model-based approaches.

3.1 Predictive Process Monitoring Approaches

We report here an overview of the main phases related to the two main families of Predictive Process Monitoring approaches, i.e., model-based approaches and approaches based on machine learning.

Figure 5 shows the main phases characterizing the model-based approaches. At training time, an explicit and conformant model (see [7]) can either be already available or can be discovered from the historical traces (see [2,4]) using the optional *Model discovery* phase in Fig. 5. The model is then enriched with information related to the data (*Model enrichment*), as for instance the remaining time extracted from the historical traces. At runtime, the enriched model is used in order to return a prediction.

One of the main model-based approaches leverages a transition system as explicit control-flow model (see Definition 1 in [4]). The transitions system is built based on a given abstraction of the representation of the events in the traces (e.g., the name of the activity), as well as of the representation of the state of the transition system, as for instance the *sequence* of activities executed so far or the *set* of activities occurred so far. For instance, let us consider the simple event log reported in Fig. 6 related to the example described in Sect. 2. Each case relates to a different patient and the corresponding sequence of events indicates the activities executed for a medical treatment of that patient. Given the variability of the process, different interplays are possible between the clinical and administrative activities. In particular in sequence σ_1 the process starts

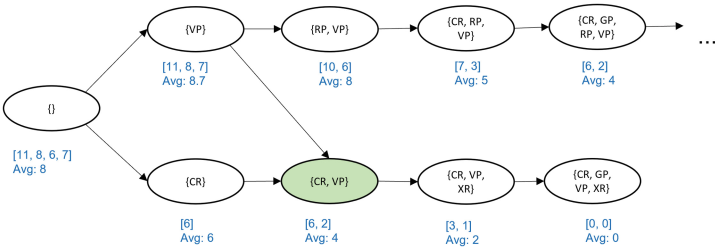


Fig. 7. Annotated transition system obtained from the log reported in Fig. 6.

directly with a visit (possibly due to urgency), while the administrative part is executed in the middle of the process; instead in sequence σ_3, the process starts with a computation of the overall price (possibly due to the request of having a quote) before proceeding further. The event timestamp of each event is reported among brackets nearby the activity. For example, trace σ_2 refers to a process execution in which the activity `Visit patient` is executed at time 08:00, the activity `Compute rate` at time 10:00 and so on. Figure 7 shows the transition system computed using as event representation abstraction the name of the activity and as state representation the activity *set*.

The transition system is then annotated, given a certain measurement function, as for instance the elapsed time or the remaining time, with the corresponding information extracted from the event log. For instance, information about the remaining time can be extracted from the traces and reported for each state of the transition system. This information is then used for making predictions, e.g., on the completion time of an ongoing trace, given a certain prediction function, as for example the average remaining execution time. The transition system in Fig. 7, for instance, is annotated (in blue) with the remaining time of each trace in the event log of Fig. 6. Moreover, for each state, the average of these values is also computed and reported. For example, the state corresponding to the empty set of activities , is annotated with the remaining time of each trace at the beginning of the execution, i.e., 11 h for σ_1, 8 h for σ_2 and so on.

When, at runtime, a prediction about the completion time of a new ongoing trace is required, the annotated transition system can be queried by looking at the state of the transition system corresponding to the ongoing case, and the value of the chosen prediction function returned. For instance, let us assume we want to predict the completion time of an ongoing case $\sigma_t = ($`Compute rate (CR)` $\{$*12:00*$\}$, `Visit patient (VP)` $\{$*13:00*$\})$. Two measurements are associated to the corresponding state of the transition system in Fig. 7 (see the state in light green), i.e., 6 and 2 hours. Considering the average as prediction function, the average value of the measurements (4 hours) can be used to compute the predicted completion time, i.e., according to the prediction, the patient will complete his process at 17:00.

Several extensions have been proposed to the original approach, such as annotating the transition systems with machine learning models like Naïve Bayes

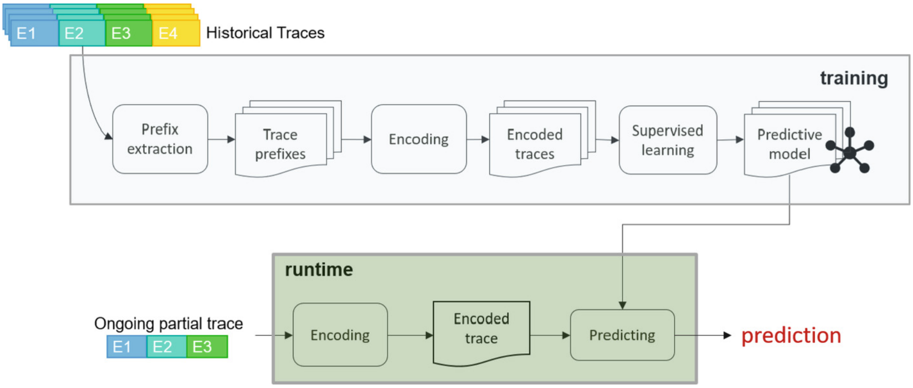


Fig. 8. Overview of the typical phases of approaches based on machine learning.

and Support Vector Regression models [34], taking into account also data payloads [35], combining the annotated transition systems with a context-driven predictive clustering approach [16,17]. Other model-based approaches consider, instead *sequence trees* [9] or stochastic Petri nets [40,41] as explicit models to predict the remaining execution time of a process instance.

Figure 8 sketches the main phases of the typical approaches based on machine learning. These approaches usually require that trace prefixes are extracted from the historical execution traces (*Prefix extraction* phase). This is due to the fact that at runtime predictions are made on incomplete traces, so that correlations between incomplete traces and what we want to predict (*target variables* or *labels*) have to be learned in the training phase. After prefixes have been extracted, prefix traces and labels (i.e., the information that has to be predicted) are encoded in the form of feature vectors (*Encoding* phase). Encoded traces are then passed to the (supervised learning) techniques in charge of learning from the encoded data one (or more) predictive model(s) (*Encoding* phase). At runtime, the incomplete execution traces i.e., the traces whose future is unknown, should also be encoded as feature vectors and used to query the predictive model(s) so as to get the prediction (*Predicting* phase).

In this chapter we will mainly focus on approaches leveraging machine learning - and in particular supervised learning - techniques.

4 Predicting Outcomes

Outcome predictions are predictions related to (categorical) case outcomes [46]. Typical examples of outcome predictions in the Predictive Process Monitoring literature are predictions related to risks or related the fulfilment of a predicate [11,29].

Given an event log L and a prefix execution trace $\sigma_i^m = <e_1, \ldots, e_m>$ of length m, the overall idea is learning a function $f_c(L, \sigma_i^m)$ returning a categorical

σ_1 (`Visit patient` {*33*, *clinic*},...,`Perform ultrasound` {*33*, *radiology*}): false

...

σ_k (`Compute rate` {*56*, *general lab*},..., `Get Payment` {*56*, *admin*}): true

Fig. 9. Running example with an outcome label

value $\overline{label_i}$, which is as close as possible to $label_i$, i.e., the actual (categorical) value of the variable that we aim to predict (e.g., whether the predicate will be actually fulfilled).

As described in the previous section, when dealing with approaches based on machine learning, one of the main steps to be carried out deals with encoding the information contained in (prefix) execution traces and corresponding labels in a format that is understandable by machine learning techniques. This would allow the technique to train, and hence learn, from encoded data a predictive model. In order to train a model, each (prefix) execution trace σ_i, (and its corresponding label) have to be represented through a feature vector $g_i = (g_{i1}, g_{i2}, ...g_{ih}, label_i)$.

In this section (and in the next two sections) we will present first the typical encodings used with the corresponding type of predictions[4] and then the main (machine-learning) pipelines/approaches used to build the predictive model and query it.

4.1 Typical Data Encodings

To exemplify the different data encoding techniques, we consider the very simple log in Fig. 9 pertaining to our running example of Sect. 2. Similarly to the log used in Sect. 3.1, also in this log each case relates to a different patient and the corresponding sequence of events indicates the activities executed for a medical treatment of that patient. `Visit patient` is the first event of sequence σ_1. Its data payload "{*33*, *radiology*}" corresponds to the data associated to attributes *age* and *department*[5]. Note that the value of *age* is static: it is the same for all the events in a case, while the value of *department* is different for every event. In the payload of an event, the entire set of attributes available in the log is considered as well. In case for some event the value for a specific attribute is not available, the value $\perp$ (*unknown*) is specified for it.

Given a case prefix, we aim at predicting whether the patient will recover soon (*true*), or not (*false*). We report the corresponding value, i.e., the corresponding label, for each case after the semicolon in Fig. 9.

Boolean Encoding. In the *boolean encoding* sequences of events are represented as feature vectors, in such a way that each feature corresponds to an event class (an activity) from the log. In particular, the boolean encoding represents a sequence

[4] Please note that some types of encodings can be used for different types of predictions. For instance encodings related to outcome-based and numerical predictions are exactly the same - except for the type of the label.

[5] We omit here for simplicity the information related to timestamps.

Table 1. Typical outcome-based encodings for the example in Fig. 9.

	`Visit patient`	`Perform ultrasound`	...	`Get Payment`	label
σ_1	1	1	...	0	false
...					
σ_k	0	0	...	1	true

(a) *boolean* encoding.

	`Visit patient`	`Perform ultrasound`	...	`Get Payment`	label
σ_1	2	1	...	0	false
...					
σ_k	0	0	...	4	true

(b) *frequency-based* encoding.

	event_1	...	event_m	label
σ_1	`Visit patient`		`Perform ultrasound`	false
...				
σ_k	`Compute rate`		`Get Payment`	true

(c) *simple-index* encoding.

	age	department_last	label
σ_1	*33*	*radiology*	false
...			
σ_k	*56*	*admin*	true

(d) *latest-payload* encoding.

	age	event_1	...	event_m	...	department_last	label
σ_1	*33*	`Visit patient`		`Perform ultrasound`	...	*radiology*	false
...							
σ_k	*56*	`Compute rate`		`Get Payment`	...	*admin*	true

(e) *index latest-payload* encoding.

	age	event_1	...	event_m	...	department_1	...	department_m	label
σ_1	*33*	`Visit patient`		`Perform ultrasound`		*clinic*		*radiology*	false
...									
σ_j	*56*	`Compute rate`		`Get Payment`		*general lab*		*admin*	true

(f) *complex index-based* encoding.

σ_i through a feature vector $g_i = (g_{i1_A}, g_{i2_A}, ...g_{ih_A}, label_i)$, where h_A is the size of the event class alphabet $A = \{a_{1_A}, \ldots, a_{h_A}\}$ and if g_{ij_A} corresponds to the event class $a_{j_A} \in A$ then:

$$g_{ij} = \begin{cases} 1 & \text{if } a_{j_A} \text{ occurs in } \sigma_i \\ 0 & \text{if } a_{j_A} \text{ does not occur in } \sigma_i \end{cases}$$

For instance, the encoding of the example reported in Fig. 9 with the *boolean* encoding is shown in Table 1a.

Frequency-Based Encoding. The *frequency-based* encoding, instead of boolean values, represents the control flow in a case with the frequency of each event class in the case. The frequency-based encoding $g_i = (g_{i1_A}, g_{i2_A}, ...g_{ih_A}, label_i)$ of σ_i, is such that, if g_{ij_A} corresponds to the event class $a_{j_A} \in A$ then:

$$g_{ij} = \begin{cases} n & \text{if } a_{j_A}\text{occurs } n \text{ times in } \sigma_i \\ 0 & \text{if } a_{j_A}\text{does not occur in } \sigma_i \end{cases}$$

Table 1b shows the *frequency-based* encoding for the example in Fig. 9, assuming that `Visit patient` occurs two times in σ_i and `Get Payment` occur four times in σ_k.

Simple-Index Encoding. Another way of encoding a sequence is by taking into account also information about the order in which events occur in the sequence, as in the *simple-index* encoding. Here, each feature corresponds to a position in the sequence and the possible values for each feature are the event classes. The resulting feature vector g_i of the simple-index encoding of an execution trace σ_i of length m is $g_i = (a_{i1}, a_{i2}, ...a_{im}, label_i)$, such that a_{ik} corresponds to the event class of the event at position k in σ_i. By using this type of encoding the example in Fig. 9 would be encoded as reported in Table 1c.

Latest-Payload Encoding. The *latest-payload* encoding takes into account both the static and the dynamic data attributes of the traces. The value of static attributes (trace attributes) is the same for all the events in the sequence, while the value of dynamic data attributes (event attributes) changes for different events. However, in this encoding, data attributes, also the dynamic ones, are all treated as static features without taking into consideration their evolution over time. Indeed, the latest-payload encoding encodes the data attributes and the data of the latest payload. The latest-payload encoding g_i of an execution trace σ_i of length m is $g_i = (s_i^1, \ldots, s_i^u, d_{im}^1, \ldots, d_{im}^r, label_i)$, where each s_i is a static feature and each d_{im} is a dynamic feature associated to the last event, i.e., the event at position m. Table 1d shows this encoding for the example in Fig. 9.

Index Latest-Payload Encoding. The *index latest-payload* encoding adds the latest encoding to the simple-index encoding. The resulting feature vector g_i, for a sequence $g_i = \sigma_i$, is $g_i = (s_i^1, \ldots, s_i^u, a_{i1}, a_{i2}, \ldots, a_{im}, d_{im}^1, \ldots, d_{im}^r, label_i)$, where each s_i is a static feature, each a_{ij} is the event class at position j and each d_{im} is a dynamic feature associated to the event at position m. Table 1e reports this encoding for the example in Fig. 9.

Complex Index-Based Encoding. In the *complex-based* encoding, the dynamic nature of the dynamic information is considered and its evolution over time is taken into account. The resulting feature vector g_i, for a sequence σ_i, is $g_i = (s_i^1, .., s_i^u, a_{i1}, a_{i2}, ..a_{im}, d_{i1}^1, d_{i2}^1, \ldots, d_{im}^1, \ldots, d_{i1}^r, d_{i2}^r, ...d_{im}^r, label_i)$, where each s_i is a static feature, each a_{ij} is the event class at position j and each d_{ij} is a dynamic feature associated to an event. The example in Fig. 9 is transformed into the encoding shown in Table 1f.

4.2 Mostly Used Approaches: Classification-Based Approaches

Different pipelines and frameworks have been proposed for providing outcome predictions. Most of them relies on classification techniques[6] (e.g., Decision Tree, Random Forest, Support Vector Machine) for the *supervised learning* phase [12, 23, 25, 29]. Moreover, most of these pipelines have been enriched with a *Bucketing* phase [46] (see the orange blocks in Fig. 10). The idea is that at training time

[6] Note that deep learning techniques can also be used for predicting outcomes [52], however we focus here on the mostly used approaches.

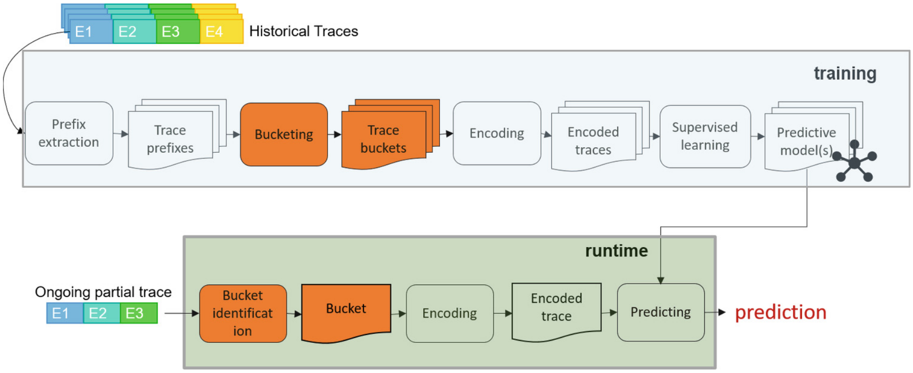


Fig. 10. Typical outcome-based pipeline

multiple predictive models are trained. Specifically, the log of prefix traces is divided in multiple buckets and each bucket is used to train a different classifier. At runtime, the most suitable bucket is identified and the corresponding classifier used for predicting the outcome.

The *Bucketing* phase has been instantiated in different ways in the Predictive Process Monitoring literature. For instance, in [12] trace clustering has been used to group prefix traces. Specifically, at training time, a clustering algorithm has been leveraged to cluster together prefix traces sharing a similar control flow. For each cluster, the data payload of the prefix traces in the cluster, once encoded in the proper format, has then been used to train a classifier. At runtime, the cluster of the incomplete ongoing trace is identified, i.e., the cluster containing the trace prefixes closest to the current incomplete trace, and the corresponding classifier queried in order to get the prediction. In [25], instead, a bucket consists of a set of prefix traces of the same length. Also in this case, at training time, a classifier for each prefix length k is built by learning from all prefix traces of length k. At runtime, the classifier of the same length of the ongoing trace is identified and the prediction returned.

5 Predicting Numeric Values

Numeric value predictions are predictions related to quantitative measures of interest of business process executions. Typical examples of numeric predictions in the Predictive Process Monitoring literature are predictions related to time, cost or generic process performance [1,8,48].

Given an event log L and a prefix execution trace $\sigma_i^m = <e_1, \ldots, e_m>$ of length m, the overall idea is learning a function $f_n(L, \sigma_i^m)$ returning a numerical value $\overline{label_i}$, which is as close as possible to $label_i$, i.e., the actual (numerical) value of the variable that we aim to predict (e.g., the remaining cycle time until the completion of the execution).

σ_1 (`Visit patient` {*33*, *clinic*}, ..., `Perform ultrasound` {*33*, *radiology*}): 3.5
...
σ_k (`Compute rate` {*56*, *general lab*}, ..., `Get Payment` {*56*, *clinic*}): 5

Fig. 11. Running example with a numeric label

Table 2. Typical numeric-based encodings for the example in Fig. 9.

	`Visit patient`	`Perform ultrasound`	...	`Get Payment`	label
σ_1	1	1	...	0	3.5
...					
σ_k	0	0	...	1	5

(a) *boolean* encoding.

	`Visit patient`	`Perform ultrasound`	...	`Get Payment`	label
σ_1	2	1	...	0	3.5
...					
σ_k	0	0	...	4	5

(b) *frequency-based* encoding.

	event_1	...	event_m	label
σ_1	`Visit patient`		`Perform ultrasound`	3.5
...				
σ_k	`Compute rate`		`Get Payment`	5

(c) *simple index* encoding.

	age	department_last	label
σ_1	*33*	*radiology*	3.5
...			
σ_k	*56*	*clinic*	5

(d) *latest payload* encoding.

	age	event_1	...	event_m	...	department_last	label
σ_1	*33*	`Visit patient`		`Perform ultrasound`	...	*radiology*	3.5
...							
σ_k	*56*	`Compute rate`		`Get Payment`	...	*clinic*	5

(e) *index latest payload* encoding.

	age	event_1	...	event_m	...	department_1	...	department_m	label
σ_1	*33*	`Visit patient`		`Perform ultrasound`		*clinic*		*radiology*	3.5
...									
σ_j	*56*	`Compute rate`		`Get Payment`		*general lab*		*clinic*	5

(f) *complex index-based* encoding.

5.1 Typical Data Encodings

Let us consider the running example of Fig. 9 and let us assume that this time we would like to predict the time required for completing the execution (reported in Fig. 11 after the semicolon).

Encodings typically used for numeric predictions are the same as the ones used for categorical predictions, except for the label, which is a numerical value rather than a boolean or a categorical value. Table 2 summarizes the boolean, frequency, simple-index, latest-payload, index latest-payload and complex-index encodings for numeric-based predictions.

5.2 Mostly Used Approaches: Regression-Based Approaches

Pipelines and frameworks proposed for numeric predictions are quite similar to the ones for outcome predictions. Most of them relies on regression techniques[7] (e.g., Regression Trees, Random Forest, XGBoost) for the *supervised learning* phase [23,29].

6 Predicting Next Events

Next event predictions are predictions related to the unfolding of the future events - until the end - of an incomplete ongoing trace [45]. Next event predictions can be related to the sequence of next event classes, but also to the next data payloads associated to the events, as for instance, the timestamps or the resources associated to the next event(s).

In case of activity predictions, given an event log L and a prefix execution trace $\sigma_i^m = <e_1, \ldots, e_m>$ of length m, the overall idea is learning a function $f_{sa}(L, \sigma^m)$ returning a sequence of next event classes that is as close as possible to $a_{m+1}, \ldots, \omega$, i.e., to the activity suffix of the current ongoing trace.

Most of the approaches for next activity predictions typically first learn a function f_{1a} that, given the first m events of a trace σ_i^m, predicts the next event class, i.e., the event class that will occur at time step $m+1$. The suffix of the ongoing trace σ_i^m is then predicted until the last event ω, by predicting the next event iteratively, that is by learning the function f_{sa}:

$$f_{sa}(L, \sigma_i^m) = \begin{cases} f_{1a}(\sigma^m) & \text{if } f_{1a}(L, \sigma_i^m) = \omega \\ f_{sa}(L, < e_1, e_2, ..., e_m, e >) & \text{otherwise} \\ \quad \text{with } f_{1a}(L, \sigma_i^m) \text{ as } e\text{'s event class} & \end{cases} \tag{1}$$

Similarly, when predicting the values of the next events' data attribute x, e.g., the next timestamps, the idea is learning a function $f_{sx}(L, \sigma^m)$ returning a sequence of values of the data attribute x that is as close as possible to the sequence of values actually held by the attribute x in the next events of the ongoing trace.

In the next subsection describing the typical data encodings, we mainly focus on the encoding for the next event class prediction. The results can then be extended to the prediction of other data attributes related to the next event, as well as to predictions related to next events, as described in (1).

6.1 Typical Data Encodings

Let us consider the running example described in Fig. 9 enriched with timestamp information and let us assume that we want to predict the next activity related to the next time step (i.e., the activity at time step $m+1$). The actual activity at time step $m+1$ is reported after the semicolon for the training traces in Fig. 12.

[7] Note that deep learning techniques can also be used for predicting numeric predictions [45], however we focus here on the mostly used approaches.

σ_1 (`Visit patient` {*33*, *08:00*, *clinic*}, ..., `Perform ultrasound` {*33*, *11:00*, *radiology*}): `Check X-ray`
...
σ_k (`Compute rate` {*56*, *13:00*, *general lab*} ,..., `Get payment` {*56*, *18:00*, *admin*}): `Emit receipt`

Fig. 12. Running example with next activity as label

Table 3. Typical sequence-based encodings for the example in Fig. 12.

	event_1	...	event_m	label
σ_1	1 0 0 0 0 0	...	0 1 0 0 0 0	0 0 0 0 1 0
...				
σ_k	0 0 1 0 0 0	...	0 0 0 1 0 0	0 0 0 0 0 1

(a) *one-hot* encoding

	event_1				...	event_m				label
	a_1	δ_1	h_1	w_1		a_m	δ_m	h_m	w_m	
σ_1	1 0 0 0 0 0	0	8	Mon	...	0 1 0 0 0 0	1	11	Mon	0 0 0 0 1 0
...										
σ_k	0 0 0 0 1 0	0	13	Sat	...	0 0 0 0 0 1	2	18	Sat	0 0 0 0 0 1

(b) *one-hot with temporal features* encoding

One-Hot Encoding. The *one-hot encoding* allows categorical data to be transformed into a numeric format. It relies on the existence of an alphabet of activities. Given the set $A = \{a_{1_A}, \ldots a_{h_A}\}$ of all possible activities, an ordering function $idx : A \rightarrow \{1, \ldots, |A|\} \subseteq \mathbb{N}$ is defined on it, such that $a_{i_A} <> a_{j_A}$ if and only if $i_A <> j_A$, i.e., two activities have the same A-index if and only if they are the same activity.

For instance, in the example in Fig. 12, if the activity alphabet is $A = \{$`Visit patient`, `Perform ultrasound`, `Compute rate`, `Get Payment`, `Check X-ray`,`Emit receipt`$\}$, the function $idx \; : A \; \rightarrow \; \{1, 2, 3, 4, 5, 6\}$ can be defined such that $idx($`Visit patient`$) = 1$, $idx($`Perform ultrasound`$) = 2$, $idx($`Computerate`$) = 3$ and so on. Each event $e_{ij} \in \sigma_i$ is then encoded as a vector (A_{ij}) where the features are all set to 0, except the one occurring at the index of its event class, which is set to 1. In the training phase, the event class of the next event e_{m+1}, which represents the target variable or label, is also encoded in the corresponding vector (A_{im}). The trace is finally encoded by composing the vectors obtained from all activities in the trace and the next activity into a matrix. The encoding of the trace σ_i is hence given by $g_i = ((A_{i1}), ..., (A_{im}), (A_{im+1}))$. The one-hot encoding related to the example in Fig. 12 is reported in Table 3a.

One-Hot Encoding with Temporal Features. The one-hot encoding, which takes into account only the activities, can be enriched with other information. For instance, another encoding used with activity sequences combines the one-hot encoding of features related to event classes and features related to time [45]. In the *one-hot encoding with temporal features*, given the set $A = \{a_{1_A}, \ldots a_{m_A}\}$ of all possible activities, each event $e_{ij} \in \sigma_i$ is encoded as the one-hot encoding of its event class enriched with three additional features pertaining to time. The first one relates to the time difference between the considered event and the one of the previous event (δ_i), the second one reports the time since midnight (h_i), thus allowing for distinguishing between working and night time, and the last one refers to the time since the beginning of the week (w_i), thus allowing for distinguishing between business and non-working days. Also in this case, in the training phase, the label, i.e., the event class of the next event e_{m+1} is also encoded with the one-hot encoding. The one-hot encoding with temporal features related to the example in Fig. 12 is reported in Table 3b.

Embedding-Based Encoding. The *embedding-based encoding* is typically used when the number of the possible values of one or more categorical variables is high and the one-hot encoding may cause an exponential growth of the feature vector dimensionality. In the embedding-based encoding, categorical data with an alphabet of possible values of size m is mapped into a n-dimensional embedding space (where n is the chosen dimensionality of the embedded space) that encodes the values of the categorical attribute so that values that are closer in the vector space are expected to be similar.

6.2 Mostly Used Approaches: LSTM-Based Approaches

Most of the approaches for next event predictions rely on Recurrent Neural Networks and, more specifically, on LSTM (Long-Short Term Memory) architectures [6,26,45].[8] This type of deep learning approaches, by using recurrent connections in a single block (LSTM cell), is indeed particularly suitable to deal with sequence problems. Different types of LSTM architectures have been proposed in the literature for predicting the label associated to the next event and its data attributes.

For instance in [45] three types of architectures have been proposed in order to predict both next activity and the timestamp of the next event and then, iteratively, suffix prediction and remaining cycle time: a first type with separate layers for activity and timestamp prediction, a second type with shared LSTM layers for both activity and timestamp prediction and finally a third one with some shared and some separate layers. The architecture proposed in [6] for predicting the next activity and its timestamp and the remaining cycle time and suffix for a running case is a composition of LSTMs and feedforward layers. In [26] an encoder-decoder framework based on LSTMs is proposed to predict the next activity and the suffix of an ongoing case. The encoder maps an input sequence into a set of high dimensional vectors and the decoder returns it back into new sequence that can be used for prediction tasks.

7 New Trends in ML-Driven Operational Support

Besides the mainstream works in the field of Predictive Process Monitoring, new research trends and directions focusing on ML-driven operational support have recently started being investigated and developed. Some of these new trends are summarised in the following subsections.

[8] Note that the usage of LSTM architectures is not limited to next event predictions - they are indeed used also for outcome and numerical predictions - nevertheless it has been widely used in the literature for this type of predictions.

Table 4. Simple-index encoding enriched with some inter-case features for the example in Fig. 9.

	event_1	event_m	simult. trace #	avg. duration	label
σ_1	`Visit patient`	`Perform ultrasound`	10	6	False
σ_k	`Compute rate`	`Get Payment`	18	8	True

7.1 Intercase Predictions

In classical works, Predictive Process Monitoring methods assume that the predicted value of interest of an ongoing case only depends on intra-case information, as for instance on the execution history of that specific case. This assumption results in encodings that include past events, inter-event durations, and other case-related attributes. However, the only intra-case assumption does not hold in many real-life scenarios. For example, in situations where cases share limited resources, the completion time of a case heavily depends on other cases that are running at the same time [42,43].

Inter-case information can be encoded in different ways, as for instance by aggregating data related to traces running simultaneously. Examples of aggregated inter-case information that can be encoded together with the intra-case features are the number of traces and the average duration of traces being executed in the same time window in which the considered trace (prefix) is being executed, e.g., the number of traces and the average duration of traces executed in the same day of the current prefix trace. Table 4 shows an example of a simple-index encoding enriched with these two simple inter-case features related to the example reported in Fig. 9, where we assume that 10 other traces are running the same day in which σ_1^m is being executed and that their average duration is 6 hours, while 18 traces are running simultaneously to σ_k^m with an average duration of 8 hours.

Taking into account the inter-case dimension is a challenging problem, since, on the one hand, we would like to take into account as much inter-case information as possible as the levels of dependencies among cases can greatly vary in different scenarios and, on the other hand, encoding several features for a large number of simultaneously running cases may lead to a feature space explosion.

7.2 Explainable Predictions

In many applications of Predictive Process Monitoring techniques, users are asked to trust a model helping them making decisions. However, users would need a certain level of trust towards the predictive model: a doctor will not operate on a patient simply because the operation has been predicted or recommended by the model. Understanding the rationale behind predictions would certainly help users decide when to trust or not to trust them.

Explainability techniques are a way to implement responsible process decision making (see [30]) and can help us to this aim. Different explainability techniques

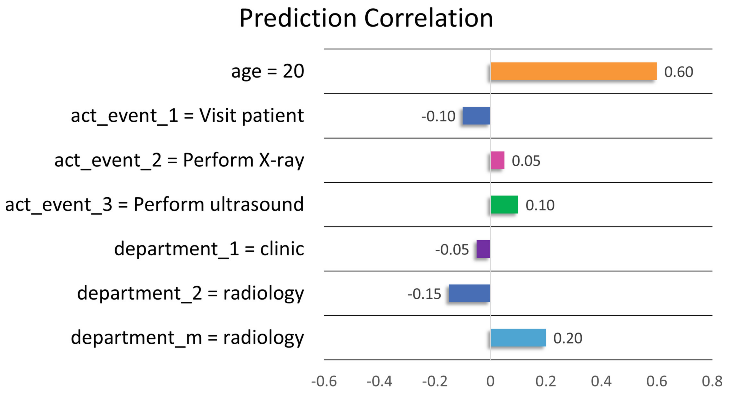


Fig. 13. Example of an explanation plot related to the prediction for σ_j.

have been proposed in the XAI (Explainable Artificial Intelligence) literature. Some of these techniques have already been experimented in the field of Predictive Process Monitoring in order to support users in understanding the overall predictive model [33] or the specific predictions it provides [18,44,51]; with model-agnostic techniques, i.e., techniques that can be applied to any predictive model, as in the case of [18] or with techniques specific to the predictive model used, as in the case of XNAP [51] and the attention layer [44] for neural networks.

As an example of prediction explanation related to a trace instance,[9] let us assume that we have trained our predictive model by encoding the training set of the example reported in Fig. 9 with the complex-index encoding (see Table 1f) and that, for our current ongoing trace σ_j (`Visit patient` {*20*, *clinic*}, `Perform X-Ray` {*20*, *radiology*}, `Perform ultrasound` {*20*, *radiology*}), which we have observed up to the event 3, the prediction of our predictive model is that the patient will recover soon. In order to understand whether we can trust or not the prediction, we would need to understand why our predictive model has returned such a prediction. Figure 13 shows an example of a possible explanation returned by a prediction explainer as LIME[37] or SHAP[27] applied to our specific Predictive Process Monitoring problem. The plot shows the impact of each feature (and related value) towards (in case of positive values) or against (in case of negative values) the fast recovery of the patient.[10] In the example, the feature

[9] Note that we provide here the idea of prediction explanations focusing on those related to a trace instance. However, aggregated trace prediction explanations (event log explanations) [18], as well as prediction model explanations [44] have also been investigated in the literature.

[10] Note that the semantics of the values on the x axis changes according to the explanation technique used for the plot. For instance, in the case of SHAP, the values on the x axis represent the SHAP values of the feature (and the related value) for the specific instance, that is the contribution of the feature towards the prediction with respect to the average value.

that has impacted most on the prediction of the fast recovery of the patient is her young age.[11]

Furthermore, the explanations used for making predictions more trustable to the users can be eventually used also for understanding the reasons why a predictive process model is wrong and hence use them to improve the model accuracy [38].

7.3 Predictions with A-Priori Knowledge

Past event logs, or more in general knowledge about the past, is not the only important source of knowledge that can be leveraged to make predictions. In many real life situations, cases exist in which, together with past execution data, some case-specific additional knowledge (*a-priori knowledge*) about the future is available and can be leveraged for improving the predictive power of a Predictive Process Monitoring technique. Indeed, this additional a-priori knowledge is what characterizes the future context of the process executions that will affect the development of the currently running cases.

We can think for instance to the occurrence of a strike, which may cause the delay or the cancellation of a flight in the travel process of a passenger, or to the temporary unavailability of a surgery room, which may delay or even rule out the possibility of executing certain activities in a patient treatment process. In this kind of scenarios, the information about the strike or about the unavailability of the surgery room is often available in advance. However, traditional Predictive Process Monitoring approaches, which only learn from the most frequent observed behaviours, are not able to take into account this knowledge. They will predict that the next activities of the passenger will be the usual ones, as if there is no strike, e.g., having the security check, moving to the boarding gate 3, boarding, ... While it is impractical to retrain the predictive algorithms to take into consideration this additional knowledge every time it becomes available, it is also reasonable to assume that considering it in some way would allow the Predictive Process Monitoring algorithm to predict for instance that the passenger will be moved to gate 2 and that there will be no boarding, and hence to improve the accuracy of the predictions on an ongoing case.

A possibility to deal with a-priori knowledge is to take into account this knowledge K at prediction time by guiding the Predictive Process Monitoring algorithm towards a solution that is compliant to the a-priori knowledge [14]. In [14] for instance, an approach using LSTM for predicting the next activities has been enriched with a mechanism able to take into account background knowledge K expressed in terms of LTL formulae in order to guide the LSTM algorithm to make predictions compliant with the a-priori knowledge. The LSTM approach keeps returning likely predictions on the suffix of the current ongoing trace (up to the last event ω) until it does not find a suffix that is compliant with K. More in detail, the LSTM network uses a beam search algorithm for

[11] Note that different types of explanations can be returned depending on the type of encoding that has been used.

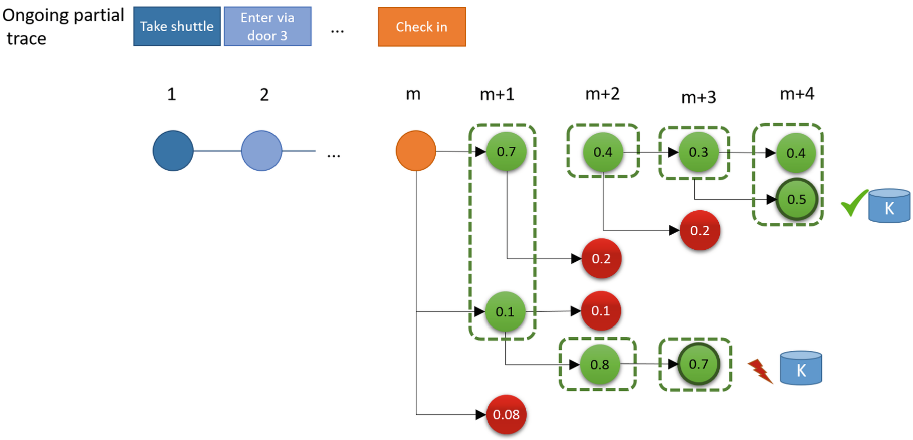


Fig. 14. Beam search in the a-priori approach.

considering at each time step the top beam-width bw most likely next events. Figure 14 shows the idea of the beam-search approach with $bw = 2$. $\sigma^m =$<`Take shuttle`, `Enter via door 3`, `Check in`> is the current ongoing trace at time step m. At time step $m+1$, among the three possible next events we take the bw most likely next events (the green nodes in Fig. 14) and keep exploring those future paths. At time step $m+2$, we again select the 2 most likely next events and keep exploring the next events of these sequences. Whenever we find a sequence that is not compliant with `K`, as at time step $m+3$, we discard that path and we keep on exploring bw compliant paths. We stop whenever we predict the last event ω (see the circle with the thicker border) and the considered trace is still compliant with `K`.

7.4 Prescriptive Process Monitoring

Predictive Process Monitoring techniques are able to predict the likelihood of a positive outcome, the time required for completing an execution or the next activities that will be executed. However, all these techniques, are limited to the prediction. They do not support further stakeholders in making decisions on whether it is worth to intervene to avoid undesired outcomes and what to do next to optimize a given Key Process Performance Indicator (KPI) [24,32,47,50].

Prescriptive Process Monitoring aims to overcome this limit of Predictive Process Monitoring by supporting or prescribing stakeholders with decisions on whether to take actions in order to prevent or mitigate the occurrence of an undesired outcome [32,47] or on the activities to take for optimizing a certain measure of interests [24,50].

In the first scenario [32,47], predictions are used in order to evaluate through a cost model the tradeoffs between the cost of intervention to mitigate undesired outcomes and the cost of compensating unnecessary interventions. For instance, in the example related to the patient recovery described in Sect. 4, if the prediction related to an ongoing trace is that the patient will not recover soon, a surgery may increase the likelihood that the patient will recover soon and hence reduce anyway the cost for the hospital. However, the surgery has a cost, so that if the surgery has been planned because of a wrong prediction, then the cost of the surgery is unnecessary and hence should be avoided.

In the second scenario [24,50], predictions are used to uncover the future of different continuations of the current trace, so as to identify and hence recommend the one(s) leading to the best value for the KPI of interest. For instance, we can consider the example of the patient recovery described in Sect. 5. If the aim is recommending next activities to minimize the remaining cycle time until the completion of the execution of an ongoing trace σ^m of length m, possible next activities at step $m+1$ can be considered. For each possible continuation of σ^m, σ^{m+1}, the remaining time until the end of the execution can be predicted and the next activity corresponding to the minimum cycle time recommended.

8 Tool Support

The research related to Predictive Process Monitoring has been paired with the development of non-commercial plugins and tools with the purpose to be used and improved by the research community. We briefly illustrate in the following three among the main open-source tools supporting Predictive Process Monitoring.

8.1 Predictive Process Monitoring in ProM

ProM [15] is one of the most used and known tool in Process Mining. It is a framework collecting a number of plugins, working independently one from the other, and each focused on implementing a specific task. Among its variety of plugins, ProM also collects several plugins implementing techniques for the prediction of outcomes (e.g., [8,29]), for the prediction of numerical values (e.g., [1,10,16,23]), as well as for the prediction of next activity sequences (e.g., [35]). Some of them leverage model-based approaches (e.g. [1]), while others rely on machine-learning solutions (e.g., [10]).

8.2 Predictive Process Monitoring in Apromore

Apromore [22] is a well known and established tool. It is an advanced process model repository that allows to hold, analyse, and re-use large sets of process models. The tool is web-based and therefore it allows the easy integration of new plug-ins in a service oriented manner. This tool aims both at allowing practitioners to deal with the challenges of stakeholders of processes, and researchers to develop and benchmark their own techniques with a strong emphasis on the separation of concerns. The only plug-in performing Predictive Process Monitoring related challenges in Apromore is the one described in [49]. This plug-in performs outcome-based, numeric-based prediction, as well as next event predictions.

8.3 Predictive Process Monitoring in Nirdizati

Nirdizati [21,39] is a web-based application for supporting users in building, comparing, and analyzing predictive models that can then be used to perform predictions on the future development of an ongoing case. Differently from the other tools, Nirdizati specifically addresses Predictive Process Monitoring problems. Nirdizati, which collects a rich set of different state-of-the-art approaches based on machine learning algorithms, supports users to deal with different predictive monitoring tasks: outcome-based, numeric and next activities predictions. Moreover, it provides services for supporting the users in tuning the hyperparameters of the specific technique, the possibility of adding some simple intercase features in the encodings, as well as some incremental algorithms, so as to be able to incrementally update the predictive model as soon as new execution traces are available. Finally, it also offers several plots for the results visualisation, thus supporting the users in the predictive model comparison tasks.

Acknowledgements. The work described in this chapter describes the effort of a number of people. We would like to thank Marlon Dumas, Marcello La Rosa, Anna Leontjeva, Fabrizio Maria Maggi, Williams Rizzi, Arik Senderovich, Irene Teinemaa, Ilya Verenich, and Anton Yeshchenko for their precious cooperation and all the (also joint) work that led to this chapter.

References

1. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Inf. Syst. **36**(2), 450–475 (2011)
2. Aalst, W.: Foundations of Process Discovery. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 37–75. Springer, Cham (2022)
3. Aringhieri, R., et al.: Leveraging structured data in predictive process monitoring: the case of the ICD-9-CM in the scenario of the home hospitalization service. In: Proceedings of "Towards smarter health Care: Can Artificial Intelligence Help? (SMARTERCARE)" Workshop. CEUR Workshop Proceedings, CEUR-WS.org (2021)
4. Augusto, A., J. Carmona, Verbeek, E.: Advanced Process Discovery Techniques. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 76–107. Springer, Cham (2022)
5. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, Newyork (2006). https://doi.org/10.1007/978-1-4615-7566-5
6. Camargo, M., Dumas, M., González-Rojas, O.: Learning accurate LSTM models of business processes. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNCS, vol. 11675, pp. 286–302. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26619-6_19
7. Carmona, J., Dongen, B., Weidlich, M.: Conformance checking: foundations, milestones and challenges. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 155–190. Springer, Cham (2022)
8. Castellanos, M., Salazar, N., Casati, F., Dayal, U., Shan, M.: Predictive business operations management. IJCSE **2**(5/6), 292–301 (2006). https://doi.org/10.1504/IJCSE.2006.014772

9. Ceci, M., Lanotte, P.F., Fumarola, F., Cavallo, D.P., Malerba, D.: Completion time and next activity prediction of processes using sequential pattern mining. In: Džeroski, S., Panov, P., Kocev, D., Todorovski, L. (eds.) DS 2014. LNCS (LNAI), vol. 8777, pp. 49–61. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11812-3_5
10. Chamorro, A.E.M., Resinas, M., Cortés, A.R., Toro, M.: Run-time prediction of business process indicators using evolutionary decision rules. Expert Syst. Appl. **87**, 1–14 (2017). https://doi.org/10.1016/j.eswa.2017.05.069
11. Conforti, R., Fink, S., Manderscheid, J., Röglinger, M.: PRISM - A Predictive Risk Monitoring Approach for Business Processes, pp. 283–400. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_22
12. Di Francescomarino, C., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. IEEE Trans. Serv. Comput. **12**(6), 896–909 (2019). https://doi.org/10.1109/TSC.2016.2645153
13. Di Francescomarino, C., Ghidini, C., Maggi, F.M., Milani, F.: Predictive process monitoring methods: which one suits me best? In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 462–479. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_27
14. Di Francescomarino, C., Ghidini, C., Maggi, F.M., Petrucci, G., Yeshchenko, A.: An Eye into the Future: Leveraging A-priori Knowledge in Predictive Business Process Monitoring, pp. 252–268. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65000-5_15
15. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM framework: a new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005). https://doi.org/10.1007/11494744_25
16. Folino, F., Guarascio, M., Pontieri, L.: Discovering context-aware models for predicting business process performances. In: Meersman, R., et al. (eds.) OTM 2012. LNCS, vol. 7565, pp. 287–304. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33606-5_18
17. Folino, F., Guarascio, M., Pontieri, L.: Discovering High-Level Performance Models for Ticket Resolution Processes, pp. 275–282. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41030-7_18
18. Galanti, R., Coma-Puig, B., de Leoni, M., Carmona, J., Navarin, N.: Explainable predictive process monitoring. In: van Dongen, B.F., Montali, M., Wynn, M.T. (eds.) 2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, 4–9 October 2020, pp. 1–8. IEEE (2020). https://doi.org/10.1109/ICPM49681.2020.00012
19. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
20. Hastie, T., Tibshirani, R., Friedman, J.: Overview of Supervised Learning, pp. 9–41. Springer, New York (2009). https://doi.org/10.1007/978-0-387-84858-7_2
21. Jorbina, K., et al.: Nirdizati: a web-based tool for predictive process monitoring. In: Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Modeling (BPM 2017), Barcelona, Spain, 13 September 2017 (2017)
22. La Rosa, M., Reijers, H.A., van der Aalst, W.M.P., Dijkman, R.M., Mendling, J., Dumas, M., García-Bañuelos, L.: APROMORE: an advanced process model repository. Expert Syst. Appl. **38**(6), 7029–7040 (2011)

23. de Leoni, M., van der Aalst, W.M.P., Dees, M.: A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Inf. Syst. **56**, 235–257 (2016). https://doi.org/10.1016/j.is.2015.07.003
24. de Leoni, M., Dees, M., Reulink, L.: Design and evaluation of a process-aware recommender system based on prescriptive analytics. In: van Dongen, B.F., Montali, M., Wynn, M.T. (eds.) 2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, 4–9 October 2020, pp. 9–16. IEEE (2020). https://doi.org/10.1109/ICPM49681.2020.00013
25. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 297–313. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_21
26. Lin, L., Wen, L., Wang, J.: MM-Pred: a deep predictive model for multi-attribute event sequence. In: Berger-Wolf, T.Y., Chawla, N.V. (eds.) Proceedings of the 2019 SIAM International Conference on Data Mining, SDM 2019, Calgary, Alberta, Canada, 2–4 May 2019, pp. 118–126. SIAM (2019). https://doi.org/10.1137/1.9781611975673.14
27. Lundberg, S.M., Lee, S.: A unified approach to interpreting model predictions. In: Guyon, I., et al. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4–9 December 2017, Long Beach, CA, USA, pp. 4765–4774 (2017). https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html
28. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.P.: Compliance monitoring in business processes: Functionalities, application, and tool-support. Inf. Syst. **54**, 209–234 (2015). https://doi.org/10.1016/j.is.2015.02.007
29. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: Jarke, M., et al. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 457–472. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07881-6_31
30. Mannhardt, F.: Responsible process mining. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 373–401. Springer, Cham (2022)
31. Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortés, A.: Predictive monitoring of business processes: a survey. IEEE Trans. Serv. Comput. **11**(6), 962–977 (2018)
32. Metzger, A., Neubauer, A., Bohn, P., Pohl, K.: Proactive process adaptation using deep learning ensembles. In: Giorgini, P., Weber, B. (eds.) CAiSE 2019. LNCS, vol. 11483, pp. 547–562. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21290-2_34
33. Pauwels, S., Calders, T.: Bayesian network based predictions of business processes. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNBIP, vol. 392, pp. 159–175. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58638-6_10
34. Polato, M., Sperduti, A., Burattin, A., de Leoni, M.: Data-aware remaining time prediction of business process instances. In: 2014 International Joint Conference on Neural Networks (IJCNN), pp. 816–823, July 2014
35. Polato, M., Sperduti, A., Burattin, A., Leoni, M.: Time and activity sequence prediction of business process instances. Computing **100**(9), 1005–1031 (2018). https://doi.org/10.1007/s00607-018-0593-x

36. Reinkemeyer, L.: Status and future of process mining: from process discovery to process execution. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 405–415. Springer, Cham (2022)
37. Ribeiro, M.T., Singh, S., Guestrin, C.: Why should I trust you?: Explaining the predictions of any classifier. CoRR abs/1602.04938 (2016). http://arxiv.org/abs/1602.04938
38. Rizzi, W., Di Francescomarino, C., Maggi, F.M.: Explainability in predictive process monitoring: when understanding helps improving. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNBIP, vol. 392, pp. 141–158. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58638-6_9
39. Rizzi, W., Simonetto, L., Di Francescomarino, C., Ghidini, C., Kasekamp, T., Maggi, F.M.: Nirdizati 2.0: new features and redesigned backend. In: Depaire, B., et al. (eds.) Proceedings of the Dissertation Award, Doctoral Consortium, and Demonstration Track at BPM 2019 co-located with 17th International Conference on Business Process Management, BPM 2019, Vienna, Austria, 1–6 September 2019, CEUR Workshop Proceedings, vol. 2420, pp. 154–158. CEUR-WS.org (2019)
40. Rogge-Solti, A., Weske, M.: Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 389–403. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45005-1_27
41. Rogge-Solti, A., Weske, M.: Prediction of business process durations using non-Markovian stochastic petri nets. Inf. Syst. **54**(Suppl C), 1–14 (2015)
42. Senderovich, A., Di Francescomarino, C., Maggi, F.M.: From knowledge-driven to data-driven inter-case feature encoding in predictive process monitoring. Inf. Syst. **84**, 255–264 (2019). https://doi.org/10.1016/j.is.2019.01.007
43. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining for delay prediction in multi-class service processes. Inf. Syst. **53**(C), 278–295 (2015). https://doi.org/10.1016/j.is.2015.03.010
44. Sindhgatta, R., Moreira, C., Ouyang, C., Barros, A.: Exploring interpretable predictive models for business processes. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNCS, vol. 12168, pp. 257–272. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58666-9_15
45. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 477–492. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_30
46. Teinemaa, I., Dumas, M., Rosa, M.L., Maggi, F.M.: Outcome-oriented predictive process monitoring: review and benchmark. ACM Trans. Knowl. Discov. Data **13**(2), 1–57 (2019). https://doi.org/10.1145/3301300
47. Teinemaa, I., Tax, N., de Leoni, M., Dumas, M., Maggi, F.M.: Alarm-based prescriptive process monitoring. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNBIP, vol. 329, pp. 91–107. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98651-7_6
48. Tu, T.B.H., Song, M.: Analysis and prediction cost of manufacturing process based on process mining. In: 2016 International Conference on Industrial Engineering, Management Science and Application (ICIMSA), pp. 1–5, May 2016
49. Verenich, I., et al.: Predictive process monitoring in Apromore. In: Mendling, J., Mouratidis, H. (eds.) CAiSE 2018. LNBIP, vol. 317, pp. 244–253. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92901-9_21

50. Weinzierl, S., Dunzer, S., Zilker, S., Matzner, M.: Prescriptive business process monitoring for recommending next best actions. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNBIP, vol. 392, pp. 193–209. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58638-6_12
51. Weinzierl, S., Zilker, S., Brunk, J., Revoredo, K., Matzner, M., Becker, J.: XNAP: making LSTM-based next activity predictions explainable by using LRP. In: Del Río Ortega, A., Leopold, H., Santoro, F.M. (eds.) BPM 2020. LNBIP, vol. 397, pp. 129–141. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-66498-5_10
52. Weytjens, H., De Weerdt, J.: Process outcome prediction: CNN vs. LSTM (with Attention). In: Del Río Ortega, A., Leopold, H., Santoro, F.M. (eds.) BPM 2020. LNBIP, vol. 397, pp. 321–333. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-66498-5_24

Assorted Process Mining Topics

Streaming Process Mining

Andrea Burattin(✉)

Technical University of Denmark, 2800 Kgs. Lyngby, Denmark
andbur@dtu.dk

Abstract. Streaming process mining refers to the set of techniques and tools which have the goal of processing a stream of data (as opposed to a finite event log). The goal of these techniques, similarly to their corresponding counterparts described in the previous chapters, is to extract relevant information concerning the running processes. This chapter presents an overview of the problems related to the processing of streams, as well as a categorization of the existing solutions. Details about control-flow discovery and conformance checking techniques are also presented together with a brief overview of the state of the art.

Keywords: Streaming process mining · Event stream

1 Introduction

Process mining techniques are typically classified according to the task they are meant to accomplish (e.g., control-flow discovery, conformance checking). This classification, though very meaningful, might come short when it is necessary to decide which algorithm, technique, or tool to apply to solve a problem in a given domain, where nonfunctional requirements impose specific constraints (e.g., when the results should be provided or when the events are recorded).

Most algorithms, so far, have been focusing on a static event log file, i.e., a finite set of observations referring to data collected during a certain time frame in the past (cf. Definition 1 [1]). In many settings, however, it is necessary to process and analyze the events as they happen, thus reducing (or, potentially, removing) the delay between the time when the event has happened in the real world and when useful information is distilled out of it. In addition, the amount of events being produced is becoming so vast and complex [22] that storing them for further processing is becoming less and less appealing. To cope with these issues, event-based systems [4] and event processing systems [19] can become extremely valuable tools: instead of storing all the events for later processing, these events are immediately processed and corresponding reactions can be taken immediately. In addition, event-based systems are also responsive systems: this means they are capable of reacting autonomously when deemed necessary (i.e., only when new events are observed).

Coping with the above-mentioned requirements in the context of data analysis led to the development of techniques to analyze streams of data [3,21].

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 349–372, 2022.
https://doi.org/10.1007/978-3-031-08848-3_11

A data stream is, essentially, an unbounded sequence of observations (e.g., events), whose data points are created as soon as the event happens (i.e., in real-time). Many techniques have been developed, over the years, to tackle different problems, including frequency counting, classification, clustering, approximation, time series analysis and change diagnosis (also known as novelty detection or concept drift detection) [20,46]. Process mining techniques applied to the analysis of data streams fall into the name of *streaming process mining* [8] and both control-flow discovery as well as conformance checking techniques will be discussed later in the chapter.

The rest of this chapter is structured as follows: this section presents typical use cases for streaming process mining and the background terminology used throughout the chapter. Section 2 presents a possible taxonomy of the different approaches for streaming process mining, which can be used also to drive the construction and the definition of new ones. Section 3 introduces the problem of streaming process discovery, by presenting a general overview of the state of the art and the details of one algorithm. Section 4 sketches the basic principles of streaming conformance checking. As for the previous case, also this section starts with a state-of-the-art summary and then dives into the details of one algorithm. Section 5 mentions other research endeavors of streaming process mining and then concludes the chapter.

1.1 Use Cases

This subsection aims at giving an intuition of potential use cases for streaming process mining. In general, every setting that requires drawing conclusions before the completion of a running process instance is a good candidate for the application of streaming process mining. In other words, streaming process mining is useful whenever it is important to understand running processes rather than improving *future* ones or "forensically" investigate those *from the past.*

Process discovery on event streams is useful in domains that require a clear and timely understanding of the behavior and usage of a system. For example, let's consider a web application to self-report the annual tax statement for the citizens of a country. Such a system, typically, requires a lot of data to be inserted over many forms and, usually, the majority of its users have to interact with help pages, FAQs, and support information. In this case, it might be useful to understand and reconstruct the flow of a user (i.e., one process instance) to understand if they are getting lost in a specific section, or in specific cycles and, if necessary, provide tailored help and guidance support. Since the ultimate goal is to improve the running process instances (i.e., helping the users currently online), it is important that the analyses process the events immediately and that corresponding reactions are implemented straight away.

Conformance checking on event streams is useful whenever it is important to immediately detect deviations from reference behavior to enact proper countermeasures. For example, let's consider operational healthcare processes [31], in most of these cases (in particular in the case of non-elective care processes, such as urgent or emergency ones) it is critically important to have a sharp understanding of each individual patient (i.e., one process instance), their treatment evolution, as well as how the clinic is functioning. For example, when treating a patient having acute myeloid leukemia it is vital to know that the treatment is running according to the protocol and, if deviations were to occur, it is necessary to initiate compensation strategies immediately.

Another relevant example of streaming conformance checking could derive from the investigation of the system calls of the kernel of an operating system when used by some services or applications. These calls should be combined in some specific ways (e.g., a file should be `open()`, then either `write()` or `read()` or both appear, and eventually the file should be `close()`) which represent the reference behavior. If an application is observed strongly violating such behavior it might be an indication of strange activities going on, for example trying to bypass some limitations or privileges. Exactly the same line of reasoning could be applied when consuming RESTful services.

Additional use cases and real scenarios are depicted in the research note [5], where real-time computing[1], to which streaming process mining belongs, is identified as one of the impactful information technology enablers in the BPM field.

1.2 Background and Terminology

This section provides the basic background on streams as well as the terminology needed in the rest of the chapter.

A stream is a sequence of observable units which evolves over time by including new observations, thus becoming unbounded[2]. An *event stream* is a stream where each observable unit contains information related to the execution of an event and the corresponding process instance. In the context of this chapter, we assume that each event is inserted into the stream when the event itself happens in the real world. The universe of observable units $\mathcal{O}$ can refer to the activities executed in a case, thus having $\mathcal{O} \subseteq \mathcal{U}_{act} \times \mathcal{U}_{case}$ (cf. Definition 1 [1]), as discussed in Sect. 3) or to other properties (in Sect. 4 the observable units refer to relations B between pairs of activities, i.e., $\mathcal{O} \subseteq (B \times \mathcal{U}_{act} \times \mathcal{U}_{act}) \times \mathcal{U}_{case}$).

Definition 1 (Event stream). *Given a universe of observable units $\mathcal{O}$, an event stream is an infinite sequence of observable units: $S : \mathbb{N}_{\geq 0} \to \mathcal{O}$.*

We define an operator *observe* that, given a stream S, it returns the latest observation available on the stream (i.e., $\mathit{observe}(S) \in \mathcal{O}$ is the latest observable unit put on S).

[1] Please note that the paper explicitly mentions that, in that context, "*real-time computing refers to the so-called near real-time, in which the goal is to minimize latency between the event and its processing so that the user gets up-to-date information and can access the information whenever required*", thus perfectly matching our notion of streaming process mining.

[2] Please note that, in the literature, it is possible to distinguish other streaming models, where elements are also deleted or updated [21]. However, in this chapter we will assume an "insert-only model".

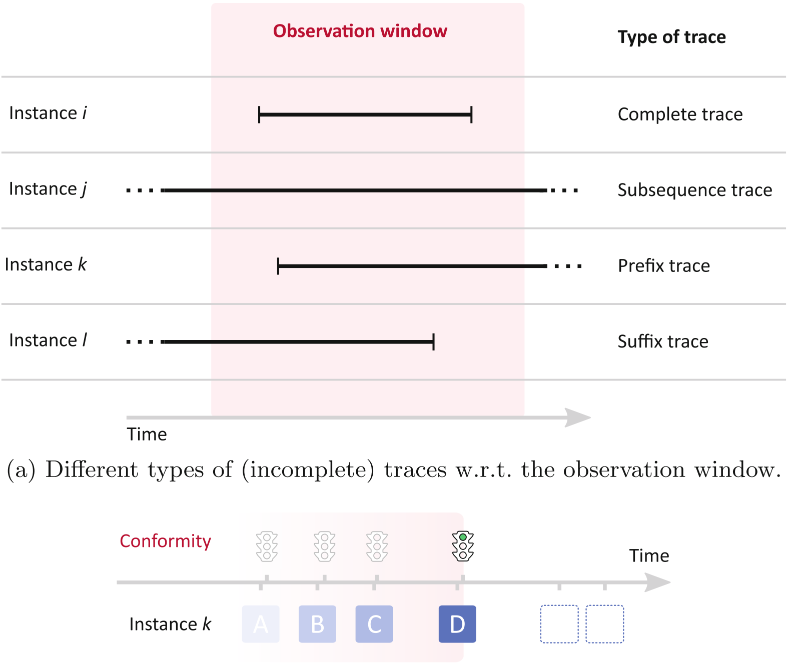
Due to the nature of streams, algorithms designed for their analyses are required to fulfill several constraints [6,7], independently from the actual goal of the analyses. These constraints are:

- it is necessary to process one event at a time and inspect it at most once;
- only a limited amount of time and memory is available for processing an event (ideally, constant time and space computational complexity for each event);
- results of the analyses should be given at any time;
- the algorithm has to adapt to changes over time.

As detailed in [21, Table 2.1], it is possible to elaborate on the differences between systems consuming data streams and systems consuming static data (from now on, we will call these "offline"): in the streaming setting, the data elements arrive incrementally (i.e., one at the time) and this imposes the analysis to be incremental as well. These events are transient, meaning that they are available for a short amount of time (during which they need to be processed). In addition, elements can be analyzed at most once (i.e., no unbounded backtracking), which means that the information should be aggregated and summarized. Finally, to cope with concept drifts, old observations should be replaced by new ones: while in offline systems, all data in the log is equally important, when analyzing event stream the "importance" of events decreases over time.

In the literature, algorithms and techniques handling data streams are classified into different categories, including "online", "incremental", and "real-time". While *real-time systems* are required to perform the computation within a given deadline – and, based on the consequences of not meeting the deadline, they are divided into hard/soft/firm –, *incremental systems* just focus on processing the input one element at the time with the solution being updated consequently (no emphasis/limit on the time). An *online system* is similar to an incremental one, except for the fact that the extent of input is not known in advance [37]. Please note that, in principle, both real-time and online techniques can be used to handle data streams, thus we prefer the more general term *streaming techniques*. In the context of this chapter, the streaming techniques are in between the family of "online" and "soft real-time": though we want to process each event *fast*, the notion of deadline is not always available, and, when it is, missing it is not going to cause a system failure but just degradation of the usefulness of the information.

When instantiating the streaming requirements in the process mining context, some of the constraints bring important conceptual implications, which are going to change the typical way process mining is studied. For example, considering that each event is added to the stream when it happens in the real world means that the traces we look at will be incomplete most of the time. Consider the graphical representation reported in Fig. 1a, where the red area represents the portion of time during which the streaming process mining is active. Only in the first case (i.e., instance i), events referring to a complete trace are seen. In all other situations, just incomplete observations are available, either because events happened before we started observing the stream (i.e., instance l, suffix trace) because the events have still to happen (i.e., instance k, prefix trace),



(a) Different types of (incomplete) traces w.r.t. the observation window.

(b) Provide immediate feedback and keep it up-to-date with respect to the input.

Fig. 1. Process mining implications of some streaming requirements.

or because of both (i.e., instance j, subsequence trace). Figure 1b is a graphical representation of what it means to give results at any time in the case of conformance: after each event, the system needs to be able to communicate the conformity value as new events come in. Also, the result might change over time, thus adapting the computation to the new observations.

2 Taxonomy of Approaches

Different artists are often quoted as saying: "Good artists copy, great artists steal". Regardless of who actually said this first[3], the key idea is the importance of understanding the state of the art to incorporate the key elements into newly designed techniques. Streaming process mining techniques have been around for some years now, so it becomes relevant to understand and categorize them in order to relate them to each other and derive new ones.

[3] Many people, including Pablo Picasso, William Faulkner, Igor Stravinsky, and several others are often referred to as the "first author" of some version of the quote. Actually, investigating the history of this quote on the Internet represents a formative yet very procrastination-prone activity (see also https://xkcd.com/214/).

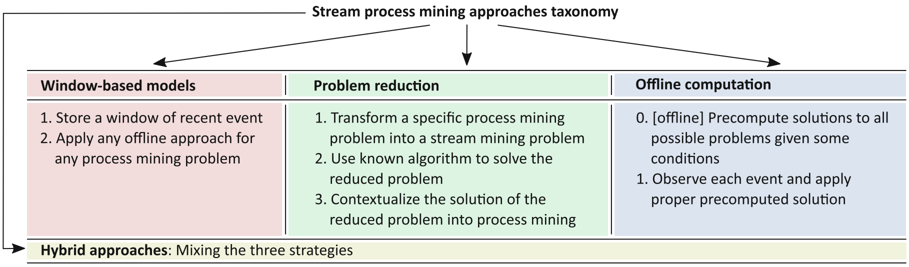

Stream process mining approaches taxonomy
Window-based models
1. Store a window of recent event
2. Apply any offline approach for any process mining problem
Problem reduction
1. Transform a specific process mining problem into a stream mining problem
2. Use known algorithm to solve the reduced problem
3. Contextualize the solution of the reduced problem into process mining
Offline computation
0. [offline] Precompute solutions to all possible problems given some conditions
1. Observe each event and apply proper precomputed solution
Hybrid approaches: Mixing the three strategies

Fig. 2. Taxonomy of the different approaches to solve the different stream process mining problems. For each technique, corresponding general steps are sketched [10].

It is possible to divide the currently available techniques for streaming process mining into four categories. A graphical representation of such taxonomy is available in Fig. 2, where three main categories are identified, plus a fourth one, which represents possible mixes of the others. In the remainder of this section, each category will be briefly presented.

Window Models. The simplest approach to cope with infinite streams consists of storing only a set of the most recent events and, periodically, analyzing them. These approaches store a "window" of information that can be converted into a static log. Then, standard (i.e., offline) analyses can be applied to the log generated from such a window. Different types of windowing models can be used and classified based on how the events are removed [34]. These models can be characterized along several dimensions, including the unit of measurement (i.e., whether events are kept according to some logical or physical units such as the time of the events or the number of events); the edge shift (so whether any of two bounds of a window is fixed to a specific time instant or if these change over time); and the progression step (i.e., how the window evolves over time, assuming that either of the bounds advances one observation at a time or several data points are incorporated at once). These profiles can create different window models, such as:

- count-based window: at every time instance, the window contains the latest N observations;
- landmark window: one of the window bounds is fixed (e.g., the starting time or the end time, far in the future) whereas the other progresses with time;
- time-based sliding window: in this case, the sliding window progresses according to two parameters: a time window duration and a time progression step;
- time-based tumbling window: similar to the time-based sliding window but where entire batches of events are observed so that two consecutive windows do not overlap.

Algorithm 1: Count-based window model process mining algorithm

```
Input: $S$: event stream
       $M$: memory
       $max_M$: number of observation to keep
       $A$: additional information (e.g., a reference model), can be $\emptyset$
forever do
    // Observe a new event
    $e \leftarrow observe(S)$
    // Memory update
    if $max(M) \geq max_M$ then
        $dequeue(M)$ // Forgetting
    end
    $insert(M, e)$
    // Mining update
    if $perform\ mining$ then
        // Memory into event log
        $L \leftarrow convert(M)$
        $ProcessMining(L, A)$
    end
end
```

Algorithm 1 reports a possible representation of an algorithm for process mining on a count-based window model. The algorithm uses as a memory model a FIFO queue and it starts with a never-ending loop which comprises, as the first step, the observation of a new event. After that, the memory is checked for maximum capacity and, if reached, the oldest event observed is removed. Then, the mining can take place, initially converting the memory into a process mining capable log and then running the actual mining algorithm on the given log.

Window-based models come with many advantages such as the capability of reusing any offline process mining algorithm already available for static log files. The drawback, however, comes from the inefficient handling of the memory: window-based models are not very efficient for summarizing the stream, i.e., the logs generated from a window suffer from strong biases due to the rigidity of the model.

Problem Reduction. To mitigate the issues associated with window models, one option consists of employing a problem reduction technique (cf. Fig. 2). In this case, the idea is to *reduce* the process mining problem at hand to a simpler yet well-studied problem in the general stream processing field in order to leverage existing solutions and derive new ones. An example of a very well studied problem is *frequency counting*: counting the frequencies of variables over a stream (another example of a relevant and well-studied problem is sampling). To properly reduce a process mining problem to a frequency counting one, it is important

Algorithm 2: Lossy Counting

```
   Input: $S$: data stream
          $\epsilon$: maximal approximation error
 1 $T \leftarrow \emptyset$ // Initially empty set
 2 $N \leftarrow 1$ // Number of observed events
 3 $w \leftarrow \lceil \frac{1}{\epsilon} \rceil$ // Bucket width
 4 forever do
 5     $e \leftarrow observe(S)$
 6     $b_{curr} \leftarrow \lceil \frac{N}{w} \rceil$
       // Is there a tuple in $T$ with $e$ as first component?
 7     if $e$ is already in $T$ then
 8         Increment the frequency of $e$ in $T$
 9     else
10         Insert $(e, 1, b_{curr} - 1)$ in $T$
11     end
12     if $N \mod w = 0$ then
13         forall the $(a, f, \Delta) \in T$ s.t. $f + \Delta \leq b_{curr}$ do
14             Remove $(a, f, \Delta)$ from $T$
15         end
16     end
17     $N \leftarrow N + 1$
18 end
```

to understand what a variable is in the process mining context and if it is indeed possible to extract information by counting how often a variable occurs.

An algorithm to tackle the frequency counting problem is called Lossy Counting [30], described in Algorithm 2 and graphically depicted in Fig. 3. Conceptually, the algorithm divides the stream into "buckets", each of them with a fixed size (line 6). The size of the bucket is derived from one of the inputs of the algorithm ($\epsilon \in [0, 1]$) which indicates the maximal acceptable approximation error in the counting. Lossy Counting keeps track of the counting by means of a data structure T, where each component (e, f, Δ) refers to the element e of the stream (the variable to count), its estimated frequency f, and the maximum number of times it could have occurred Δ (i.e., the maximum error). Whenever a new event is observed (line 5), if the value is already in the memory T, then the counter f is incremented by one (line 8), instead, if there is no such value, a new entry is created in T with the value e corresponding to the observed variable, frequency $f = 1$, and maximum error equal to the number of the current bucket minus one (from here it is possible to understand that since the buck size depends on the maximum allowed error, the higher the error, the larger the bucket size and hence the higher the approximation error) (line 10). With a fixed periodicity (i.e., every time a new conceptual bucket starts) the algorithm cleans the memory, by removing elements not frequently observed (lines 12–16). Please note that this specific algorithm has no memory bound: the size

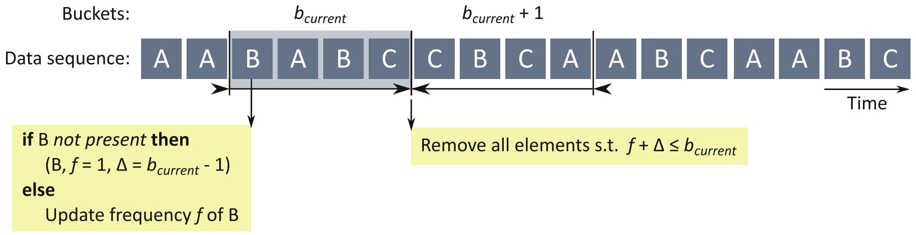

Buckets:
$b_{current}$
$b_{current} + 1$
Data sequence:
A A B A B C C B C A A B C A A B C
Time
if B *not present* then
(B, f = 1, Δ = $b_{current}$ - 1)
else
Update frequency f of B
Remove all elements s.t. $f + \Delta \leq b_{current}$

Fig. 3. Graphical representation of the Lossy Counting algorithm.

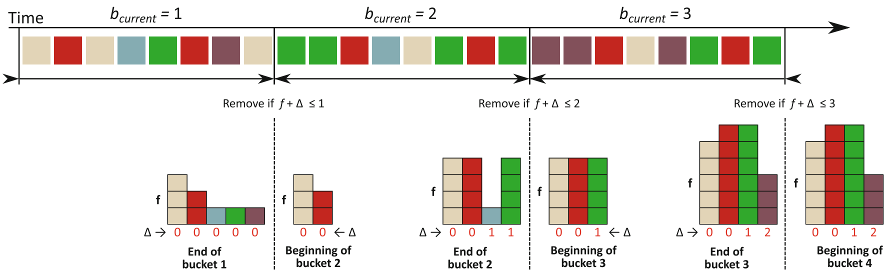


Fig. 4. Demonstration of the evolution of the internal data structure constructed by the Lossy Counting on a simple stream. Each color refers to a variable.

of its data structure T depends on the stream and on the max approximation error (i.e., if the error is set to 0 and the observations never repeat, the size of T will grow indefinitely). Variants of the algorithm enforcing a fixed memory bound is available as well [18] but are not described in detail here. In the rest of this chapter, a set whose entries are structured and updated using the Lossy Counting algorithm (i.e., T in Algorithm 2) will be called Lossy Counting Set.

Figure 4 shows a demonstration of the evolution of the Lossy Counting Sets over time (at the end and at the beginning of each virtual bucket) for the given stream. In this case, for simplicity purposes, the background color of each box represents the variable that we are counting. The counting is represented as the stacking of the blocks, and below, in red, the maximum error for each variable is ported.

The most relevant benefit of reducing the problem to a known one is the ability to employ existing as well as new solutions in order to improve the efficiency of the final process mining solution. Clearly, this desirable feature is paid back in terms of the complexity of the steps (both conceptual and computational) required for the translation.

Offline Computation. Due to some of the constraints imposed by the streaming paradigm, one option consists of moving parts of the computation offline (cf. Fig. 2) so that performance requirements are met when the system goes online. This idea implies decomposing the problem into sub-problems and reflecting on whether some of them can be solved without the actual streaming data. If that

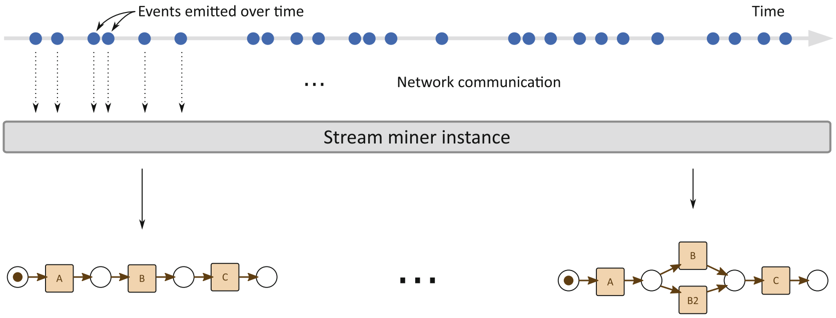


Fig. 5. Conceptualization of the streaming process discovery. Figure from [16].

is the case, these sub-problems will be solved beforehand and corresponding pre-computed solutions will be available as the events are coming in.

Such an approach comes with the advantage of caching the results of computations that would otherwise require extremely expensive computations. This approach, still, suffers from several limitations since it is not possible to apply this approach to all streaming process mining problems. Additionally, by computing everything in advance, we lose the possibility of adapting the pre-computed solutions to the actual context, which might be uniquely specific to the running process instance.

Hybrid Approaches. As a final note, we should not rule out the option of defining *ensemble* methods that combine different approaches together (see Fig. 2).

3 Streaming Process Discovery

After introducing the general principles and taxonomy of techniques to tackle streaming process mining, in this section, we will specifically analyze the problem of streaming process discovery.

A graphical conceptualization of the problem is reported in Fig. 5: the basic idea is to have a source of events that generates an event stream. Such an event stream is consumed by a miner which keeps a representation of the underlying process model updated as the new events are coming in.

3.1 State of the Art

In this section, the main milestones of the streaming process discovery will be presented. The first available approach to tackle the streaming process discovery problem is reported in [15,16]. This technique employs a "problem reduction" approach (cf. Fig. 2) rephrasing the Heuristics Miner [38] as a frequency counting problem. The details of this approach will be presented in Sect. 3.2. In [23], authors present StrProM which, similarly to the previous case, tracks the direct following relationship by keeping a prefix tree updated with Lossy Counting with Budget [18].

More recently, an architecture called S-BAR, which keeps an updated abstract representation of the stream (e.g., direct follow relationships), is used as starting point to infer an actual process model, as described in [44]. Different algorithms (including α. [39], Heuristics Miner [38] and Inductive Miner [26]) have been incorporated to be used with this approach. Also in this case authors reduced their problem to frequency counting, thus using Lossy Counting, Space Saving [32], and Frequent [24].

Declarative processes (cf. Chapter 4) have also been investigated as the target of the discovery. In [12,14,27], authors used the notion of "replayers" – one for each Declare [35] template to mine – to discover which one are fulfilled. Also in this case, Lossy Counting strategies have been employed to achieve the goal. A newer version of the approach [33], is also capable of discovering data conditions associated with the different constraints.

3.2 Heuristics Miner with Lossy Counting (HM-LC)

This section describes in more detail one algorithm for streaming process discovery: Heuristics Miner with Lossy Counting (HM-LC) [16].

The Heuristics Miner algorithm [38] is a discovery algorithm which, given the frequency of the direct following relations observed (reported as $|a > b|$ and indicating the number of times b is observed directly after a), calculates the *dependency measure*, a measure of the strength of the causal relation between two activities a and b:

$$a \Rightarrow b = \frac{|a > b| - |b > a|}{|a > b| + |b > a| + 1} \in [-1, 1]. \qquad (1)$$

The closer the value of such metric is to 1, the stronger the causal dependency from a to b. Based on a given threshold (parameter asked as input), the algorithm considers only those dependencies with values exceeding the threshold, deeming the remaining as noise. By considering all dependencies which are strong enough, it is possible to build a dependency graph, considering one node per activity and one edge for each dependency. In such a graph, however, when splits or joins are observed (i.e., activities with more than one outgoing or incoming connection) it is not possible to distinguish the type of the splits. In the case of a dependency from a to b and also from a to c, Heuristics Miner disambiguates between an AND and an XOR split by calculating the following metric (also based on the frequency of direct following relations):

$$a \Rightarrow (b \wedge c) = \frac{|b > c| + |c > b|}{|a > b| + |a > c| + 1} \in [0, 1]. \qquad (2)$$

When the value of this measure is high (i.e. close to 1), it is likely that b and c can be executed in parallel, otherwise, these will be mutually exclusive. As for the previous case, a threshold (parameter asked as input) is used to make the distinction.

It is important to note that the two fundamental measures employed by the Heuristics Miner rely on the frequency of the directly-follows measure (e.g. $|a > b|$), and so the basic idea of Heuristics Miner with Lossy Counting is to consider such values as "variables" to be observed in a stream, thus reducing the problem to frequency counting.

As previously mentioned, Lossy Counting is an algorithm for frequency counting. In particular, the estimated frequencies can be characterized both in terms of lower and upper bounds as follows: given the estimated frequency f (i.e., the frequency calculated by the algorithm), the true frequency F (i.e., the actual frequency of the observed variable), the maximum approximation error ϵ and the number of events observed N, these inequalities hold

$$f \leq F \leq f + \epsilon N.$$

To calculate the frequencies, Lossy Counting uses a set data structure where each element refers to the variable being counted, its current (and approximated) frequency, and the maximum approximation error in the counting of that variable.

For the sake of simplicity, in Heuristics Miner with Lossy Counting, the observable units of the event stream comprises just the activity name and the case id (cf. Definition 1). In other words, each event observed from the stream comprises two attributes: the activity name and the case id (cf. Definition 1 [1], Sect. 3.2 [1]), so an event e is a tuple with $e = (c, a)$, where $\#_{case}(e) = c$ and $\#_{act}(e) = a$.

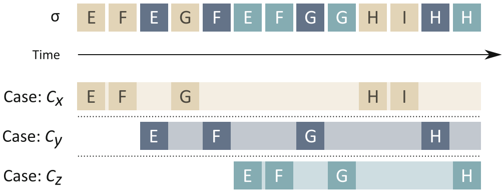


Fig. 6. Conceptualization of the need to isolate different traces based on a single stream. Boxes represent events: their background colors represent the case id, and the letters inside are the activity names. First line reports the stream, following lines are the single cases. Figure from [16].

The pseudocode of HM-LC is reported in Algorithm 3. The fundamental idea of the approach is to count the frequency of the direct following relations observed. In order to achieve this goal, however, it is necessary to *identify* the direct following pairs in the first place. As depicted in Fig. 6, to identify direct following relations it is first necessary to disentangle the different traces that are intertwined in an event stream. To this end, the HM-LC instantiates two Lossy Counting Sets: $\mathcal{D}_C$, and $\mathcal{D}_R$. The first keeps track of the latest activity observed in each process instance whereas the second counts the actual frequency of the directly follow relations. These data structures are initialized at the first line of the algorithm, which is followed by the initialization of the counter of observed events (line 2) and the calculation of the size of the buckets (line 3). After the initial setup of the data structure, a never-ending loop starts by observing events from the stream (line 5, cf. Definition 1), where each

Algorithm 3: Heuristics Miner with Lossy Counting (simplified)

```
   Input: $S$: event stream
          $\epsilon$: approximation error
 1 Initialize Lossy Counting Sets $\mathcal{D}_C$ and $\mathcal{D}_R$
 2 $N \leftarrow 1$                                  // Counter of observed events
 3 $w \leftarrow \lceil \frac{1}{\epsilon} \rceil$                                        // Bucket size
 4 forever do
 5     $(c_N, a_N) \leftarrow observe(S)$
 6     $b_{curr} = \lceil \frac{N}{w} \rceil$                  // Calculate the current bucket id
       // Step 1: Update the Lossy Counting Sets
 7     if $\exists((c, a_{last}), f, \Delta) \in \mathcal{D}_C$ such that $c = c_N$ then
 8         Remove the entry $((c, a_{\text{last}}), f, \Delta)$ from $\mathcal{D}_C$
 9         $\mathcal{D}_C \leftarrow \mathcal{D}_C \cup \{((c, a_N), f + 1, \Delta)\}$
           // Update the $\mathcal{D}_R$ data structure
10         $r_N \leftarrow (a_{\text{last}}, a_N)$ // Build relation $r_N$ as $a_{\text{last}} \rightarrow a_N$
11         if $\exists(r, f, \Delta) \in \mathcal{D}_R$ such that $r = r_N$ then
12             Remove the entry $(r, f, \Delta)$ from $\mathcal{D}_R$
13             $\mathcal{D}_R \leftarrow \mathcal{D}_R \cup \{(r, f + 1, \Delta)\}$
14         else
15             $\mathcal{D}_R \leftarrow \mathcal{D}_R \cup \{(r_N, 1, b_{curr} - 1)\}$
16         end
17     else
18         $\mathcal{D}_C \leftarrow \mathcal{D}_C \cup \{((c_N, a_N), 1, b_{curr} - 1)\}$
19     end
       // Step 2: Periodic cleanup
20     if $N \equiv 0 \mod w$ then
21         forall the $((c, a), f, \Delta) \in \mathcal{D}_C$ such that $f + \Delta \leq b_{curr}$ do
22             Remove $((c, a), f, \Delta)$ from $\mathcal{D}_C$
23         end
24         forall the $(r, f, \Delta) \in \mathcal{D}_R$ such that $f + \Delta \leq b_{curr}$ do
25             Remove $(r, f, \Delta)$ from $\mathcal{D}_R$
26         end
27     end
28     $N \leftarrow N + 1$
       // Step 3: Consumption of the data structure to update the model
29     Update the model using $\mathcal{D}_R$
30 end
```

event is the pair (c_N, a_N), indicating that the case id observed as event N is c_N (resp., the activity is a_N). The id of the current bucket is calculated right afterwards (line 6). The whole algorithm is then divided into three conceptual steps: in the first the data structures are updated; in the second periodic cleanup takes place; in the third the data structures are used to construct and update the actual model.

Step 1: Updating the Data Structure. The Lossy Counting Set $\mathcal{D}_C$ has been defined in order not only to keep a count of the frequency of each case id observed in the events but also to keep track of the latest activity observed in the given trace. To achieve this goal, the entries of the data structure are tuples themselves, comprising the case id as well as the name of the latest activity observed in the case. Therefore, the first operation within the step consists of checking for the presence of an entry in $\mathcal{D}_C$ matching the case id of the observed event (but not the activity name), as reported in line 7. If this is the case, the data structure $\mathcal{D}_C$ is updated, not only by updating the frequency but also by updating the latest activity observed in the given case (lines 8 and 9). In addition, having already an entry in $\mathcal{D}_C$ means that a previous event within the same trace has already been seen and, therefore, it is possible to construct a direct following relation (cf. line 10 of Algorithm 3). This relation is then treated as a normal variable to be counted and the corresponding Lossy Counting Set is updated accordingly (lines 11–16). In case $\mathcal{D}_C$ did not contain an entry referring to case id c_N, it means that the observed event is the first event of its process instance (up to the approximation error) and hence just a new entry in $\mathcal{D}_C$ is inserted and no direct following relation is involved (line 18).

Step 2: Periodic Cleanup. With a periodicity imposed by the maximum approximation error (ϵ), i.e., at the end of each bucket (line 20), the two Lossy Counting Sets are updated by removing entries that are not frequent or recent enough (lines 21–26). Please note that the algorithm expects that observing an event belonging to a process instance that has been removed from $\mathcal{D}_C$ corresponds to losing one direct following relation from the counting in $\mathcal{D}_R$. From this point of view, the error on the counting of the relations is not only affected by $\mathcal{D}_R$ but, indirectly, also by the removal of instances from $\mathcal{D}_C$ which causes a relation not to be seen at all (and therefore, it cannot be counted).

Step 3: Consumption of the Data Structures. The very final step of the algorithm (line 29) consists of triggering a periodic update of the model. The update procedure (not specified in the algorithm) extracts all the activities involved in a direct following relations from $\mathcal{D}_R$ and uses the dependency measure (cf. Eq. 1) to build a dependency graph, by keeping the relations with dependency measure above a threshold. To disambiguate AND/XOR splits Eq. 2 is used. Both these measures need to be adapted in order to retrieve the frequency of the relations from $\mathcal{D}_R$.

The procedure just mentioned recomputes the whole model from scratch. However, observing a new event will cause only local changes to a model. Hence a complete computation of the whole model is not necessary. In particular, it is possible to rearrange Eqs. 1 and 2 in order to signal when a dependency has changed. Specifically, given a dependency threshold τ_{dep}, we know that a dependency should be present if these inequalities hold:

$$|a > b| \geq \frac{|b > a|(1 + \tau_{dep}) + \tau_{dep}}{1 - \tau_{dep}} \quad \text{or} \quad |b > a| \leq \frac{|a > b|(1 - \tau_{dep}) - \tau_{dep}}{1 + \tau_{dep}}$$

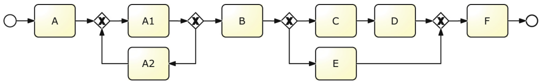


Fig. 7. Reference process model used to calculate the conformance of traces in Table 1, from [17].

Table 1. Example traces with corresponding offline conformance.

Trace	Conf.
$t_1 = \langle A, A1, B, E, F\rangle$	1.00
$t_2 = \langle A, A1, A2, A1, B\rangle$	0.80
$t_3 = \langle B, C, D, F\rangle$	0.78
$t_4 = \langle B, C, D\rangle$	0.62

In a similar fashion, we can rewrite Eq. 2 so that, given an AND threshold parameter τ_{and}, a split (i.e., from activity a to both activities b and c) has type AND if all these inequalities hold:

$$\begin{aligned}
|b > c| \leq& \tau_{and}\,(|a > b| + |a > c| + 1) - |c > b| \\
|c > b| \leq& \tau_{and}(|a > b| + |a > c| + 1) - |b > c| \\
|a > b| \leq& \frac{|b > c| + |c > b|}{\tau_{and}} - |a > c| - 1 \\
|a > c| \leq& \frac{|b > c| + |c > b|}{\tau_{and}} - |a > b| - 1
\end{aligned}$$

If this is not the case, the type of the split will be XOR. Therefore, by monitoring how the frequencies of some of the relations in $\mathcal{D}_R$ (which should be used as an approximation of the direct following frequencies) are evolving, it is possible to pinpoint the changes appearing in a model, with no need for rebuilding it from scratch all the times.

In this section, we did not exhaustively cover the reduction of the Heuristics Miner to Lossy Counting (for example, we did not consider the absolute number of observations for an activity or parameters such as the relative-to-best) but we focused on the core aspects of the reduction. The goal of the section was to present the core ideas behind a streaming process discovery algorithm while, at the same time, showing an example of an algorithm based on the problem reduction approach (cf. Fig. 2).

4 Streaming Conformance Checking

Computing the conformity of running instances starting from events observed in a stream is the main goal of streaming conformance checking.

Consider, for example, the process model reported in Fig. 7 as a reference process, and let's investigate the offline conformance (calculated according to the alignment technique reported in [2]) for the traces reported in Table 1. Trace t_1 is indeed conforming with respect to the model as it represents a possible complete execution of the process. This information is already properly captured by the offline analysis. Trace t_2, on the other hand, is compliant with the process but just up to activity B, as reported by the conformance value 0.8:

offline systems assume that the executions are complete, and therefore observing an incomplete trace represents a problem. However, as previously discussed and as shown in Fig. 1a, in online settings it could happen that parts of the executions are missing due to the fact that the execution has not yet arrived at this part of the computation. This *could* be the case with trace t_2 (i.e., t_2 is the prefix of a compliant trace). Trace t_3 suffers from the opposite problem: the execution is conforming to the model, but just from activity B onward. While offline conformance, in this case, is calculated to the value of 0.68, as for the previous case, we cannot rule out the option that the trace is actually compliant but, since the trace started before the streaming conformance checker was online, it was incapable of analyzing the beginning of it (i.e., t_3 is the suffix of a compliant trace). Trace t_4, finally, seems compliant just between activities B and D. Though offline conformance, in this case, is 0.62, as for the previous two cases, in a streaming setting, we cannot exclude that the issue actually derives from the combination of the trace starting before the streaming conformance was online and the trace not being complete (i.e.,t_4 is a subsequence of a compliant trace).

Hopefully, discussing the previous examples helped to point out the limit of calculating the extent of the conformance using only one numerical value in a streaming setting. Indeed, when the assumption that executions are complete is dropped, the behavior shown in the traces of Table 1 could become 100% compliant since the actual issue does not lie in the conformity but in the amount of observed behavior.

4.1 State of the Art

Computing the conformity of a stream with respect to a reference model has received a fairly large amount of attention, in particular in the case of declarative processes. Under the name "*operational support*", research has been focusing [28, 29] on understanding if and which constraints are violated and satisfied as new events are coming in. In particular, each constraint is associated with one of four possible truth values: permanently or temporarily violated or fulfilled which are computed by representing the behavior as an automaton with all executions replayed on top of it.

Streaming conformance checking on imperative models has also received attention, though more recently. Optimal alignments can be computed for the prefix (i.e., prefix-alignments) of the trace seen up to a given point in time [41], resulting in a very reliable approach which, however, meets only to some extent the streaming scenario (cf. Sect. 1.2). A more recent approach [36] is capable of improving the performance of calculating a prefix-alignment, by rephrasing the problem as the shortest path one and by incrementally expanding the search space and reusing previously computed intermediate results.

A different line of research focused on calculating streaming conformance for all scenarios (cf., Fig. 1a). In this case, techniques employed "offline computation" approaches [11,17,25] to construct data structures capable of simplifying the computation when the system goes online. These approaches not only compute the conformity of a running instance but also try to quantify the amount of behavior observed or still to come.

In addition to these, one of the first approaches [45] focused on a RESTful service capable of performing the token replay on a BPMN model (via a token pull mechanism). No explicit guarantees, however, are reported concerning the memory usage, the computational complexity, or the reliability of the results, suggesting that the effort was mostly on the interface type (i.e., *online* as in *RESTful*).

4.2 Conformance Checking with Behavioral Patterns

This section presents in more detail one algorithm for streaming conformance checking using behavioral patterns [17]. The algorithm belongs to the category of offline computation (cf. Fig. 2), where the heaviest computation is moved before the system goes online, thus meeting the performance requirement of streaming settings.

The fundamental idea of the approach is that using just one metric to express conformity could lead to misleading results, i.e. cases that already started and/or that are not yet finished get falsely penalized. To solve these issues, the approach proposes to break the conformity into three values:

1. *Conformance*: indicating the amount of actually correct behavior seen;
2. *Completeness*: providing an estimate of the extent to which the trace has been observed since the beginning; and
3. *Confidence*: indicating how much of the trace has been seen, and therefore to what extent the conformance is likely to remain stable.

A graphical representation of these concepts is reported in Fig. 8. In addition, the approach does not assume any specific modeling language for the reference process. Instead, the approach takes the reference process as a constraining of the relative orders of its activities. Such constraints are defined in terms of behavioral patterns, such as weak ordering, parallelism, causality, and conflict. Such behavioral patterns (with the corresponding activities involved) represent also what the conformance checking algorithm observes. In the context of this chapter, we will consider the directly follow relation as a pattern.

Please note that the input of the algorithm is not a stream of events, but a stream of observed behavioral patterns, which could require some processing of the raw events. This, however, does not represent a problem for the behavioral pattern considered (i.e., directly follow relation), since these can be extracted using the technique described in Sect. 3.2.

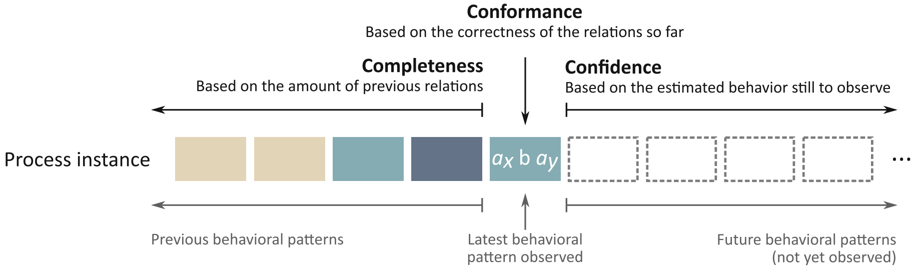

Conformance
Based on the correctness of the relations so far
Completeness
Based on the amount of previous relations
Confidence
Based on the estimated behavior still to observe
Process instance
a_x b a_y
...
Previous behavioral patterns
Latest behavioral pattern observed
Future behavioral patterns (not yet observed)

Fig. 8. General idea of the 3 conformance measures computed based on a partially observed process instance: *conformance*, *completeness*, and *confidence*. Figure from [17].

As previously mentioned, the technique offloads the computation to a preprocessing stage which takes place offline, before the actual conformance is computed. During such a step, the model is converted into another representation, better suited for the online phase. Specifically, the new model contains:

1. The set of behavioral patterns that the original process prescribes;
2. For each of the behavioral patterns identified, the minimum and maximum number of distinct prescribed patterns that must occur *before* it, since the very beginning of the trace;
3. For each behavioral pattern, the minimum number of distinct patterns *still to observe* to reach a valid accepting state of the process (as prescribed by the reference model).

These requirements drive the definition of the formal representation called "Process Model for Online Conformance" (PMOC). A *process model for online conformance* $M = (B, P, F)$ is defined as a triplet containing the set of prescribed behavioural patterns B. Each pattern $b(a_1, a_2)$ is defined as a relation b (e.g., the directly follow relation) between activities $a_1, a_2 \in \mathcal{U}_{act}$ (cf. Definition 1 [1]). P contains, for each behavioral pattern $b \in B$, the pair of minimum and maximum number distinct prescribed patterns (i.e., B) to be seen before b. We refer to these values as $P_{\min}(b)$ and $P_{\max}(b)$. For each pattern, $b \in B$, $F(b)$ refers to the minimum number of distinct patterns (i.e., B) required to reach the end of the process from b.

Once such a model is available, the conformance values can be calculated according to Algorithm 4 which executes three steps for each event: updating the data structures, calculating the conformance values, and housekeeping cleanup. After two maps are initialized (lines 1, 2), the never-ending loop starts and, each observation from the stream (which refers to a behavioral patter b for case id c, cf. Definition 1) triggers and update of the two maps: if the pattern refers to a prescribed relation, then it is added to the $\mathsf{obs}(c)$ set (line 6)[4], otherwise, the value of incorrect observations for the process instance $\mathsf{obs}(c)$ is incremented (line 8)[5]. In the second step, the algorithm calculates the new conformance values.

[4] If obs has no key c, $\mathsf{obs}(c)$ returns the empty set.

[5] If inc has no key c, then $\mathsf{inc}(c)$ returns 0.

Algorithm 4: Conformance Checking with Behavioral Patterns

```
   Input: $S$: stream of behavioural patterns
          $M = (B, P, F)$: process model for online conformance
1  Init map obs          // Maps case ids to set of observed patterns from $M$
2  Init map inc                                   // Maps case ids to integers
3  forever do
4      $(c, b) \leftarrow observe(S)$        // New observation of pattern $b$ for case $c$
       // Step 1: update internal data structures
5      if $b \in B$ then
6          $\mathsf{obs}(c) \leftarrow \mathsf{obs}(c) \cup \{b\}$     // If $b$ already in $\mathsf{obs}(c)$, then no effect
7      else
8          $\mathsf{inc}(c) \leftarrow \mathsf{inc}(c) + 1$
9      end
       // Step 2: compute online conformance values
10     $\mathsf{conformance}(c) \leftarrow \frac{|\mathsf{obs}(c)|}{|\mathsf{obs}(c)| + \mathsf{inc}(c)}$
11     Notify new value of $\mathsf{conformance}(c)$
12     if $b \in B$ then
13         if $P_{\min}(b) \leq |obs(c)| \leq P_{\max}(b)$ then
14             $\mathsf{completeness}(c) \leftarrow 1$
15         else
16             $\mathsf{completeness}(c) \leftarrow \min \left\{ 1, \frac{|\mathsf{obs}(c)|}{P_{\min}(b) + 1} \right\}$
17         end
18         $\mathsf{confidence}(c) \leftarrow 1 - \frac{F(b)}{\max_{b' \in B} F(b')}$
19         Notify new values of $\mathsf{completeness}(c)$ and $\mathsf{confidence}(c)$
20     end
       // Step 3: cleanup
21     if size of obs and inc is close to max capacity then
22         Remove oldest entries from obs and inc
23     end
24 end
```

The actual conformance, which resembles the concept of *precision*, is calculated (lines 10, 11) as the number of distinct observed prescribed patterns in c (i.e., $|\mathsf{obs}(c)|$) divided by the sum of the number of prescribed observed patterns and the incorrect patterns (i.e., $|\mathsf{obs}(c)| + \mathsf{inc}(c)$): 1 indicates full conformance (i.e., only correct behaviour) and 0 indicates no conformance at all (i.e., only incorrect behaviour). Completeness and confidence are updated only when a prescribed behavioral pattern is observed (line 12) since they require locating the pattern itself in the process. Concerning completeness, we have perfect value if the number of distinct behavioral patterns observed so far is within the expected interval for the current pattern (lines 13, 14). If this is not the case, we might have seen fewer or more patterns than expected. If we have seen fewer patterns, the completeness is the ratio of observed patterns over the minimum expected; otherwise, it's just 1 (i.e., we observed more patterns than needed, so the completeness is not an issue). Please bear in mind that these numbers confront the *number* of distinct patterns, not their type, thus potentially leading to false positives (line 16). The confidence is calculated (line 18) as 1 minus the proportion of patterns to observe (i.e., $F(b)$) and the overall maximum number of future patterns (i.e., $\max_{b' \in B} F(b')$): a confidence level 1 indicates strong confidence (i.e., the execution reached the end of the process), 0 means low confidence (i.e., the execution is still far from completion, therefore there is room for change). The final step performs some cleanup operations on obs and inc (lines 21–23). The algorithm does not specify how old entries should be identified and removed, but, as seen on the previous section, existing approaches can easily handle this problem (e.g., by using a Lossy Counting Set).

It is important to note once again that the actual algorithm relies on a data structure (the PMOC) that is tailored to the purpose and that might be computational very expensive to obtain. However, since this operation is done only once and before any streaming processing, this represents a viable solution. The details on the construction of the PMOC are not analyzed in detail here but are available in [17]. Briefly, considering the directly follow relation as the behavioral pattern, the idea is to start from a Petri net and calculate its reverse (i.e., the Petri net where all edges have opposite directions). Both these models are then unfolded according to a specific stop criterion and, once corresponding reachability graphs are computed, the PMOC can be easily derived from the reachability graph of the unfolded original Petri net and the reachability graph of the unfolded reverse net.

Considering again the traces reported in Table 1 and the reference model in Fig. 7, all traces have a streaming conformance value of 1 (when calculated using the approach just described). The completeness is 1 for t_1 and t_2, 0.6 for t_3, and 0.5 for t_4. The confidence is 1 for t_1 and t_3, 0.5 for t_2, and 0.75 for t_4. These values indeed capture the goals mentioned at the beginning of this section: do not penalize the conformance but highlight the actual issues concerning the missing beginning or end of the trace.

As for the streaming process discovery case, in this section, we did not exhaustively cover the algorithm for streaming conformance checking presented. Instead, we focused on the most important aspects of the approach, hopefully also giving an intuition of how an offline computation approach could work (cf. Fig. 2).

5 Other Applications and Outlook

It is worth mentioning that the concepts related to streaming process mining have been applied not only to the problem of discovery and conformance but, to a limited extent, to other challenges.

Examples of such applications are the discovery of cooperative structures out of event streams, as tackled in [43], where authors process an event stream and update the set of relationships of a cooperative resource network. In [40], several additional aspects of online process mining are investigated too.

Supporting the research in streaming process mining has also been a topic of research. Simulation techniques have been defined both as standalone applications [9], as ProM plugins [42], or just as communication protocols [13].

Finally, from the industrial point of view, it might be interesting to observe that while some companies are starting to consider some aspects related to the topics discussed in this chapter (e.g., Celonis' Execution Management Platform supports the real-time data ingestion, though not the analysis), none of them offers actual solutions for streaming process mining. A report from Everest Group[6] explicitly refers to real-time monitoring of processes as an important process intelligence capability not yet commercially available.

This chapter presented the topic of streaming process mining. While the field is relatively young, several techniques are already available both for discovery and conformance checking.

We presented a taxonomy of the existing approaches which, hopefully, can be used proactively, when new algorithms need to be constructed, to identify how a problem can be tackled. Then two approaches, one for control-flow discovery and one for conformance checking, are presented in detail which, in addition, belong to different categories of the taxonomy. Alongside these two approaches, window models can also be employed, yet their efficacy is typically extremely low compared to algorithms specifically designed for the streaming context.

It is important to mention that streaming process mining has very important challenges still to be solved. For example, dealing with a stream where the arrival time of events does not coincide with their actual execution. In this case, it would be necessary to reorder the list of events belonging to the same process instance before processing them. Another relevant issue might be the inference of the termination of process instances. Finally, so far, we always considered an insert-only stream model, where events can only be added in a monotonic fashion. Scenarios where observed events can be changed or removed (i.e., insert-delete models) are yet to be considered.

References

1. van der Aalst, W.M.P.: Chapter 1 - Process mining: a 360 degrees overview. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. Lecture Notes in Business Information Processing, pp. ??-??, vol. 448. Springer-Verlag, Berlin (2022)
2. Adriansyah, A.: Aligning observed and modeled behavior. Ph.D. thesis, Technische Universiteit Eindhoven (2014)
3. Aggarwal, C.C.: Data Streams: Models and Algorithms. Advances in Database Systems. Springer, Boston (2007). https://doi.org/10.1007/978-0-387-47534-9
4. Berry, R.F., McKenney, P.E., Parr, F.N.: Responsive systems: an introduction. IBM Syst. J. **47**(2), 197–206 (2008)

[6] https://www2.everestgrp.com/reportaction/EGR-2020-38-R-3808/Marketing.

5. Beverungen, D., et al.: Seven paradoxes of business process management in a hyperconnected world. Bus. Inf. Syst. Eng. **63**(2), 145–156 (2021)
6. Bifet, A., Gavaldà, R., Holmes, G., Pfahringer, B.: Machine Learning for Data Streams. The MIT Press, Cambridge (2018)
7. Bifet, A., Kirkby, R.: Data stream mining: a practical approach. Technical report, Centre for Open Software Innovation - The University of Waikato (2009)
8. Burattin, A.: Process Mining Techniques in Business Environments. Lecture Notes in Business Information Processing, vol. 207. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-17482-2
9. Burattin, A.: PLG2: multiperspective process randomization with online and offline simulations. In: Online Proceedings of the BPM Demo Track (2016). CEUR-WS.org
10. Burattin, A.: Streaming process discovery and conformance checking. In: Sakr, S., Zomaya, A., (eds.) Encyclopedia of Big Data Technologies. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-63962-8_103-1
11. Burattin, A., Carmona, J.: A framework for online conformance checking. In: Teniente, E., Weidlich, M. (eds.) BPM 2017. LNBIP, vol. 308, pp. 165–177. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74030-0_12
12. Burattin, A., Cimitile, M., Maggi, F.M., Sperduti, A.: Online discovery of declarative process models from event streams. IEEE Trans. Serv. Comput. **8**(6), 833–846 (2015)
13. Burattin, A., Eigenmann, M., Seiger, R., Weber, B.: MQTT-XES: real-time telemetry for process event data. In: CEUR Workshop Proceedings (2020)
14. Burattin, A., Maggi, F.M., Cimitile, M.: Lights, camera, action! Business process movies for online process discovery. In: Proceedings of the 3rd International Workshop on Theory and Applications of Process Visualization (TAProViz 2014) (2014)
15. Burattin, A., Sperduti, A., van der Aalst, W.M.P.: Heuristics Miners for Streaming Event Data. ArXiv CoRR, December 2012
16. Burattin, A., Sperduti, A., van der Aalst, W.M.P.: Control-flow discovery from event streams. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 2420–2427. IEEE (2014)
17. Burattin, A., van Zelst, S.J., Armas-Cervantes, A., van Dongen, B.F., Carmona, J.: Online conformance checking using behavioural patterns. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 250–267. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_15
18. Da San Martino, G., Navarin, N., Sperduti, A.: A lossy counting based approach for learning on streams of graphs on a budget. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, pp. 1294–1301. AAAI Press (2012)
19. Dayarathna, M., Perera, S.: Recent advancements in event processing. ACM Comput. Surv. **51**(2), 1–36 (2018)
20. Gaber, M.M., Zaslavsky, A., Krishnaswamy, S.: Mining data streams: a review. ACM SIGMOD Rec. **34**(2), 18–26 (2005)
21. Gama, J.: Knowledge Discovery from Data Streams. Chapman and Hall/CRC, London (2010)
22. Gandomi, A., Haider, M.: Beyond the hype: big data concepts, methods, and analytics. Int. J. Inf. Manage. **35**(2), 137–144 (2015)
23. Hassani, M., Siccha, S., Richter, F., Seidl, T.: Efficient process discovery from event streams using sequential pattern mining. In: 2015 IEEE Symposium Series on Computational Intelligence, pp. 1366–1373 (2015)

24. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A simple algorithm for finding frequent elements in streams and bags. ACM Trans. Database Syst. **28**(1), 51–55 (2003)
25. Jonathan Lee, W.L., Burattin, A., Munoz-Gama, J., Sepúlveda, M.: Orientation and conformance: a HMM-based approach to online conformance checking. Inf. Syst. **102**, 1–38 (2020)
26. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_17
27. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: A knowledge-based integrated approach for discovering and repairing declare maps. In: Salinesi, C., Norrie, M.C., Pastor, Ó. (eds.) CAiSE 2013. LNCS, vol. 7908, pp. 433–448. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38709-8_28
28. Maggi, F.M., Montali, M., van der Aalst, W.M.P.: An operational decision support framework for monitoring business constraints. In: Proceedings of 15th International Conference on Fundamental Approaches to Software Engineering (FASE), pp. 146–162 (2012)
29. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: an approach based on colored automata. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 132–147. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23059-2_13
30. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: Proceedings of International Conference on Very Large Data Bases, pp. 346–357. Morgan Kaufmann, Hong Kong, China (2002)
31. Mans, R., van der Aalst, W.M.P., Vanwersch, R.J.B.: Process Mining in Healthcare. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16071-9
32. Metwally, A., Agrawal, D., El Abbadi, A.: Efficient computation of frequent and top-k elements in data streams. In: Eiter, T., Libkin, L. (eds.) ICDT 2005. LNCS, vol. 3363, pp. 398–412. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30570-5_27
33. Navarin, N., Cambiaso, M., Burattin, A., Maggi, F.M., Oneto, L., Sperduti, A.: Towards online discovery of data-aware declarative process models from event streams. In: Proceedings of the International Joint Conference on Neural Networks (2020)
34. Patroumpas, K., Sellis, T.: Window specification over data streams. In: Proceedings of Current Trends in Database Technology - EDBT, pp. 445–464 (2006)
35. Pešić, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: Proceedings of EDOC, pp. 287–298. IEEE (2007)
36. Schuster, D., van Zelst, S.J.: Online process monitoring using incremental state-space expansion: an exact algorithm. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNCS, vol. 12168, pp. 147–164. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58666-9_9
37. Sharp, A.M.: Incremental Algorithms: Solving Problems in a Changing World. Ph.D. thesis, Cornell University (2007)
38. van der Aalst, W.M.P., Ton, A.J., Weijters, M.M.: Rediscovering workflow models from event-based data using little thumb. Integr. Comput. Aid. Eng. **10**(2), 151–162 (2003)

39. van der Aalst, W.M.P., Ton, A.J., Weijters, M.M., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**, 1128–1142 (2004)
40. van Zelst, S.J.: Process mining with streaming data. Ph.D. thesis, Technische Universiteit Eindhoven (2019)
41. van Zelst, S.J., Bolt, A., Hassani, M., van Dongen, B., van der Aalst, W.M.P.: Online conformance checking: relating event streams to process models using prefix-alignments. Int. J. Data Sci. Anal. **8**, 269–284 (2017)
42. van Zelst, S.J., van Dongen, B., van der Aalst, W.M.P.: Know what you stream: generating event streams from CPN models in ProM 6. In: CEUR Workshop Proceedings, pp. 85–89 (2015)
43. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Online discovery of cooperative structures in business processes. In: Debruyne, C., et al. (eds.) OTM 2016. LNCS, vol. 10033, pp. 210–228. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48472-3_12
44. van Zelst, S.J., van Dongen, B., van der Aalst, W.M.P.: Event stream-based process discovery using abstract representations. Knowl. Inf. Syst. **54**, 1–29 (2018)
45. Weber, I., Rogge-Solti, A., Li, C., Mendling, J.: CCaaS: online conformance checking as a service. In: Proceedings of the BPM Demo Session 2015, vol. 1418, pp. 45–49 (2015)
46. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Mach. Learn. **23**(1), 69–101 (1996)

Responsible Process Mining

Felix Mannhardt(✉)

Eindhoven University of Technology, Eindhoven, The Netherlands
f.mannhardt@tue.nl

Abstract. The prospect of data misuse negatively affecting our life has lead to the concept of responsible data science. It advocates for responsibility to be built, by design, into data management, data analysis, and algorithmic decision making techniques such that it is made difficult or even impossible to intentionally or unintentionally cause harm. Process mining techniques are no exception to this and may be misused and lead to harm. Decisions based on process mining may lead to *unfair* decisions causing harm to people by amplifying the biases encoded in the data by disregarding infrequently observed or minority cases. Insights obtained may lead to *inaccurate* conclusions due to failing to considering the quality of the input event data. *Confidential* or personal information on process stakeholders may be leaked as the precise work behavior of an employee can be revealed. Process mining models are usually white-box but may still be difficult to interpret correctly without expert knowledge hampering the *transparency* of the analysis. This chapter structures the topic of responsible process mining based on the FACT criteria: Fairness, Accuracy, Confidentiality, and Transparency. For each criteria challenges specific to process mining are provided and the current state of the art is briefly summarized.

Keywords: Fairness · Accuracy · Confidentiality · Transparency

1 Introduction

Data-based decisions affect our society and our daily life. Organizations leverage data to obtain *objective insights* that are based on *facts* rather than on guesswork. Being data-driven to guide decisions is in itself hardly new and, certainly, decisions should be based on data rather than being based on arbitrary factors. In fact, the scientific method itself is based on meticulously analysing data to derive trustworthy conclusions.

What changed in recent years, and is increasingly changing every aspect of our life, is the abundance of data and compute power available to most people and organizations. The capability of collecting and analysing a large amount of data is now within the reach for most organization. What used to be a costly and time consuming operation involving a great degree of planning what data to be collected and what methods to build, can now be done ad-hoc on large amounts of *stockpiled* data.

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 373–401, 2022.
https://doi.org/10.1007/978-3-031-08848-3_12

This abundance of data together with the emergence of a wide variety of analysis techniques has led to the formation of the *data science* field. Data science technique are not limited to giving decision support to human decision makers but increasingly *Artificial Intelligence* (AI) is used to automate decisions based on predictive models. *Process mining* is a *data science* method that focuses on improving an organization's processes by leveraging event logs. The core of event logs are timestamped data about all kinds of events that occur in the context of work or business processes [1]. Process mining techniques have been very successfully deployed in numerous organizations and have helped to remove inefficiencies and improve the quality of processes [2].

However, this increased use of data leads to an increased risk of creating negative effects from its usage by accidental or intentional *irresponsible usage* of data [3]. Irresponsible usage of data ranges from invading the privacy of individuals over flawed analysis of data with poor quality or inappropriate methods to unfair automated decisions of systems trained on data biased towards majority groups. The potential misuse of this power gives rise to calls for the *responsible* use of data by creating knowledge and awareness about possible negative consequences and researching technical and socio-technical solutions to prevent these negative consequences.

1.1 Responsible Data Science and AI

Many initiatives have called for research and development on methods that can be broadly categorized under the umbrella term *responsible data science* under which sub themes such as *responsible AI* [4] are included. Depending on the individual perspective different criteria or principles that are relevant to obtain *responsible* methods have been proposed.

- Aalst et al. and the Responsible Data Science consortium[1] call for methods that follow the FACT criteria, which stands for *Fairness*, *Accuracy*, *Confidentiality*, *Transparency* [5].
- The ACM FAccT Conference[2] calls for research on *Fairness*, *Accountability*, and *Transparency* principles.
- In Information Retrieval, the FACTS-IR critera include Fairness, Accountability, Confidentiality, Transparency, and also Safety [6].
- Dignum advoates that systems should be designed to follow the principles of *Accountability*, *Responsibility*, and *Transparency* (ART) [4].
- The European Commission provided Ethics Guidelines for Trustworthy Artificial Intelligence[3] mentioning principles such as *Human agency*, *Technical Robustness*, *Privacy*, *Transparency*, *Fairness*, and *Accountability*.

Several other organizations developing or using AI technology have published manifestos or best practices also include similar principles such as *fairness*, *privacy* or *confidentiality*, *accountability*, as well as often also *interpretability* and

[1] https://redasci.org.

[2] https://facctconference.org.

[3] https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai.

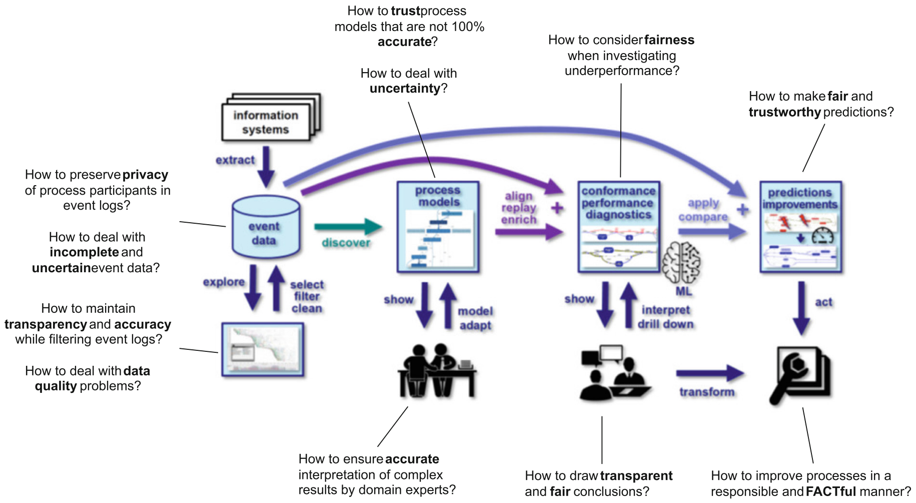

How to **trust** process models that are not 100% **accurate**?
How to deal with **uncertainty**?
How to consider **fairness** when investigating underperformance?
How to make **fair** and **trustworthy** predictions?
information systems
extract
How to preserve **privacy** of process participants in event logs?
How to deal with **incomplete** and **uncertain** event data?
event data
discover
process models
align replay enrich
conformance performance diagnostics
apply compare
predictions improvements
explore
select filter clean
show
model adapt
show
interpret drill down
ML
act
How to maintain **transparency** and **accuracy** while filtering event logs?
How to deal with **data quality** problems?
transform
How to ensure **accurate** interpretation of complex results by domain experts?
How to draw **transparent** and **fair** conclusions?
How to improve processes in a responsible and **FACTful** manner?

Fig. 1. Example challenges for responsible process mining in context of the 360 degree overview on process mining [7]

safety. Whereas originating from different perspectives and following slightly different definitions, there is great overlap on the major principles that are deemed relevant for leveraging data in a responsible manner. Naturally, the importance of the criteria differs depending on the application area. Considering *fairness* is crucial when designing AI systems based on machine learning that may possibly discriminate against individuals, whereas *safety* would be important when using such a system for controlling an industrial process. At the core of these "calls for action" is the realization that methods from the standard tool set of data science rarely follow all the desired criteria or principles by themselves. Additional effort is required, either by the analyst or system designer, to ensure their responsible use. This often requires ethical considerations since perfect technological solutions commonly do not exist.

1.2 Responsible Process Mining

This chapter instantiates the responsible data science challenges for process mining and summarises the state-of-the-art research on *responsible process mining*. Some of the challenges are specific to process mining and the event log data format whereas others are comparable to any other data science or AI approach. The context in which process mining operates means that many of the responsible data science principles and challenges are highly relevant. Figure 1 provides a non-comprehensive overview on some of the major challenges for responsible process mining in the context of the different process mining tasks. We discuss and, at least partially, answer some of these questions.

The subject of investigation in process mining is a business process, e.g., the handling of loan applications. So, the process mining analysis is not directly

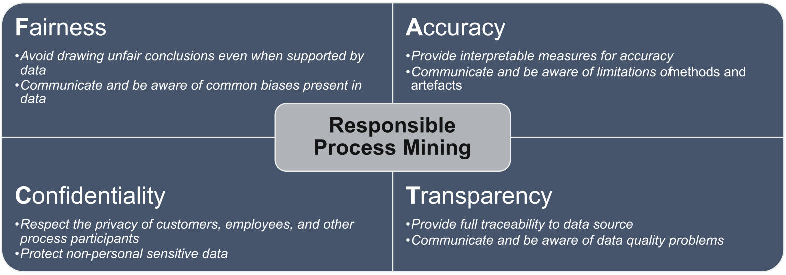

Fairness
•Avoid drawing unfair conclusions even when supported by data
•Communicate and be aware of common biases present in data
Accuracy
•Provide interpretable measures for accuracy
•Communicate and be aware of limitations ofmethods and artefacts
Responsible Process Mining
Confidentiality
•Respect the privacy of customers, employees, and other process participants
•Protect non-personal sensitive data
Transparency
•Provide full traceability to data source
•Communicate and be aware of data quality problems

Fig. 2. FACT principles for responsible process mining adapted from [5]

focused on individuals. Rather it looks at the manner in which the work is organized and performed. When analysing the loan application process, event logs are commonly not used for deciding the outcome of the loan application but for deciding how to improve the handling of applications to create a better process. Here, better may refer to being more efficient, less costly, more transparent, or any other indicator of process performance. At first glance process mining seems to not have the same impact on individuals as, e.g., deploying face recognition, predictive policing, or automatically scoring applicants for a job using AI methods. However, the manner in which business processes are performed can have an effect on various stakeholders (customers, employees, etc.).

As any other data science method, process mining relies on data to reconstruct how processes were performed and how process can be improved. Thus, the results are highly dependant on the quality of the used event data and the possible biases contained. Some additional quality and *confidentiality* challenges arise from the required sequential ordering of events, grouping of events to a specific process cases, and events being related to activities. In principle, process mining aims to discover *human-interpretable* models that are supposed to be accurate and transparent. However, for *complex* process behaviour process mining techniques often attempt to generalise from incomplete and noisy data. This creates *accuracy* and *transparency* challenges even in the process mining setting.

We follow the definitions of the FACT principles brought forward in [3,5] and illustrated in Fig. 2 to structure the discussion of process mining related challenges. First, we discuss *fairness* and its relevance to process mining in Sect. 2. Then, in Sect. 3, we briefly illustrate aspects of *accuracy* including data quality and model quality. Section 4 is a major part of this chapter and is devoted to *confidentiality*, which is about protecting and respecting sensitive data in event logs including the privacy of individuals. We close the chapter in Sect. 5 with a look at *transparency* focusing on generalization and the interpretability of process mining results.

2 Fairness

Algorithmic fairness or fairness of automated systems [8] has been an increasingly prominent topic [9] when it comes to the development and usage of AI systems that are based on black-box machine learning models. Statistical biases embedded in training data may lead to systems making unfair decisions or clearly discriminating against certain groups of people. Prominent examples of such bias are the COMPAS system for predicting the risk of criminals to re-offend, which seem exhibit racial bias by having a higher false positive rate among blacks[4], or gender stereotypes exhibited by automated translation systems such as Google Translate, which applies male gender when translating typically male dominated job names from gender neutral Turkish to English [10]. There are many more examples and we refer to the first chapter of the Fair ML book [10] for a comprehensive introduction.

An important realization regarding bias in data and their usage in any kind of data-based system is that: “Data and data sets are not objective; they are creations of human design” [11]. Data may be incomplete for a certain context leading to *representation bias* that is reflected in the learned model or the data analysis. Even when not being incomplete, data can reinforce existing discrimination that is embodied in the available data (*historical bias*). This cannot be avoided by simply discarding “problematic” attributes from the datasets since bias may be hidden in highly correlated attributes [10]. Many more data biases can be defined depending on the context [9], a notable one being Simpson’s Paradox which describe the situation that a statistic may be very different or even opposite for subgroups of a dataset compared to the statistic on the aggregate entire dataset including all those subgroups.

2.1 Process Mining Perspective

It seems that the discussion on algorithmic fairness is not directly relevant to process mining. The impact of process mining on individuals is usually indirect, so direct discrimination by a process mining analysis seems unlikely to occur. However, the potential reach of decision made based on process mining may have impacts on individuals. Employees working in an analysed process may be subject to unfair decision, customers may be rejected based on predictive process mining techniques, or processes may be redesigned in a way that is discriminating minorities. These are unfair results that are hidden behind the scenes and may not make headlines in the newspaper, unless discovered. Based on the process illustrated in Fig. 3, we give two examples on how fairness challenges can be part of a process mining project.

Automated decision making can be part of process mining as it may result in redesigned processes with changed decision making. As shown in Fig. 3 additional extensive checks may be added to a loan application process for certain

[4] https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

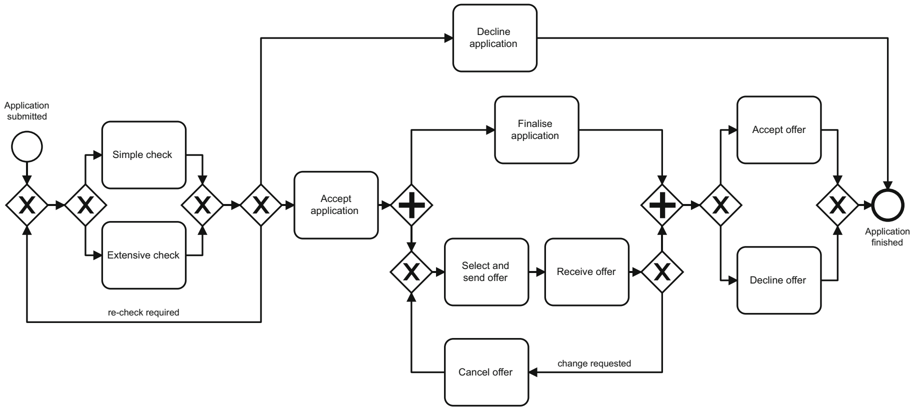


Fig. 3. Loan application process in BPMN adapted from the process used in [12]. Additional activities that indicate the kind of checks performed on the loan application before considering it have been added. Based on some criteria either a simple check or a more extensive check of the application is performed and in some cases the check is repeated.

cases leading to *fairness* challenges. This process re-design may be the outcome of a process mining analysis with the goal to minimize the cost of background checks. To further minimize the cost, methods for predictive process mining, action-oriented process mining, or the integration with robotic process automation is used to make the decision whether additional or extensive checks are necessary. Thus, process mining has directly affected the outcome of some process cases. Whereas the final decision is still made by a human, some applicants need to endure much more extensive background checks. This decision is based on machine learning techniques and, thus, inherits all the fairness issue associated with algorithmic decision making.

A second example of a *fairness* challenge that may arise in a process mining context would be affecting the employees working in the process. For example, it may be detected that when certain workers are involved in the processing of the loan application the throughput time is much longer. However, care must be taken not to draw unfair conclusions as those workers may simply handle more difficult cases [3], which leads to biased event data. If the nature of the loan application request is not included in the event log, e.g., due to confidentiality concerns, these confounding factors are difficult to detect and require careful human interpretation.

Besides the obvious ethical concerns that make it relevant to investigate fairness in the context of process mining, there are also upcoming regulations such as the EU Artificial Intelligence Act [13] that may constitute legal threats to consider fairness in any kind of automated data analysis. In the remainder of Sect. 2, we summarise the relevance of fairness for process mining along the main definitions that attempt to formalise fairness for algorithms. For each of the definitions, we instantiate them in the context of process mining and summarise existing work if available.

2.2 Algorithmic Discrimination

In the literature on algorithmic fairness, several types of discrimination that can arise from unfair algorithms have been defined. Similarly, a wide variety of definitions on how fair algorithmic systems can be designed have been researched. We sketch the main fairness definitions and discrimination's in the light of how they are relevant to the different process mining tasks as illustrated in Fig. 1.

Many possible types of discrimination are possible. It is important to realize that discrimination or unfairness does not always need to be caused by direct discrimination [9]. Direct discrimination would be a decision that is solely based on a sensitive or protected attribute a decision is made that negatively affects them. For example, if a predictive process monitoring would be trained on a somehow biased dataset and learn that female applicants for a loan should always received an extra background check causing a worse service quality or an increase rate of rejection. Clearly, this type of discrimination would be easily detected and mitigated. However, often discrimination can be *indirect discrimination* or *statistical discrimination* [9]. In these cases, some negative effect is applied but it is not directly based on a sensitive attribute. Rather the attribute or some statistical distribution is strongly correlated to a sensitive attribute. For example, when analysing the performance of a process with process mining methods one may identify a group of workers as being slower than other as they are assigned more difficult cases [3] or receive less support than others. Similarly, when improving a process design, one may focus on the 80% most frequent variants and, thereby, discriminate against minority groups with special needs that trigger infrequent activities and, thus, are often not visible in the standard process mining visualization.

2.3 Algorithmic Fairness

To counter and detect discrimination, there are attempts to formalize the notion of fairness of an algorithmic decision based on data. Again, there are many definitions that formalize different kinds of fairness that can be provided by algorithms [14,15]. It is important to realize that none of them is universally applicable and that it depends on the context which one is suitable. Often fairness definitions are introduced on the example of a simple binary classification task. The four main types of fairness notions based on [15] are: (1) based solely on the predicted outcome, (2) based on the predicted outcome and comparing it with the actual outcome (ground truth), (3) taking additionally into account the probability of the predictions, (4) notions based on similarity of the non-sensitive attributes, and (5) notions based on causal reasoning.

We introduce a few selected of these notions in the context of process mining and assume a simple binary classification model with the protected attribute *gender* for concise presentation as done in [15].

- Group fairness or statistical parity is of type (1) and satisfied when subjects from the protected group, e.g., females, have equal probability of being assigned a positive outcome. However, this notion is only applicable if there is no other, unprotected, attribute that would justify a difference in probability to be assigned a positive outcome.
- Predictive parity is of type (2) and satisfied when the precision or positive predictive value of the classifier is equal for both groups. The fraction of females and males that are predicted to be in the positive group from those that are in the positive group in the ground truth is the same. So, there are equal chances for a positive prediction for those that are in the positive group in the training data. Thus, the definition only works if there are really similar probabilities to be in the positive class.
- Treatment equality is of type (2) and satisfied when both groups have the same ratio of false negative and false positives. This allows to compare if the number of misclassifications (either positive or negative) is different between the groups.
- Fairness through unawareness is of type (4) and satisfied if the sensitive or protected attribute is removed from the dataset. Clearly, this will not resolve the issue of other correlated attributes.
- Fairness through awareness is of type (4) and uses a distance metric between individuals and compares the distance of the outcome with the distance between individuals. However, how to define that distance measure if not always easy.
- Counterfactual fairness is based on causal reasoning, thus, requires the definition of a causal graph instead of any binary classifier. Here the definition is satisfied if the predicted outcome does not depend on the protected attribute in the causal graph [15]. However, building causal graphs generally requires domain knowledge.

Being only a small selection of possible fairness notions, we refer to [15] for a comprehensive overview. None of the provided definitions is universally accepted and provides fairness in every sense of the concept.

The only work so far that directly addresses the challenge of fairness from a process mining viewpoint is written by Qafari et al. [16]. Here the problem of creating a fair classifier for data extracted from an event log that is enriched with process performance information is investigated. The approach firstly advocates to exclude the sensitive attribute or feature from building the classifier and then builds a C4.5 decision tree based on a discrimination-aware decision tree learning method. As fairness decision *predictive parity* is employed. An interesting problem is raised that *relabeling* may not always be desirable, in which case the fairness guarantees cannot be achieved. This is left as future work.

Though not explicitly addressing fairness, several proposals for applying causal machine learning techniques in the context of process mining have been made. For example, Bozorgi et al. [17,18] looked at discovering causal rules from event logs as well as taking some form of cost into account when making suggestions for intervention in running cases as part of a prescriptive process mining approach. By making the causalities explicit its may be feasible to include fairness constraints into decisions.

2.4 Open Challenges

Many open research challenges for considering fairness in process mining exists. So far, there is hardly any research on fairness that is specific to process mining neither from a technological nor from an organizational perspective, with the notable exception of [16]. A clear research challenge is to develop specific notions for fairness in process mining from the more generic fairness definitions. Whereas one could take the stance that the existing definitions from the wider machine learning field are sufficient, we motivated the need to consider fairness explicitly also regarding process mining techniques.

3 Accuracy

Models need to be *accurate* to be useful in the real world. An analyst relying on a statistical analysis or an engineer developing a machine learning model for classification needs to have confidence that the analysis or the model captures the real-world phenomenon correctly. Differently to a model based on, e.g., physical laws or logic that can be shown to be correct in any application setting the kind of statistical models often used in data science can rarely be proven to be correct. Thus, the level of accuracy with which a real-world phenomenon is captured or the level of confidence that a user can have when using that model are important aspects of any such model. The accuracy of models depends on many factors and it is often not straightforward to measure it properly. A classification model may, on average, be classifying near perfectly between pictures showing different breeds of dogs on an independent test set but if the relevant breed is highly underrepresented the classifier may still be unusable in the real world due to the class imbalance. It may also be that the classifier provides very good accuracy but makes its decision based on the wrong features picking up on spurious correlations introduced when preparing the training data: a data quality problem.

3.1 Process Mining Perspective

Understanding and being able to measure the *accuracy* of a process mining analysis is an integral part of responsible process mining. Whereas it may seem obvious to only use results that accurately reflect the process reality, this is frequently impaired in practice by the need to abstract from that reality.

Process discovery techniques are often unable to create the perfectly accurate model but are forced to balance between several quality dimensions [1] that are competing with each other. For example, to obtain a process model that is understandable by a human analyst, some observed behavior may need to be omitted. In some cases, the process behavior is too complex to be captured by a single case notion and multi dimensional or multi entity representation are required to avoid drawing inaccurate conclusions [19]. Conformance checking techniques such as alignments [20] often face the challenge that there are multiple possible explanations for a non-conformance between observed and prescribed process behavior. However, it may be infeasible to show all of them due to the large number of possibilities. Finally, the quality of the input data is often a substantial issue when applying process mining in a real-world scenario [12,21].

This brief look at possible challenges for *accuracy* indicates that the topic is very broad and difficult to discuss comprehensively in the scope of this chapter. Thus, we limit ourselves to briefly describe several challenges and selected solution proposals. We categorize them into solutions for *data quality* and *model quality*.

3.2 Data Quality

Data quality is known to be often poor [22] and this may lead to non-factual or misleading representation of the real business process. Garbage-in garbage-out is a often used phrase to illustrate this issue. Whereas the data quality issue is not particular to process mining there are some peculiarities of event logs that call for specific solutions.

Often data quality problems in process mining are related to the strict data requirements on timestamps (R1), case identifiers (R2), and event labels (R3) [23]. Wrong or coarse granular timestamps lead to discovering wrong causalities in process models or parallelism where none exists. Inconsistent event labels make it difficult to assign clear semantics to the activities of a discovered process model. These are just two examples of how data quality issue impair process mining. Automated repair approaches to combat some of the data quality problems exist. For example, in [24] autoencoders are used to add missing values. However, any such method may affect *transparency* [25] as it is unclear what part of the data was inferred and what part of the data can be considered truthful beyond doubt. A discovered process model may be perfectly *accurate*, but when it is based on data with poor quality any conclusions become disputable. Notions of data quality and remedies are already introduced and discussed in [12], therefore we go not further into detail on the data quality challenge.

One noteworthy topic connected to data quality is *uncertainty* at the level of the event log data [26], e.g., by adding metadata to express the uncertainty [27]. Pegoraro et al. [26] advocate to explicitly encode the uncertainty about events and traces in order to leverage it in a transparent manner during the analysis. Based on this event log with explicit uncertainty representation conformance checking techniques can be adaped [28] to an obtain more trustworthy diagnostics that also provide more transparency about the possible different scenarios compatible with the (uncertain) observations.

3.3 Model Quality

How to decide whether a process model is of good quality? In fact, even when it comes to the question on how to measure *accuracy* there is hardly an agreement in process mining. Classically, process mining quality dimensions consist of fitness, precision, generalization, and simplicity as introduced in [1]. For most of these quality dimensions measures have been proposed that are based on conformance checking, e.g., through alignments as indicated in [20]. However, this common practice of measuring model quality has been challenged at least for precision with Tax et al. [29] proposing several axioms that the prevalent measures do not fulfil.

The issue with initially proposed quality measures led to several new methods and definitions for measuring various model quality dimensions being proposed [30–34]. Main complications for model quality in process mining are that process models commonly exhibit infinite behaviour (through loops) and the absence of negative examples, i.e., behaviour that the model should not contain [1].

Recently, there have been several proposals that aim to extend process discovery and the model quality measures to the stochastic setting in which process models include probabilities and the likelihood of observing a certain trace is taken into account [35,36] allowing to better estimate the relevant subset of the behavior modelled. This may help to truly quantify the confidence that an analyst can have in a model.

A somewhat related issue on the confidence an analyst can put in the performance of a process discovery algorithm was brought up by Van der Werf et al. [37]. They observed that process discovery techniques not always discover better process models when provided with a better sample of the process behavior, i.e., a larger event log with observations of process behavior.

3.4 Outlook and Challenges

The extensive discussion around how to measure quality shows that even defining *accuracy* for process discovery is not straightforward. In practice, this creates the challenge to choose which measure should be used in which context and when can a model be considered good for an analysis purpose. Another very relevant perspective for responsible process mining regarding model quality is how the discovered process model representation is understood by the user of such model. We will come back to this issue when considering *transparency.*

4 Confidentiality

Confidentiality generally refers to the protection of certain sensitive data or information from disclosure. In the context of an organization many different kind of information is usually confidential. Intellectual property such as the design of machines or software may be confidential to protect it from competitors but also general information on the business such as the amount of sales in a certain area is usually kept confidential. A subset of the confidential information in the sphere of an organization relates to personal data. Here, the concern is on the right to *privacy* for individuals of which personal data is processed by the organization. Personal data may relate to customers, employees, suppliers or other people that interact an organization's processes. Privacy rights have received a lot of attention with several high-profile data breaches and increased regulation such as Europe's General Data Protection Regulation (GDPR) [38].

4.1 Process Mining Perspective

In the context of process mining, the information contained in event logs may be sensitive for several reasons. Event logs contain data providing detailed *information on the operations of an organization*, e.g., the order volume or the production capacity. Uncontrolled disclosure of such information may be undesired as it could negatively affect the organization. Event logs contain *information on individuals*, e.g., customers, which may be subject to the privacy regulations.

Assume a hospital process is analyzed. Case data is related to the individual patient and *confidentiality* challenges to protect sensitive data and *privacy* are obvious [39]. However, the employees that work in processes are often also directly affected by process mining results and may be directly represented in the event logs e.g. via the resource attribute in XES. This can create an additional *confidentiality* challenges to prevent work surveillance [40,41].

Protecting the privacy of individuals in event logs is difficult, as sequential event data is highly vulnerable to re-identification [42]. In fact, when assuming some background information, privacy leakages exists in the vast majority of presumably anonymous event logs that are used in the process mining community [42]. As events are linked together through a case, and often the traces in an event log are highly unique, already very limited background knowledge on some attributes or events can reveal the identity of an individual.

This "privacy problem" creates challenges in the practical application of process mining. Data gathering is more difficult or impossible when privacy concerns are raised. For example, the hospital may fear that privacy regulations (GDPR, HIPAA [43]) are violated when analysing patient trajectories [39,44] or a works council may object to the usage of process mining technology due to fear of worker surveillance [41]. Regulations threaten organizations with high fines when personal data is used without legitimate purpose or consent. The fines in GDPR may be as high as 4% of the organizations worldwide annual revenue [38]. Thus, there is a clear need for privacy-preserving or protecting techniques for process mining. Such approaches aim to retain the utility of the data without the risk of accidental disclosure of personal data. Please note that we use *protection* here in the sense of *anononymity* and *unlinkability* requirements. Next to those, other requirements such as *notice*, *transparency*, and *accountability* are often imposed by regulations [45]. Note that most privacy-preserving techniques differ from the wide variety of best-effort pseudonymization, perturbation, and generalization

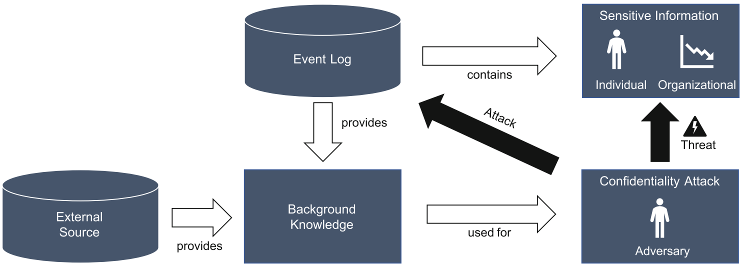


Fig. 4. The main aspects of any confidentiality scenario for process mining: What is the sensitive information contained in the event log that needs to be protected? Which background knowledge can be assumed (including provided by external sources)? What are the attacks used by the adversary and which threats are posed?

methods that are used by commercial tools[5]. Unfortunately, it has been shown such naïve replacement of identifiers is often not sufficient to keep information secure in many scenarios.

For each confidentiality scenario we need to characterize at least the *sensitive information* (Sect. 4.2) and the *background knowledge* (Sect. 4.3) of the attacker or adversary as illustrated in Fig. 4. Then, we can identify *confidentiality attacks* (Sect. 4.4) that are assumed to be employed an the resulting *threats* that should be mitigated. Based on the analysis of the available threats, protection techniques have been proposed to mitigate these threats under certain assumptions (Sect. 4.5).

4.2 Sensitive Information

Several kinds of sensitive information may be derived from event logs. We consider both the scenario in which an event log contains some business information that needs to be secured as well as the scenario in which personal data of individuals that took part in the process should not be revealed. These individuals could be customers that are the *subject* of the process or workers that perform activities withing the process.

We assume that the **sensitive information** is contained in a given event log as shown in Fig. 4. Sensitive information may be obtained *directly* from the attribute values of individual events of or it may be *derived* by performing some computation over several events in. Often, in the scenario in which personal data of individuals is at risk the sensitive information in the event log is assumed to be connected to the individual through the process cases each of which is about a single individual. We now illustrate several types of sensitive information with

[5] Most commercial tools provide some kind of pseudonymization technique to replace sensitive data by a hashing or replacement. An example is given here: https://fluxicon.com/blog/2017/11/privacy-security-and-ethics-in-process-mining-part-3-anonymization/.

Table 1. Example of a loan application event log that contains several types of sensitive information and may be subject to confidentiality attacks revealing this information to an adversary possessing suitable background knowledge.

SSN	Activity	Time	Resource	Amount	Age	Type	Postcode	Income
617-07-5604	SA: Submit appl.	09-02-22 23:39		200k	30	Home	94121	50k
617-07-5604	SC: Simple check	11-02-22 08:38	Alice	200k	30	Home	94121	50k
617-07-5604	AA: Accept appl.	12-02-22 11:35	Joe	200k	30	Home	94121	50k
617-07-5604	SO: Send offer	12-02-22 12:32	Joe	200k	30	Home	94121	50k
617-07-5604	RO: Receive offer	13-02-22 08:14		200k	30	Home	94121	50k
617-07-5604	FA: Finalise appl.	15-02-22 16:30	Alice	200k	30	Home	94121	50k
617-07-5604	AO: Accept offer	19-02-22 23:31		200k	30	Home	94121	50k
528-41-8024	SA: Submit appl.	01-03-22 12:32		60k	42	Car	37287	75k
528-41-8024	SC: Simple check	02-03-22 15:23	Joe	60k	42	Car	37287	75k
528-41-8024	EC: Extensive check	05-03-22 07:31	John	60k	42	Car	37287	75k
528-41-8024	AA: Accept appl.	11-03-22 12:21	Alice	60k	42	Car	37287	75k
528-41-8024	SO: Send offer	11-03-22 15:44	Joe	60k	42	Car	37287	75k
528-41-8024	RO: Receive offer	12-03-22 12:33		60k	42	Car	37287	75k
528-41-8024	FA: Finalise appl.	15-03-22 16:54	Robert	60k	42	Car	37287	75k
528-41-8024	AO: Accept offer	18-03-22 18:23		60k	42	Car	37287	75k
330-80-8169	SA: Submit appl.	02-03-22 23:30		500k	22	Home	32984	45k
330-80-8169	EC: Extensive check	05-03-22 08:30	John	500k	22	Home	32984	45k
330-80-8169	DA: Decline appl.	10-03-22 11:30	John	500k	22	Home	32984	45k
526-34-5246	SA: Submit appl.	15-04-22 23:31		100k	30	Home	75755	30k
526-34-5246	SC: Simple check	17-04-22 12:47	Joe	100k	30	Home	75755	30k
526-34-5246	AA: Accept appl.	18-04-22 11:59	Alice	100k	30	Home	75755	30k
526-34-5246	SO: Send offer	18-04-22 12:29	Alice	100k	30	Home	75755	30k
526-34-5246	RO: Receive offer	19-04-22 07:52		100k	30	Home	75755	30k
526-34-5246	CO: Cancel offer	24-04-22 21:34		100k	30	Home	75755	30k
526-34-5246	SO: Send offer	28-04-22 09:21	John	100k	30	Home	75755	30k
526-34-5246	RO: Receive offer	29-04-22 10:12		100k	30	Home	75755	30k
526-34-5246	FA: Finalise appl.	02-05-22 15:43	Alice	100k	30	Home	75755	30k
526-34-5246	AA: Accept offer	05-05-22 05:23		100k	30	Home	75755	30k

the event log in Table 1 that was obtained from the previously introduced loan application process.

An example for sensitive information related to an individual that can be directly obtained is the social security number of the applicant stored in the column *SSN*, which also acts as case identifier here. Obviously, using such direct identifiers of individuals poses a privacy risk as it would allow to directly link all the remaining information contained in the event log to individuals. Analogously, the *Resource* column contains the full name of the employee responsible for handling the process activities. This information would enable direct profiling of the work performance of individual employees, which may be against company policies or forbidden by work regulations. It is easy to remove directly personally identifiable information such as names or identifying numbers of customers or workers as they are not necessary for process mining. For example, it would be trivial to replace both the *SSN* column and the *Resource* in Table 1 with a surrogate case identifier based on a mapping obtained through one-way hash function or a simple lookup table.

However, it has been shown that obscuring the direct identifiers is not sufficient as also not directly identifying attributes can be problematic [42,46]. *Quasi-identifiers* are values not directly revealing the identity of a person but may be used to do so in combination with other attributes. Common quasi-identifiers are attributes such as gender, birth dates, or postcodes that taken together are often unique for an individual. For example, in Table 1 the combination of columns *Age*, *Type*, and *Postcode* would very likely be uniquely identifying a single customer leading to disclosure of other sensitive information contained in the event log such as the yearly *Income* of the applicant.

So far, we gave examples of sensitive information that is directly stored in the event attributes. However, also the presence of a certain activity in the event log or derived information such as the sequence of events that occurred for a certain case may be considered sensitive. Take for instance the third case in Table 1 in which the loan application is declined (DA) after an extensive check (EC). Knowledge of such details on how the loan application process was carried out may be used against the individual. Thus, even the sequence of activities performed for an individual case may be regarded as sensitive information. At the same time, the sequence of activities performed may also act as a quasi-identifier as it is often unique and identifies an individual such as an applicant or a patient [42,47].

When it comes to sensitive business information one may think about attributes encoding the cost of a certain activity or information on prices paid by different customer segments (e.g., the interest offered on the loan). Similarly to the case of personal information, the sensitive information may not only reside in the attribute values but also be derived from the sequence of events that occurred or their timestamps, e.g., the throughput times computed for different organizational units may be considered sensitive.

It is important to realize that these computations may also be based on the artefacts that are returned by the classical process mining tasks: intermediate data structures, process models, and conformance checking results. Thus, direct access to the original event log may not be required to gain access to sensitive information. For example, the utilisation of a certain department or group may be determined by considering the number of traces in a certain time period and could be considered sensitive. The cross-organisational process mining scenario is also commonly considered when it comes to motivating the need of protecting sensitive information for process mining. Here, two organizations want to compare their processes to learn from each other or analyse a process that is jointly performed (e.g., supplier and integrator). However, certain sensitive data should not be shared.

4.3 Background Knowledge

Apart from the trivial case in which an individual or an organizational entity can be directly identified, attackers often need to possess certain limited background knowledge about the individual, i.e., the process case, the entity, or about remaining parts of the dataset. This is reflected in Fig. 4 by assuming the adversary to use some knowledge to facilitate attacks on sensitive information. Some protection models assume the worst-case scenario in which no restriction on the background knowledge of an attacker is assumed and still some kind of privacy guarantee should be given. However, in many cases it is reasonable to assume only limited background knowledge to be available.

Background knowledge may be fully derived from the event log or it may also contain information that is not present in the event log but related to specific cases or events. Thus, it can be any kind of knowledge that gives an attacker information that can be used to identify sensitive information. We keep the definition of the background knowledge deliberately vague as it may be defined in various ways and include arbitrary external data sources. Two more precise definitions for event logs have been introduced in the literature.

Rafei et al. [48] provide several definitions for possible background information in a process mining context. They assume that background knowledge is defined over a simple view of process traces as sequences of event labels, e.g., the third trace in Table 1 would be seen as sequence $\langle SA, EC, DA \rangle$. Three categories are defined: *Set knowledge*, *Multiset knowledge*, and *Sequence knowledge*. The knowledge refers the occurrence of activity labels in the to be attacked process case at one of the three abstraction levels. Thus, an attacker can either know only about the *presence* of activities (set abstraction), their *frequency* (multiset abstraction), or have in-depth knowledge about a certain *ordering* of activities (sequence abstraction). In Table 1, the third trace $\langle SA, EC, DA \rangle$ would already be uniquely identified when having the set background knowledge $\{DA\}$ since that is the only trace in which an application is declined. As another example, the multiset background knowledge of $[SO^2]$ would uniquely identify the fourth case. In many setting such knowledge of process events may be easy to obtain, e.g., one may know that their neighbours received two loan offers in a specific time period.

Von Voigt et al. [42] quantify the re-identification risk of individual cases by assuming different kinds of background knowledge. In addition to knowledge of activity labels as in [48] also case-level attributes are considered to be candidates for background knowledge. For example, in the well-known BPI Challenge 2018 dataset [49] case attributes have been generalized to provide some level of privacy protection. However, still when considering the combinations of all case attributes 84.5% of all cases are unique.

Many other similar abstraction and definitions of background knowledge are possible but have not yet been investigated. For example, partial orders of activities or knowledge about time or resource involved. An adversary may know that two medical diagnostic tests have been performed on the same day and two days later the patient was re-invited for a discussion by the same doctor.

Also knowledge on the absence of a certain activity in the case to be attacked could be informative. As Fig. 4 illustrates also external data source may provide complementary background knowledge. A famous example that involved using external background knowledge is the successful attack on a Netflix dataset by using information from the public IMBD movie ratings [50], which included full names for some users, and compared them to the ratings in the Netflix dataset thereby identifying users in the supposedly anonymized Netflix dataset.

In summary, a precise analysis of background knowledge assumed is important to provide meaningful guarantees against uncovering sensitive information.

4.4 Threats and Attacks

Several attacks on confidential data in event logs are possible. We follow Elkoumy et al. [45] and focus on a honest-but-curious attacker scenario. An adversary has access to data or results and tries to identify some sensitive information without trying to break into systems. So, we do not consider scenarios in which access control or similar security measures are broken.

We structure confidentiality attacks structured according to the threat that they pose, i.e., the kind of sensitive information that an attacker or adversary tries to reveal. As already motivated, it is important to consider the kind of background knowledge that is assumed in the analysis of a specific threat or attack to find reliable mitigation strategies. Attacks on confidentiality use this background knowledge to reveal sensitive information that is contained in the event log as shown in Fig. 4.

So, a very general definition of a confidentiality attack on an event log can be given as follows. Given an event log and some sensitive information that is related to that log, a **confidentiality attack** uses some background knowledge, which may be derived from the log or from other available sources, to reveal some subset of sensitive information that is part of the log. We distinguish four general types of threats based on the goal of an attacker and the employed attack method following the categorization in [45].

Membership Disclosure Threats. A basic threat is that an adversary could establish that an individual was taking part of the process that is described by the event log. A *membership inference* attack combines background knowledge about the individual to the information released by an event log or a process mining analysis. So, the sensitive information obtained from the event log would consist of the identifiers for a subset of individuals that took part in the process. Whereas this does not reveal the exact case in which an individual took part, it still often allows to draw conclusions about which activities and events an individual was involved in. Let us assume that the event log obtained in our example loan application process scenario only contains loans for starting a business. Already, the information that an individual is part of that event log, i.e., they were applying for such a specific loan type can be sensitive information.

Re-identification Threats. Threats that cause the disclosure of the identity of a individual to which some data belongs are called re-identification threats. So, the sensitive information is the subject of a certain case, e.g., the patient identity, or the subject of a certain event, e.g., the identity of the resource or worker that performed the activity recorded by the event. Example attacks are linkage attacks and intersection attacks [45]. *Linkage attacks* use background knowledge to reveal the identity, e.g., a certain combination of attribute values or a certain sequence of events is known to be connected to an individual. In Table 1, knowing that an individual received two offers, i.e., multiset background knowledge of $[SO^2]$, and that their data is part of the event log uniquely re-identifies identity of the applicant in the fourth case. *Intersection attacks* try to establish a mapping between two separately released event logs revealing the identity of an individual. Here the information revealed in a second separately released dataset is assumed to be directly linkable to an individual without containing any sensitive information. However, in combination this information can be used as background knowledge and reveal the sensitive information in the first event log.

Reconstruction Threats. In some cases it may be possible to partially or fully infer the original event from seemingly protected data. Here, the sensitive information to be retrieved would be the entire event log. The two main attack methods for reconstruction are *difference attacks* and *model-inversion attacks*. The basic idea for both is to repeatedly consult a model or a statistic with slightly different queries and, thereby, uncovering sensitive data.

Cryptanalysis Threats. Data may have been pseudonymized, as often done by commercial tools, or encrypted in an attempt to provide confidentiality. However, naïvely pseudonymized or even fully encrypted event logs are vulnerable to attacks based on the analysis of the frequency [51]. Please note that this may lead in turn to re-identification, membership disclosure, or reconstruction, but may also simply leak sensitive business information such as the number of certain activity executions. The main attack method is a *frequency analysis* based on background knowledge on the activities of the process and their prevalence.

4.5 Protection Approaches

Whereas still in an early stage, the research on privacy and confidentiality has received increased attention in the past years and several protection techniques with diverse assumptions and guarantees that protect against the mentioned threats have been proposed. However, none of the proposed methods is generally applicable to any possible confidentiality and privacy problems. Each of them makes certain assumptions regarding the attack scenario including the background knowledge of the assumed attacker. Conversely, depending on the input log the methods result in some loss of utility. Thus, the goal of the process mining analysis (discovery, conformance, etc.), their data requirements, and the characteristics of the process that generated the event log need to be considered.

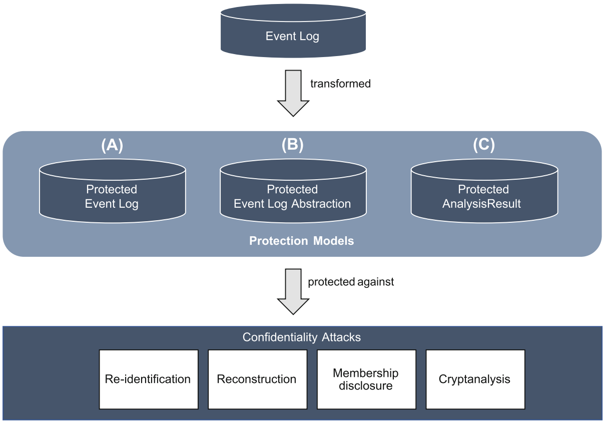


Fig. 5. Different protection models have been proposed that protect the data contained in an event log by transforming it into protected representations: a protected event log, a protected abstraction the event log, or a protected analysis result.

Protection models can work at different levels of a process mining analysis as shown in Fig. 5. Following [48], we differentiate between several tasks for protection models. Some models protecting the event log itself and provide a protected copy of the original event log. Other techniques provide protected abstractions over the original event log, e.g., a directly-follows graph representation which can only be used for certain process mining activities. In [48], these two tasks are denoted as Privacy-Preserving Data Publishing (PPDP) and Privacy-Preserving Process Mining (PPPM), respectively. We add a third possible task, which is to protect process mining results, e.g. a process model or a conformance checking result, without an intermediate representation.

Regardless of the task at hand, techniques can also be distinguished into roughly three categories of protection models [45]: *group-based privacy models*, *indistinguishability-based models*, and *confidentiality frameworks* including encryption. We now introduce the main properties of the three different protection model categories and briefly introduce exemplary techniques.

Group-Based Privacy. The prototype of a group-based privacy protection model are those that provide *k-anonymity* [52]. The basic idea is that a tabular dataset containing rows with information about individuals is *k-anonymous* when the values for each combination of sensitive attributes or columns (quasi-identifiers) appear at least k times. So, data similarity is used as a criterion here. The intuitive idea is that the individual will have the same sensitive data as the $k-1$ other individuals in the same group and, thus, with sufficiently large k it protects against re-identification. Usually, this is achieved by data *suppression* or *generalization* until the $k-anonymity$ property is achieved. Whereas this model is interpretable and easy to understand, unfortunately, it has been shown to be suspect to certain attacks based on background knowledge [53]. Several extensions have been proposed that mitigate some of those including: l-diversity [53], and t-closeness [54].

For process mining, two methods are providing group-based privacy protection models. The TLKC model by Rafaei et al. [47] and the PRETSA approach by [55]. Both aim at the release of a proctected event log, option (A) in Fig. 5. PRETSA utilizes generalisation based on a prefix-tree that is build on top of the activity sequence in the event log and provides *k-anonymity* and *t-closeness* guarantee to prevent the disclosure (membership and re-identification) of resources or workers that performed certain activities. The TLKC model protects the identity of cases, e.g., a customer, and provides a relaxed variant of *k-anonymity*. Additional, it supports protecting information in the time and organizational perspective. Both approaches make assumptions on the background knowledge. Maintaining data utility is challenging for both methods when many unique traces exist.

Indistinguishability-Based Privacy. Differently to the group-based models providing guarantees such as k-anonymity for a given dataset, indistinguishability-based privacy models give a guarantee that two versions of a dataset are indistinguishable to a certain degree. A central model is *Differential Privacy* (DP). The idea is that there is one datasets A without an individuals information and another one A' including an individuals information. A mechanism provides DP with a parameter ϵ when the results of a (randomized) query mechanism statistically differ between A and A' only by a small factor that is controlled by the ϵ-parameter [56]. This provides a strong guarantee that is independent of the background knowledge as the guarantee needs to hold for any dataset A and A'. There have been many variants of the differential privacy concept [57]. For example, adding a relaxation parameter δ to better tuning the utility while loosing some of its strengths ((ϵ, δ)-DP), only requiring the values of the datasets to not differ too much and ignoring addition and removal of items (bounded DP), or extending the guarantees in the case individuals appear multiple times in the dataset (group DP), which is a possible scenario for event logs.

For process mining, several adaptions have been proposed. The first one was given by Mannhardt et al. in [58] who assume a protected event log to be queries through a privacy engine. Laplacian noise is added to the counts returned by each query, thereby guaranteeing DP with regard to the individual cases. Queries are defined for both directly-follows relations [1] and complete activity sequences. The method was later extended in [59] to also protect contextual information that is encoded in the attributes of the event log. Furthermore, the guarantee was extended to *local DP*, which means that a perturbed event log itself can be released. One major issue of these methods is that obviously invalid process behavior may be added. Recently, the approach was improved to consider the semantic of the added noise [60]. Contextual information, in particular process performance indicators, is also protected in the work by Kabierski et al. [61]. Finally, there is a very recent proposal by Elkoumy et al. [62] that provides only a bounded DP guarantee but improves the utility of the protected data by using an oversampling approach instead of adding noise.

Confidentiality Frameworks. The third type that we distinguish are protection models that are not directly targeted at protecting individuals but any kind of sensitive information in event logs. Here, mainly encryption schemes have been proposed. A major family of techniques are those based on on homomorphic encryption [63] schemes. The goal is to enable certain computations on an encrypted version of the data. For process mining, this idea is taken up by Rafei et al. in [51] and embedded in a framework that aims to protect against frequency or background knowledge-based attacks by disassociating events from their respective cases. It could be used to outsource computations on secured data or in a cross-organizational setting. However, it does not protect the resulting analysis results (B) and (C) from an internal process analyst. The cross-organization setting is also targeted by Elkoumy et al. in [64]. A secure multi-party computation [65] method is proposed that avoids to leak sensitive information in the cross-organisational process mining scenario.

The above categorization and list of techniques covers the major share of the work in the process mining field so far.

4.6 Outlook and Challenges

Protecting the privacy and confidentiality of data while keeping it useful for analysis is a difficult problem. Information needs to be hidden while the objective is to get as much signal from data as possible. Unsurprisingly, many open challenges exists for *confidentiality* in process mining and, apart from academic prototypes [66,67], none of the proposed techniques has seen uptake in commercial solutions. Seven main challenges for research in the field of privacy and confidentiality in process mining are identified by [45]:

– *Interpretable Quantification of Privacy Disclosure.* Protection mechanisms should be interpretable when it comes to the remaining risk. Guarantees and attacks are often not obvious for non-experts making adoption by industry difficult.
– *Balancing Risk and Utility.* Any protection mechanism may impair the utility of the source data and poses a trade-off that needs to be made upfont. In the exploratory process mining setting this is a challenge for adoption. In [68] it is proposed that mechanisms need to be *utlility aware.*
– *Level of Granularity.* Process mining analyses happen at various levels of granularity and various perspectives. Some tools require only activity sequences and timestamps, which most current protection models focus upon. Others also consider the resource perspective including potential sensitive data on employees. In some cases, access to an event log may not be necessary and privacy guarantees should be given at the level of a released process model as proposed in [69]. A one-fits-all approach to privacy is unlikely to work, which opens opportunities for further research.

- *Distributed Privacy.* In many settings attempts on data sharing between organizations are made which creates the problem of protecting privacy in an inter-organizational setting. This setting is currently less well researched.
- *Computational Challenges.* Some of the approaches proposed are computationally expensive. Thus, research on making those suitable for real-life settings is required.
- *Traceability and Transparency Challenges.* Often personal data still needs to be collected and stored at some point during the analysis. GDPR requires to trace the processing and usage of data to fulfill the different rights (right to consent, right to access, right to be forgotten [38]). This is challenging for process mining where data comes from different distributed data sources. Similarly, GDPR requires organizations to be transparent about the usage of data. Traceability is a pre-requisite but not sufficient for achieving transparency. Investigating how to provide information on the purpose for which data was used in process mining is a research challenge.

Many of these challenges are geared towards the improving technological solutions that provide some form of privacy guarantee in various settings. However, as already reported in [40] many aspects of privacy and confidentiality as well as the compliance to regulations such as GDPR cannot be solved by technological measures alone. However, there is little research from the organizational side apart from anecdotal discussion on the role of privacy in real-life process mining projects [41]. To conclude, it is notable that process mining has also been used to check conformance to privacy regulations [70]. Thus, process mining can also help in uncovering confidentiality issues that are present in an organizations processes.

5 Transparency

Transparency has been a widely discussed topic for AI systems that are based on machine learning. Often, a key concern is the explainability of black-box classifiers such as Deep Learning models: Why is a certain classification or prediction made and what features are important in the decision of the model?

The core process mining tasks of process discovery and conformance checking aim to provide *white-box* process models that can be interpreted by process stakeholders. Explainability of the discovered models and, thus, transparency is key objective of process mining. Still, there are several aspects of process mining in which transparency is at risk. In the next two section, we focus on two exemplary transparency challenges for process mining: achieving *generalization* without hampering transparency and the *interpretability* of the discovered process model representations.

Besides these two transparency challenges all the common transparency issues of predictive models are inherited when building predictive process mining models. Therefore, we do not discuss this in detail since many resources on explainable machine learning are available and [71] gives a brief overview of how to obtain explainable predictions in the context of process mining.

5.1 Generalization

Process discovery aims to abstract from the exact behaviour observed in the event log and return a *concise* model of the underlying process. This often requires to disregard *infrequent behaviour* to obtain simpler process models. Conversely, process discovery techniques often attempt to generalize beyond the observed behaviour since they cannot be assumed to have observed all possible incarnations of the process, particularly in the presence of parallel process behavior. This aspiration creates a *transparency* challenges.

Disregarding infrequent behaviour may hide important parts of the observed data. In particular, infrequent patterns may be of high interest [44]. Very few techniques have been focusing on retaining infrequent data, e.g., in [72] certain infrequent dependencies are not filtered if they can be reliably predicted from data attributes and in [73] it is explored how to selectively include infrequent behaviour by filtering over multiple ranges of parameter values.

In a orthogonal direction, the frequency and probability with which behaviour is observed gets more attention in approaches that can be labeled as: *stochastic process mining*. In [35], Leemans et al. proposed a new conformance checking method with the goal of taking into account routing probabilities, which improves the accuracy of the diagnostics.

5.2 Interpretation of Results

Interpretation of results based on process model notations or visualizations can be difficult for stakeholders leading to *transparency* challenges. For example, the presence of loops together with optional activities may enable non obvious process behaviour and the filtering of edges in a directly-follows graph may lead to invalid statistics as is illustrated for many commercial tools in [74].

However, also for discovery approaches based on clear semantics misinterpretations are possible. As an example, the models discovered by the Inductive Miner often contain silent transitions that allow to skip certain behaviour that in combination with loops allow any behaviour. This may be difficult to spot for a non-expert. Whereas there exists research on the comprehension of process models [75], little work has yet been done in the context of automated process discovery.

Recently the question of interpretability of process mining results has been touched upon by Mendling et al. [76] who raise the issue that the quality of process mining results needs to be judged in light of the tasks of a process analyst using the models. A first technical contribution for process discovery in this direction was provided by Fahland et al. [77] with a new variant of the Inductive Miner that was evaluated in a user study in which an analyst's trust in the model as considered. Overall, there has been surprisingly little research on this topic given the claim of process mining to provide white-box models.

6 Conclusion

This chapter defined the concept of Responsible Process Mining under the umbrella of Responsible Data Science. Based on the FACT criteria put forward in [5] (Fairness, Accuracy, Confidentiality, and Transparency), we gave an overview of challenges related to these criteria and introduced state-of-the-art approaches for addressing each of them. Due to the broad scope of the FACT criteria, we can provide only a high-level introduction and discussion for each of them. We refer to the individual work or relevant surveys for further details.

In some areas the research on responsible process mining is already much further developed than in others. Little attention has been devoted to *fairness* in the context of process mining, at least compared to its prominence in the machine learning field. The trend to more automated decision taking in process mining may change this in the future. In contrast, the *confidentiality* challenge has been recognized in the process mining research community and has recently received much attention in research. However, adoption by commercial process mining tools has not yet started even though the problem has also been recognized by industry [41].

Criteria such as *accuracy* and *transparency* are very broad and many approaches touch these issues; however, with the notable exception of the work on data quality [12] they are rarely addressed explicitly under the umbrella of responsible process mining. More work is required to develop and address these criteria more explicitly in future process mining research.

References

1. Aalst, W.: Foundations of process discovery. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
2. Reinkemeyer, L.: Status and future of process mining: from process discovery to process execution. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
3. van der Aalst, W.M.P.: Responsible data science: using event data in a "People Friendly manner. In: Hammoudi, S., Maciaszek, L.A., Missikoff, M.M., Camp, O., Cordeiro, J. (eds.) ICEIS 2016. LNBIP, vol. 291, pp. 3–28. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62386-3_1
4. Dignum, V.: Responsible Artificial Intelligence. Springer (2019)
5. van der Aalst, W.M.P., Bichler, M., Heinzl, A.: Responsible data science. Bus. Inf. Syst. Eng. **59**(5), 311–313 (2017)
6. Olteanu, A., Garcia-Gathright, J., Rijke, M.d., Ekstrand, M.D.: FACTS-IR: Fairness, accountability, confidentiality, transparency, and safety in information retrieval. ACM SIGIR Forum **53**(2), 20 (2019)
7. Aalst, W.: Process mining: a 360 degrees overview. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
8. Friedman, B., Nissenbaum, H.: Bias in computer systems. ACM Trans. Inf. Syst. **14**(3), 330–347 (1996)

9. Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., Galstyan, A.: A survey on bias and fairness in machine learning. ACM Comput. Surv. **54**(6) (2021)
10. Barocas, S., Hardt, M., Narayanan, A.: Fairness and Machine Learning. fairmlbook.org (2019). http://www.fairmlbook.org
11. Crawford, K.: The hidden biases in big data. Harvard Bus. Rev. **1**(4) (2013)
12. De Weerdt, J., Wynn, M.T.: Foundations of process event data. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
13. Commission, E.: Proposal for a regulation of the European parliament and of the council laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts (2021)
14. Gajane, P., Pechenizkiy, M.: On formalizing fairness in prediction with machine learning. CoRR abs/1710.03184 (2018)
15. Verma, S., Rubin, J.: Fairness definitions explained. In: FairWare@ICSE, pp. 1–7. ACM (2018)
16. Qafari, M.S., van der Aalst, W.: Fairness-aware process mining. In: Panetto, H., Debruyne, C., Hepp, M., Lewis, D., Ardagna, C.A., Meersman, R. (eds.) OTM 2019. LNCS, vol. 11877, pp. 182–192. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33246-4_11
17. Bozorgi, Z.D., Teinemaa, I., Dumas, M., Rosa, M.L., Polyvyanyy, A.: Process mining meets causal machine learning: discovering causal rules from event logs. In: ICPM, pp. 129–136. IEEE (2020)
18. Bozorgi, Z.D., Teinemaa, I., Dumas, M., Rosa, M.L., Polyvyanyy, A.: Prescriptive process monitoring for cost-aware cycle time reduction. In: ICPM, pp. 96–103. IEEE (2021)
19. Fahland, D.: Process mining over multiple behavioral dimensions with event knowledge graphs. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
20. Carmona, J., Dongen, B., Weidlich, M.: Conformance checking: foundations, milestones and challenges. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
21. Accorsi, R., Lebherz, J.: A practitioner's view on process mining adoption, event log engineering and data challenges. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
22. Wynn, M.T., Sadiq, S.: Responsible process mining - a data quality perspective. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNCS, vol. 11675, pp. 10–15. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26619-6_2
23. Suriadi, S., Andrews, R., ter Hofstede, A.H.M., Wynn, M.T.: Event log imperfection patterns for process mining: towards a systematic approach to cleaning event logs. Inf. Syst. **64**, 132–150 (2017)
24. Nguyen, H.T.C., Lee, S., Kim, J., Ko, J., Comuzzi, M.: Autoencoders for improving quality of process event logs. Expert Syst. Appl. **131**, 132–147 (2019)
25. Martin, N., Martinez-Millana, A., Valdivieso, B., Fernández-Llatas, C.: Interactive data cleaning for process mining: a case study of an outpatient clinic's appointment system. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) BPM 2019. LNBIP, vol. 362, pp. 532–544. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37453-2_43
26. Pegoraro, M., van der Aalst, W.M.P.: Mining uncertain event data in process mining. In: ICPM, pp. 89–96. IEEE (2019)

27. Pegoraro, M., Uysal, M.S., van der Aalst, W.M.P.: An XES extension for uncertain event data. In: BPM (PhD/Demos). Volume 2973 of CEUR Workshop Proceedings, pp. 116–120. CEUR-WS.org (2021)
28. Pegoraro, M., Uysal, M.S., van der Aalst, W.M.P.: Conformance checking over uncertain event data. Inf. Syst. **102**, 101810 (2021)
29. Tax, N., Lu, X., Sidorova, N., Fahland, D., van der Aalst, W.M.P.: The imprecisions of precision measures in process mining. Inf. Process. Lett. **135**, 1–8 (2018)
30. van Dongen, B.F., Carmona, J., Chatain, T.: A unified approach for measuring precision and generalization based on anti-alignments. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 39–56. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_3
31. Kalenkova, A., Polyvyanyy, A., La Rosa, M.: A framework for estimating simplicity of automatically discovered process models based on structural and behavioral characteristics. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNCS, vol. 12168, pp. 129–146. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58666-9_8
32. Polyvyanyy, A., Solti, A., Weidlich, M., Ciccio, C.D., Mendling, J.: Monotone precision and recall measures for comparing executions and specifications of dynamic systems. ACM Trans. Softw. Eng. Methodol. **29**(3), 17:1–17:41 (2020)
33. Augusto, A., Armas-Cervantes, A., Conforti, R., Dumas, M., Rosa, M.L.: Measuring fitness and precision of automatically discovered process models: a principled and scalable approach. IEEE Trans. Knowl. Data Eng. **34**(4), 1870–1888 (2022)
34. Polyvyanyy, A., Kalenkova, A.A.: Conformance checking of partially matching processes: an entropy-based approach. Inf. Syst. **106**, 101720 (2022)
35. Leemans, S.J., van der Aalst, W.M., Brockhoff, T., Polyvyanyy, A.: Stochastic process mining: earth movers' stochastic conformance. Inf. Syst. **102**, 101724 (2021)
36. Alkhammash, H., Polyvyanyy, A., Moffat, A., García-Bañuelos, L.: Entropic relevance: a mechanism for measuring stochastic process models discovered from event data. Inf. Syst. **107**, 101922 (2022)
37. van der Werf, J.M.E.M., Polyvyanyy, A., van Wensveen, B.R., Brinkhuis, M., Reijers, H.A.: All that glitters is not gold. In: La Rosa, M., Sadiq, S., Teniente, E. (eds.) CAiSE 2021. LNCS, vol. 12751, pp. 141–157. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79382-1_9
38. Regulation, E.G.D.P.: Regulation (eu) 2016/679 of the european parliament and of the council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation) 2016. OJ L 119(1) (2016)
39. Pika, A., Wynn, M.T., Budiono, S., ter Hofstede, A.H.M., van der Aalst, W.M.P., Reijers, H.A.: Towards privacy-preserving process mining in healthcare. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) BPM 2019. LNBIP, vol. 362, pp. 483–495. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37453-2_39
40. Mannhardt, F., Petersen, S.A., Oliveira, M.F.: Privacy challenges for process mining in human-centered industrial environments. In: Intelligent Environments, pp. 64–71. IEEE (2018)
41. Mannhardt, F., Koschmider, A., Biermann, L., Lange, J., Tschorsch, F., Wynn, M.T.: Trust and privacy in process analytics. Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model. **15**, 8:1–8:4 (2020)
42. Nuñez von Voigt, S., et al.: Quantifying the re-identification risk of event logs for process mining. In: Dustdar, S., Yu, E., Salinesi, C., Rieu, D., Pant, V. (eds.) CAiSE 2020. LNCS, vol. 12127, pp. 252–267. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49435-3_16

43. Centers for Medicare & Medicaid Services: The Health Insurance Portability and Accountability Act of 1996 (HIPAA) (1996). Online at http://www.cms.hhs.gov/hipaa/
44. Martin, N., et al.: Recommendations for enhancing the usability and understandability of process mining in healthcare. Artif. Intell. Med. **109**, 101962 (2020)
45. Elkoumy, G., et al.: Privacy and confidentiality in process mining - threats and research challenges. ACM Trans. Manage. Inf. Syst. (2021) accepted
46. Sweeney, L.: Simple demographics often identify people uniquely. Health (San Francisco) **671**(2000), 1–34 (2000)
47. Rafiei, M., van der Aalst, W.M.P.: Group-based privacy preservation techniques for process mining. Data Knowl. Eng. **134**, 101908 (2021)
48. Rafiei, M., van der Aalst, W.M.P.: Towards quantifying privacy in process mining. In: Leemans, S., Leopold, H. (eds.) ICPM 2020. LNBIP, vol. 406, pp. 385–397. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72693-5_29
49. van Dongen, B., Borchert, F.F.: Bpi challenge 2018 (2018)
50. Narayanan, A., Shmatikov, V.: Robust de-anonymization of large sparse datasets. In: IEEE Symposium on Security and Privacy, pp. 111–125, IEEE Computer Society (2008)
51. Rafiei, M., von Waldthausen, L., van der Aalst, W.M.P.: Supporting confidentiality in process mining using abstraction and encryption. In: Ceravolo, P., van Keulen, M., Gómez-López, M.T. (eds.) SIMPDA 2018-2019. LNBIP, vol. 379, pp. 101–123. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-46633-6_6
52. Sweeney, L.: k-anonymity: a model for protecting privacy. Int. J. Uncertain. Fuzz. Knowl. Based Syst. **10**(05), 557–570, 101962 (2002)
53. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: l-diversity: privacy beyond k-anonymity. ACM Trans. Knowl. Discov. Data **1**(1) 3-es (2007)
54. Li, N., Li, T., Venkatasubramanian, S.: t-closeness: privacy beyond k-anonymity and l-diversity. In: 2007 IEEE 23rd International Conference on Data Engineering, pp. 106–115, IEEE (2007)
55. Fahrenkrog-Petersen, S.A., van der Aa, H., Weidlich, M.: Pretsa: event log sanitization for privacy-aware process discovery. In: 2019 International Conference on Process Mining (ICPM), pp. 1–8. IEEE (2019)
56. Dwork, C.: Differential privacy: a survey of results. In: Agrawal, M., Du, D., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 1–19. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79228-4_1
57. Desfontaines, D., Pejó, B.: SOK: differential privacies. Proc. Priv. Enhancing Technol. **2020**(2), 288–313, 101962 (2020)
58. Mannhardt, F., Koschmider, A., Baracaldo, N., Weidlich, M., Michael, J.: Privacy-preserving process mining - differential privacy for event logs. Bus. Inf. Syst. Eng. **61**(5), 595–614 (2019)
59. Fahrenkrog-Petersen, S.A., van der Aa, H., Weidlich, M.: PRIPEL: privacy-preserving event log publishing including contextual information. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNCS, vol. 12168, pp. 111–128. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58666-9_7
60. Fahrenkrog-Petersen, S.A., Kabierski, M., Rösel, F., van der Aa, H., Weidlich, M.: Sacofa: semantics-aware control-flow anonymization for process mining. In: ICPM, pp. 72–79. IEEE (2021)
61. Kabierski, M., Fahrenkrog-Petersen, S.A., Weidlich, M.: Privacy-aware process performance indicators: framework and release mechanisms. vol. 12751, pp. 19–36 (2021)

62. Elkoumy, G., Pankova, A., Dumas, M.: Mine me but don't single me out: differentially private event logs for process mining. In: ICPM, pp. 80–87. IEEE (2021)
63. Gentry, C.: Computing arbitrary functions of encrypted data. Commun. ACM **53**(3), 97–105 (2010)
64. Elkoumy, G., Fahrenkrog-Petersen, S.A., Dumas, M., Laud, P., Pankova, A., Weidlich, M.: Secure multi-party computation for inter-organizational process mining. In: Nurcan, S., Reinhartz-Berger, I., Soffer, P., Zdravkovic, J. (eds.) BPMDS/EMMSAD -2020. LNBIP, vol. 387, pp. 166–181. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49418-6_11
65. Lindell, Y.: Secure multiparty computation. Commun. ACM **64**(1), 86–96 (2021)
66. Bauer, M., Fahrenkrog-Petersen, S.A., Koschmider, A., Mannhardt, F., van der Aa, H., Weidlich, M.: Elpaas: event log privacy as a service. In: BPM (PhD/Demos). Volume 2420 of CEUR Workshop Proceedings., CEUR-WS.org, pp. 159–163 (2019)
67. Rafiei, M., van der Aalst, W.M.P.: Practical aspect of privacy-preserving data publishing in process mining. In: BPM (PhD/Demos). Volume 2673 of CEUR Workshop Proceedings., CEUR-WS.org, pp. 92–96 (2020)
68. Elkoumy, G., Pankova, A., Dumas, M.: Utility-aware event log anonymization for privacy-preserving process mining. EMISA Forum **41**(1), 37–38 (2021)
69. Maatouk, K., Mannhardt, F.: Quantifying the re-identification risk in published process models. In: ICPM Workshops, vol. 433, pp. 382–394. LNBIP. Springer (2021). https://doi.org/10.1007/978-3-030-98581-3_28
70. Zaman, R., Hassani, M.: On enabling GDPR compliance in business processes through data-driven solutions. SN Comput. Sci. **1**(4), 210 (2020)
71. Di Francescomarino, C., Ghidini, C.: Predictive process monitoring. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
72. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Data-driven process discovery - revealing conditional infrequent behavior from event logs. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 545–560. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_34
73. Vidgof, M., Djurica, D., Bala, S., Mendling, J.: Cherry-picking from spaghetti: multi-range filtering of event logs. In: Nurcan, S., Reinhartz-Berger, I., Soffer, P., Zdravkovic, J. (eds.) BPMDS/EMMSAD -2020. LNBIP, vol. 387, pp. 135–149. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49418-6_9
74. van der Aalst, W.M.: A practitioner's guide to process mining: limitations of the directly-follows graph. Procedia Comput. Sci. 164, 321–328 (2019). (CENTERIS 2019 - International Conference on ENTERprise Information Systems/ProjMAN 2019 - International Conference on Project MANagement/HCist 2019 - International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN/HCist 2019)
75. Figl, K.: Comprehension of procedural visual business process models - a literature review. Bus. Inf. Syst. Eng. **59**(1), 41–67, 101962 (2017)
76. Mendling, J., Djurica, D., Malinova, M.: Cognitive effectiveness of representations for process mining. In: Polyvyanyy, A., Wynn, M.T., Van Looy, A., Reichert, M. (eds.) BPM 2021. LNCS, vol. 12875, pp. 17–22. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85469-0_2
77. Brons, D., Scheepens, R., Fahland, D.: Striking a new balance in accuracy and simplicity with the probabilistic inductive miner. In: ICPM, pp. 32–39. IEEE (2021)

Industrial Perspective and Applications

Status and Future of Process Mining: From Process Discovery to Process Execution

Lars Reinkemeyer(✉)

VP Customer Transformation, Celonis SE, Munich, Germany
lars@reinkemeyer.de, l.reinkemeyer@celonis.com

Abstract. During the last two decades Process Mining has seen a rapid global adoption: first in academics and then in corporate business. It has evolved into a foundational technology, allowing users to discover actual process flows with unprecedented transparency, speed, and detail. In a business environment Process Mining has no purpose of its own, but companies leverage it to identify process inefficiencies, improve process execution and ultimately drive value. Process discovery and transparency does not provide immediate business value, but requires specific use cases combined with human intelligence to identify and deploy levers for process improvement. In this article we argue that the future focus and evolution of Process Mining shall not focus on lateral expansion - i.e. with further processes and discoveries - but vertically by enhancing the depth of added value for business users with artificial intelligence, proactive and predictive enablement and other levers which boost process execution. In essence, focus should be on deploying smarter technologies for driving business value in process areas where Process Mining has shown impact.

1 Setting the Stage

1.1 The Evolution of Process Mining in Operational Business

Process Mining was invented at the end of last millennium by Wil van der Aalst and has seen a strong adoption by academics in the first decade of this millennium. In the second decade of this millennium companies started to use Process Mining for transparency, to discover, understand and improve actual processes. To this respect, numerous use cases i.e. in horizontal support functions such as Procurement and Order Management have been defined and deployed by companies like BMW, Siemens, Uber and many more around the world, across all industries, in organizations of any sizes and for processes along the whole value chain, as the following selected examples show:

- Procurement Experts use Process Mining for order processing e.g., to discover duplicate payments, payment term deviations and maverick buying.
- Logistic Experts use Process Mining to discover reasons for late deliveries, improve supply chain resilience and assure on-time deliveries.
- Order Manager use Process Mining to discover customer order processing, identify inefficiencies resulting from rework and improve customer satisfaction.

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 405–415, 2022.
https://doi.org/10.1007/978-3-031-08848-3_13

- Plant Managers use Process Mining to find bottlenecks in manufacturing processes, conduct value stream analysis and improve efficiencies.
- Sustainability Manager use Process Mining to discover operational root causes for waste and CO_2 impact of single process decisions.

All these sample have in common that Process Mining is used to discover actual process flows. Then human intelligence is applied to interpret the achieved transparency, identify root causes for process inefficiencies, and turn these into business value. However, the evolution on Process Mining should progress, similar to the evolution of the imaging method in healthcare.

For an analogy, the evolutions in Healthcare and Process Mining show many similarities: prior to inventing xRay at the end of the 19th century, a Medicus needed to guess what is happening in the human body and what are the root causes for a particular disease. Similarly – before using Process Mining – process experts had to use process models and subjective assumptions to guess actual process flows and define process improvement. The invention of xRay allowed to discover the root causes for diseases and thus decide on appropriate remediation. Similarly, Process Mining enables users to discover process gaps and decide on improvements. Imaging methods have become smarter and capable to interpret the images, identify diseases and propose curative measures. Furthermore, medical devices are trained not only to "read" the images, but also to propose and conduct treatments. In a similar form, we expect Process Mining to develop more "intelligence": in a first step by automatically identifying process gaps and proposing measures for remediation to the users who will then execute the action. And in the future even "learn" to execute process activities autonomously based on defined criteria, with only exception based human interference.

1.2 Achievements in the Decade Starting 2010

Process Mining has seen some impressive developments in the last decade and enabled many companies towards a data- and fact-based culture, using single sources of truth for process assessment and optimization. The evolution from Business Process Modeling (BPM, the design how processes should happen) to Process Mining (full transparency how processes actually happen) allows organizations to understand and improve processes.

Many organizations started using Process Mining in single functional silos (e.g. audit, procurement) and then expanded the usage across different functions. While there is an amazing variety of use cases, experience shows that the biggest impact is achieved in the core processes Accounts Payable (A/P), Accounts Receivable (A/R), Purchase-to-Pay (P2P) and Order-to-Cash (O2C). These horizontal support processes are critical to any company and are typically executed in transactional systems, providing sufficient digital event logs with a high degree of standardization, and requiring a high degree of automation. Many companies, which started their Process Mining journey with a focus on these core processes, achieved short term transparency and operational impact e.g., by eliminating duplicate payments, identifying payment term deviations, and reducing rework [1].

1.3 Hurdles and Challenges

While the concept of Process Mining has seen a rapid expansion, operational adoption has faced key challenges:

- Focus on the right *Purpose*: some initial use cases – built with passion for this exciting new technology – did lack appropriate purpose. Providing process transparency is great, but it is only valuable if it serves a clearly defined business purpose, e.g., improving throughput times or on-time delivery. This purpose must be defined prior to starting any Process Mining project and the deployment should be continuously measured regarding impact and value.
- Engaging the right *People*: as part of human nature, most people are reluctant to leave their comfort zone. Discovering process inefficiencies does not always resonate well with people, who have designed or operated a particular process for many years. Reactance is a typical reaction which needs to be managed proactively with appropriate change management and initially engaging people who are open to change.
- Process Mining can enable an *operational transformation*, which requires the right organizational setup. Experience shows, that the majority of companies, which have successfully deployed Process Mining, established a Center of Excellence as central accelerator to drive change, process transformation and value.
- *Technical performance*: since my very first days with Process Mining, I have personally seen a continuous race between business requirements and technical performance. Once the concept of Process Mining was understood and data access had been arranged, the demand for transparency grew exponentially in the organization. This obviously goes in hand with an exponentially growing demand for data, which must be extracted, analyzed and presented to the operational users. Technology is required to assure performance for high customer experience, and has seen a similar exponential evolution to match demand, with cloud technology and standard connectors being only two sample for innovation.

1.4 The Power of Processes

Processes represent the lifeblood of any organization and efficient process execution is a critical success factor to stay competitive. Amazon is probably one of the most efficient companies, when it comes to process execution and the following quote from Jeff Bezos shows his reluctance to adopt rigid process frameworks, but rather continuously adjust and improve processes to maintain Day 1 efficiency and agility:

"You stop looking at outcomes and just make sure you're doing the process right. Gulp. It's not that rare to hear a junior leader defend a bad outcome with something like, "Well, we followed the process." A more experienced leader will use it as an opportunity to investigate and improve the process. The process is not the thing. It's always worth asking, do we own the process or does the process own us? In a Day 2 company, you might find it's the second." [2].

2 The Future of Process Execution

Process Mining has enabled thousands of organizations around the world to better understand their actual processes, to fuel data- and fact-based discussions and thus derive process improvements. However, the ultimate goal for any organization must be to minimize transactional cost, i.e. the cost induced by executing business processes. The digital age has seen the raise of an increasing number of digital native companies, which are built on highly efficient, automated processes e.g. for sales order or purchase order processing. Think of Amazon's Marketplace, with a maximum degree of automation in order processing leading to a minimum degree of transactional costs. Traditional companies, with a grown legacy of IT infrastructure, are challenged to compete against these digital native companies. While Process Mining is focused on process discovery, process execution focuses on enabling the companies to execute processes more efficiently and thus reduce transactional cost by leveraging smart technologies.

2.1 Process- and Organizational Transformation

Leveraging Process Mining and process execution implies a significant transformation in the way a company operates. Transformation as a popular buzzword comes in multiple flavors, such as e.g. digital transformation or transformation of the business model – which shall not be discussed in further detail. Our focus shall rather be on process transformation and transformation of the operating model, thus focusing on the way a company executes its processes and how it is setting up operations to drive change. Processes are essential for the value generation of any company, and typically show a high degree of inertia as employees got used to a certain way of doing things and typically show reluctance to change. While Process Mining can discover process inefficiencies and process execution can provide solutions for increasing efficiency, operational transformation is a crucial factor for success, which needs to be managed proactively. Experience shows that many Process Mining projects fail due to organizational / human reluctance in respect to change. Tools and technology represent only one side of the equation for an organizational transformation, as the following statement shows: "While cutting edge technology and talent are certainly needed, its equally important to align a company's culture, structure, and ways of working to support broad AI adoption. In most firms that aren't born digital, mindsets run counter to those needed for AI" [3].

Driving process- and organizational transformation implies a range of different success factors:

- Value focus: the target value needs to be defined, e.g. strategically (transforming towards a data driven organization) or operational in measurable value such as e.g. reduction in working capital or rework, improvement of payment term deviations or degree of automation. A very tangible definition of measurable value, which is turned into operational targets, is crucial for a successful transformation.
- Executive sponsorship with a clear focus on targets: all transformational efforts must be sponsored by a senior executive for priority and guidance. Operationalization via defined strategic targets, which are ideally aligned with individual incentives, have yielded maximum success.

– Change Management as a conscious approach to prepare, support and enable individuals and teams to adopt process transformations.
– Organizational setup with a defined accelerator to drive transformation, e.g. a Center of Excellence (CoE), which is enabled and empowered by an executive sponsor to drive process transformation across the organization.

2.2 Trends

The following discussion on Trends results from numerous discussions with process owners and experts, other Process Mining evangelists and market players.

2.2.1 Intelligent Process Execution

While Process Mining has enabled users to discover process inefficiencies with human intelligence, the concept of intelligent process execution builds on this discovery and supports users by providing just that kind of information which is relevant. While Process Mining can screen millions of purchase orders, intelligent process execution provides the individual users with only those purchase orders which require attention or call for immediate action. While Process Mining can discover millions of manual activities, intelligent process execution enables the user to execute multiple activities in one step in a suitable user interface. In essence, intelligent process execution takes Process Mining to the next level by leveraging AI, proactive and predictive solutions for the benefit of providing users only with the relevant information and smart forms of process execution.

2.2.2 Proactive Solutions

While big data and new tools allow unprecedented transparency, most software provides insights for users to search for relevant issues. Process Mining allows insights where users can identify e.g. late deliveries, rework effort, process delays and much more. But should users apply human intelligence to search for relevant issues, spending high effort and wasting precious time while searching for relevant issues? We don't think so. Virtual assistants should provide proactive, customized and individual support. Intelligent process execution is capable to "learn" current operations and develop skills to propose relevant exceptions proactively to the users. The software is evolving into a smart companion, which is capable to discover the operational process, understand exceptional issues and propose these proactively to the user. E.g. overdue payments can be presented to the user per push-mail or pop-up message, delayed customer deliveries will be flagged out and potentials for automation proposed. Dedicated execution Apps condense execution gaps or exceptions for the user to decide how to proceed in these cases.

2.2.3 Predictive Solutions

Upcoming events can be predicted to enable users to take preventative measures. It might – to take an example from procurement - be helpful to get a prediction, which purchase order will not be delivered on time. Equally, in logistics it is helpful to get predictions, which shipment will not be delivered on time. Based on historical data, predictions are calculated and presented with probability thresholds: as one example, predictions can identify all supplies, which will not be received on an expended date with a defined probability threshold. Those kinds of solutions have been developed for several years, e.g. based on algorithms programmed with Python on R-server, analyzing open and closed orders including times for process execution, leading to a vast number of operational execution support cases.

2.2.4 Usability

Application development shows a strong focus on the consumer, with a requirement to provide intuitive user interfaces (UIs) which are fun to use and quick in interaction. Usability is equally relevant for standard Apps as well as for individual data analytics:

- Standard Execution Apps: Horizontal core processes such as A/P or A/R should be standardized to a maximum and can be executed with standard Apps, which provide the users with a convenient way of interaction. Execution Apps can provide one single layer across multiple traditional transactional systems and thus allow users to focus on relevant process exceptions and executions in one user interface.
- A second trend supports dynamic sets of data analytics cubes, which can be consumed individually and intuitively, thus moving away from predefined, static Dashboards which had been designed centrally in the past to provide standard transparency frameworks.

2.2.5 Impact on Digital Workforce and Data Democratization

With the trends for process transformation and organizational change, methodologies such as Process Execution and AI gain increasing relevance. New digital tools and data democratization have a major influence on the digital workforce, with changing requirements and roles. Process Execution enables and supports data analysts to drive execution efficiency, but at the same time requires new skills, roles and responsibilities. A new generation of experts has been educated, with a thorough understanding of computer science, data and IT.

The mindset of a digital workforce differs significantly to the traditional mindset e.g. regarding access to data: while the traditional approach was extremely restricted in respect to data access – typically with "eyes only" principles - the democratization of data is a trend which drives major change towards open data access. Access to data as well as the preparation and analytics of data was traditionally rather a task conducted by specialized experts in organizational silos. As the general perception changes towards an understanding, that data is essential for today's business, data ubiquity, accessibility, and usability for everybody becomes a standard requirement.

2.2.6 Data Collection and Preparation

While projects in the past required high effort to identify, collect and prepare event logs, there is a trend towards usage of standard connectors. In particular structured data from homogeneous systems (e.g. SAP ERP) can easily be identified and read by standard extractors. Discovering e.g. P2P processes across multiple systems has become possible with much less effort due to standard connectors, which require little customization. Automated discovery of event logs is expected to become possible, building on the growing experience gained from data preparation and technical innovations. Machine learning algorithms will understand the format and structure of data in similar source systems, facilitating an automation of data collection and preparation. In addition, transactional ERP systems as well as workflow platforms such as Pegasystems and ServiceNow play an increasing role for process automation and execution. Data collection and preparation across different types of platforms allows seamless execution for e.g. financial and customer data.

2.2.7 Task Mining

While Process Mining is based on event logs from backend systems, Task Mining allows for process insights based on recorded activities from individual users, typically from front office systems. Samples for captured activities are mouse clicks, keystrokes, application inputs and field entries, thus providing a much deeper understanding of an individual working behavior. Task mining allows to discover actual human activities with the purpose to identify potentials for improvement. Any activity can be recorded, including phone calls, eMail or excel documentation, where no log files are available, and data is stored in unstructured format. While task mining provides a micro-picture of individual behavior and thus allows optimization of individual tasks, it does not allow insights into overarching operational processes, which can only be visualized with Process Mining. Task mining typically complements Process Execution as a "magnified" analysis of actual user behavior e.g. in Call Centers. Solutions have matured quickly and become a valuable support for operational experts.

2.2.8 Cloud Technology

Storing digital traces in a public cloud has become commonly accepted and will support the possibilities to use proven algorithm for extraction and customization of data, deploy standard use cases and benefit from analytics available in the Cloud. Hosted AI is expected to become attractive and available in form of Software as a Service (SaaS) and accessible with standardized Application Programming Interfaces (APIs) to provide applications, technology, and best practices to a wide number of users.

2.2.9 IIoT Platforms

The Industrial Internet of Things (IIoT) has set the technical foundation for an extensive access to event logs, as devices become connected to an internet hosted platform, thus allowing easier access to digital footprints, which are generated from these devices. IIoT platforms such as MindSphere already today receive data from millions of single devices, including relevant event logs. Value can be generated for example by understanding manufacturing processes based on the event logs from multiple machines – even across machines at different sites. The collection of event logs from different machines, sites, and companies on one common IIoT platform will allow new use cases such as the visualization of cross-company supply chain processes or inter-company benchmarking. As a crucial benefit, the IIoT platforms provide a standardized and secured environment and protocol, which has been adopted to industrial requirements.

2.3 Midterm Future

2.3.1 Self-learning and -Optimizing Systems

With AI becoming more mature and suitable to assist even in environments where profound high domain knowledge is required, technology will evolve towards self-learning and -optimization. Imagine a process execution system, which is autonomously capable to learn, i.e. to detect and resolve process inefficiencies. Like self-driving cars, there will be "self-driving" Process Execution tools which are capable to learn factors which determine efficient process flows and autonomously suggest or even initiate measures to optimize process efficiency including optimization of variants and reduction of process exceptions.

2.3.2 Artificial Intelligence

While the impact of AI, which has been experienced in operational use cases to date, has been limited, it will grow up to its promises. Some innovative providers show exciting use cases with virtual process analysts discovering and documenting actual processes by imitation learning. A virtual digital companion learns from actual and optimum process handling and is thus trained to become an accepted artificial co-worker, understanding also complex domain know-how, which is the big challenge in the B2B environment. Virtual companions are trained to identify and remediate process flaws, which can start with simple, repeatable process tasks such as the removal of delivery blocks. Besides all excitement about AI, it must remain explainable in order to ensure ethical data usage with clear transparency about what and how AI is applied. AI governance will play an increasing role and will have a significant impact on the acceptance of these new technologies in particular in a corporate environment.

2.3.3 Benchmarking

Process Mining makes process efficiency measurable and transparent. As it is based on big data and facts, it is predetermined for benchmarking purposes. Standard processes such as P2P and O2C will be benchmarked on operational performances such as automation rate, throughput time or rework across different organizations. With digital traces available on standard platforms and in the cloud this will also become available as a self-service, where companies can access benchmark data – based on appropriate data anonymization – to assess their own performance versus other market players. And consulting companies will be able to lift cross-company benchmarking analysis to a new level of data foundation, as benchmarking can be conducted based on the full set of all relevant events from different players.

2.4 Longterm Future

2.4.1 Inter-Company

The long term perspective provides significant economic and ecologic benefits through optimization of cross-company supply chains, based on data from different companies and sources. Process optimization will become possible for inter-company value chains, including supplier, manufacturer, freight forwarder and customer. Companies like Slync already today offer multi-party supply chain interaction with a high degree of automation, across different organizations and multiple data sources. The value proposition offers logistics orchestration across manufacturers, suppliers, freight forwarder and customers. With Process Execution, this could be taken to a new level by understanding the extended end-to-end process chains. On-time delivery, integrated manufacturing and optimization of stock/working capital are just a few benefits of a transparent supply chain processes, which can be monitored and managed with the support of Process Execution. Empowering the business partner with access to own process data will allow all parties to benefit. Besides economic benefits this will lead to a sustainable ecological optimization due to the wholistic approach, which will allow to reduce e.g. the number of empty deliveries, reduce waste and allow for a better resource management and more sustainable business.

2.4.2 Sustainability

The sustainability revolution should be supported by technological innovations such as Process Execution. Think about process inefficiencies in your immediate environment and how better process efficiency could support sustainability: from traffic congestions to waiting times in hospitals, from wasted time in call center queues to waiting times for bureaucratic decisions, from delayed goods deliveries to delayed flight arrival. Process inefficiencies are omnipresent, producing friction, waste and avoidable emission. Understanding the end to end processes allows to track down inefficiencies and reduce waste in time and resources. While Supply Chain Management is probably the primary field, where Process Mining can support a sustainability revolution, CRM and other functions can equally support as ecological driver. The management of resources in ERP systems (financials, materials, assets and HR) will become more efficient with Process Mining, thus allowing to optimize scarce resources. In a world with more than 7.9 billion people and increasing issues due to limited resources this will become a strong purpose.

2.4.3 B2C

While the primary focus on Process Execution to date has been on business-to-business (B2B) processes, there is a huge potential for process optimization in the business-to-consumer (B2C) field. Understanding consumer interactions with the additional dimensions of time and activity sequences allows to better interpret and predict e.g. consumer behavior. B2C use cases could include for example activity tracking for the timing and sequence of user clicks on shopping pages or in social media platforms. Understanding of strategies, how users approach challenges such as search for restaurants or music, appear valuable and might allow for trail prediction. As another example, the insight into the search sequence for web offerings could – based on large amounts of activities – not only be interesting for psychometric analysis, but also for product management and sales.

2.5 Vision of a Digital Enabled Organization

Imagine an organization which has been automated for most standard processes, such as procurement of indirect material, financial transactions, order deliveries and customer order processing. Standard tasks are conducted automatically, supported by an AI, which is capable to learn not only how to execute standard cases, but also minor exceptions, conducting immediate actions and corrections. This "intelligent system" processes most of all activities with zero human touch, and humans only interfere exception based, thus providing a high process reliability at minimum transactional cost.

Data ingestion from diverse source systems is supported by AI, which allows to identify and customize structured and unstructured data from various sources such as ERP or workflow systems. Cloud technology is commonly established as basis for data hosting, collaboration, and data mining, with the application providers applying continuous monitoring and optimization. Streamed event data allows real time process analytics for immediate reaction e.g. for customer interaction. Platforms offer standard Apps for process execution in a secure environment and share best practices for process handling and monitoring.

As most operational processes have been fully automated, the focus of Process Execution changes. Based on this vision, there will be less demand for transparency and discovery in respect to today's focus areas. Standard support processes such as P2P and O2C provide decreasing marginal benefits, as they are mostly optimized and the focus shifts towards more challenging processes such as e.g. customer interaction, manufacturing, HR and legal proceedings. Besides inter-company automation, process optimization is happening cross-organization in integrated supply chain process flows. Exception based activities remain in focus, as they require optimization with appropriate digital tools. Similar to tele-medicine, remote diagnosis and optimization of processes based on smart automation will be available through dedicate Process Mining Analysts, who are alerted by intelligent virtual assistants, which conduct a continuous real-time monitoring and provide predictive and proactive alerting.

The role of humans has changed significantly: mundane tasks have been completely automated and new tasks and roles emerged instead. The focus of human responsibility has changed towards data analytics and steering, using tools which are provided by the digital enabled organization. Process analysts use digital tools such as virtual assistants, which collect data from Process- and Task Mining, thus empowering the digital enabled organization. Value generation shifts towards service innovation. In their book "Dreams and Details" Snabe and Trolle describe how to reinvent business from a position of strength and with a compelling vision. An innovative "'Digital Enabled Organization" could provide the dream to set the mindset and framework to unleash the human and digital potential.

As a positive ecological contribution, the process optimization has yielded significant reduction in carbon footprint e.g. due to reduction of empty trips and optimization of routings. Transactional costs have been reduced to a minimum.

3 Conclusion

While the first two decades of Process Mining have been focused on transparency and discovery, the real impact in a corporate environment is driven through intelligent execution management. Process Mining provides an excellent foundation, which will be enhanced with standard process execution Apps, common extractors, process transformation capabilities and artificial intelligence in order to execute business processes in an easier, smarter and more efficient manner. Thus Process Mining is the base for a much wider field which is still to be developed.

References

1. Reinkemeyer, L.: Process Mining in Action. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-40172-6
2. Isaacson, W.: Invent and Wander: The Collected Writings of Jeff Bezos. Blackstone Publishing, Ashland (2021)
3. Hardvard Business Review 08/19 (2019)

Using Process Mining in Healthcare

Niels Martin[1,2](✉), Nils Wittig[3], and Jorge Munoz-Gama[4]

[1] Hasselt University, Martelarenlaan 42, 3500 Hasselt, Belgium
niels.martin@uhasselt.be
[2] Research Foundation Flanders (FWO), Egmontstraat 5, 1000 Brussels, Belgium
[3] KMS Vertrieb und Services AG, Inselkammerstraße 1,
82008 Unterhaching, Germany
nils.wittig@kms.ag
[4] Pontificia Universidad Católica de Chile, Av. Vicuña Mackenna 4860,
7820436 Macul, Chile
jmun@uc.cl

Abstract. This chapter introduces a specific application domain of process mining: healthcare. Healthcare is a very promising domain for process mining given the significant societal value that can be generated by supporting process improvement in a data-driven way. Within a healthcare organisation, a wide variety of processes is being executed, many of them being highly complex due to their loosely-structured and knowledge-intensive nature. Consequently, performing process mining in healthcare is challenging, but can generate significant societal impact. To provide more insights in process mining in healthcare, this chapter first provides an overview of healthcare processes and healthcare process data, as well as their particularities compared to other domains. Afterwards, an overview of common use cases in process mining in healthcare research is presented, as well as insights from a real-life case study. Subsequently, an overview of open challenges to ensure a widespread adoption of process mining in healthcare is provided. By tackling these challenges, process mining will become able to fully play its role to support evidence-based process improvement in healthcare and, hence, contribute to shaping the best possible care for patients in a way that is sustainable in the long run.

Keywords: Process mining · Healthcare · Evidence-based process improvement

1 Introduction

The prior chapters of this book introduced various process mining topics. In contrast to these preceding chapters, this chapter focuses on introducing a specific application domain of process mining. In particular, this chapter focuses on *healthcare*. In process mining research, healthcare illustrations are often used to demonstrate new techniques, or a healthcare problem is the starting point

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 416–444, 2022.
https://doi.org/10.1007/978-3-031-08848-3_14

of the research project altogether [55]. This can be, at least partly, explained by the great societal value related to efforts to improve the healthcare system. In many countries, the long-term sustainability of the healthcare system is an important societal issue due to trends such as the increasing life expectancy, and the raising prevalence of chronic diseases [29]. Improvements in terms of healthcare processes is an indispensable piece of the puzzle to sustain the healthcare system, while continuously improving the quality of care delivered to the patient.

Within the healthcare domain, many different processes are being performed in a wide variety of healthcare organisations. Many processes in healthcare are complex as they are loosely-framed and knowledge-intensive [20,55,58]. While the former indicates that healthcare processes can typically be executed in a large number of distinct ways [58], the latter indicates that the trajectory that is followed strongly depends upon complex decisions made by knowledge workers such as physicians and nurses [20]. These healthcare processes are increasingly being supported by health information systems [53], which capture data about the real-life execution of a process in their databases. This data can be leveraged to compose an event log, the key input for process mining [55].

There has been a steady growth in research interest on process mining in healthcare in recent years [17]. Despite the great potential of process mining to support process improvement in healthcare and the increasing number of methods specifically designed for the healthcare context, the systematic uptake of process mining in healthcare organisations outside the research context is still fairly limited [55]. Hence, there are still challenges ahead that need to be overcome, which is consistent with the fact that process mining in healthcare is a rather young research area. Moreover, healthcare is a highly dynamic field as processes change due to advances in, for instance, medicine and technology [29,55]. For instance, the increasing presence of wearable devices and mobile health applications provides opportunities to collect richer data about a particular process, but also presents new challenges, e.g. in terms of merging all data sources [37,55]. Even though it will require continued efforts, it is worthwhile to benefit from opportunities and tackle challenges as it will enable process mining to fully play its pivotal role to instigate evidence-based process improvement in healthcare [55].

The goal of this chapter is to introduce the reader to healthcare as an application domain for process mining. To this end, the remainder of this chapter is structured as follows. Section 2 provides a primer on healthcare processes and healthcare process data, with an emphasis on its particularities. Section 3 introduces the reader to the common use cases of process mining in healthcare from a research point of view. Section 4 discusses a case study, which illustrates the potential of process mining in the context of a specific hospital. Section 5 outlines the key open challenges that the community is confronted with when it aspires a broad uptake of process mining in healthcare. The chapter ends with a brief conclusion in Sect. 6.

2 A Primer on Healthcare Processes and Process Data

Before providing an overview of common use cases in the process mining from healthcare literature, this section sets the stage by providing an overview of healthcare organisations and healthcare processes (Sect. 2.1). Moreover, the particularities of healthcare processes and healthcare process data are introduced (Sect. 2.2).

2.1 Healthcare Organisations and Healthcare Processes

Some readers might implicitly equate healthcare to the care that patients receive in a hospital. Hospitals, either general hospitals or specialised hospitals [57], play an important role in the provision of healthcare services. As will become apparent in Sect. 3, many process mining applications are also situated within the hospital context. However, it should be noted that curative care, i.e. care focused on the treatment of diseases to increase life expectancy [88], is organised in various types of healthcare organisations [57]. For instance: long-term care facilities provide care to patients suffering from a chronic disease or patients needing long-term rehabilitation after a hospital discharge. Psychiatric care organisations, in their turn, provide therapy for patients with mental problems. Home-based care organisations, another category of healthcare organisations, deliver care services in the comfort of the patient's home [57].

Within a particular healthcare organisation, a wide variety of healthcare processes is being performed. A basic distinction between medical treatment processes and organisational processes is introduced by Lenz and Reichert [46]. *Medical treatment processes*, also commonly referred to as clinical processes, have a direct link to the patient and are connected to the therapeutic-diagnostic cycle. This implies that, in these processes, healthcare professionals takes informed decisions regarding the patient's diagnosis or therapy based on medical knowledge and the available patient-related information. *Organisational processes*, in their turn, cover all processes that support medical treatment processes by coordinating actions between different healthcare professionals and supporting staff, potentially even belonging to various departments. Examples include appointment or procedure scheduling processes, as well as logistical processes of patients or goods [46,67].

An alternative categorisation of healthcare processes is provided by Mans et al. [52]. Their classification solely takes processes that are directly related to the patients into account, but considers both medical activities as the preparation of these activities (such as booking the appointment) as being part of the same process. Against this background, Mans et al. [52] make a distinction between elective care processes and non-elective care processes. The execution of *elective care processes* can responsibly be postponed for several days or weeks. Within this subcategory, a further distinction is made between standard, routine, and non-routine care processes. For *standard care processes*, a structured treatment trajectory is available, containing information about the activities that need to be performed, as well as the timing that needs to be respected.

In a *routine care process*, various treatment trajectories can be followed to obtain an outcome that is typically known. The latter does not hold for *non-routine care processes* as a physician will need to determine the next step in the treatment trajectory based on the patient's reaction on the current process step. While elective care can be postponed for several days or weeks, *non-elective care processes* refers to unexpected medical treatments that need to be performed promptly. Here, a distinction is made between *emergency care processes*, which should be executed immediately, and *urgent care*, which can be postponed for a limited period of time (e.g. a few days) [52].

From the previous, it follows that healthcare is a highly versatile domain, with a large variety of healthcare organisations and a mix of different processes being executed at these organisations. These processes can be fairly structured (e.g. standard care processes) or highly unstructured (e.g. non-routine care processes) [52]. The close interconnection between processes, even across different healthcare organisations, adds to the complexity of the healthcare domain. For instance: the trajectory of a patient suffering from a chronic disease might consist of surgery at a specialised hospital, several check-ups at a local general hospital, as well as multiple therapies taken at home under the supervision of a home nurse [55]. Even within a single healthcare organisation, processes are closely intertwined as, e.g., efficiently carrying out surgical processes depends on the timely execution of logistical processes, both regarding patient transportation and the material flow.

2.2 Particularities of Healthcare Processes and Process Data

To really grasp the challenging nature of healthcare as an application domain for process mining, it is important to understand the particularities of healthcare processes and healthcare process data. Munoz-Gama et al. [59] defined ten distinguishing characteristics of healthcare processes, which also impact the process data that will be recorded. While some of these characteristics might also be relevant for other sectors, their combined occurrence in the healthcare context needs to be reckoned with and will generate challenges when conducting process mining analyses. The ten key particularities of healthcare processes and healthcare process data, as defined in Munoz-Gama et al. [59], are discussed in the remainder of this subsection.

Exhibit Significant Variability. An important contributing factor to the complexity of healthcare processes is their significant variability [63,67]. Variability is caused, amongst others, by the diversity of activities that can be performed (e.g. a wide variety of examinations and treatments) in various orders, and the different characteristics of patients (e.g. they can suffer from various combinations of co-morbidities, influencing the way the process is executed) [67]. As a consequence, in many healthcare contexts, almost every case will have a unique trajectory through the process, leading to challenges within the context of, e.g., control-flow discovery [59].

Value the Infrequent Behaviour. In many domains, process mining is used to better understand the typical behaviour of a process. Hence, as infrequent behaviour would complicate, e.g., the discovered control-flow model, it is often removed in the pre-processing stage of a process mining project [15]. However, in healthcare, infrequent behaviour can be a source of valuable knowledge about the process. It might, for instance, highlight infrequent treatment paths that result in the same clinical outcome, unveiling knowledge about alternative treatment options for a particular disease [22,59]. Understanding infrequent behaviour is important as solely focusing on models representing the typical behaviour could generate blind spots, which constitute missed innovation opportunities for healthcare processes [59].

Use Guidelines and Protocols. Within the field of medicine, various clinical practice guidelines and protocols are available, which build upon evidence-based information on a certain topic [79,87]. This implies that, for clinical processes, reference processes are often available, which does not hold in many other domains [35]. This opens opportunities for process mining to, e.g., analyse the adherence to these guidelines and protocols [34,59].

Break the Glass. While clinical practice guidelines and protocols aim to achieve standardisation in clinical processes, medical doctors and healthcare professionals might need to deviate from guidelines and protocols when confronted with specific situations. For example: the discovery of specific co-morbidities of a patient might require an alternative course of action [62,72]. Another situation that might require a deviation from protocols is an unexpected surge in the number of arriving patients that should be coped with by a department [59]. The occurrence of such *'break the glass'* situations will also be reflected in the data, highlighting the crucial importance to take into account context information when using process mining in healthcare to fully understand the process behaviour [59,80].

Consider Data at Multiple Abstraction Levels. In a healthcare context, data about the execution of a process can originate from various data sources, both for clinical processes and organisational processes [45,55]. These data sources will capture data at multiple levels of abstraction. Medical equipment such as surgical robots or wearable devices will often generate large volumes of very fine-grained data, which should be aggregated to retrieve meaningful patterns [59,85]. High-level data, typically recorded in administrative systems, tends to be directly interpretable, but might provide an insufficiently detailed view on the process. Hence, when performing process mining in healthcare, it might be required to integrate data from various sources, potentially bridging clinical and administrative systems, as well as different data abstraction levels [59].

Involve a Multidisciplinary Team. Healthcare processes typically have a multidisciplinary character, with healthcare professionals (physicians from various disciplines, nurses, etc.) and supporting staff with various backgrounds being involved [55,67]. Given the critical importance of expertise from the healthcare domain, a multidisciplinary team needs to be involved during all stages of a process mining initiative, ranging from the specification of the problem to the translation of process mining insights to practical actions. This implies that attention needs to be attributed to the use of the appropriate medical terminology and customs to assure mutual understanding [59].

Focus on the Patient. When considering healthcare processes, the key role of the patient should be emphasised. Patients are, directly or indirectly, at the core of nearly all healthcare processes. Hence, when performing process mining in healthcare, specific attention should be attributed to support the provision of patient-centred care, a key care quality indicator [11]. When focusing on the patient journey, i.e. the trajectory of a patient over the course of a disease or treatment [49], it is important to note that (s)he typically receives services from various healthcare organisations (e.g. the hospital, the general practitioner, and the physiotherapist). This also causes the patient journey data to be spread over several organisations, with its associated challenges [55,59].

Think About White-Box Approaches. Recent advances in artificial intelligence and machine learning have provided techniques to support physicians in taking complex clinical decisions. One of the biggest hurdles for the adoption of such techniques is the physician's reluctance to use systems that they do not fully understand, i.e. to use *black-box* approaches [65]. Hence, to support decisions in a healthcare context, there is a need for *white-box* approaches, enabling healthcare professionals to understand where recommendations originate from. Process mining is perceived as such a white-box approach [39]. Nevertheless, the understandability of process mining outcomes for healthcare professionals should remain a permanent point of attention [55,59].

Generate Sensitive and Low Quality Data. Healthcare processes, especially clinical processes, generate sensitive data as it typically contain information regarding a patient's health condition, co-morbidities, ongoing treatments, etc. Consequently, ethics in general and *data privacy* in particular need to be first-class citizens when working with healthcare processes [74]. Moreover, strict regulations are typically in place regarding the use, storage and transfer of sensitive healthcare data [64]. Besides data privacy, *poor data quality* also characterises data collection regarding healthcare processes [54,86]. Data quality, a topic which has been discussed in Chapter 6 [18] is highly relevant in the healthcare domain, where data might suffer from various quality issues such as missing events, incorrect timestamps and imprecise timestamps [52,86]. One of the key reasons for data quality issues in healthcare is the fact that many events are recorded after a manual interaction between a healthcare professional and an information system. This might cause inaccuracies in the recorded data as some actions might not be recorded in the system, other actions might be recorded in the system well after they have been executed, etc. Data quality issues have to be handled with great care when conducting process mining in healthcare [59].

Handle Rapid Evolutions and New Paradigms. As the healthcare domain is rapidly and continuously evolving, this also holds for processes in healthcare. Changes are induced both by advances in clinical research, leading to changes in diagnostic or treatment processes [24], as well as advances in technology, e.g. the rise of remote monitoring due to the development of robust mobile health solutions [76]. New healthcare paradigms also surface, which also have an impact on healthcare processes. For instance: patient-centred care has become a core paradigm in healthcare, implying that care should attribute significant attention to the needs and preferences of the individual patient [66]. When working on process mining in healthcare, researchers and practitioners should be aware of these rapid evolutions and emerging new paradigms, as well as be able to cope with them [59].

3 Use Cases in Process Mining in Healthcare Research

Against the background of the previous section, this section aims to highlight some typical use cases for process mining in healthcare as reported in published research articles. While many of the papers that will be referenced below make important methodological contributions, the focus of the discussion in this section is mainly on how process mining techniques were applied in a particular healthcare context. To structure the outline, the six process mining types introduced in Chapter 1 [1] are used: process discovery (Sect. 3.1), conformance checking (Sect. 3.2), performance analysis (Sect. 3.3), comparative process mining (Sect. 3.4), predictive process mining (Sect. 3.5), and action-oriented process mining (Sect. 3.6). At the end of the section, some recommendations for further reading are provided (Sect. 3.7).

3.1 Process Discovery

Process discovery focuses on the discovery of a process model from an event log. As holds for process mining in general, process discovery is also, by far, the most prominent use case of process mining in healthcare [17,37]. Papers on process discovery in healthcare typically center around the discovery of the control-flow, i.e. the order of activities, from an event log [17].

When focusing on control-flow discovery, various algorithms have been used to *automatically* retrieve a visualisation of the activity order from an event log. Based on a literature review, Guzzo et al. [37] conclude that Heuristics Miner is the most commonly used algorithm, followed by Fuzzy Miner and Inductive Miner. Control-flow discovery has been applied in various healthcare contexts. For instance: Caron et al. [14] use the Heuristics Miner to retrieve a process model for the radiotherapy department within the context of gynaecologic oncology. Duma and Aringhieri [25] use both Heuristics Miner and ‘Inductive Miner - Infrequent’ to study the patient trajectory at the emergency department of an Italian hospital. To limit the complexity of the data, they preprocess the event log by merging consecutive events referring to the same activity in the process.

Despite these pre-processing efforts, the Heuristics Miner discovers a spaghetti model, which is not understandable. The 'Inductive Miner - Infrequent', in its turn, generates a very simple, but imprecise model, meaning that the discovered model allows for a lot of behaviour that is not observed in the event log [25]. Using, amongst others, Heuristics Miner and Fuzzy Miner, Kim et al. [40] focus on the patient trajectory in an outpatient clinic in Korea. They explicitly compare the process models discovered from data to a process model that has been developed solely based on a discussion with domain experts. The process mining insights surface some important trajectories that are not included in the domain experts' model, highlighting the added value of process mining [40].

Besides automated control-flow discovery, *interactive control-flow discovery* also receives some attention in literature. A distinguishing characteristic of interactive control-flow discovery is that a domain expert is interactively involved while the model is being discovered from the event log [10]. In this way, domain knowledge is embedded in the discovery processes, instead of being used to interpret the output of an automated algorithm. Using a case study of the patient trajectory of lung cancer patients, Benevento et al. [10] show that the interactive process discovery approach of Dixit et al. [23] generates control-flow models which are both accurate and understandable. In contrast, automated control-flow discovery algorithms might experience difficulties to generate such an accurate and understandable model. Even though the advanced algorithms discussed in Chapter 3 [7] will prove helpful, it might still be difficult to discover accurate and understandable control-flow models automatically. This can be, at least partly, explained by the fact that the order of tasks in healthcare processes often depends on highly specialised background knowledge, which is not embedded in the event log [10]. While interactive control-flow discovery received fairly little attention so far, it is highly promising for domains in which processes are highly knowledge-intensive and loosely-structured, which holds for many healthcare processes [55]. For a more extensive introduction on interactive process mining in healthcare, the reader is referred to Fernandez-Llatas [29].

A important challenge in control-flow discovery in healthcare, especially for medical treatment processes, is the great variability [59]. As many different paths through the process tend to occur, applying a control-flow discovery algorithm often results in a spaghetti model, which is very complex or even impossible to understand [51]. To handle this problem, trace clustering techniques can be used to create more homogeneous patient subgroups, which can be studied separately in an effort to reduce complexity. For instance: Mans et al. [51] use trace clustering on an event log of gynaecological oncology patients from a Dutch hospital to generate patient groups that follow a similar trajectory. Despite the potential of trace clustering, Lu et al. [48] also recognise some challenges. These include the fact that individual clusters might still contain thousands of distinct activities performed for patients, which would still be highly problematic for control-flow discovery purposes. Moreover, suppose clusters are created based on the medical condition of patients, each cluster might still contain a wide variety of patient trajectories as the same condition might be handled in a variety of ways. Against this background and with the ambition to generate clusters that are meaningful to domain experts, Lu et al. [48] develop a novel trace clustering method. Their method starts from a small sample set of patients, based on input from domain experts, to generate clusters. An evaluation of the method at a Dutch hospital highlights that the resulting control-flow models presented meaningful behavioural patterns for medical experts [48].

While the majority of control-flow discovery contributions take data from the hospital information system as a starting point, other types of input data are also occasionaly taken into consideration [37]. For example: Fernandez-Llatas et al. [31] use real-time indoor location systems data, which track the movement of patients throughout the surgery area of a Spanish hospital. Using this data, PALIA is used to discover a process model that represents the order of locations that a patient has visited [31]. Another illustration is the work of Lira et al. [47], where video recordings of a surgical procedure, i.e. the ultrasound-guided central venous catheter placement, are used as input data. These video recordings are tagged to generate an event log, which is used as an input for control-flow discovery [47].

All of the aforementioned papers focus on the discovery of control-flow models. However, as highlighted in Chapter [1] process discovery can also relate to other perspectives of the process, such as the resource perspective. For instance, Alvarez et al. [3] identify collaboration patterns between healthcare professionals within the emergency department of a hospital. The resulting process model sheds valuable insights in the interactions between physicians, nurses, medical assistants and technicians [3]. Similarly, one of the analyses conducted by Agnostinelli et al. [2] centers around the identification of interactions between different subdepartments in an Italian outpatient clinic. These examples highlight the potential of process mining to discover valuable process models in healthcare, also beyond the control-flow perspective.

3.2 Conformance Checking

As highlighted in Sect. 2.2, a multitude of clinical practice guidelines and protocols are available in the healthcare domain, which can act as reference processes [59]. Conformance checking, the topic of Chapter 5 [13] and a second common use case for process mining in healthcare, enables assessing the adherence of the real-life healthcare process (as captured by the event log) to clinical guidelines and protocols, as well as to study where reality deviates from an already existing process model [55]. For instance: Mannhardt and Blinde [50] use the public sepsis event log and aim to assess the conformance of the real-life process with two rules put forward by the sepsis guidelines at that time: (i) the time difference between the moment at which the triage document is completed and the admission of intravenous antibiotics should be less than 1 h, and (ii) the time difference between the moment at which the triage document is completed and the measurement of lactic acid should be less than three hours. Through the use of multi-perspective conformance checking, the authors conclude that the first rule is violated for 58.5% of the patients, while the second rule is only violated for 0.7% of patients. This observation constitutes a basis to look into the adherence to medical guidelines in more detail [50]. Another example is the work by Rinner et al. [68], who use alignment-based conformance checking to assess the compliance between the European guideline on melanoma treatment and an event log from an Austrian medical university. This analysis is highly relevant as the authors indicate that patients which comply to the guidelines have a significantly better prognosis than deviating patients [68]. Also focusing on clinical guidelines, Huang et al. [38] propose an approach to detect both global and local anomalies between a clinical pathway and an event log. While the former refers to patient trajectories that significantly deviate from the clinical pathway, the latter represents a deviation in a particular part of the trajectory. This approach is applied to an event log containing trajectories of unstable angina patient at a Chinese hospital [38].

While conformance checking offers great potential, Sato et al. [75] highlight the challenge that clinical guidelines and protocols are often defined at a different level of aggregation than the events in the event log. To tackle this problem and using the pre-operative phase of bariatric surgery as an illustration, the high-level activities in the reference model are explicitly mapped to the events included in the event log. Besides the potential discrepancy in terms of the level of aggregation, Bottrighi et al. [12] also highlight that clinical guidelines typically focus on patients in general, while clinical practice often requires adapting general guidelines to the specificities of individual patients and contexts. For instance: patients might have several co-morbidities and certain equipment might not be available in a particular situation. As a consequence, physicians add what is called basic medical knowledge in order to alter clinical guidelines to the specific patient and contextual characteristics. This adds a dimension to conformance checking: besides checking the adherence to the clinical guideline, the basic medical knowledge that the physician adds also needs to be taken into consideration [12].

The aforementioned examples use clinical guidelines and protocols as the reference model. While this is a common situation in the healthcare domain, it should be noted that conformance checking techniques can also generate valuable insights when the reference model originates from a different source. For instance: Kirchner et al. [41] perform conformance checking within the context of the liver transplantation process. To create the process model to compare the event log with, an interdisciplinary team consisting of physicians and modelling experts was brought together [41]. This example highlights that conformance checking is a versatile toolkit to assess whether hospital processes are performed in reality as intended according to any form of reference model.

3.3 Performance Analysis

Regarding the evaluation of healthcare process performance, various types of performance measures can be used. A basic distinction can be made between clinical, financial and operational key performance indicators. A *clinical* key performance indicator relates to a measure of the patient's medical condition, a *financial* key performance indicator reflects the financial effect of the execution of the process, and an *operational* key performance indicator represents a measure regarding the operational execution of the process. The category of operational key performance indicators can be further subdivided in *time-related* and *resource-related* key performance indicators. The former can, for example, be the waiting time of a patient or the length of stay, while the latter can relate to the bed occupancy rate or staff utilisation at a particular department [17].

Based on a systematic literature review, De Roock and Martin [17] conclude that less than half of the reviewed paper reports on a specific key performance indicator for their process mining analysis. When a key performance indicator is used, time-related key performance indicators are used the most frequently, followed by clinical key performance indicators. Financial and resource-related key performance indicators are rarely used in literature [17]. A commonly used time-related key performance indicator is the length of stay of a patient, which represents the time between the arrival of a patient and his/her departure [89].

Rojas et al. [70] use the length of stay when conducting a performance analysis of processes at the emergency department of a Chilean hospital. Based on their analysis, they identified that two key steps in the emergency department process contribute to higher length of stay values for patients. Firstly, the number of examination-treatment loops that the patient goes through, indicating the amount of time that is needed to uncover the true problem. Secondly, the need for a validation examination, which is an examination by a physician to ensure that the patient is ready to be discharged from the emergency department. In the same context and with the same key performance indicator, the length of stay at the emergency department of a hospital, Andrews et al. [5] conduct a process performance analysis at the St. Andrew's War Memorial Hospital in Australia. They conclude that a key contributor to high length of stay values is the time that elapses between the moment at which it is decided that a patient should be admitted and the moment at which the patient can actually move to the relevant ward [5].

3.4 Comparative Process Mining

Comparative process mining, e.g. the comparison of various patient groups, time periods or healthcare organisations, has also been used in the healthcare domain. With respect to the comparison of *patient groups*, Rojas and Capurro [69] study the medication use process for patients suffering from sepsis in the MIMIC-II database. To this end, three patient groups are distinguished, based on whether vasodilators, vasopressors, or systemic antibacterial antibiotics were used. Another example is Pebesma et al. [61], where three patient groups are separated to model the trajectory of cardiovascular risks for patients with type 2 diabetes: a high-risk, medium-risk and low-risk group. After modelling the evolution of the risk level for each group, the gender distribution within each group is determined, suggesting that female patients tend to be in lower risk states compared to their male counterparts. A final example is the research by Andrews et al. [6], who study the pre-hospital care process for victims of road traffic accidents. In this respect, they consider three groups: (i) persons who do not require ambulance transportation, (ii) persons who are transported to e.g. local medical practices or elderly care facilities, and (iii) persons who are transported to a hospital [6].

Other papers *compare different time periods*, which is another type of comparative process mining. For instance, Yoo et al. [92] use process mining to assess the impact of commissioning new buildings of a hospital, where, e.g., the cancer centre and clinical neuroscience centre have moved to the same floor and additional administrative counters have been added. To determine the impact of the move to the new building, as well as the associated new facilities that became available, the results of a process mining analysis before the move are compared to the results using an event log of a period after the move. Their findings highlight that processes run more efficiently in the new facilities, both for the cancer centre and the clinical neuroscience centre. Moreover, the consultation waiting time decreased [92]. A different example is situated within the context of an emergency department. Within that context, Stefanini et al. [77] compare the summer period to the winter period. In their comparison, they both incorporate the patients' trajectory as well as a variety of key performance indicators. One finding is that urgent patients, on average, have to wait longer before their first consultation in summer than in winter [77].

Regarding the *comparison of healthcare organisations*, a prime example is the work by Partington et al. [60]. They compare four Australian hospitals in terms of the pathway of patients who presented themselves at the emergency department and are suspected to suffer from acute coronary syndrome. The comparison focuses on the control-flow and time perspectives of the process. Regarding the time perspective, measures such as waiting times, throughput time and length of stay are taken into consideration. Various valuable insights were retrieved from the comparative analysis, e.g. some hospitals use an angiography (i.e. an X-ray of a patient's blood vessels) significantly more often than other hospitals. Moreover, significant differences in the length of stay of patients were discovered [60]. The work of Partington et al. [60] highlights the great potential of comparative process mining to compare local practices and process performance values. This can constitute a fruitful basis for mutual learning and, hence, the improvement of healthcare processes. However, it requires a culture of transparency, which has been highlighted as a challenge for process mining adoption within the broader process mining field [56].

3.5 Predictive Process Mining

While the aforementioned process mining types are backward-looking, process mining in healthcare research has also focused on forward-looking approaches, i.e. predictive process mining (see also Chapter 10 [21]). Two key research topics are data-driven prediction models and data-driven process simulation. An example of the former category, *data-driven prediction models*, is Benevento et al. [9], which focus on predicting the waiting time of patients at the emergency department. To this end, various predictor variables are taken into consideration, such as patient variables (e.g. their age or the assigned triage code), temporal variables (e.g. the hour of the day), staff-based variables (e.g. the nurses' schedules, the physicians' schedules). They also consider queue-related variables in the prediction model (e.g. the number of patients who received a triage code, but were not yet treated), which were identified in an event log. The empirical evidence suggests that adding the queue-related variables improves the performance of the waiting time prediction model. In a very different context, van der Spoel et al. [82] use a combination of data mining and process mining techniques to predict the cashflow of a Dutch hospital. In this respect, they focus on predicing the treatment trajectory based on the diagnosis and the start of the trajectory, as well as on predicting the duration of this trajectory [82].

Several papers have investigated the potential of process mining within the context of process simulation in healthcare. These efforts belong to the domain of *data-driven process simulation*, which refers to the extensive use of an event log during the development of a simulation model [19]. For example: Tamburis and Esposito [78] investigate how process mining could be used to support the development of a simulation model of the cataract treatment process at an ophthalmology department. Kovalchuck et al. [42], in their turn, simulate the process that patients suffering from acute coronary syndrome follow, using process mining to support the model development process. To demonstrate the developed simulation model, they focus on the effect of the availability of angiography equipment, which is important to quickly detect the presence of acute coronary syndrome. In particular, the influence of varying the number of angiography instruments on output measures such as the length of stay and the average waiting time is predicted [42]. Franck et al. [33] use a simulation-based analysis of the process of stroke patients at the emergency department. Process mining is used to determine the order of activities from an event log. Using the simulation model, various scenarios are defined in terms of the number of neurovascular intensive care unit beds required to provide patients with care according to the optimal clinical pathway.

van Hulzen et al. [84] use data-driven process simulation to explore potential future scenarios to support capacity management decisions for the radiology department of a Belgian hospital. Within the context of the construction of new facilities, which involves a centralisation of different geographically separated campuses, department management needs to provide input regarding the required number of radiological devices (X-ray, CT scanner, etc.), the size of the waiting area for ambulatory patients, and the required number of receptionists. In particular, the study centers around three key questions formulated by the department management: (i) what is the effect of the centralisation of services on the required resource capacities?, (ii) what is the impact of abolishing the need for patients to drink contrast fluid on the throughput time and required waiting area size?, and (iii) what would be the effect of an online registration system for ambulatory patients on the reception staff requirements and the size of the waiting area? To develop a simulation model to answer these questions, an event log originating from the radiology information system is intensively used.

While the case study clearly demonstrates the potential of data-driven process simulation in healthcare, van Hulzen et al. [84] also highlight challenges such as data quality issues, as well as the lack of support to interactively involve domain experts during the development of a simulation model.

3.6 Action-Oriented Process Mining

As highlighted in Chapter 1 [1], action-oriented process mining focuses on translating process mining insights into actions. This is also a crucial step within the healthcare domain as only then process mining will reach its full potential as a catalyst of evidence-based process improvement [55]. Despite its great importance, research efforts focusing on the translation of process mining insights in actions are scarce in the healthcare domain. This is confirmed by the review of De Roock and Martin [17], where the need for more research on the translation of process mining outcomes to actionable process improvement ideas is indicated as one of the key recommendations for the future development of the research field.

A first step in the direction of action-oriented process mining is ensuring that process mining endeavors start from specific questions put forward by healthcare professionals [55]. Several research papers explicitly report on this matter, such as the work by van Hulzen et al. [84] on data-driven process simulation for capacity management at the radiology department. In a similar vein, Agostinelli et al. [2] explicitly devote attention to defining the questions of healthcare professionals in a process mining project in cooperation with the San Carlo di Nancy hospital. Better understanding three key processes was the central objective of the process mining analysis, including the hospitalisation process of patients. However, Agostinelli et al. [2] claimed that it was difficult to elicit specific questions from healthcare professionals because they had no background knowledge on process mining. The knowledge gap between process mining experts and domain experts is an important consideration to take into account when moving towards action-oriented process mining.

3.7 Further Reading

This section had the ambition to provide an intuitive overview of common use cases in process mining in healthcare literature. Hence, it does not constitute a full overview of all scientific contributions in the field. For a more detailed outline of the state of the art in literature, the reader is referred to one of the literature reviews on process mining in healthcare that have been published. Some reviews focus on a particular subdomain in healthcare: Kurniati et al. [43] on oncology, Kusuma et al. [44] on cardiology, Williams et al. [90] on primary care, and Farid et al. [28] on frail elderly care. Other reviews take a more generic perspective and consider process mining in healthcare as a whole: Ghasemi and Amyot [36], Rojas et al. [71], Batista and Solanas [8], Erdogan and Tarhan [27], Rule et al. [73], Dallagassa et al. [16], Guzzo et al. [37], and De Roock and Martin [17]. All review papers significantly differ in terms of the review dimensions that are taken into consideration and whether time trends are taken into consideration [17]. De Roock and Martin [17] provide an overview of the similarities and differences amongst 11 published literature reviews.

4 Case Study

The previous sections introduced healthcare processes, their particularities, and common use cases in process mining in healthcare literature. This section presents a real-life case study of conducting a process mining analysis in a hospital. The case study is situated in the *Superfluid Hospital* project conducted at the hospital of Braunschweig, led by Dr. Andreas Goepfert and Lars Anwand together with Nils Wittig. The project has the overarching ambition of ensuring that processes run smoothly within the hospital in order to improve the well-being of patients and employees, the quality of care, as well as the hospital's financial performance. To outline the case study, the project goal and IT-infrastructure is discussed (Sect. 4.1), followed by the outcomes of the process mining analysis (Sect. 4.2).

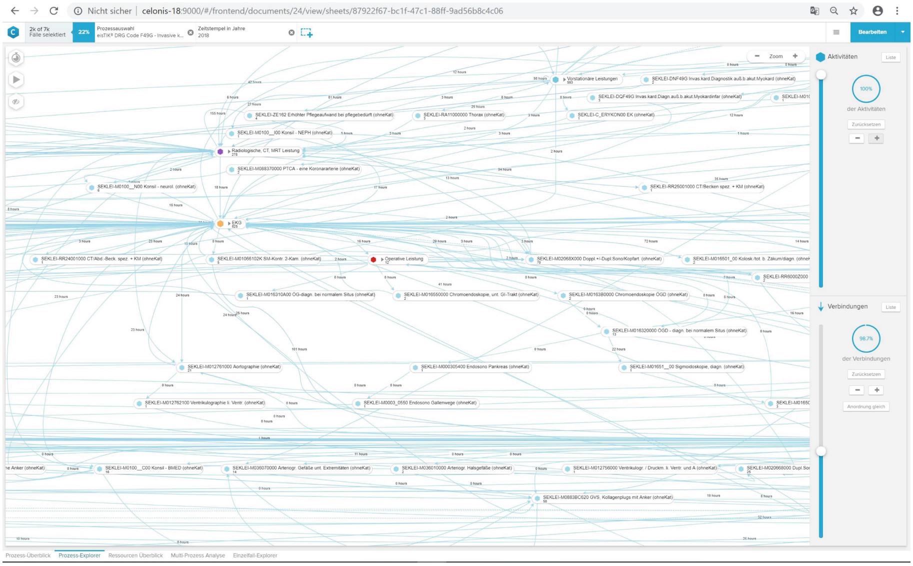
4.1 Project Goal and IT-Infrastructure

The specific goal of the *Superfluid Hospital* project is discovering medical treatment processes within the hospital. To this end, readily available process execution data and process mining has been used in order to avoid any additional documentation work for healthcare professionals. The fact that no additional data needs to be recorded could play an important role in nurturing acceptance for process mining and to stimulate its use on a continuous basis (e.g. also to track and evaluate the effect of process changes).

Hospitals typically use a variety of IT systems, implying that process execution data will also be scattered over various systems. In order to be able to analyse all relevant data centrally, the Braunschweig hospital uses data warehouse infrastructure as a starting point for process mining. This data warehouse already gathers the relevant data from various underlying information systems in the hospital. In particular, this case study uses the data warehouse infrastructure and business intelligence solution *eisTIK* from *KMS Vertrieb und Services AG*, which combines process execution data from different data sources such as the Hospital Information System, the Laboratory Information System, the Radiology Information System, etc. For the process mining analysis, an integrated version of the tool *Celonis* has been used within the data warehouse. Hence, process mining is no longer a standalone tool, which lowers the efforts for healthcare professionals to perform process mining.

4.2 Outcomes of the Process Mining Analysis

This subsection illustrates the outcomes of conducting process discovery at the case study hospital in Braunschweig. In particular, the focus will be on the medical treatment process of cardiology patients, which is a cohort of 1566 patients



Nicht sicher | celonis-18:9000/#/frontend/documents/24/view/sheets/87922f67-bc1f-47c1-88ff-9ad56b8c4c06

2k of 7k Fälle selektiert 22%

Prozessauswahl: eisTIK® DRG Code F49G - Invasive k...

Zeitstempel in Jahre: 2018

Aktivitäten: 100% der Aktivitäten

Verbindungen: 98.7% der Verbindungen

Bearbeiten
Zoom
Liste
Zurücksetzen
Anordnung gleich
Vorstationäre Leistungen 593
SEKLEI-DNF49G Invas.kard.Diagnostik auß.b.akut.Myokard (ohneKat) 2
SEKLEI-DQF49G Invas.kard.Diagn.auß.b.akut.Myokardinfar (ohneKat) 3
SEKLEI-M010
SEKLEI-ZE162 Erhöhter Pflegeaufwand bei pflegebedürft (ohneKat) 4
SEKLEI-RA11000000 Thorax (ohneKat) 3
SEKLEI-C_ERYKON00 EK (ohneKat) 1
SEKLEI-M0100__I00 Konsil - NEPH (ohneKat) 1
Radiologische, CT, MRT Leistung 278
SEKLEI-M088370000 PTCA - eine Koronararterie (ohneKat) 2
SEKLEI-M0100__N00 Konsil - neurol. (ohneKat) 6
SEKLEI-RR25001000 CT/Becken spez. + KM (ohneKat) 1
EKG 825
SEKLEI-RR24001000 CT/Abd.-Beck. spez. + KM (ohneKat) 1
SEKLEI-M01066102K SM-Kontr. 2-Kam. (ohneKat) 5
Operative Leistung 12
SEKLEI-M02068X000 Doppl.+/-Dupl.Sono/Kopfart. (ohneKat) 70
SEKLEI-M016501_00 Kolosk./tot. b. Zäkum/diagn. (ohneK
SEKLEI-RR6000Z000 2
SEKLEI-M016310A00 ÖG-diagn. bei normalem Situs (ohneKat) 1
SEKLEI-M016550000 Chromoendoskopie, unt. GI-Trakt (ohneKat) 2
SEKLEI-M0163B0000 Chromoendoskopie ÖGD (ohneKat) 2
SEKLEI-M016320000 ÖGD - diagn. bei normalem Situs (ohneKat) 13
SEKLEI-M012761000 Aortographie (ohneKat) 21
SEKLEI-M000305400 Endosono Pankreas (ohneKat) 1
SEKLEI-M01651__00 Sigmoidoskopie, diagn. (ohneKat) 1
SEKLEI-M012762100 Ventrikulographie li. Ventr. (ohneKat) 1
SEKLEI-M0003_0550 Endosono Gallenwege (ohneKat) 1
SEKLEI-M0165C 3
ne Anker (ohneKat)
SEKLEI-M0100__C00 Konsil - BMED (ohneKat) 16
SEKLEI-M036070000 Arteriogr. Gefäße unt. Extremitäten (ohneKat) 14
SEKLEI-M036010000 Arteriogr. Halsgefäße (ohneKat) 2
SEKLEI-M012756000 Ventrikulogr. / Druckm. li. Ventr. und A (ohneKat) 1
SEKLEI-M020668000 Dupl.So 25
SEKLEI-M0883BC620 GVS, Kollagenplugs mit Anker (ohneKat) 58
Prozess-Überblick
Prozess-Explorer
Ressourcen Überblick
Multi-Prozess Analyse
Einzelfall-Explorer

Fig. 1. Detailed view of the trajectories of patients receiving cardiology services, showing only a cut-out of the whole process.

in the data warehouse. It was the ambition of the project team, consisting of process analysts and healthcare professionals, to gain a deep understanding in the treatment of cardiology patients in order to identify areas for improvement towards the future.

Figure 1 provides an overview of the trajectories of patients receiving cardiology services, in particular a coronary angiography, containing all activities that have been conducted. As becomes apparent from the visualisation, this level of detail is unsuitable to gain insights into potential problems in the process. As a consequence, the amount of activities represented in the process model is reduced by means of filtering. Visualising only the most important activities, as shown in Fig. 2, leads to a less complex process model. The key difference between Figs. 1 and 2 is that the percentage of included activities is reduced from 100% in Fig. 1 to 53% in Fig. 2. Moreover, the number of connections between activities is also significantly reduced to about 40% in Fig. 2.

When studying Fig. 2 in more detail, it follows that particular diagnostics have already been performed for some patients before they actually go to the hospital. In particular, for 593 patients, an electrocardiogram and other check-ups (*'Vorstationäre Leistungen'*) have already been executed before they were admitted to the hospital. Note that all results that patients bring with them will still be checked to ensure that the patient is eligible for the procedure. Patients that do not have prior check-up results generally take one of the following paths from hospital admission (*'Aufnahme'*, blue hexagon) onwards:

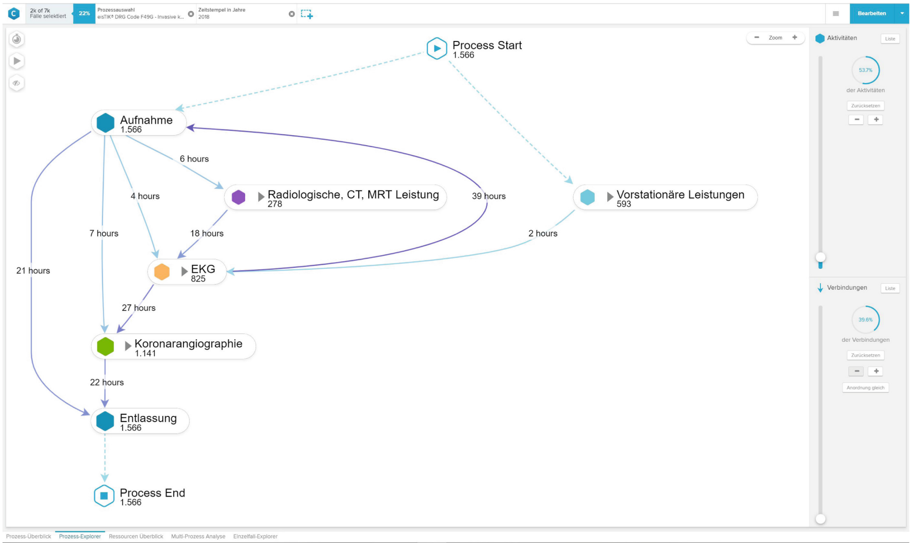


Fig. 2. Filtered view of the process for cardiology patients receiving a coronary angiography, all grouped by DRG F49G (which is a diagnosis-related grouping that is used as a billing system in Germany). (Color figure online)

- ***Path 1 – on average 7 h to intervention:*** Patients following the first path directly proceed to the coronary angiography (*'Koronarangiographie'*, green hexagon). It takes, on average, seven hours before this intervention with a coronary angiography can be performed (e.g. due to the need for a general consultation, the analysis of a blood sample, etc.). This implies that, when the patient arrives at the hospital in the morning, the intervention occurs on the same day.
- ***Path 2 – on average 31 h to intervention:*** Patients following the second path receive an electrocardiogram (*'EKG'*, yellow hexagon) on average four hours after their arrival at the hospital. When the results of the electocardiogram are available, the patient is ready for the coronary angiography. It takes, on average, 27 h before the intervention is actually carried out.
- ***Path 3 – on average 51 h to intervention:*** Patients which follow the third path receive an X-ray (*'Radiologische, CT, MRT Leistung'*, purple hexagon), on average, six hours after their arrival at the hospital. Afterwards, on average 18 h pass before the patient receives an electocardiogram (*'EKG'*, yellow hexagon). Finally, a coronary angiography takes place, on average, another 27 h later.

Note that Fig. 2 also contains a connection between the execution of an electrocardiogram (*'EKG'*, yellow hexagon) and hospital admission (*'Aufnahme'*, blue hexagon). This connection represents patients which are temporarily discharged from the hospital, but return the following day to continue the process. Another interesting connection was revealed by analysing the data i.e. the direct connection from admission (*'Aufnahme'*, blue hexagon) to discharge (*Entlassung*, blue hexagon) in Fig. 2 within 21 h. This connection can be explained by the existence of a specific group of patients for whom the treatment has been recorded in a different logic. These patients have previously not been included in the internal performance measurement. This shows that process mining can also highlight relevant deviations in the documentation. In this way, important areas of action for the improvement of data quality have been identified, generating additional added value for the hospital.

As mentioned in Sect. 3, it is important that process mining insights are also translated to actions. Based on the analysis, of which some highlights have been presented above, several actions have been specified in the process, as will be exemplified here. Firstly, patients will be encouraged to bring all relevant radiological imaging and recent electrocardiogram reports with them. This will enable them to get treated much faster by following the first path described above. Secondly, measures have been taken to accelerate the second path to make sure that patients receive the intervention during their first day of hospitalisation. Due to organisational adjustments, patients now receive the ECG with higher priority. This makes it possible that, after a faster diagnosis, they often receive the actual intervention in the afternoon of the day of admission. Finally, the third path outlined above should be combined with the second path by registering patients for both the radiological and cardiological diagnostic services at the moment of admission. The relevant preliminary examinations can be carried out and evaluated over the course of a day. In this way, the procedure can take place the day after admission, provided that there are no medical reasons for not doing so.

Healthcare professionals provided positive feedback on the conducted process mining analysis, both with respect to the analysis procedure, as well as with regards to the insights that have been gathered. The conducted analysis made healthcare professionals aware of the improvement potential in their processes, which will result in shorter hospitalisations and improved care quality for patients. Especially changes that resulted in a reduction of unnecessary waiting times in the patient's trajectory are considered highly useful. While the insights and improvement actions presented in this section are based on an analysis of historical data, it should be noted that the use of the data warehouse with integrated process mining functions also enables real-time analyses. As a consequence, it is possible to create a live view of the process, which opens options to take action in the process while the process for a patient is still running.

5 Open Challenges

Section 3 and Sect. 4 demonstrate the great potential of process mining in healthcare, as well as the research that has been conducted in the research field. However, it has been reported that the uptake of process mining in healthcare, beyond case studies in a research context, is fairly limited [55]. Hence, there are still significant challenges ahead to ensure a widespread adoption of process mining in healthcare. The remainder of this section provides an overview of ten key challenges for the field, based upon the recent work by Martin et al. [55] and Munoz-Gama et al. [59].

Create a Standardised Terminology. In the healthcare domain, there is a tradition of using standardised terminologies to ensure a common understanding of concepts [26]. An illustration is the *International Classification of Diseases* (ICD), which defines about 55000 codes to label injuries, diseases, and causes of death in a standardised way [91]. In the process mining field, standardisation often focuses on the data structure level (e.g. the XES and OCEL standards), but less on the terminology level. Terms such as event, case, activity, and trace might be used in an ambiguous way based on the working definitions of individuals or research groups. This is especially troublesome when working in an interdisciplinary context as it can lead to problematic communication. Hence, there is a need to develop a standardised terminology to support process mining in healthcare, which should (i) provide a clear definition of process mining concepts in a healthcare context, and (ii) link to existing terminologies in the healthcare domain whenever possible [55].

Tackle Real-World Healthcare Problems. To support the uptake of process mining, it is important that process mining methods help to solve real-word problems of healthcare professionals. In order to capture and thoroughly understand these problems, close and ongoing interaction between the process mining community and healthcare professionals is needed. Only then, methods can be developed that actually support healthcare professionals to solve these problems [55,59]. Progress still needs to be made as, based on a systematic literature review, De Roock and Martin [17] conclude that only 12.5% of the reviewed papers reported that healthcare professionals were actively involved during the problem definition stage of a process mining project. Besides eliciting problems from healthcare professionals instead of assuming that a particular issue is relevant, it is also key to evaluate process mining methods using real-life data from an authentic healthcare context. Besides enabling the researcher to fine-tune the developed method based on the complexity of real-life data, a real-life demonstration will also build confidence among healthcare professionals in process mining's ability to tackle real-world problems [55,59].

Deal with Low Quality Data. The healthcare domain has been shown to suffer from low quality process execution data, the key input for process mining. As applying process mining techniques to low quality data can lead to counterintuitive and even misleading results [4], data quality is an important challenge for process mining in healthcare (see also Chapter 6 [18]). Data quality issues include missing events (i.e. events that took place, but which were not registered in the system), incorrect timestamps (i.e. timestamps that do not correspond to the time at which the event actually took place), and imprecise resource information (i.e. resource information that does not refer to a specific healthcare professional) [52,54]. While approaches have recently been developed to assess the event log quality or to handle specific event log quality issues using targeted heuristics [54], data quality remains a challenge for process mining in healthcare. In this respect, it is also important that healthcare organisations are made aware of the need to improve data registration at the source in order to fully leverage the potential of process mining. Potential initiatives include raising awareness among healthcare professionals and facilitating data registration when designing user interfaces [55,59].

Identify the Most Suitable Process Modelling Language. Within the context of control-flow discovery, process mining enables retrieving a visual representation of how a healthcare process is performed in reality. In order to effectively use a process model as a communication instrument and, hence, as a basis for process improvement, it is important to determine the most suitable process modelling language within a healthcare context. Within the business process management domain, a wide variety of process modelling languages have been developed such as BPMN, Petri nets and Declare. At the same time, modelling languages to represent clinical guidelines such as GLIF3 have been proposed in the healthcare domain. Given the plethora of available languages and as it has been shown that the modelling language impacts model understandability [32], thorough benchmarking research is required. Such research should focus on both the expressive power of the considered modelling language, as well as the understandability of the resulting control-flow model for healthcare professionals. Regarding the latter, a wide range of healthcare contexts and healthcare professionals should be taken into account. By carefully understanding the strengths and weaknesses of existing process modelling languages, both from the business process management and the healthcare domain, valuable lessons can be drawn on the visualisation of process mining outcomes in healthcare [55].

Move Beyond Control-Flow Discovery. While Sect. 3 aimed at providing a broad view on process mining in healthcare, it should be recognised that control-flow discovery remains the most dominant use case of process mining in healthcare [17,37]. While there is a clear need for control-flow discovery algorithms that are designed with the particularities of healthcare processes in mind, it is important that targeted methods are also developed for other process mining types such as conformance checking, predictive process mining or to discover insights from the time or resource perspective [59]. Moreover, as follows from Sect. 3, more research on action-oriented process mining in healthcare is needed as this is the key for process mining to actually contribute to the generation of societal value in healthcare. With respect to the various perspective of a process, analyses that span over several process perspectives, e.g. which combine the control-flow perspective with the time or resource perspective, also have the potential to generate great value for healthcare. Such multi-perspective analyses can provide healthcare professionals with rich insights, e.g. about how the control-flow of the process gives rise to particular resource behaviour [55,59].

Look Beyond the Hospital Walls. As highlighted in Sect. 2.2, patients are at the core of healthcare processes. Patients, especially patients with a chronic disease, often have a therapeutic relationship with various healthcare organisations.

However, the great majority of the research on process mining in healthcare is still focused on what happens with patients in the context of a hospital visit or admission. Exceptions such as Fernandez-Llatas et al. [30], who focus on supporting nursing home design using process mining, are scarce. Even when a part of the patient's diagnosis and treatment process takes place in a hospital, it is important to note that a significant portion of the process might also be executed outside the hospital's walls. For instance: an oncological patient might have surgery at a specialised hospital, (s)he might have regular check-ups scheduled at a local general hospital and might receive specific treatments at home, supported by a home healthcare organisation. When process mining has the ambition to provide healthcare professionals with valuable insights in the patient journey, it will probably not be sufficient to only study the process fragment that takes place in the hospital. As process execution data will be spread over the information systems of several healthcare organisations, this will pose challenges in terms of obtaining data and connecting all data sources. Moreover, careful consideration has to be given to data privacy and security. While privacy and security are relevant for all process mining endeavours, involving several healthcare organisations will add an additional layer of complexity [55,59].

Give Control to Healthcare Professionals. Currently, process mining initiatives in healthcare are often carried out by a multidisciplinary team, consisting of both healthcare professionals and process mining experts. Process mining experts play an important role given the technical skills which are required to prepare an event log and perform the appropriate analyses. In the long run, it should be the ambition of the process mining community to develop tools which are so intuitive that healthcare professionals can autonomously use them, instead of depending on (potentially external) process mining experts. While this is far from trivial given the high complexity of many healthcare processes, as well as due to complicating factors such as data quality issues, efforts to give control to healthcare professionals are highly valuable. A first step would, for instance, be to ensure that healthcare professionals are actively involved in the specification of analysis targets. In order to make informed judgements and clearly delineate their questions, it would be highly valuable if healthcare professionals have a minimal level of data and process literacy [2,17]. Moreover, enhanced training might also nurture a mindset in which process execution data is considered as a strategic asset that the healthcare organisation wishes to leverage to the largest extent possible. Additional efforts to gradually give control to healthcare professionals involve specific attention to elements such as the use of unambiguous terminology and the clear visualisation of outcomes when developing tools to perform process mining in healthcare [55,59].

Integrate Process Mining Functionalities in Existing Systems. The positioning of process mining as a standalone tool constitutes a major barrier for the systematic use of process mining in healthcare practice. Nowadays, in order to use process mining, data often need to be extracted from the health information system, reformatted to the required event log structure, and imported in a process mining tool. While this is feasible for a one-off research project, this is impractical in the daily work setting of healthcare professionals. Hence, to support the use of process mining in healthcare, process mining functionalities need to be integrated in the information systems that are used by healthcare professionals. To this end, a strong partnership between the process mining community and health information system vendors needs to be established. Moreover, healthcare organisations can include the need for data-driven process analysis functions when formulating update requests to their vendors [55]. The case study presented in Sect. 4 presents a first step towards tackling this challenge as process mining functionalities were integrated with the data warehouse solution used by the hospital under consideration.

Develop Tailored Methodologies for Process Mining in Healthcare. The particularities of healthcare show the need for the development of tailored methodologies for process mining in healthcare. Such methodologies should provide specific guidelines for the various phases of a typical process mining initiative in a healthcare context, ranging from the specification of the research problem, over the composition of the event log, the execution of the analysis, to the interpretation of the final results, and the actions that will be linked to the findings. When establishing methodologies, inspiration should evidently be drawn from efforts in the broader process mining field such as the L*-methodology [81], and the PM^2-methodology [83]. However, it is key to also take the particularities of the healthcare domain into consideration, as well as the wide variety of contexts in which process mining can be used in the domain. The presence of solid methodological support might also persuade healthcare organisations that are considering the adoption of process mining, but still have concerns regarding the rigour of a relatively young research domain, as well as regarding how the process mining effort should exactly be approached [55,59].

Evolve in Symbiosis with Evolutions in the Healthcare Domain. As highlighted in Sect. 2.2, the healthcare domain is in constant evolution due to advances in various fields such as medicine and technology. Moreover, new paradigms such as patient-centred care give rise to new care approaches. Against this background, it will be an ongoing challenge for process mining to follow-up on these evolutions and to ensure that the provided support matches the expectations of healthcare organisations. To appreciate the latter statement, it is important to realise that process mining will always be a means to an end, rather than a goal in itself. Consequently, the impact of process mining in the healthcare field will depend on its ability to add value within a constantly changing context. From that perspective, process mining in healthcare should evolve in symbiosis with evolutions in the healthcare domain. While the foregoing represents a more reactive perspective, it is important to note that process mining can also actively contribute to evolutions in healthcare. For instance: process mining techniques can be used to efficiently compare various treatment processes with respect to the clinical and patient experience outcomes they generate and, hence, can contribute to shaping the clinical pathways of the future. Similarly, by providing profound insights in the usage patterns of mobile health applications, process mining can help to optimise the user-friendliness and, hence, patient satisfaction with respect to telemonitoring instruments [55,59].

6 Conclusion

This chapter introduced a specific application domain of process mining: healthcare. Healthcare is a promising domain in which process mining can create significant societal value by helping healthcare organisations to better understand and improve their processes. Besides highlighting and illustrating the potential of various types of process mining in healthcare, the complex nature of many of its processes was also discussed. The specific characteristics of healthcare processes, such as the high level of variance and the widespread presence of guidelines and protocols, necessitate the development of dedicated process mining methods. In this respect, it is important to note that process mining in healthcare can build upon an active and committed research community, who are keen to develop novel methods that start from real-world problems experienced in healthcare. This will definitely be needed as the systematic uptake of process mining in healthcare, beyond the research context, is still fairly limited. A multitude of challenges is still ahead.

While current literature still predominantly focuses on the hospital setting, as was clearly reflected in the examples used in this chapter, it is important to also consider other types of healthcare organisations such as elderly care organisations, psychiatric care organisations and home-based care organisations. These organisations are also confronted with immense challenges and are likely to have even less resources available for advanced analytics than hospitals. Even though these other types of healthcare organisations might even be more challenging for process mining than the hospital context, e.g. because of their lower maturity in terms of data registration, they would greatly benefit from open access and user-friendly instruments from the research community to gain data-driven insights in their processes.

As a final reflection, we would like to make a message explicit that might have already become apparent while reading through this chapter: process mining in healthcare is not merely about technology and algorithms, but also about people. Actionable insights to improve healthcare processes will always emerge from the interplay between the process mining outcomes and the profound domain knowledge of healthcare professionals. Hence, it is crucial that healthcare professionals build trust in the potential of process mining and the results it generates. While healthcare professionals are a crucial actor in process mining in healthcare, another stakeholder should always remain at the center of attention: the patient. In the end, healthcare organisations, healthcare professionals, process miners and many others join forces for a single goal: to provide the best possible care to patients in a way that is sustainable in the long run. Without disregarding the numerous challenges that are still ahead, this chapter demonstrated that process mining can (and should) play an important role in achieving that goal.

References

1. van der Aalst, W.M.P.: Process mining: a 360 degrees overview. In: van der Aalst, W.M.P., Carmona, J. (eds.) PMSS 2022. LNBIP, vol. 448, pp. 3–34. Springer, Cham (2022)
2. Agostinelli, S., Covino, F., D'Agnese, G., De Crea, C., Leotta, F., Marrella, A.: Supporting governance in healthcare through process mining: a case study. IEEE Access **8**, 186012–186025 (2020)
3. Alvarez, C., et al.: Discovering role interaction models in the emergency room using process mining. J. Biomed. Inform. **78**, 60–77 (2018)
4. Andrews, R., Suriadi, S., Ouyang, C., Poppe, E.: Towards event log querying for data quality. In: Panetto, H., Debruyne, C., Proper, H.A., Ardagna, C.A., Roman, D., Meersman, R. (eds.) OTM 2018. LNCS, vol. 11229, pp. 116–134. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02610-3_7
5. Andrews, R., Suriadi, S., Wynn, M., ter Hofstede, A.H.M., Rothwell, S.: Improving patient flows at St. Andrew's War Memorial Hospital's emergency department through process mining. In: vom Brocke, J., Mendling, J. (eds.) Business Process Management Cases. MP, pp. 311–333. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-58307-5_17
6. Andrews, R., Wynn, M.T., Vallmuur, K., Ter Hofstede, A.H.M., Bosley, E.: A comparative process mining analysis of road trauma patient pathways. Int. J. Environ. Res. Public Health **17**(10), 3426 (2020)
7. Augusto, A., Carmona, J., Verbeek, E.: Advanced process discovery techniques. In: van der Aalst, W.M.P., Carmona, J. (eds.) PMSS 2022. LNBIP, vol. 448, pp. 76–107. Springer, Cham (2022)
8. Batista, E., Solanas, A.: Process mining in healthcare: a systematic review. In: Proceedings of the 2018 International Conference on Information, Intelligence, Systems and Applications, pp. 1–6. IEEE (2018)
9. Benevento, E., Aloini, D., Squicciarini, N., Dulmin, R., Mininno, V.: Queue-based features for dynamic waiting time prediction in emergency department. Meas. Bus. Excell. **23**(4), 458–471 (2019)
10. Benevento, E., Dixit, P.M., Sani, M.F., Aloini, D., van der Aalst, W.M.P.: Evaluating the effectiveness of interactive process discovery in healthcare: a case study. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) BPM 2019. LNBIP, vol. 362, pp. 508–519. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37453-2_41
11. Berghout, M., Van Exel, J., Leensvaart, L., Cramm, J.M.: Healthcare professionals' views on patient-centered care in hospitals. BMC Health Serv. Res. **15**(1), 1–13 (2015). https://doi.org/10.1186/s12913-015-1049-z
12. Bottrighi, A., Chesani, F., Mello, P., Montali, M., Montani, S., Terenziani, P.: Conformance checking of executed clinical guidelines in presence of basic medical knowledge. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011. LNBIP, vol. 100, pp. 200–211. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28115-0_20
13. Carmona, J., van Dongen, B., Weidlich, M.: Conformance checking: foundations, milestones and challenges. In: van der Aalst, W.M.P., Carmona, J. (eds.) PMSS 2022. LNBIP, vol. 448, pp. 155–190. Springer, Cham (2022)
14. Caron, F., Vanthienen, J., Vanhaecht, K., Van Limbergen, E., De Weerdt, J., Baesens, B.: Monitoring care processes in the gynecologic oncology department. Comput. Biol. Med. **44**, 88–96 (2014)

15. Conforti, R., La Rosa, M., ter Hofstede, A.H.M.: Filtering out infrequent behavior from business process event logs. IEEE Trans. Knowl. Data Eng. **29**(2), 300–314 (2016)
16. Dallagassa, M.R., dos Santos Garcia, C., Scalabrin, E.E., Ioshii, S.O., Carvalho, D.R.: Opportunities and challenges for applying process mining in healthcare: a systematic mapping study. J. Ambient. Intell. Humaniz. Comput. **13**, 165–182 (2021). https://doi.org/10.1007/s12652-021-02894-7
17. De Roock, E., Martin, N.: Process mining in healthcare - an updated perspective on the state of the art. J. Biomed. Inform. **127**, 103995 (2022)
18. De Weerdt, J., Wynn, M.T.: Foundations of process event data. In: van der Aalst, W.M.P., Carmona, J. (eds.) PMSS 2022. LNBIP, vol. 448, pp. 193–211. Springer, Cham (2022)
19. Depaire, B., Martin, N.: Data-driven process simulation. In: Encyclopedia of Big Data Technologies, pp. 607–614 (2019)
20. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-intensive processes: characteristics, requirements and analysis of contemporary approaches. J. Data Semant. **4**(1), 29–57 (2015). https://doi.org/10.1007/s13740-014-0038-4
21. Di Francescomarino, C., Ghidini, C.: Predictive process monitoring. In: van der Aalst, W.M.P., Carmona, J. (eds.) PMSS 2022. LNBIP, vol. 448, pp. 320–346. Springer, Cham (2022)
22. DiMatteo, M.R., Giordani, P.J., Lepper, H.S., Croghan, T.W.: Patient adherence and medical treatment outcomes a meta-analysis. Med. Care **40**(9), 794–811 (2002)
23. Dixit, P.M., Verbeek, H.M.W., Buijs, J.C.A.M., van der Aalst, W.M.P.: Interactive data-driven process model construction. In: Trujillo, J.C., et al. (eds.) ER 2018. LNCS, vol. 11157, pp. 251–265. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00847-5_19
24. Djulbegovic, B., Guyatt, G.H.: Progress in evidence-based medicine: a quarter century on. The Lancet **390**(10092), 415–423 (2017)
25. Duma, D., Aringhieri, R.: Mining the patient flow through an emergency department to deal with overcrowding. In: Cappanera, P., Li, J., Matta, A., Sahin, E., Vandaele, N.J., Visintin, F. (eds.) ICHCSE 2017. SPMS, vol. 210, pp. 49–59. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66146-9_5
26. Engelhorn, M.: Semantics and big data semantics methods for data processing and searching large amounts of data. In: Langkafel, P. (ed.) Big Data in Medical Science and Healthcare Management, pp. 177–196. Walter de Gruyter, Berlin (2016)
27. Erdogan, T.G., Tarhan, A.: Systematic mapping of process mining studies in healthcare. IEEE Access **6**, 24543–24567 (2018)
28. Farid, N.F., De Kamps, M., Johnson, O.A.: Process mining in frail elderly care: a literature review. In: Proceedings of the 12th International Joint Conference on Biomedical Engineering Systems and Technologies-Volume 5: HEALTHINF, vol. 5, pp. 332–339. SciTePress, Science and Technology Publications (2019)
29. Fernandez-Llatas, C.: Interactive Process Mining in Healthcare. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-53993-1
30. Fernández-Llatas, C., et al.: Behaviour patterns detection for persuasive design in nursing homes to help dementia patients. In: Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp. 6413–6417. IEEE (2011)
31. Fernandez-Llatas, C., Lizondo, A., Monton, E., Benedi, J.-M., Traver, V.: Process mining methodology for health process tracking using real-time indoor location systems. Sensors **15**(12), 29821–29840 (2015)

32. Figl, K.: Comprehension of procedural visual business process models. Bus. Inf. Syst. Eng. **59**(1), 41–67 (2017). https://doi.org/10.1007/s12599-016-0460-2
33. Franck, T., Bercelli, P., Aloui, S., Augusto, V.: A generic framework to analyze and improve patient pathways within a healthcare network using process mining and discrete-event simulation. In: Proceedings of the 2020 Winter Simulation Conference, pp. 968–979. IEEE (2020)
34. Gatta, R., et al.: Clinical guidelines: a crossroad of many research areas. Challenges and opportunities in process mining for healthcare. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) BPM 2019. LNBIP, vol. 362, pp. 545–556. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37453-2_44
35. Gatta, R., et al.: What role can process mining play in recurrent clinical guidelines issues? A position paper. Int. J. Environ. Res. Public Health **17**(18), 6616 (2020)
36. Ghasemi, M., Amyot, D.: Process mining in healthcare: a systematised literature review. Int. J. Electron. Healthc. **9**(1), 60–88 (2016)
37. Guzzo, A., Rullo, A., Vocaturo, E.: Process mining applications in the healthcare domain: a comprehensive review. Wiley Interdisc. Rev. Data Min. Knowl. Discov. **12**(2), e1442 (2022)
38. Huang, Z., Dong, W., Ji, L., Yin, L., Duan, H.: On local anomaly detection and analysis for clinical pathways. Artif. Intell. Med. **65**(3), 167–177 (2015)
39. Jans, M., Soffer, P., Jouck, T.: Building a valuable event log for process mining: an experimental exploration of a guided process. Enterp. Inf. Syst. **13**(5), 601–630 (2019)
40. Kim, E., et al.: Discovery of outpatient care process of a tertiary university hospital using process mining. Healthc. Inform. Res. **19**(1), 42–49 (2013)
41. Kirchner, K., Herzberg, N., Rogge-Solti, A., Weske, M.: Embedding conformance checking in a process intelligence system in hospital environments. In: Lenz, R., Miksch, S., Peleg, M., Reichert, M., Riaño, D., ten Teije, A. (eds.) KR4HC/ProHealth - 2012. LNCS (LNAI), vol. 7738, pp. 126–139. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36438-9_9
42. Kovalchuk, S.V., Funkner, A.A., Metsker, O.G., Yakovlev, A.N.: Simulation of patient flow in multiple healthcare units using process and data mining techniques for model identification. J. Biomed. Inform. **82**, 128–142 (2018)
43. Kurniati, A.P., Johnson, O., Hogg, D., Hall, G.: Process mining in oncology: a literature review. In: Proceedings of the 2016 International Conference on Information Communication and Management, pp. 291–297. IEEE (2016)
44. Kusuma, G.P., Hall, M., Gale, C.P., Johnson, O.A.: Process mining in cardiology: a literature review. Int. J. Biosci. Biochem. Bioinform. **8**(4), 226–236 (2018)
45. Lenz, R., Peleg, M., Reichert, M.: Healthcare process support: achievements, challenges, current research. Int. J. Knowl.-Based Organ. **2**(4) (2012)
46. Lenz, R., Reichert, M.: IT support for healthcare processes - premises, challenges, perspectives. Data Knowl. Eng. **61**(1), 39–58 (2007)
47. Lira, R., et al.: Process-oriented feedback through process mining for surgical procedures in medical training: the ultrasound-guided central venous catheter placement case. Int. J. Environ. Res. Public Health **16**(11), 2019 (1877)
48. Lu, X., Tabatabaei, S.A., Hoogendoorn, M., Reijers, H.A.: Trace clustering on very large event data in healthcare using frequent sequence patterns. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNCS, vol. 11675, pp. 198–215. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26619-6_14
49. Manchaiah, V.K.C., Stephens, D., Meredith, R.: The patient journey of adults with hearing impairment: the patients' views. Clin. Otolaryngol. **36**(3), 227–234 (2011)

50. Mannhardt, F., Blinde, D.: Analyzing the trajectories of patients with sepsis using process mining. In: CEUR Workshop Proceedings, vol. 1859, pp. 72–80 (2017)
51. Mans, R.S., Schonenberg, M.H., Song, M., van der Aalst, W.M.P., Bakker, P.J.M.: Process mining in healthcare. In: Proceedings of the 2008 International Conference on Health Informatics, pp. 118–125 (2008)
52. Mans, R.S., van der Aalst, W.M.P., Vanwersch, R.J.B.: Process Mining in Healthcare: Evaluating and Exploiting Operational Healthcare Processes. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-16071-9
53. Mans, R.S., van der Aalst, W.M.P., Russell, N.C., Bakker, P.J.M., Moleman, A.J.: Process-aware information system development for the healthcare domain - consistency, reliability, and effectiveness. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 635–646. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12186-9_61
54. Martin, N.: Data quality in process mining. In: Fernandez-Llatas, C. (ed.) Interactive Process Mining in Healthcare. HI, pp. 53–79. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-53993-1_5
55. Martin, N., et al.: Recommendations for enhancing the usability and understandability of process mining in healthcare. Artif. Intell. Med. **109**, 101962 (2020)
56. Martin, N., et al.: Opportunities and challenges for process mining in organizations: results of a Delphi study. Bus. Inf. Syst. Eng. **63**(5), 511–527 (2021). https://doi.org/10.1007/s12599-021-00720-0
57. Maurer, F.A., Smith, C.M.: Community/Public Health Nursing Practice: Health for Families and Populations. Elsevier Saunders, St. Louis (2013)
58. Mertens, S., Gailly, F., Poels, G.: Towards a decision-aware declarative process modeling language for knowledge-intensive processes. Expert Syst. Appl. **87**, 316–334 (2017)
59. Munoz-Gama, J., et al.: Process mining for healthcare: characteristics and challenges. J. Biomed. Inform. **127**, 103994 (2022)
60. Partington, A., Wynn, M., Suriadi, S., Ouyang, C., Karnon, J.: Process mining for clinical processes: a comparative analysis of four Australian hospitals. ACM Trans. Manag. Inf. Syst. **5**(4), 1–18 (2015)
61. Pebesma, J., et al.: Clustering cardiovascular risk trajectories of patients with type 2 diabetes using process mining. In: Proceedings of the 2019 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp. 341–344. IEEE (2019)
62. Peleg, M.: Computer-interpretable clinical guidelines: a methodological review. J. Biomed. Inform. **46**(4), 744–763 (2013)
63. Pereira Detro, S., Santos, E.A.P., Panetto, H., De Loures, E., Lezoche, M., Cabral Moro Barra, C.: Applying process mining and semantic reasoning for process model customisation in healthcare. Enterp. Inf. Syst. **14**(7), 983–1009 (2020)
64. Pika, A., Wynn, M.T., Budiono, S., ter Hofstede, A.H.M., van der Aalst, W.M.P., Reijers, H.A.: Privacy-preserving process mining in healthcare. Int. J. Environ. Res. Public Health **17**(5), 1612 (2020)
65. Poon, A.I.F., Sung, J.J.Y.: Opening the black box of AI-medicine. J. Gastroenterol. Hepatol. **36**(3), 581–584 (2021)
66. Rathert, C., Wyrwich, M.D., Boren, S.A.: Patient-centered care and outcomes: a systematic review of the literature. Med. Care Res. Rev. **70**(4), 351–379 (2013)
67. Rebuge, Á., Ferreira, D.R.: Business process analysis in healthcare environments: a methodology based on process mining. Inf. Syst. **37**(2), 99–116 (2012)

68. Rinner, C., Helm, E., Dunkl, R., Kittler, H., Rinderle-Ma, S.: An application of process mining in the context of melanoma surveillance using time boxing. In: Daniel, F., Sheng, Q.Z., Motahari, H. (eds.) BPM 2018. LNBIP, vol. 342, pp. 175–186. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11641-5_14
69. Rojas, E., Capurro, D.: Characterization of drug use patterns using process mining and temporal abstraction digital phenotyping. In: Daniel, F., Sheng, Q.Z., Motahari, H. (eds.) BPM 2018. LNBIP, vol. 342, pp. 187–198. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11641-5_15
70. Rojas, E., Cifuentes, A., Burattin, A., Munoz-Gama, J., Sepúlveda, M., Capurro, D.: Analysis of emergency room episodes duration through process mining. In: Daniel, F., Sheng, Q.Z., Motahari, H. (eds.) BPM 2018. LNBIP, vol. 342, pp. 251–263. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11641-5_20
71. Rojas, E., Munoz-Gama, J., Sepúlveda, M., Capurro, D.: Process mining in healthcare: a literature review. J. Biomed. Inform. **61**, 224–236 (2016)
72. Roulin, D., Muradbegovic, M., Addor, V., Blanc, C., Demartines, N., Hübner, M.: Enhanced recovery after elective colorectal surgery-reasons for non-compliance with the protocol. Dig. Surg. **34**(3), 220–226 (2017)
73. Rule, A., Chiang, M.F., Hribar, M.R.: Using electronic health record audit logs to study clinical activity: a systematic review of aims, measures, and methods. J. Am. Med. Inform. Assoc. **27**(3), 480–490 (2020)
74. Safran, C., et al.: Toward a national framework for the secondary use of health data: an American Medical Informatics Association white paper. J. Am. Med. Inform. Assoc. **14**(1), 1–9 (2007)
75. Sato, D.M.V., de Freitas, S.C., Dallagassa, M.R., Scalabrin, E.E., Portela, E.A.P., Carvalho, D.R.: Conformance checking with different levels of granularity: a case study on bariatric surgery. In: Proceedings of the 2020 International Congress on Image and Signal Processing, Biomedical Engineering and Informatics, pp. 820–826. IEEE (2020)
76. Silva, B.M.C., Rodrigues, J.J.P.C., de la Torre Díez, I., López-Coronado, M., Saleem, K.: Mobile-health: a review of current state in 2015. J. Biomed. Inform. **56**, 265–272 (2015)
77. Stefanini, A., Aloini, D., Benevento, E., Dulmin, R., Mininno, V.: Performance analysis in emergency departments: a data-driven approach. Meas. Bus. Excell. **22**(2), 130–145 (2018)
78. Tamburis, O., Esposito, C.: Process mining as support to simulation modeling: a hospital-based case study. Simul. Model. Pract. Theory **104**, 102149 (2020)
79. Ten Teije, A., et al.: Improving medical protocols by formal methods. Artif. Intell. Med. **36**(3), 193–209 (2006)
80. van Andel, V., Beerepoot, I., Lu, X., van de Weerd, I., Reijers, H.A.: DEUCE: a methodology for detecting unauthorized access of electronic health records using process mining. In: Proceedings of the European Conference on Information Systems, p. 1340 (2021)
81. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
82. van der Spoel, S., van Keulen, M., Amrit, C.: Process prediction in noisy data sets: a case study in a Dutch hospital. In: Cudre-Mauroux, P., Ceravolo, P., Gašević, D. (eds.) SIMPDA 2012. LNBIP, vol. 162, pp. 60–83. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40919-6_4
83. van Eck, M.L., Lu, X., Leemans, S.J.J., van der Aalst, W.M.P.: PM^2: a process mining project methodology. In: Zdravkovic, J., Kirikova, M., Johannesson, P.

(eds.) CAiSE 2015. LNCS, vol. 9097, pp. 297–313. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19069-3_19

84. van Hulzen, G., Martin, N., Depaire, B., Souverijns, G.: Supporting capacity management decisions in healthcare using data-driven process simulation. J. Biomed. Inform. **129**, 104060 (2022)
85. van Zelst, S.J., Mannhardt, F., de Leoni, M., Koschmider, A.: Event abstraction in process mining: literature review and taxonomy. Granular Comput. **6**(3), 719–736 (2020). https://doi.org/10.1007/s41066-020-00226-2
86. Vanbrabant, L., Martin, N., Ramaekers, K., Braekers, K.: Quality of input data in emergency department simulations: framework and assessment techniques. Simul. Model. Pract. Theory **91**, 83–101 (2019)
87. Wang, D., et al.: Representation primitives, process models and patient data in computer-interpretable clinical practice guidelines: a literature review of guideline representation models. Int. J. Med. Inform. **68**(1–3), 59–70 (2002)
88. Wang, F.: The roles of preventive and curative health care in economic development. PLoS ONE **13**(11), e0206808 (2018)
89. Wiler, J.L., Welch, S., Pines, J., Schuur, J., Jouriles, N., Stone-Griffith, S.: Emergency department performance measures updates: proceedings of the 2014 emergency department benchmarking alliance consensus summit. Acad. Emerg. Med. **22**(5), 542–553 (2015)
90. Williams, R., Rojas, E., Peek, N., Johnson, O.A.: Process mining in primary care: a literature review. Stud. Health Technol. Inform. **247**, 376–380 (2018)
91. World Health Organization: WHO releases new international classification of diseases (ICD 11) (2018). https://www.who.int/news-room/detail/18-06-2018-who-releases-new-international-classification-of-diseases-(icd-11). Accessed 08 Apr 2022
92. Yoo, S., et al.: Assessment of hospital processes using a process mining technique: outpatient process analysis at a tertiary hospital. Int. J. Med. Inform. **88**, 34–43 (2016)

Process Mining for Financial Auditing

Mieke Jans[1,2](✉) and Marc Eulerich[3]

[1] Hasselt University, Martelarenlaan 42, 3500 Hasselt, Belgium
mieke.jans@uhasselt.be
[2] Maastricht University, Minderbroedersberg 4-6, 6211 LK Maastricht, Netherlands
[3] University Duisburg-Essen, Lotharstr. 65, 47057 Duisburg, Germany
marc.eulerich@uni-due.de

Abstract. Over the last years, process mining has increasingly demonstrated its potential as a valuable tool for internal and external auditors. Thereby, the possible use cases in the field of auditing are manifold. This chapter focuses especially on the use of process mining in the context of financial audits, which are relevant for both, internal and external auditors. Beside a short explanation of the different types of auditors, this chapter aims to connect process mining to the different process steps of an internal (and later also external) audit and discusses the similarities and differences between both areas.

Keywords: Financial auditing · Internal auditing · External auditing · Process mining

1 Introduction

Financial auditing refers to an external independent party that examines the financial statements of an organization and formulates an opinion on how well those statements present a true and fair view of its financial performance and position. Apart from hiring external auditors to conduct such investigations, larger companies also have an internal department that conducts comparable audits, albeit through a wider lens. Where external auditing is only concerned with assuring the quality of financial reporting, internal auditing extends this with an efficiency perspective on the entire functioning of an organisation. Independent from the business units, the internal audit department examines the organisation's governance mechanisms. A key aspect for both external and internal audits is to assess whether processes are in control, whether prominent risks are mitigated (partly by their process design), and whether the input data for the financial statements are complete, accurate, and valid. Consequently, both internal and external audits can benefit from process mining, since it provides the auditor with a realistic view on how processes, that indirectly impact the financial reporting, are being executed. Not surprisingly, process mining has in recent years increasingly demonstrated its potential as a valuable tool for financial auditing.

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 445–467, 2022.
https://doi.org/10.1007/978-3-031-08848-3_15

Running a process mining analysis in the context of an audit, internal or external, requires a specific approach that takes into account the preliminaries of audit engagements. This chapter will take the reader through these audit-specific concerns. The chapter starts with a short introduction into financial auditing. Both internal and external audit will be introduced, along with the connection between the two audits. Readers that are familiar with this topic, can immediately proceed with the next section, that discusses process mining in the internal audit function. All phases of the internal audit are explained first, and then revisited while integrating process mining in it. Section 4 brings the external audit in the picture. How does a process mining approach differ between the external and the internal auditor? Sects. 5 and 6 deal with the practical organisation of bringing the right expertise in-house and how to move from data to audit evidence. We end the chapter with open challenges in Sect. 7 and conclude in Sect. 8.

2 Financial Auditing

Financial statements are key when a stakeholder wishes to inform him- or herself about an organization. Investors, banks, employees, customers, vendors, etc. are all parties that might be interested in the financial situation of an organization before partnering up. To this end, the officially published financial statements of an organization are the primary documents to consult. These statements are prepared by the organization, adhering to (national or international) accounting standards. The statements include minimally a *balance sheet* and an *income statement*. Depending under which legislation the organization reports, also a cash-flow statement is included. The balance sheet presents an overview of the assets, liabilities, and capital that the organization possesses at a particular moment of time. The income statement provides an overview of the revenues and the incurred expenses over a period of time, mostly one year. The combination of the revenues and expenses presents the monetary gain or loss that the organization realized over that accounting year.

It goes without saying that it is important that the statements are reliable, given the numerous decisions that are taken based on this information: investors start, continue, or quit investing, banks offer loans or not, customers churn or not. The guiding principle is that the statements need to present a ‘true and fair view’ of the financial situation of the organization. It is the key responsibility of the auditor to safeguard this principle: they provide *reasonable assurance* that the statements indeed present such a view. This assurance is primarily given by the external auditor (legal requirement for companies from a certain size onward), but this can also be assisted by the internal auditor.

This section will provide a general overview on the governance mechanism that financial auditing holds for companies. It will explain the goals and characteristics of both external and internal auditing and the interaction between these two. The internal audit department has the latitude to fully implement process mining at the core of the business in a continuous fashion. The findings of these continuous monitoring efforts can be passed on to the external auditor who can use these findings as input for their own investigation. Alternatively, the external auditor can run their own 'one shot'-process mining analysis during the annual audit engagement. The biggest traction of process mining in the auditing field is achieved through the internal audit, due to its possible embedding in the core of the organization. The interplay between these two audit settings is elaborated on further in this section.

2.1 Purpose of the External Financial Audit

The external auditor typically conducts an annual audit [1]. The auditor audits and reports on the procedures and the recorded transactions relied upon to prepare the financial statements. When the auditor reports a 'clean opinion,' the financial statements are presumed to be free of material misstatements and hence reliable to share- and stakeholders for decision making [2].

As mentioned, the objective of an external audit is to obtain reasonable assurance about whether the financial statements are free of material misstatement. It is intended to increase the reliability of the information contained in the annual financial statement. Nevertheless, an audit must be carried out efficiently, which might create tension with the goal of providing assurance. To meet the two requirements, efficiency and reasonable assurance, in the context of the audit, the so-called risk-based audit approach is applied. Following this approach, the external auditor first assesses the risks of the organisation in general, but also per department or business process. Based on this risk assessment, resources are allocated to the riskiest parts of the organization. If, for example the sales process is assessed as a key process to have under control, the auditor will put more emphasis on this process. Differently stated, more resources are allocated to auditing this element, compared to other processes that are assessed less risky.

The concept of risk-based auditing is also regulated through the relevant standard setters. For example the International Auditing and Assurance Standards Board (IAASB) issued the revised auditing standard ISA 315 (Revised 2019) "Identifying and Assessing the Risks of Material Misstatement". This standard establishes the risk identification and assessment procedures that form the basis for a risk-based financial statement audit. The risk assessment procedures are described *"to obtain an understanding of the entity and its environment, including the entity's internal control, to identify and assess the risks of material misstatement..."*. It is clear that the auditor is expected to understand how the organisation (the ‘entity’) is organized and how they mitigate risks by their internal control system. Precisely this *internal control system* is also a responsibility of the internal audit department, tying the goals of the internal audit and the external audit to each other. A *control* refers to a measure that is implemented to mitigate a certain risk. An example is the design of proper access rights to the financial accounting module to mitigate the risk of having unauthorized bookings in the financial ledger.

2.2 Purpose of the Internal Financial Audit

Internal auditing is a support unit of the company’s management that is embedded in the organization and supports the company on two levels. On one hand it aims to detect and manage potential misstatement risks and on the other hand guards the operational performance [3]. The officially established definition of internal auditing by the global Institute of Internal Auditors (IIA) is as follows:

> *“Internal auditing is an independent, objective assurance and consulting activity designed to add value and improve an organization’s operations. It helps an organization accomplish its objectives by bringing a systematic, disciplined approach to evaluate and improve the effectiveness of risk management, control, and governance processes.”* [4]

Furthermore, the IIA defines a mission of internal auditing, which states that the value of an organization is to be increased and protected through risk-oriented and objective auditing, consulting and insights.

As for external auditing, similar risk assessment standards exist for the internal audit. The IIA addresses the risk-based audit planning in their Standards 2010 - Planning, 2010.A1, 2010.A2, and 2010.C1. These standards stipulate how the Chief Audit Executive (CAE) has the responsibility to develop a plan of all upcoming internal audit engagements based on a risk assessment that is performed at least annually.

2.3 Internal and External Audit: Interplay and Common Challenges

In many respects, the practical procedure of conducting an audit is similar for both internal and external auditors. Especially since both audits include the investigation of recorded financial transactions in the light of the prepared financial statements. In the course of all audits of a material[1] and formal nature, the regularity and reliability of the generated data must be assessed. Hence, the overall aim is to ensure quality control of all published financial information, taking into account the processes that precede the reporting.

[1] Meaning ‘significant’ in an audit context.

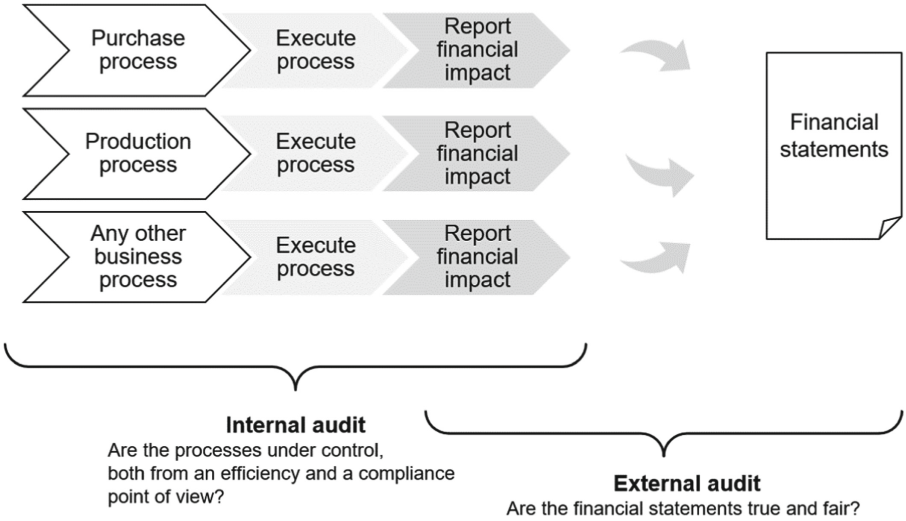


Fig. 1. Interplay between internal and external audit

Figure 1 provides a simplified overview of the primary responsibilities of the external and the internal audit. The external auditor is ultimately concerned with the accuracy of the financial statements, which is basically a summary of the recorded business transactions that are encapsulated in business processes. These processes typically integrate one or more recording steps since executing a business transaction alters the financial situation of the organisation and this change needs to be recorded. Figure 2 visualizes an example business process (purchase-to-pay) and its relationship to the financial statements. The process envisions an efficient execution of the purchase, but it also incorporates controls like approving once or twice a purchase order, before the purchase is placed. These type of control measures increase the level of assurance that the reported financial information is accurate and valid. In the designed process three activities trigger the reporting of a financial impact: entering a Goods Receipt document in the system should be reflected in the books by increasing the assets and booking an invoice increases your liability (you owe money to a vendor), while paying the invoice clears that liability again. Hence, during the execution of the procurement process, in parallel to the business transactions the impact on the financial situation of the company is tracked in the general ledger. All these bookings together form the basis for preparing the financial statements that are issued and audited once a year.

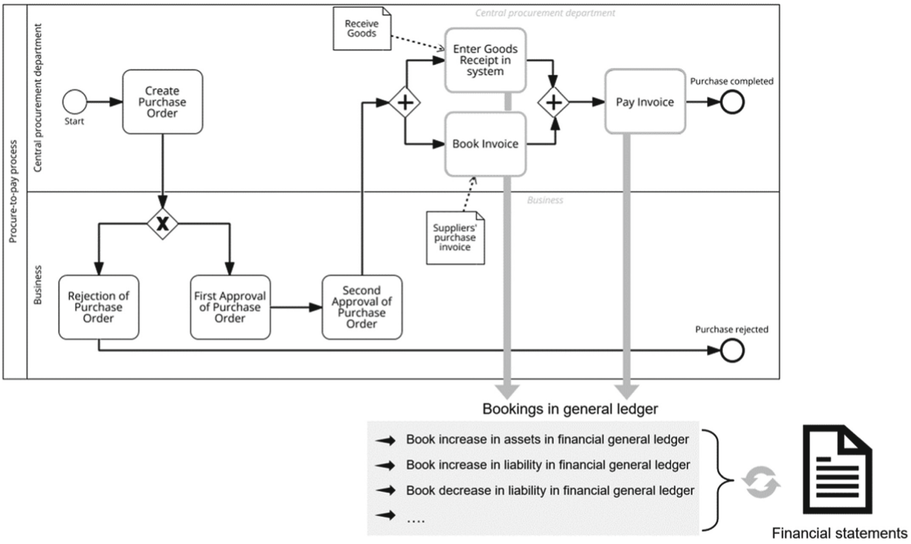

Procure-to-pay process
Central procurement department
Business
Start
Create Purchase Order
Receive Goods
Enter Goods Receipt in system
Book Invoice
Pay Invoice
Purchase completed
Suppliers' purchase invoice
Rejection of Purchase Order
First Approval of Purchase Order
Second Approval of Purchase Order
Purchase rejected
Bookings in general ledger
Book increase in assets in financial general ledger
Book increase in liability in financial general ledger
Book decrease in liability in financial general ledger
....
Financial statements

Fig. 2. Example relationship between the purchase process and the financial statements

The external audit traditionally focuses on the bookings in the general ledger and the financial statements, whereas the internal audit typically starts from the designed procedures and how they translated into financial bookings. In a world without resource limitations, the external auditor could trace back every recorded transaction to its origin and double-check whether the recorded transaction is backed-up by a real transaction (Is there for example evidence of delivering goods or encountering certain expenses?). In reality, however, the external auditor examines the organization's controls to ensure that only legitimate transactions get recorded. The auditor investigates the design of the controls and tests their effectiveness. In our procurement process, the auditor might test whether it is indeed not possible to enter an invoice and have it paid, without creating a purchase order and having it approved by someone else than the auditor. These checks make part of 'understanding the entity's control environment', an essential aspect of risk assessment as stipulated in ISA 315 and mentioned before. The working assumption is that if the processes, foreseen of enough controls, are under control, the generated financial data is accurate.

The above described examination of the control environment is not only the responsibility of the external auditor, but is also part of the internal auditor's function. Although the internal auditor includes an additional efficiency point of view, auditing the control structure and installed control measures is a core responsibility of the internal audit department. Consequently, the external auditor may rely on the internal auditor's findings. Of course, additional checks are always required.

3 Process Mining in the Internal Audit Function

Given the increasing complexity and availability of information in accounting, digital data analysis has emerged as an innovative audit approach to perform financial audits by internal and external auditors [5]. Given the strong connection between the internal audit function and the organization, we start from the perspective of internal auditing. How can process mining support the internal audit? Subsequently, this perspective will be expanded by looking at the application of process mining by external auditors over the course of their audit engagements.

3.1 Internal Auditing Background

The range of internal auditing tasks is subject to constant change, which is reflected in not only a shift of focus within the individual audit areas but also a varying understanding of the role of internal auditing. Within the traditional range of auditing activities, a distinction is made between audits of financial processes, operational processes, and management processes [6]:

- **Audits of financial processes** describe formal audits in finance and accounting with the primary objective of determining compliance with laws and to guarantee a reliable financial reporting process. The audit is aimed at ensuring the appropriateness, correctness, and reliability of the financial information. Within the scope of the audit, the aspect of compliance is expanded to include determining whether all relevant legal framework conditions and internal guidelines have been adhered to. Thus, this type of audit is focusing especially on the conformance with laws and regulations.
- **Audits of operational processes** refer to the audit of systems and processes in an organization. Its purpose is to examine the structural and procedural organization of a company by looking at the present or the future. The aim of this audit procedure is to determine whether the design of corporate processes, structures, and systems is appropriate. At the very center of this process is a review of the appropriateness and cost-effectiveness of essentially all corporate processes to ensure future viability. In this context, no common standards can be established as criteria of comparability. The target values must first be determined by internal auditing through analyses of relevant processes. This type of internal audit increasingly adopts the function of an advisory activity and can be seen in the light of continuous process improvement, a key goal of business process management.
- **Audits of management processes** focus on assessing the performance of management processes and institutions. The audit process in this field includes a past-oriented root cause analysis, paired with a potential identification of future points of weakness. In contrast to operational auditing, the focus of the review is no longer on the operational process but on management and its strategic decisions. Just as in operational auditing, the audit criteria - efficiency and effectiveness - remain to be the focus, and thus contribute to the future safeguard of the organization.

A major trend can be noted in the field of internal auditing activities. The solely past-oriented audit is increasingly complemented by future-oriented auditing activities. This expansion is accompanied by a further development of auditing activities. Namely, internal auditing is more and more intended to initiate approaches to solve organizational problems. Providing improvement recommendations can therefore be referred to as the overall mission of all internal audit activities. Consequently, the internal audit shifts from a purely control-oriented view towards an enterprise-wide view.

3.2 The Internal Audit Process

The internal audit process generally pertains to the structure and standardized procedure of auditing activities of the internal audit function and can be structured following the so-called *phase model* (see Fig. 3). The phase model of the audit activity is organized according to a sequence of audit phases. These audit phases are inherently separate units in terms of both content and methodology, yet there is a predetermined order for their execution. In fact, they are connected in such a manner that the start of the respective phase is directly linked to the completion of the preceding phase. As a result, the phase model is in effect the process model of the internal audit.

1. The traditional internal audit process typically starts with the (risk-based) *planning of the (annual) audit schedule* in order to allocate the internal audit resources (e.g. staff) to various potential audit objects. Usually, this phase is performed by the chief audit executive and defines the areas of the organization that will be audited in the forthcoming year.
2. Subsequently, the *planning of the process audit* is being conducted. Thus, each auditor or audit team plans and prepares their upcoming audit engagement in terms of objectives, scoping, and needed audit methodologies.
3. The third phase of the audit process consists of *conducting the audit* on-site, namely the actual audit. In this phase, the internal auditors apply different methodologies, such as data analysis, interviews, walk-throughs, etc., to gather audit evidence and achieve the overall objective of the audit.
4. The final phase of each audit process is the preparation and finalization of an audit report for the audited entity and relevant stakeholders, such as the CEO, CFO, or the Audit Committee to *communicate the results.* This report encompasses the main objectives of the audit and outlines any performed audit steps and the obtained evidence. Ultimately, the report evaluates the findings and offers additional recommendations.
5. As an additional phase - directly linked to the specific audit engagements - a *follow-up* attempts to monitor the enhancement and improvement of the audited entity based on the previously defined recommendations.

Figure 3 visualizes the phase model of the internal audit, along with possible ways to integrate process mining activities in the different phases. The following paragraphs will describe these starting points in greater depth. We present a running example to further explain the connection between internal auditing and process mining.

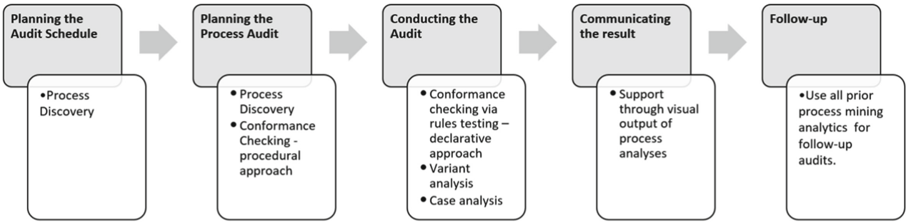

Planning the Audit Schedule
•Process Discovery
Planning the Process Audit
• Process Discovery
• Conformance Checking - procedural approach
Conducting the Audit
• Conformance checking via rules testing – declarative approach
• Variant analysis
• Case analysis
Communicating the result
• Support through visual output of process analyses
Follow-up
•Use all prior process mining analytics for follow-up audits.

Fig. 3. Integration of process mining activities in the internal audit process

3.3 Planning the Audit Schedule

Imagine an exemplary audit engagement that should assess the functioning and the exposure to risk of a manufacturing company. Through an internal audit, the auditor needs to determine whether the internal rules and guidelines (controls) are fulfilled, if the processes are efficient, if there are specific risks that are not mitigated by appropriate controls and whether there is room for improvement.

The first phase of this audit entails planning the audit; determining the allocation of resources. In order to do so effectively, the auditor could visualize the organization's core business processes and then analyze them in terms of conformity and process efficiency. This would require the use of process discovery algorithms (as described in [7,8]). Any variances, weaknesses, and risks identified throughout this phase can subsequently serve as indicators to guide the allocation of resources. In other words, this phase involves an attempt to "explore" the processes and process discovery is well-positioned to support this phase. In a more mature setting, where the core business processes are efficiently logged and event logs can be extracted automatically, a quick discovery step can yield insights in which processes are highly structured and which aren't simply by looking at the discovered process models and their level of 'spaghettiness'[2] (see [9]).

Measures of 'structuredness' are necessary to turn this step objective, to select which process to give prior attention. For example, one could identify processes with a very high number of variants. In the running example, the purchase-to-pay (P2P) process might show a high number of variants, whereas the hiring process only exhibit a low number of process variants. This is an indication of a myriad of possible execution variants in the P2P process, accompanied with higher risk exposure. However, indicators such as '20 variants per 100 cases' are very generic. This can relate to two extremes (and everything in between): one variant representing 81 cases and 19 variants each representing a single case is one extreme, versus all 20 variants representing five cases as the other extreme. Consequently, the distribution of variants might be more insightful. Possible measures for structuredness are variance, self-loops, repetition and batch-processing [10]. This enables the identification of potential audit objects (risky processes) and, preferably, a simultaneous assessment of the risks

[2] 'Highly structured' is directly associated with 'less risky'.

inherent to these objects. In this phase of the audit process, process mining is consequently used to support the creation of the risk-based audit plan.

For Example, Table 1 presents a set of different structuredness measures to gain an insight in which process is more or less structured than other processes. The structuredness measures are calculated for the P2P process of different plants, helping the auditor to classify the individual risk level of each business unit. Based on this exploratory phase, the audit schedule would reserve resources to an audit of the P2P process in the Norway facility in *year n*, and leave the audit of the P2P process of the USA for *year n+1*, and the audit of Germany for *year n+2*, perhaps together with the Belgium plant. Also the other processes would be integrated in the audit schedule, typically covering a cycle of four to six years.

Table 1. Measures of process structuredness, used to plan the Audit Schedule

Business unit	Variants per 100 cases	Repetition per 100 cases	Self-loops	Classification
USA	1.5	198	24	High risk
Norway	6	151	56	Very high risk
Belgium	2	206	19	Very low risk
Germany	1	76	23	Low risk
...	...	...	...	...

3.4 Planning the Audit

Once the audit object of a specific audit is identified -the P2P process in the Norwegian facility in our example- the audit is scheduled and the audit engagement needs to be planned in more detail. Starting with process discovery, the individual steps of the audited unit can be visualized and analyzed before the actual on-site audit. This helps the auditor gain a better understanding of the area to be audited and familiarize with the unique features of the process environment. The deduction of the process model based on available transactions facilitates identifying parallel process steps, loops, and undesired process skips [11,12]. This approach bears the advantage of verifying the assurance of the process flows on one hand and revealing process steps that require further examination on the other hand. A first scan of the discovered process model involves a critical look at the discovered edges. Even when not looking at complete process executions from start to end, examining the most frequent direct flows yields interesting information.

For example, when the default discovered process shows an edge between ‘book invoice’ and ’first approval order’, it is clear that unexpected sequences are present in a significant part of the transactions. This information is valuable when planning the audit, since it provides indicators of which directions should be investigated more thoroughly.

After process discovery, the auditor can perform a first conformance check against the normative model to verify whether the individual process steps of the examined transactions comply with the previously defined process. In contrast with the exploratory process discovery step, the focus now shifts towards complete process executions. Since the auditor is still in the planning phase, it is recommended to compare the logged transactions with a procedural normative process model (like a BPMN-model that represents the 'to be'-model). The idea is to have a first impression of the level of business alignment: "Are the real process and the process model properly aligned?" [13]. This approach enables identifying variants that are not in line with the (often overly simplified) normative process model. Further investigation during the audit will reveal the real, associated risks. However, during the phase of planning the audit, the auditor can already have a first look at variants that represent a majority of non-conforming cases.

For example, Table 2 presents a set of variants in our P2P process that deviate from the normative model in Fig. 2 and that could be skimmed during this audit phase (see how to analyze deviations between observed and modeled behavior in [14]).

- The first variant presents a double second approval, which indeed deviates from the process model, but would not trigger any additional audit inquiries in the next phase. This is called an *exception*: a deviation from the normative model, but not presenting a risk according to the auditor. In these cases, the auditor can *clear* the deviation'.
- The second example shows that 16% of the cases are not associated with a receipt of goods. Although this comes across worrisome initially, there might be a perfectly reasonable explanation. Perhaps these purchases relate to services and not to goods. The auditor should test this hypothesis in the next audit phase, however. As for now, this deviation is classified as *potential compliance issue*.
- The last example deviating variant in Table 2 is an example where the auditor cannot formulate any hypothesis on situations where this deviation would not represent a risk. As such, the deviation can directly be classified as an *anomaly*. This, too, will be taken as input to the next audit phase.

Table 2. Example output of non-conforming variants

Deviating variant	Presence in log	Classification
Create PO - First approval - Second approval - Second approval - Enter Goods Receipt - Book invoice - Pay invoice	21%	Exception
Create PO - First approval - Second approval - Book invoice - Pay invoice ...	16% ...	Potential compliance issue
Create PO - Pay invoice ...	10% ...	Anomaly

So within the scope of planning the process audit, both process discovery and conformance checking can be used to determine the focus of the audit. The insights that are gained during this phase provide guidance on which special features or possible deviations of the prescribed process warrant further investigation. Sometimes, these analyses already allow for the identification of potential findings before the actual audit takes place. It should be noted that although process deviations might be identified, it does not necessarily represent a financial statement risk. Further tests are required to uncover potential "false positives" [15]. This will be elaborated on in the next phase.

3.5 Conducting the Audit

When conducting the audit, the auditor takes a deep dive into the control and operational environment. Specific analyses are conducted during this audit phase. Whereas the previous phases were preparatory, this phase is targeted to identifying risks and weaknesses in the reviewed process. As mentioned before, the opinion that auditors issue at the end of the engagement is partly based on an evaluation of the existing controls in terms of their effectiveness and efficiency [16].

The rationale of how internal controls are evaluated by making use of transactional data is presented as a process in Fig. 4 [17]. During the previous two audit phases, potential violations of internal controls have been identified. Also, a preliminary start of deviation analysis is taken in these phases (see example in Table 2). When conducting the audit, this preliminary analysis is extended. The purpose is to classify all deviations that stem from a procedural conformance check as either an exception or an anomaly. To date, in practice these conformance checks are solely using a control-flow perspective. In theory, however, this can be extended to a multi-paradigm conformance check that, for example, includes a Segregation of Duties control between two activities.

Starting from deviations, an iterative cycle presents itself, until all deviations are classified as either an anomaly or an exception. When the deviation is classified as a potential compliance issue, a follow-up investigation is triggered. Taking back our example of missing the receipt of goods, this might–or might not–be a compliance issue. It was not classified as anomaly, because a possible explanation could be formulated: perhaps the purchase related to services, making the receipt of goods an illogical activity. The auditor should test this potential explanation in order to reach a conclusion on whether this deviation is an anomaly or an exception to the normative process. To do so, the auditor collects all cases where the receipt of goods is missing, and subjects this (filtered) log to a conformance check where the formulated hypothesis is tested. Hence, a declarative constraint is checked on this set of transactions: 'if the receipt of goods is missing, the purchase relates to services'. The cases that follow this rule can be cleared and listed as exception. If there are still cases that are not cleared by this possible explanation, the same approach is repeated.

The cycle, as described above, presents the theory. In practice, too many deviations are presented to inspect all of them and auditors fall back to a sampling approach. Current research is looking into weak supervision and active learning to support the auditor in the iterative cycle such that a full-population testing can be reached [18]. The goal is to present deviations in an intelligent way to the auditor whom provides a classifier with the labels *anomaly*, *exception*, or *uncertain*. In case of uncertainty, the auditor provides a possible rule to check (the hypothesis). Based on the iterative human input the classifier can support a classification of all identified deviations, preferably with minimal expert knowledge input [17].

By identifying deviations that can be tracked to the document level, the auditor can also direct the auditing activities in a target-oriented manner. More specifically, in-depth variant analysis and case analysis can be used to evaluate both the functioning of the internal controls and the process performance. The majority of process mining solutions offer various metrics in this regard, such as duration of the process, number of process steps, number of loops, number of variants, etc. Internal auditors can create additional value through this third dimension by auditing, analyzing, and identifying improvement opportunities and utilize process mining for consulting activities in addition to its conventional auditing activities.

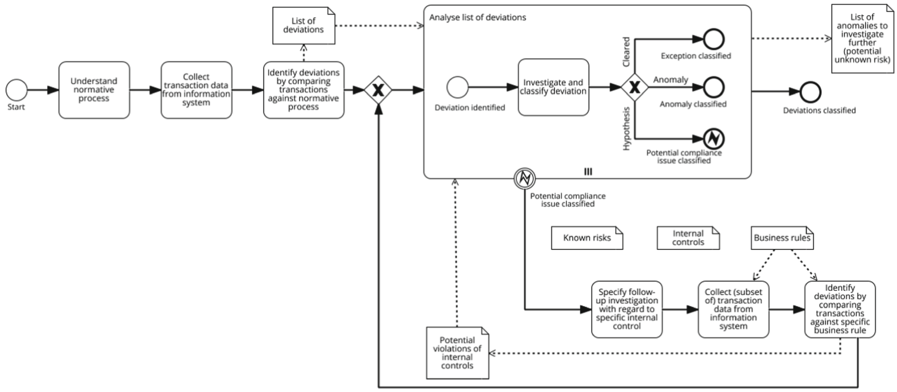

Start
Understand normative process
Collect transaction data from information system
List of deviations
Identify deviations by comparing transactions against normative process
Analyse list of deviations
Deviation identified
Investigate and classify deviation
Cleared
Exception classified
Anomaly
Anomaly classified
Hypothesis
Potential compliance issue classified
List of anomalies to investigate further (potential unknown risk)
Deviations classified
Potential compliance issue classified
Known risks
Internal controls
Business rules
Specify follow-up investigation with regard to specific internal control
Collect (subset of) transaction data from information system
Identify deviations by comparing transactions against specific business rule
Potential violations of internal controls

Fig. 4. Internal control testing rationale, including process mining (Source: [17])

3.6 Communicating the Result

The advantages of visualizing processes and existing process deviations as well as conformance checks and identification of control weaknesses can also be used by the auditor as part of the audit report and the presentation of audit results. The visualization generally leads to the audited entity or report addressees being even more receptive to the potential findings or recommendations for improvement. Clearly, this area is of limited relevance compared to those mentioned previously, yet it provides an advantage that should not be overlooked.

3.7 Follow-up

During the follow-up of an audit, the auditor normally tries to check if the problems and negative findings of the initial audit were solved and whether the recommendations were implemented. Thus, the auditor can actually use all of the prior process mining analysis to double-check if the expected improvements were realized.

3.8 Maturity Levels

Although a match between the different internal audit phases and separate process mining activities is presented in the previous sections, sometimes an internal auditor only relies on process mining during the core of the audit (the phase 'Conducting the audit'), as visually presented in Fig. 5, or another single phase. After building the event log, process discovery can help the auditor to understand the existing process variants and identify potential risk areas. Furthermore, the check of the existing process structure compared to the process model allows the internal auditor to clarify existing deviations or identify additional risks. Checking the process executions against business rules offers an additional way to gather evidence of, for example internal control weaknesses or potential fraud cases. The variant analysis allows the auditor to compare different variants to identify additional risks, but also opportunities, since the as is-process might be a more efficient or effective way to organize the process instead of the intended process model. Finally, the case analysis allows a deep-dive of the auditor on the transaction level to analyze the identified cases. It goes without saying that integrating process mining throughout the entire internal audit is next-level in terms of maturity, compared to the integration in only one phase.

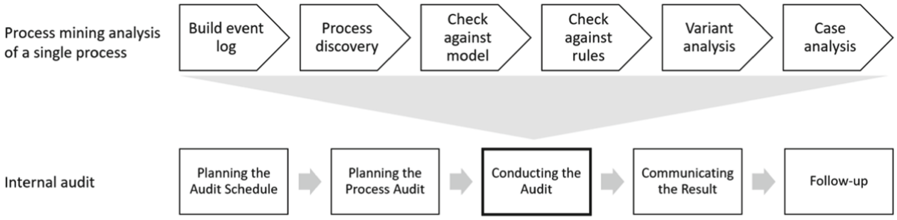


Fig. 5. Example of how process mining can be integrated in a single phase of the internal audit

4 The Symbiosis Between Internal and External Auditing When Using Process Mining

As mentioned before, the external and internal audit have partially overlapping goals. Although internal audits include the investigation of process improvement from an efficiency point of view, both audits have the goal to assure the validity of the reported financial statements. As a result, the audit phases of the external and internal audit are not that different in nature. However, differences exist; some in nature and some in terminology.

4.1 External Auditing and Process Mining

Where the internal audit starts with planning the audit schedule for a period of one or several years, the external audit does not have such a long-term setting. The external audit standards also describe a phase of planning the audit, but this is in the light of a running audit engagement and relates to the audit of that specific accounting year. In order to plan the audit approach, a risk assessment phase takes place. Based on the risk assessment, the auditor decides where to allocate most resources to. This is comparable with the internal audit phase 'planning the process audit'. Similarly, this phase would rely mostly on process discovery and a rather high-level conformance check against a procedural model (Fig. 6).

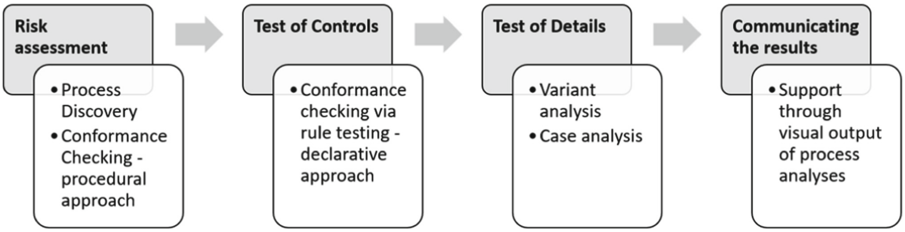

Risk assessment
• Process Discovery
• Conformance Checking - procedural approach
Test of Controls
• Conformance checking via rule testing - declarative approach
Test of Details
• Variant analysis
• Case analysis
Communicating the results
• Support through visual output of process analyses

Fig. 6. Integration of process mining activities in the external audit process

Based on the assessed risks, targeted business processes will be investigated through 'tests of controls' and 'tests of details'. The tests of controls are related to examining the design and implementation of the organisation's control environment. The tests of details are checks that take place at the transaction level. This distinction stems from before the digital era, where tests of controls were not executed at a detailed level. Nowadays, however, the distinction is less clear. For example, checking whether all documents in a system have been approved, is a test of details that also checks the effectiveness of a control. Given the lesser delineation between these two concepts, these tests are often intertwined and form, together with other analytical procedures[3], the core of the external audit. Similar to the internal audit phase of conducting the audit, these tests would heavily rely on checking rules, a more in-depth variant analysis and case analysis

[3] Analytical procedures in the context of an external audit are defined as "... evaluations of financial information through analysis of plausible relationships among both financial and non-financial data. Analytical procedures also encompass such investigation as is necessary of identified fluctuations or relationships that are inconsistent with other relevant information or that differ from expected values by a significant amount." [19].

[11,20]. The driver of the tests is currently stemming from the traditional audit. The check-lists that were used before, are now automated and extended with additional dimensions. Still, the core of the audit did not yet change drastically. This can be devoted to the standards that remain unchanged, creating some reluctance with the auditors to turn to new techniques.

After the tests of details and controls, the results are communicated. As with the internal audit, the visual aspect of process mining is an important characteristic. A graphical presentation of the phases of the internal and external audit, along with the process mining analysis phases that can be used, is given in Fig. 7.

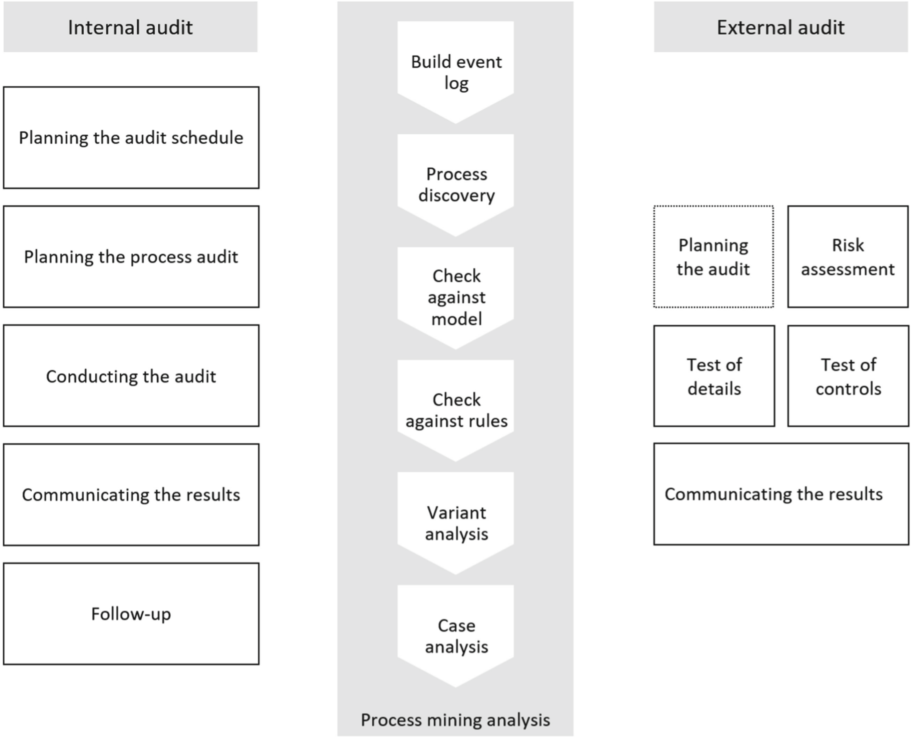


Fig. 7. Parallels between internal and external auditing and the process mining analysis phases that support the audits.

4.2 Relying on Internal Audit's Process Mining Efforts

If the external auditor can start from the process mining efforts of the internal auditor, obviously, more can be reached than if this is not the case. In this setting, the external auditor would have to examine the process that was followed by the internal auditor when conducting the process analysis. Following questions will be important to have a clear answer to (and hence clear documentation on):

- Which information is used to build the event log?
- How exactly is the event log built? (which process instance was selected, which activities were included, based on which tables and fields in the information system,...)
- Which filters were applied before starting the analyses?
- How is dealt with running cases?
- How are the analyses run? (in which tool, with which commands,...)
- How are the results written away? Is there an unmodified audit trail?
- ...

So although the external auditor does not have to start from scratch, a lot of effort will still be devoted to having assurance over the process of process mining. Only with a reasonable level of assurance of this internal process, the external auditor can rely on the outcome of the internal auditor.

The alternative is that the auditor takes full control of the process mining analysis. They can still start from the internal auditor's expertise, which will speed-up the process of building the event log for instance. But the external auditor would extract the data from the information systems himself, build the event log and run the analysis himself. This trade-off between control and depth of analysis is related to the personal preference of the external auditor.

5 Organizational Integration of Process Mining in the Auditing Function

Aside from the theoretical integration of process mining in the auditing process, organizational integration is equally important. There are different approaches how internal and external auditors can implement process mining in the auditing process. The following options present potential approaches and should be examined individually for the respective auditor. Of course, the best-fitting solution always depends on the financial, technical and human resources available. Also, the time factor for the implementation of process mining and the necessary training of the employees are not to be neglected.

5.1 Individual Process Mining Experts

Solutions with one or few process mining experts are conceivable, especially in smaller and medium-sized audit departments. The required profile is comparable to the members of a specialized team described above. Aside from profound process mining knowledge, the expert should have an excellent command of data analysis tools and, consequently, design and manipulate queries and data easily. This enables the expert to create the analyses necessary for successful use in the auditing process. As a direct contact person, the expert becomes a shorter link to the auditor than when working with process mining teams. On one hand, the advantage is the lower financial investment and the possibility to easily upscale the team. On the other hand, a challenge could be that the expert's capacity is too low when there are frequent requests.

5.2 Specialized Process Mining Team

It is a good idea to set up an independent process mining team, especially for sizeable internal audit functions or functions with solid data analysis activities. This team specializes in data analysis and process mining and prepares reports to support each audit. Consequently, this team belongs to the core of the audit function and is heavily involved in audit preparations. In such a team, especially auditors with profound ERP systems and process expertise should be involved. This know-how enables the team to develop and prepare target-oriented analyses of data and processes, such that the auditors outside the team reap significant benefits from this. Since the experts can reuse some procedures to prepare different dashboards and process analyses, learning effects can be assumed. While the high degree of specialization of the team brings numerous advantages, the disadvantages of this approach should not be neglected either. The team members are primarily "remote" active, potentially leading to isolation from the actual audit process. Such an approach is also associated with high personnel costs, so it must be examined per company to what extent this can be realistically implemented. If individual teams for data analysis already exist in the auditing function, this would, of course, be a sensible starting point for this approach.

5.3 Training of All Staff

When process mining becomes an integral part of the auditing function, comprehensive training of all auditors working with it is unavoidable. Depending on the selected process mining software, specific levels of training are required. The training should include the connection to the ERP system: understanding the ERP data structure is a prerequisite to building a suitable event log. The connection to the corresponding process specifications of the company is also of central importance against this background. What do the processes in our company look like? What controls have been implemented? Where are the critical points? All these questions need to be translated into clear questions to feed the process analysis of the auditor. These sample questions show how demanding training on this topic is. The high demands on training must also be understood against the background of the local or global orientation of the internal audit function of the respective company. If the auditing functions have several locations, possibly even in different nations, a corresponding global training concept to roll out to the different nations is recommended. Overall, training should help standardize procedures, develop appropriate competencies, and build a shared knowledge base across the audit function. Only with a sound thought-out training concept, the auditing department can successfully implement process mining in their processes.

5.4 Process Mining Competencies of Other Departments or Outsourcing

In addition to building up process mining competencies within the auditing team, the auditing function can also draw on support from outside. Numerous companies have now implemented process mining solutions in various areas of the company, which is why a cooperation with other functions in the context of process mining analyses is conceivable. This approach is particularly useful if the auditing department has not yet been able to think through the process mining approach or if initial trial analyses are to be carried out. The great advantage of this approach is that the auditing team has to expand relatively limited resources in order to fundamentally evaluate the possible applications. An important note is, however, that audit-specific aspects that should be included during the log building phase, might not be included. To mitigate this risk, it is important to team-up with a partner that has process mining expertise in an auditing context. Another important characteristic that holds for every outsourcing act, is that the auditor doesn't develop deep in-house expertise. Alternatively, an outsourcing is possible, in order to maintain the flexibility of the auditing function. However, outsourced analyses will probably produce comparable costs due to the enormous implementation effort later on.

6 From Data to Audit Evidence

When the auditor decides to apply a process mining approach, there are two preparatory steps that are key to success: identify the scope of your process under investigation and formulate (upfront!) the most important questions that need to be answered. The process can be either a core business process, like purchase or sale, or a supporting process like an incident management or change management process. It is paramount to identify which activities are subject (and which are not) to the audit. Aside from the process scope, the formulated key questions are of paramount importance. Based on these questions, the event log can be built. Although research on object-centric process mining is on the rise, in practice, process mining for audit is still document-centric: a case is either an order, or a journal posting, or an invoice, or another document of relevance.

Based on the audit questions, the relevant information can be extracted from the information system. Unfortunately, the selection of a case identifier potentially creates noise on the analyses to follow. When many-to-many relationships exist between a document at the beginning of the process and another document that is created near the end, choosing one or the other document always has its up- and downsides. In general, it seems that the auditor prefers to select a document that leads to a financial transaction and at the same time is earlier in the process [21]. If a choice needs to be made on whether to follow that document on its header level, or on a more detailed level, the auditor is attracted to examine the case at the lower level.

To select activities to include in the event log, most ERP systems give a plethora of options to enrich the log with. A lot of recorded events could be included on top of the most straight forward (process) activities. It is for example possible to include both, the moment an invoice was put in the system and the date that it was posted, where the former is cpu-timestamped and the latter timestamp is manually entered. These additional insights are still to be integrated in regular audits. To date, the auditor is not yet fully grasping these opportunities. In a future audit, perhaps when standards have been updated, these type of activities should find their way into typical standard analyses.

As the event log serves as 'audit evidence'[4] in the context of an external audit, it is tied to audit regulations. To assemble the audit evidence, and hence build the event log, the auditor is expected to *obtain sufficient appropriate audit evidence*. This leaves room for discussion on whether, according to this stipulation, an auditor can ask full access to a client's information system or not. If not, the auditor will need to submit a detailed data request. This is the standard procedure for regular audits where you can request 'information'. However, when you are interested in the data underneath the information, this is more difficult. Submitting a sufficient data request to build an event log later is only possible if the auditor is acquainted with the company's information system.

Once the auditor has access to information, the auditor has to decide on whether it is suited to use as audit evidence and base an opinion on this information. Particularly relevant to the case of process mining in auditing are the stipulations on *audit evidence that has been prepared using the work of a management's expert*. In such cases, the auditor is expected to "evaluate the competence, capabilities and objectivity of that expert; obtain an understanding of the work of that expert; and evaluate the appropriateness of that expert's work as audit evidence for the relevant assertion." [22]. *When using information produced by the entity*, the auditor is expected to have a view on the accuracy and completeness of the information, and whether the information is sufficiently precise and detailed [22]. Taking these regulations into account makes it clear that it is not straightforward for an external auditor to rely on other parties to conduct process mining analyses and build further on this.

7 Open Challenges

Although the preceding discussion demonstrates the numerous benefits of applying process mining within internal and external auditing, numerous fundamental challenges also exist. These challenges can be divided into four areas:

- Data quality
- Auditor skill-set
- Stakeholder
- Full-population testing

All four areas have varying degrees of relevance to their respective organizations and include numerous broader challenges and components.

[4] Audit evidence is defined in ISA 500 as "Information used by the auditor in arriving at the conclusions on which the auditor's opinion is based. Audit evidence includes both information contained in the accounting records underlying the financial statements and other information." [19].

Data Quality. With regard to data, the application of process mining within the audit is jeopardized if the available data is either not usable or irrelevant. Also, data integrity and compatibility play a key role against this backdrop. For example, having different source systems implies that the creation of a usable data model for process mining is not straightforward. Consequently, all known problems and challenges of data analyses and IT systems are also valid for process mining in the context of auditing.

Auditor Skill-Set. Both, internal and external auditors must have the required know-how and appropriate training for the use of process mining in the context of the audit. However, information systems, data analytics, and process mining are no standard topics in auditing education curricula. Although programs are increasingly including these topics in courses and providing good starting points, the acceleration must be sustained during career development. Therefore, it is equally important that audit companies (or departments) sharpen these skills with the new hires. Only then can this lead to a know-how flow through the entire audit firm.

Stakeholder. It is important for the auditor to clarify the compliance with auditing standards during the financial audit with process mining and get the commitment from the audit committee or the audited entity while using process mining. The regulator or the respective professional association must also support the use of new technologies for obtaining an audit result accordingly.

Full-Population Testing. Process mining often extends the traditional sampling approach, where a sample is taken from a population as audit evidence. Process mining, on the other hand, generally uses the entire population of transactions, so that the potential alpha and beta errors of a sample no longer exist. However, this consideration becomes difficult if, for example, several hundred or even thousands of deviations ('red flags') are identified during the audit. In such a case, it is necessary to decide how the auditor arrives at the intended reasonable assurance. Does the auditor draw a sample from the identified red flags, because of resource constraints? Or does the auditor use additional resources, because the risk of not evaluating all red flags impairs the audit judgment? In practice, the latter is often not feasible, pushing auditors sometimes back to traditional sampling approaches. This is of course not the way forward when new techniques are around to move closer to full-population testing. The answer should be sought in providing support to dealing with all these deviations (see Sect. 3.5).

In addition to these general areas, further challenges can be identified, such as the high entry barriers and costs for smaller audit functions or audit firms.

8 Conclusion and Outlook

The use of process mining in internal and external auditing offers a magnitude of potential benefits. Through the visualization and analysis of process steps, especially in combination with an in-depth data analysis, auditors can use numerous new ways and approaches to generate unique insights. For example, process mining can support compliance with (global) process governance and thereby improve the process landscape in national and international organizations. Moreover, by combining data from different areas of the company, completely new contexts can be mapped and a true added value for auditing can be created. Process mining can support all areas of auditing work. This includes the identification of risk areas or compliance violations, the audit of the internal control system, and the compliance with governance requirements.

Of course, process mining is a tool on the process level, which is why the link to real business processes and data is of decisive importance for a successful implementation in auditing. When reaching this process connection through the audit, process mining offers numerous approaches and applications for both, experienced and novice auditors. Depending on the focus and experience of the auditor, the field of application can be completely different. Successfully applying process mining in auditing is hitting the balanced combination of the right process mining techniques and skills with the right level of audit expertise. Only running an analysis, without the interpretation of a domain expert, is like any other data analysis meaningless. The added value is found in the powerful combination of techniques and domain expertise. This is where future investigations of this topic should focus at.

References

1. IAASB: International Standard on Auditing 200: Overall objectives of the independent auditor and the conduct of an audit in accordance with international standards on auditing (2009)
2. IAASB: International Standard on Auditing 700: Forming an opinion and reporting on financial statements (2016)
3. Carcello, J.V., Eulerich, M., Masli, A., Wood, D.A.: Are internal audits associated with reductions in perceived risk? Auditing: J. Pract. Theory **39**(3), 55–73 (2020)
4. The Institute of Internal Auditors (IIA): The Professional Practices Framework. The Institute of Internal Auditors, Altamonte Springs (2017)
5. Christ, M.H., Eulerich, M., Wood, D.A.: Internal Auditors' Response to Disruptive Innovation, 1st edn. Internal Audit Foundation, Lake Mary, Florida (2019)
6. Eulerich, M., Eulerich, A.: What is the value of internal auditing? - A literature review on qualitative and quantitative perspectives. Maandblad Voor Accountancy en Bedrijfseconomie **94**, 83–92 (2020)
7. Aalst, W.: Foundations of process discovery. In: Aalst, W., Carmona, J., (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 37–75. Springer, Cham (2022)
8. Augusto, A., Carmona, J., Verbeek, E.: Advanced process discovery techniques. In: Aalst, W., Carmona, J., (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 76–107. Springer, Cham (2022)
9. Aalst, W.: Process mining: a 360 degrees overview. In: Aalst, W., Carmona, J., (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 3–34. Springer, Cham (2022)

10. Swennen, M., Janssenswillen, G., Jans, M., Depaire, B., Vanhoof, K.: Capturing process behavior with log-based process metrics. In Ceravolo, P., Rinderle-Ma, S. (eds.) SIMPDA. CEUR Workshop Proceedings, vol. 1527, pp. 141–144. CEUR-WS.org (2015)
11. Jans, M., Alles, M.G., Vasarhelyi, M.A.: A field study on the use of process mining of event logs as an analytical procedure in auditing. Account. Rev. **89**(5), 1751–1773 (2014)
12. Folino, F., Greco, G., Guzzo, W., Pontieri, L.: Discovering multi-perspective process models: the case of loosely-structured processes. In: Filipe J., Cordeiro J. (eds.) Enterprise Information Systems. ICEIS 2008 (2009)
13. Rozinat, A., Van der Aalst, W.M.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)
14. Carmona, J., Dongen, B., Weidlich, M.: Conformance checking: foundations, milestones and challenges. In: Aalst, W., Carmona, J., (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 155–190. Springer, Cham (2022)
15. Baader, G., Krcmar, H.: Reducing false positives in fraud detection: combining the red flag approach with process mining. Int. J. Account. Inf. Syst. **31**, 1–16 (2018)
16. Chiu, T., Jans, M.: Process mining of event logs: a case study evaluating internal control effectiveness. Account. Horiz. **33**(3), 141–156 (2019)
17. Jans, M., Hosseinpour, M.: How active learning and process mining can act as continuous auditing catalyst. Int. J. Account. Inf. Syst. **32**, 44–58 (2019)
18. Laghmouch, M., Jans, M., Depaire, B.: Classifying process deviations with weak supervision. In: 2020 2nd International Conference on Process Mining (ICPM), pp. 89–96. IEEE (2020)
19. IAASB: International Standard on Auditing 520: Analytical procedures (2009)
20. Jans, M., Alles, M., Vasarhelyi, M.: The case for process mining in auditing: sources of value added and areas of application. Int. J. Account. Inf. Syst. **14**(1), 1–20 (2013)
21. Jans, M.: Auditor choices during event log building for process mining. J. Emerg. Technol. Account. **16**(2), 59–67 (2019)
22. IAASB: International Standard on Auditing 500: Audit evidence (2009)

Robotic Process Mining

Marlon Dumas[1,3](✉), Marcello La Rosa[2,3], Volodymyr Leno[1,2,3], Artem Polyvyanyy[2], and Fabrizio Maria Maggi[4]

[1] University of Tartu, Tartu, Estonia
marlon.dumas@ut.ee
[2] University of Melbourne, Melbourne, Australia
{marcello.larosa,artem.polyvyanyy}@unimelb.edu.au
[3] Apromore, Melbourne, Australia
volodymyr.leno@apromore.com
[4] University of Bozen-Bolzano, Bolzano, Italy
maggi@inf.unibz.it

Abstract. User interaction logs allow us to analyze the execution of tasks in a business process at a finer level of granularity than event logs extracted from enterprise systems. The fine-grained nature of user interaction logs open up a number of use cases. For example, by analyzing such logs, we can identify best practices for executing a given task in a process, or we can elicit differences in performance between workers or between teams. Furthermore, user interaction logs allow us to discover repetitive and automatable routines that occur during the execution of one or more tasks in a process. Along this line, this chapter introduces a family of techniques, called Robotic Process Mining (RPM), which allow us to discover repetitive routines that can be automated using robotic process automation technology. The chapter presents a structured landscape of concepts and techniques for RPM, including techniques for user interaction log preprocessing, techniques for discovering frequent routines, notions of routine automatability, as well as techniques for synthesizing executable routine specifications for robotic process automation.

1 Introduction

The rigidity and complexity of legacy applications, particularly in large organizations, engender situations in which workers are required to perform repetitive routines to transfer data from one application to another via their user interfaces. Examples of such repetitive routines include:

- Downloading and opening an Excel workbook attached to an inbound email (e.g. a list of academic credentials of a prospective student) and copying data records from one of the sheets in this workbook (e.g. the credential details of the student) into a student admission system accessed via a web browser.
- Accesing a legacy ERP system to retrieve one or more purchase orders of a given customer, copying data from each of these purchase orders into a consolidated sheet, and sending the resulting spreadsheet to a customer by email.

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 468–491, 2022.
https://doi.org/10.1007/978-3-031-08848-3_16

The automation of such routines can eliminate tedious and demotivating manual work, reduce cycle times, and enhance data quality. Advances in Robotic Process Automation (RPA) technology [1,41] make it possible to automate routines like the above ones. However, building and maintaining RPA bots requires a significant investment and hence, it is important for organizations to make the right decisions as to which bots they should build. In a typical organization, there may be tens of thousands of types of tasks, and any of them may involve one or more repetitive routines. Some routines are sufficiently frequent and widespread across the organization that they can be identified and scoped via interviews, focus groups, and workshops with workers. Other routines, however, may be less widespread or performed sporadically, but still sufficiently often that it is beneficial to automate them.

Robotic Process Mining (RPM) is a family of techniques to discover repetitive routines that can be automated using RPA technology, by analyzing interactions between one or more workers and one or more software applications, during the performance of one or more tasks in a business process. In general, RPM techniques take as input User Interaction logs (*UI logs*).[1] These UI logs are recorded while workers interact with one or more applications, typically desktop applications. Based on these logs, RPM techniques produce specifications of one or more routines that can be automated using RPA or related tools.

Depending on the type of technique, the discovered routine specifications may be conceptual (i.e. non-executable) or *executable*. A conceptual routine specification provides guidance to analysts and developers to help them scope a repetitive routine and to build an executable script to fully or partially automate the routine. For example, a non-executable specification of a routine could take the form of a textual description (in natural language), or a sequence of screenshots corresponding to repetitive sequences of interactions, or a sequence of user interactions (e.g. ["open sheet","select cell", "edit cell", "copy cell contents", ...]). An executable routine specification is a specification that contains all the information required to fully reproduce the routine via a dedicated execution engine or to synthesize a script that can be executed using an RPA tool or a similar type of automation tool.

This chapter reviews the state of the art in the field of RPM and provides a structured overview of the steps of a typical RPM pipeline, the techniques that may be employed in each of these steps, as well as open research challenges on the way to realizing mature RPM tool sets.

The chapter is partially based on a previous journal article [32]. The chapter extends this journal article by positioning the vision of RPM within the broader context of task mining and process mining, and by providing an updated review of related work in the field.

The rest of the chapter is structured as follows. Section 2 gives an overview of techniques related to robotic process mining, including task mining and process mining, and gives an overview of existing work on identification of task automa-

[1] In this chapter, we use the acronym *UI* to refer to a *user interaction*, not to be confused with a *user interface* which is another common use of this acronym.

tion opportunities. Section 3 presents a framework for robotic process mining and introduces techniques covering each component of the framework. Finally, Sect. 4 discusses open challenges in the field of robotic process mining.

2 Background

2.1 Robotic Process Automation

RPA is a class of tools to automatically execute sequences of steps (herein called *routines*) involving interactions between a user and a software application, or interactions between multiple applications via Application Programming Interfaces (APIs). In an RPA tool, the execution of a routine is driven by a pre-specified script, which consists of atomic steps corresponding to individual interactions, assembled together via control-flow structures (if-then-else statements, repeat-until loops, etc.) [43]. A common characteristic of RPA tools is that they are able to "operate on the user interfaces of computer systems in the way a human would do" [1]. For example, an RPA tool may perform clicks or keystrokes on the user interface of a desktop application to mimic a sequence of steps that would normally be performed by a human operator. Examples of RPA tools, as of the time of writing this chapter, include Automation Anywhere RPA Workspace[2], Blue Prism Intelligent Automation Platform[3], Microsoft Power Automate Desktop[4], RocketBot[5], and UiPath Platform.[6]

Typically, RPA tools include a design environment, where different types of users, ranging from software developers to business users, may specify and test scripts to automate one or more routines. Each such script is then embedded into a so-called *software bot*. A bot is a unit of execution in an RPA tool. A bot is responsible for executing a given script whenever a given type of trigger occurs. Bots are operated via so-called control dashboards, which allow human operators to oversee the work performed by a collection of bots.

Depending on how the control dashboard is used, we can distinguish two RPA use cases: *attended* and *unattended* [43]. In attended use cases, the bot is triggered by a user. During its execution, an attended bot may provide data to a user and take in data from a user. In these use cases, the user may run the bot's script step-by-step, pause or stop the bot, or otherwise intervene during the script's execution. Attended bots are suitable for routines where dynamic inputs are required (i.e. inputs gathered during a routine execution), where some decisions or checks require human judgment, or when the routine is likely to have unforeseen exceptions. For example, entering data from an invoice in a spreadsheet format into a financial system is an example of a routine suitable for attended RPA, given that in this setting, some types of errors may have financial

[2] https://www.automationanywhere.com/.

[3] https://www.automationanywhere.com/.

[4] https://powerautomate.microsoft.com/.

[5] https://www.rocketbot.com/.

[6] https://www.uipath.com/.

consequences. Unattended RPA bots, on the other hand, execute scripts without human involvement and do not take inputs during their execution. Unattended RPA bots are suitable for executing deterministic routines where all execution paths (including exceptions) are well understood and can be codified. Copying records from one system into another via their user interfaces through a series of copy-paste operations is an example of a routine that an unattended bot could execute. In this chapter, we focus on unattended RPA bots.

Figure 1 presents a simple lifecycle model of RPA bots, which we use below to position the role of robotic process mining.[7] According to this lifecycle model, an RPA bot goes through four phases:

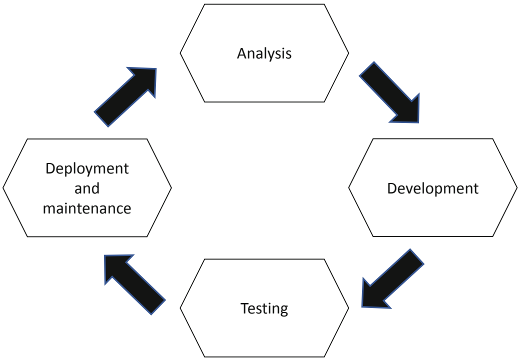


Fig. 1. Simple RPA bot lifecycle [23]

- ***Analysis.*** In this phase, analysts identify candidate routines for automation, examine the current ways of their execution (e.g. by constructing the *as-is* process model), assess the costs and benefits of their automation as well as the related risks, and analyze whether the identified routines can be automated without being redesigned.
- ***Development.*** In this phase, the routines identified earlier are automated. This involves constructing a process model representing the desired execution of the routines to be automated (i.e. the *to-be* process model). Then RPA developers implement the routine using a specialized development environment by creating an executable software script, a.k.a. RPA bot. Depending on the complexity of the task to be automated, this requires a different amount of coding. Large enterprise RPA tools such as UiPath or Automation Anywhere allow for the creation of the scripts by dragging and dropping the required functions (e.g. open a file, copy a cell). Since this step requires a large amount of manual, error-prone work, a code review and script evaluation are required.

[7] For the sake of conciseness, the RPA bot lifecycle model discussed here consists of four coarse-grained phases. A finer-grained RPA bot lifecycle can be found, for example, in [16].

- ***Testing.*** In this phase, the implemented bot undergoes testing in a pre-production environment. It is evaluated in the different scenarios to examine whether it works as intended and how it handles exceptions. If the tests are successful, the bot proceeds to the deployment phase. If the tests fail, it is sent back to the developers to identify and fix the identified issues.
- ***Deployment and maintenance.*** After successful testing, the bot is deployed in the production environment and is ready to be used via a control dashboard. As the bot performs its work, certain issues may arise. In this case, the bot may be sent back to the testing or development phases.

In this chapter, we focus on techniques that leverage UI logs to support the analysis and development phases of RPA bots.

2.2 Task Mining

Task mining is a collection of techniques for analyzing the execution of tasks performed by human workers, based on records of interactions between these workers and one or more software applications. Depending on the goal of the analysis, we can distinguish between three use cases of task mining [26]: (i) task discovery and optimization; (ii) resource and workforce optimization; and (iii) task automation.

Task Discovery and Optimization. In this use case, the goal is to discover how a task is performed by one or more workers, to identify deviations with respect to policies or work instructions related to that task, and/or to uncover ways of improving the performance of the task. By applying task mining techniques to a task, we may discover that different workers perform the task in different ways. For example, one worker might open all the desktop windows required to perform a task upfront (e.g. an email client, a spreadsheet application, and a browser window connected to a CRM system), and only once all windows are open, they start navigating across these windows to complete the task. Another worker might start performing the task in one desktop window (e.g. the email client's window) and then open the other windows incrementally. Similarly, one worker might usually execute a task in a single go, without interruptions, while another might interleave the execution of the task with other work, or might multitask.

Having identified how a task is performed by one or more resources, task mining can help us to identify steps in a task that are responsible for delays (bottlenecks), as well as common rework loops or workarounds with respect to normative work instructions. Task mining also allows us to relate the sequences of steps that different workers perform with performance measures, such as the mean cycle time of a task or the defect rate of a task. For example, task mining may help us to identify that when a given step, such as clicking on a given cell number in a sheet, is repeated multiple times, the mean cycle time of a task is significantly higher than when this cell is visited only once.

Resource and Workforce Optimization. In this use case, the goal is to identify inefficiencies in the way tasks are assigned to resources, or conversely, to uncover ways to improve the assignment of tasks. For example, by analyzing UI logs, we may find that when an invoice entry task relates to an invoice from a company in country X, it takes more time for worker A to perform the task (rather than another worker B) whereas the opposite holds for invoices coming from country Y. We might also find that when worker A performs an invoice data entry task after 4:30pm, the task gets completed faster, but when this happens, some fields in the invoice are left unfilled, which might then be causing issues downstream.

Task Automation and Robotic Process Mining. In this use case, the goal is to discover opportunities to automate a task or part of a task. The automation of a task can be achieved using a variety of technologies. For example, if a task involves information flows between multiple applications, one could use middleware technology to programmatically connect these applications, thus replacing the manual information flow with an automated (programmatic) flow. Another approach is to develop and RPA bot to transfer data from one application to another by replicating the user interactions that a human worker would do to achieve this. Robotic Process Mining (RPM) refers to the use case of task mining where UI logs are analyzed in order to identify frequent routines that can be automated by means of one or more RPA bots. The rest of this chapter focuses on this latter use case of task mining.

2.3 Relations Between Task Mining and Process Mining

Task mining is in many ways related to process mining, particularly to techniques for automated process discovery (cf. Sects. 2 and 3). However, task mining and process mining differ in several respects. These differences stem from the differences in the inputs of these techniques. Process mining take as input event logs extracted from enterprise systems that support the execution of one or more business processes in an organization – e.g. Enterprise Resource Planning (ERP) or Customer Relationship Management (CRM) systems, as discussed in [2]. Meanwhile, task mining techniques take as input UI logs, consisting of records of micro-steps performed by workers while they interact with software applications to perform individual tasks in a process. Both types of logs consist of timestamped records, such that each record refers to the execution of an action (or task) by a user. Also, each record may contain a payload consisting of one or more attribute-value pairs. However, UI logs and event logs differ in at least four ways.

First, event logs intended for process mining consist of events at a finer level of granularity than UI logs. An event in an event log typically refers to the start, completion or other significant state change in the execution of a task within a business process, such as *Check purchase order* or *Transfer student records*. Such tasks can be seen as a composition of lower-level (micro-)steps, which may be recorded in an UI log. For example, task *Transfer student records* may involve multiple actions to copy the records associated with a student (name, surname, address, course details) from one application to another. In other words, an UI log may contain dozens or even hundreds of entries per task execution, whereas an event log would typically only contain one or a handful of entries per task execution. Also, the payload of the events in an event log may contain low-level information such as the specific cell or the pixel coordinates involved in a user interaction, or it may be associated to a screenshot taken during a user interaction. In contrast, event logs contain business-relevant attributes, such as the amount of a loan offer, the interest rate, the repayment term, etc.

Second, UI logs do not come with a notion of *case identifier* (or process instance identifier), whereas event logs typically do. In other words, events in an UI log are not explicitly correlated. A typical UI log consists of thousands of user interactions recorded during a period of several hours on the workstation(s) of one or more workers. Prior to being used, such UI logs needs to be segmented into logical units corresponding to task executions, as discussed later in this chapter.

Third, a record in an event log often does not contain all input or output data used or produced during the execution of the corresponding task. For example, a record in an event log corresponding to an execution of task *Transfer student records*, is likely not to contain all attributes of the corresponding student (e.g. address). Meanwhile, an UI log typically collects all the data observed during the execution of a task, particularly when the UI log is intended to be used for RPM purposes. Indeed, if some input or output attributes are missing in the UI log, the resulting routine specification would be incomplete, and hence the resulting RPA bot would not perform the routine correctly.

A fourth difference is that event logs are typically obtained as a by-product of transactions executed in an information system, rather than being explicitly recorded for analysis purposes. The latter characteristic entails that event logs are more likely to suffer from incompleteness, including missing attributes as discussed above, but also missing events. For example, in a patient treatment process in a hospital, it may be that the actual arrival of the patient to the emergency room is not recorded when a patient arrives by themselves, but it is recorded when a patient arrives via an ambulance. In other words, the presence or absence of an event in an event log depends on whether or not the information system is designed to record it, and whether or not the workers actually record it. Meanwhile, an UI log is recorded specifically for analysis purposes, which allows all relevant events to be collected subject to the capabilities of the UI recording tool.

The above differences in the input entail that it is often not possible nor desirable to use the same techniques for process mining as for task mining. In the field of process mining, a typical visualization consists of a graph with one node per activity. The emphasis of these techniques is to show the most frequent control-flow dependencies between the activities of the process. This approach is not feasible in the context of task mining because the steps are fine-grained and therefore too numerous to be displayed in their entirety. Besides, only certain steps are relevant for a given use-case, specifically those that are part of a frequent routine. Accordingly, a task mining technique typically starts by pre-processing the UI log in order to extract only the most frequent sequences of steps (i.e. the most frequent routines) using sequence pattern mining techniques, or using event abstraction techniques such as those developed in the field of process mining [44].

Notwithstanding these differences, several commercial process mining vendors, such as Apromore[8], Celonis[9], and Minit[10], take advantage of the commonalities between UI logs and business process event logs to offer task mining features. Typically, these tools discover directly-follows graphs (cf. [3]) from UI logs or from combinations of event logs and UI logs. For example, these tools may discover directly-follows graphs to visualize the sequences of screens visited by a user during the performance of one or more tasks, or to visualize the most frequent or the slowest steps during the performance of a task.

These visualizations are suitable when analyzing tasks for the purpose of task optimization and workflow optimization (cf. the first two use-cases above). They can also help users to visually detect candidate routines for automation, when those routines have a simple structure (e.g. perfect sequences of steps). However, beyond simple scenarios, these visualizations do not allow users to determine if a given task contains routines that can be automated by means of an RPA bot. In this respect, RPM techniques complement task mining techniques by explicitly addressing the questions of: (1) how to identify candidate routines for automation? and (2) how to derive an executable specification of a routine that has been identified as a candidate for automation?

3 Robotic Process Mining: A Framework

RPA tools are able to automate a wide range of routines, raising the question *how to identify routines in an organization that may be beneficially automated using RPA?* [41] To address this question, we envision a new class of tools, namely Robotic Process Mining (RPM) tools.

We define RPM as *a class of techniques and tools to analyze data collected during the execution of user-driven tasks to support identifying and assessing candidate routines for automation and discovering routine specifications that RPA bots can execute.* In this context, a *user-driven task* is a task that involves interactions between a user (e.g. a worker in a business process) and one or more software applications.

Accordingly, the primary source of data for RPM tools consists of user interaction (UI) logs. RPM aims at assisting the analysts in drawing a systematic inventory of candidate routines for automation and help them to produce executable specifications that can be used as a starting point for their automation.

[8] https://apromore.com.

[9] https://celonis.com.

[10] https://minit.io.

3.1 UI Logs and Routines

Figure 2 presents a class diagram capturing the core concepts and RPM and their relations. In this class diagram, the two main concepts are User Interaction log (*UI log*) and Routine. UI logs are the input of RPM, while routines (represented as routine specifications or as RPA scripts) are the output of RPM.

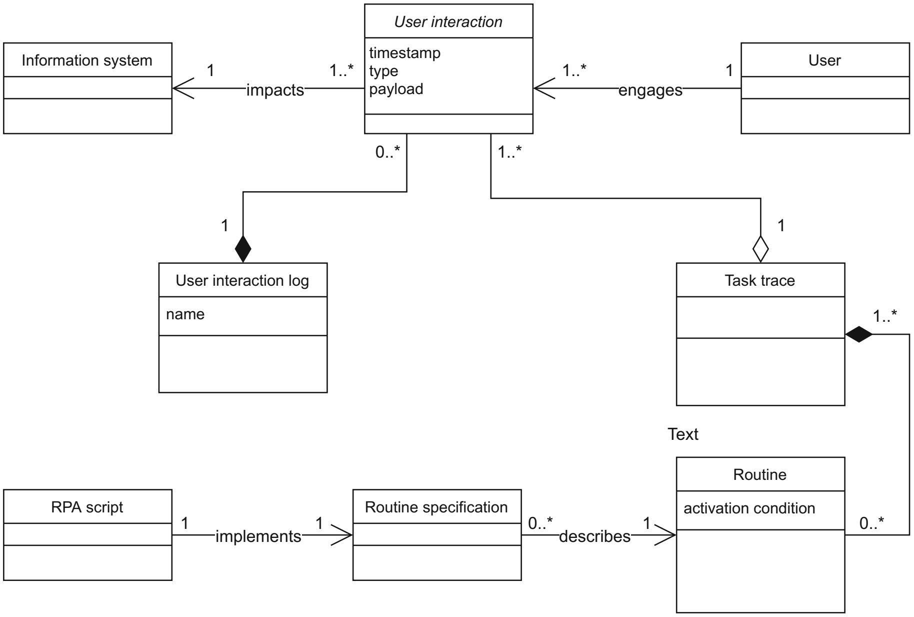


Fig. 2. Class diagram of RPM concepts

An UI log is a chronologically ordered sequence of user interactions, or UIs in short, performed by a single user in a single workstation and involving interactions across one or more applications (including web and desktop applications). An example of an UI log, which we use herein as a running example, is given in Table 1.

Each row in this example corresponds to one UI (e.g. clicking a button or copying the content of a cell). Each UI is characterized by a *timestamp*, a *type*, and a set of *parameters*, or *payload* (e.g. application, button's label or value of a field). To be useful in the context of RPA, the payload should contain sufficient information for a software bot to reproduce the performed activity. For example, for a UI that refers to clicking a button, it is important to store a unique identifier of this button (e.g. either the element identifier, or its name if this is unique in the page). Likewise, for an event that refers to editing a field, an identifier of the field as well as a new value assigned to that field are required attributes. The payload of a UI is not standardized and depends on the UI type and application.

Table 1. Fragment of a user interaction log

Row	UI Timestamp	UI Type	Payload					
			P_1	P_2	P_3	P_4	P_5	P_6
1	2019-03-03T19:02:23	Navigate to (web)	https://www.unimelb.au	204	Google search	–	–	–
2	2019-03-03T19:02:26	Click button (web)	https://www.unimelb.au	New record	newRecord	Button	–	–
3	2019-03-03T19:02:28	Select cell (Excel)	StudentRecords	Sheet1	A	2	“John”	–
4	2019-03-03T19:02:31	Select field (web)	https://www.unimelb.au	First name	First	Input	“”	–
5	2019-03-03T19:02:37	Edit field (web)	https://www.unimelb.au	First name	First	Input	“John”	–
6	2019-03-03T19:03:56	Create new tab (web)	https://chrome/new-tab/	219	New tab	–	–	–
7	2019-03-03T19:03:56	Select tab (web)	https://chrome/new-tab/	219	New tab	–	–	–
8	2019-03-03T19:04:05	Navigate to (web)	https://www.facebook.com	219	New tab	–	–	–
9	2019-03-03T19:07:50	Select tab (web)	https://www.unimelb.au	204	New record	–	–	–
10	2019-03-03T19:08:02	Select field (web)	https://www.unimelb.au	Last name	Last	Input	“”	–
11	2019-03-03T19:08:05	Edit field (web)	https://www.unimelb.au	Last name	Last	Input	“Do3”	–
12	2019-03-03T19:08:08	Select field (web)	https://www.unimelb.au	Last name	Last	Input	“Do3”	–
13	2019-03-03T19:08:12	Edit field (web)	https://www.unimelb.au	Last name	Last	Input	“Doe”	–
14	2019-03-03T19:08:16	Select field (web)	https://www.unimelb.au	Birth date	Date	Input	“”	–
15	2019-03-03T19:08:20	Edit field (web)	https://www.unimelb.au	Birth date	Date	Input	“18-11-1992”	–
16	2019-03-03T19:08:24	Select field (web)	https://www.unimelb.au	Country of residence	Country	Input	“”	–
17	2019-03-03T19:08:27	Edit field (web)	https://www.unimelb.au	Country of residence	Country	Input	“Australia”	–
18	2019-03-03T19:08:31	Click button (web)	https://www.unimelb.au	Submit	Submit	Submit	–	–
19	2019-03-03T19:08:35	Click button (web)	https://www.unimelb.au	New record	newRecord	Button	–	–
20	2019-03-03T19:08:38	Select cell (Excel)	StudentRecords	Sheet1	A	3	“Albert”	–
21	2019-03-03T19:08:40	Copy cell (Excel)	StudentRecords	Sheet1	A	3	“Albert”	“Albert”
22	2019-03-03T19:08:42	Select field (web)	https://www.unimelb.au	First name	First	Input	“”	–
23	2019-03-03T19:08:43	Paste (web)	https://www.unimelb.au	First name	First	Input	“”	“Albert”
24	2019-03-03T19:08:44	Edit field (web)	https://www.unimelb.au	First name	First	Input	“Albert”	–
25	2019-03-03T19:08:47	Select cell (Excel)	StudentRecords	Sheet1	B	3	“Rauf”	–
26	2019-03-03T19:08:49	Copy cell (Excel)	StudentRecords	Sheet1	B	3	“Rauf”	“Rauf”
27	2019-03-03T19:08:52	Select field (web)	https://www.unimelb.au	Last name	Last	Input	“”	–
28	2019-03-03T19:08:53	Paste (web)	https://www.unimelb.au	Last name	Last	Input	“”	“Rauf”
29	2019-03-03T19:08:54	Edit field (web)	https://www.unimelb.au	Last name	Last	Input	“Rauf”	–
30	2019-03-03T19:08:59	Select cell (Excel)	StudentRecords	Sheet1	C	3	“08/09/1989”	–
31	2019-03-03T19:09:02	Copy cell (Excel)	StudentRecords	Sheet1	C	3	“08/09/1989”	“08/09/1989”
32	2019-03-03T19:09:07	Select field (web)	https://www.unimelb.au	Birth date	Date	Input	“”	–
33	2019-03-03T19:09:10	Paste (web)	https://www.unimelb.au	Birth date	Date	Input	“”	“08/09/1989”
34	2019-03-03T19:09:12	Edit field (web)	https://www.unimelb.au	Birth date	Date	Input	“08-09-1989”	–
35	2019-03-03T19:09:17	Select cell (Excel)	StudentReords	Sheet1	D	3	“Germany”	–
36	2019-03-03T19:09:21	Copy cell (Excel)	StudentRecords	Sheet1	D	3	“Germany”	“Germany”
37	2019-03-03T19:09:26	Select field (web)	https://www.unimelb.au	Country of residence	country	Input	“”	–
38	2019-03-03T19:09:32	Paste (web)	https://www.unimelb.au	Country of residence	country	Input	“”	“Germany”
39	2019-03-03T19:09:35	Edit field (web)	https://www.unimelb.au	Country of residence	country	Input	“Germany”	–
40	2019-03-03T19:09:48	Edit field (web)	https://www.unimelb.au	International Student	international	checkbox	TRUE	–
41	2019-03-03T19:09:54	Click button (web)	https://www.unimelb.au	Submit	submit	submit	–	–
...	...	...	...	...	...	...		

Consequently, the UIs recorded in the same log may have different payloads. For example, the payload of UIs performed within a spreadsheet contains information regarding the spreadsheet name and the location of the target cell (e.g. the cell's row and column). In contrast, the payload of the UIs performed in a web browser contains information regarding the webpage URL, the name and identifier of the UI's target HTML element, and its value (if any).

An UI log consists of interactions of different types. To illustrate the types of interactions that may be exploited in the context of robotic process mining, Table 2 provides the concrete list of UI types (and associated parameters) supported by the Action Logger tool [33]. Action Logger is an open-source UI recording tool designed to record events generated by browsers and desktop applications, in a way that enables the discovery of automatable routines.

Note that in Table 2, the UI types are grouped into three groups: navigation, read, and write UIs. Navigation UIs correspond to actions that affect the state of the user interface, but without reading or writing any data. This includes, for example, moving from one tab to another in a broader, or selecting a cell in an Excel spreadsheet. Read actions are those where some data item is accessed, for example in order to copy it into the clipboard. Meantime, "write" actions are those where data is written into an element of the UI, for example, pasting the contents of the clipboard into the currently selected cell of an Excel spreadsheet.

Table 2. User interaction types and their parameters

UI	UI	Parameter names					
group	type	P1	P2	P3	P4	P5	P6
Navigate	Create new tab (web)	URL	ID	Title			
	Select tab (web)	URL	ID	Title			
	Close tab (web)	URL	ID	Title			
	Navigate To (web)	URL	Tab ID	Tab title			
	Add worksheet (Excel)	Workbook	Worksheet				
	Select worksheet (Excel)	Workbook	Worksheet				
	Select cell (Excel)	Workbook	Worksheet	Cell column	Cell row	Value	
	Select range (Excel)	Workbook	Worksheet	Range columns	Range rows	Value	
	Select field (web)	URL	Name	ID	Type	Value	
Read	Copy (web)	URL	Name	ID	Value	Copied content	
	Copy cell (Excel)	Workbook	Worksheet	Cell column	Cell row	Value	Copied content
	Copy range (Excel)	Workbook	Worksheet	Range columns	Range rows	Value	Copied content
Write	Paste into cell (Excel)	Workbook	Worksheet	Cell column	Cell row	Value	Pasted content
	Paste into range (Excel)	Workbook	Worksheet	Range columns	Range rows	Value	Pasted content
	Paste (web)	URL	Name	ID	Value	Pasted content	
	Click button (web)	URL	Name	ID	Type		
	Click link (web)	URL	Inner text	Href			
	Edit field (web)	URL	Name	ID	Type	Value	
	Edit cell (Excel)	Workbook	Worksheet	Cell column	Cell row	Value	
	Edit range (Excel)	Workbook	Worksheet	Range columns	Range rows	Value	

To obtain an UI log suitable for RPM, all UIs related to a particular task have to be recorded. This recording procedure can be long-running, covering a session of several hours of work if the user performs multiple instances of this task one after the other. During such a session, a worker is expected to perform a number of tasks of the same or different types. The UI log shown in the example above describes the execution of a task corresponding to transferring student data from a spreadsheet into the web form of a study information system.

The web form requires information such as the student's first name, last name, date of birth, and country of residence. If the country of residence is not Australia, the worker needs to perform one more step, indicating that the student will be registered as an international student.

Each execution of a task (herein also called a *task instance*) is represented by a *task trace*. In our running example, there are two traces belonging to a "new record creation" task. From the log, we can see that the worker performed this task in two different ways. In the first case, she manually filled in the form (UIs 1 to 18), while in the second case, she copied the data from a worksheet and pasted it into the corresponding fields (UIs 19 to 41).

Given a collection of task traces, the goal of RPM is to identify a repetitive sequence of UIs that can be observed in multiple task traces, herein called a *routine*, and to identify routines amenable for automation. For each such routine, RPM then aims at discovering an executable specification (herein called a *routine specification*). This routine specification may be initially captured in a platform-independent manner and then compiled into a platform-dependent *RPA script* to be executed in a given RPA tool.

3.2 RPM Phases

We distinguish three main phases in RPM: (1) collecting and pre-processing UI logs corresponding to the executions of one or more tasks; (2) discovering candidate routines for RPA; and (3) discovering executable RPA routines.[11]

Collecting and Pre-processing UI Logs. We decompose the first phase into the recording step itself and two preprocessing steps, namely the segmentation of the log into task traces and the simplification of the resulting task traces. We map the second phase into a single step. Then, we decompose the third phase into three steps: the discovery of platform-independent routine specifications, the aggregation of routines with the same effects, and the compilation of the discovered specifications into platform-specific executable scripts. This decomposition of the three phases into steps is summarized in the RPM pipeline depicted in Fig. 3. Below we discuss each step of this pipeline.

[11] Once an RPA routine has been automated via an RPA bot, a fourth phase is to monitor this bot to detect anomalies or performance degradation events that may signal that the bot may need to be adjusted and re-implemented or retired. While relevant from a practical perspective, this phase is orthogonal to the three previous phases since it is relevant both for bots developed manually and bots developed using RPM techniques. Furthermore, previous work has shown that existing process mining tools are suitable for analyzing logs produced by RPA bots for monitoring purposes [20].

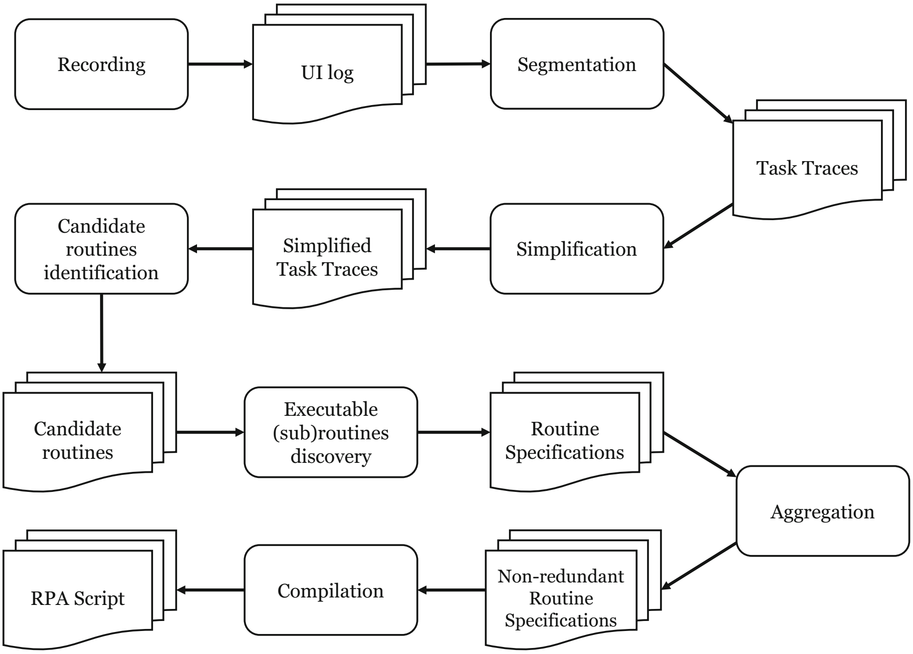


Fig. 3. RPM pipeline

The recording of an UI log involves capturing low-level UIs, such as selecting a field in a form, editing a field, opening a desktop application, or opening a web page. UI log recording may be achieved by instrumenting the software applications (including web browsers) used by the workers via plug-in or extension mechanisms. Logs collected by such plug-ins or extensions may be merged to produce a raw UI log corresponding to the execution of one or more tasks by a user during a period of time. This raw log usually needs to be preprocessed to be suitable for RPM.

The main challenge in this step is to identify what UIs must be recorded. The same UI (e.g. mouse click) can either be important or irrelevant in a given context. For example, a mouse click on a button is an important UI, but a mouse click on a web page's background is an irrelevant UI. Also, when a worker selects a web form, we need to record UIs at the level of the web page (the Document Object Model – DOM) in order to learn routines at the level of logical input elements (e.g. fields) and not at the level of pixel coordinates, which are dependent on screen resolution and window sizes. Existing UIs recording tools, such as JitBit Macro Recorder[12], TinyTask[13], and WinParrot[14], save all the UIs performed by the user at a too low level of granularity, with reference to

[12] https://www.jitbit.com/macro-recorder/.
[13] https://www.tinytask.net/.
[14] http://www.winparrot.com/.

pixel coordinates (e.g. click the mouse at coordinates 748,365). As a result, the UI logs generated by these tools are not suitable for extracting useful routines. The RPA tools mentioned in Sect. 2.1 (e.g. UiPath and Automation Anywhere) provide recording functionality. However, this functionality is intended to record RPA scripts. These tools do not capture details about different fields' values, as these values are not relevant for RPA script generation. For example, an RPA script must know which cell in a spreadsheet has to be copied, and it is agnostic to the value stored in that cell. Hence, a new family of recording tools is needed to record UI logs required for RPM.

In [33], we introduced a tool to record UI logs in a format that is suitable for RPM. The tool records not only the UI actions (selecting a field, editing a field, copying into or pasting from the clipboard) but also the values associated with these actions (e.g. the value of a field after an editing event). The tool supports MS Excel and Google Chrome. The tool also simplifies the recorded UI logs by removing redundant events (e.g. double-copying without pasting, navigation between cells in Excel without modifying or copying their content). The applicability of such tool, however, is limited to desktop applications that provide APIs for listening to UI events and accessing the data consumed and produced by these events. To achieve a more general solution, it may be necessary to combine this latter approach with OCR technology in order to detect UI events and associated data from application screenshots, as outlined in [35,38].

In its raw form, an UI log consists of one single sequence of UIs recorded during a session. During this session, a user may have performed several executions of one or multiple tasks, that may be mixed up in the log. Moreover, in case of multi-tasking, UIs of multiple concurrent task executions may be mixed together. Before identifying candidate routines for automation, an UI log has to be segmented into task traces, such that each trace corresponds to the execution of one task instance. This involves the identification of the boundaries of the tasks and the assignment of UIs to specific task traces. Given the fragment of the UI log demonstrated in the running example, we can extract two segments, each corresponding to the processing of a specific entry in the spreadsheet containing students' data (UIs 1 to 18 and 19 to 41 in Table 1).

The problem of extracting segments from an UI log corresponding to task instances is similar to that of web session reconstruction [40], where the goal is to identify the beginning and the end of web navigation sessions in server log data (e.g. streams of clicks and web page navigation) [40]. Methods for session reconstruction are usually based on heuristics that rely on the structural organization of websites or time intervals between events. The former approach covers only the cases where all the user interactions are performed in the web applications. In contrast, the latter approach assumes that users make breaks in-between two consecutive segments – in our case, two routine instances.

The problem of segmentation is also related to that of preprocessing so-called *uncorrelated event logs* in process mining. As discussed in [2,3] each event in a log should include, as a minimum, a case identifier, a timestamp, and an activity label. When the events of an event log do not have a case identifier, the log is said to be uncorrelated. Various methods have been proposed to extract correlated (i.e. regular) event logs from uncorrelated ones. However, existing methods in this field address the problem in restrictive settings. Specifically, some approaches [17] assume that the underlying process is acyclic, while others [10,11] assume that an explicit process model is given as input (in addition to the uncorrelated event log). These assumptions do not hold in the context of RPM, where no explicit process model is available, and a routine may contain repetitions. Also, the above approaches sometimes produce inaccurate results, whereas in the context of RPM, we need to identify routines with high levels of confidence (preferably 100% confidence), since an inaccurate replication of a routine by an unattended RPA bot may lead to costly errors.

In some scenarios, segmentation may be accomplished by combining transactional data recorded by enterprise information systems and user interactions logs, as proposed in [35]. However, a shortcoming of this approach is that such transactional data often provides only limited information about the process context, which is not enough to identify the boundaries of tasks captured in the user interactions logs.

Recent work on UI log segmentation [5,7] proposes to use trace alignment between the logs and the corresponding interaction models to identify the segments. In practice, however, such interaction models are not available beforehand.

Another related work [30] proposes to discover segments in the log by identifying cycles in the graph constructed from this log. These cycles represent repetitive behavior in the log and thus potentially correspond to task instances recorded in the log. However, this approach assumes that the task instances recorded in the log do not overlap and occur consequently one after the other.

In the context of desktop assistants, research proposals such as TaskTracer and TaskPredictor have tackled the problem of analyzing UI logs generated by desktop applications to identify the current task performed by a user and to detect switches between one task and another [15,39]. These approaches can potentially be used to split the UI logs into segments corresponding to different tasks. However, such approaches are not able to distinguish different instances of the same task.

Ideally, UIs recorded in a log should only relate to the execution of the task(s) of interest. However, in practice, a log often also contains UIs that do not contribute to completing the recorded task(s). We can consider such UIs to be *noise*. Examples of noise UIs include a worker browsing the web (e.g. social networking) while executing a task that does not require doing that, or a worker committing mistakes (e.g. filling a text field with an incorrect value or copying a wrong cell of a spreadsheet). UIs 6, 7, 8, 9, 10, and 11 are noise in our running example. During the creation of the student record, the worker decided to make a small pause, switched to a new tab in the web browser (6–7), and navigated to Facebook (8), where she spent almost 4 min browsing the news feed, before going back to the tab with the active student form (9). All these UIs do not have any relation to the task being recorded; thus, they constitute noise. When performing the task, the worker selected a surname field in the form (10) and made a mistake by accidentally misspelling the surname of the student (11). She then had to select the same field again (12) and fill it in with the correct value (13). Although the UIs 10 and 11 belong to the performed task, their effects are overwritten by successive UIs (e.g. UI 11 is overwritten by UI 13) and, therefore, they do not affect the outcome of the routine and are considered to be noise. The presence of the noise may negatively affect the subsequent steps of the RPM pipeline (e.g. the discovery of the candidate routines). Accordingly, the next step in the RPM pipeline is *simplification*, which aims at noise identification and removal. The UIs in the log are removed so that the resulting log captures the same effects as the original one while being simpler (i.e. having fewer UIs).

One of the challenges that arises during the pre-processing step of the RPM pipeline is to separate irrelevant UIs (i.e. noise) from those UIs that do contribute to the completion of a task. A possible approach is to assume that noise takes the form of chaotic events that may happen anywhere during process execution. One technique for filtering out such chaotic events is described in [42]. However, if noise gravitates towards one particular state or set of states in the task (e.g. towards the start or the end of the task), techniques such as the one mentioned above may not discover it and consequently not filter it out. Moreover, some UIs can be mistakenly removed due to the different ways the same task can be performed and induce what may mistakenly appear to be chaotic sequences of UIs. Thus, it is important to consider the data perspective, i.e. values of data objects that are manipulated by the UIs. In this way, one can identify the UIs that share the same parameter values (e.g. copying a value from a worksheet and then pasting it in a web form), or have the same source/origin (e.g. all the UIs are performed on the same website). The UIs that do not share any data parameters and/or values or originate from different sources most likely constitute noise.

Discovering Candidate Routines for Automation. Given a set of simplified task traces, the next phase is to identify candidate routines for automation. This phase aims at extracting repetitive sequences of UIs that occur across multiple task traces, a.k.a. routines, and to identify which of those routines are amenable for automation. The output of this step is a set of candidate routines for automation.

Even though an automated RPM tool can considerably reduce the effort required to automate routine, there is still a lot of development, quality assurance, and maintenance effort required to automate a routine in a real-life setting. Also, the automation of a routine may require re-training and re-allocation of human workers involved in the process. And if the routine is only partially automated (as opposed to fully automated), some handoffs will have to be put in place between the manual and the automated parts of a routine. As a result, the costs of automating a routine may sometime (or even often) outweigh the benefits. Thus, the cost-benefit analysis of routine automation is an important step in an end-to-end RPM method. To perform this analysis, a first step is to assess is a routine is suitable for automation.

Mindful of this requirement, Lacity and Willcocks [27] propose high-level guidelines for determining if a task is a candidate for automation in the context of a case study at Telefonica. The guidelines, however, do not provide a formal and precise definition of what makes a routine suitable for automation.

In a recent systematic review of the RPA literature, Syed et al. [41] conclude that "there is a need for formal, systematic and evidence-based techniques to determine the suitability of tasks for RPA.". In other words, a major challenge in the field of RPM is how to formally characterize what makes a routine amenable for automation via RPA or other automation technologies.

Two necessary criteria for a routine to be amenable for automation are:

1. *Frequency* [20] The more frequently a routine is performed, the more its automation is likely to lead to significant reductions in processing times, waiting times, and defects (due to human mistakes).
2. *Determinism* [12,34]. A candidate routine for automation should be such that a software bot is always able to determine the next step it should perform next in order to complete an execution of the routine. In other words, a routine can be automated only if: (1) every UI in the routine is deterministically activated, meaning that we know when to execute it (e.g. the box *International* is ticked whenever the student's country of residence is not Australia); and (2) every UI in the routine relies only on data produced by previous UIs (e.g. one of the UIs in the routine consists in entering the country of birth of a student into a field of a web form, and this data item has been previously copied from a cell of a spreadsheet in a previous UI).

Considering the running example provided in Table 1 and assuming that the identified task traces frequently occur in the log, we would discover two candidate routines, handling the domestic and international students, respectively. Note that the routine in the first task trace is only partially automatable. The worker manually filled in the form by looking at the corresponding entry values in the spreadsheet. Since she did not read the data values explicitly (e.g. by copying the values to the clipboard), these values are unknown for the recording tool. Hence, it is not possible to understand how the values used for editing the form's fields were obtained. On the other hand, the routine from the second task trace is fully automatable, as it is clear how to compute the values for the fields of the web form in the target application (i.e. by copying them from the spreadsheet).

Several techniques proposed in the field of UI log mining address the problem of identifying routines that fulfill the "frequency" criterion. Dev and Liu [14] have noted that the problem of frequent routine identification from (segmented) UI logs can be mapped to that of frequent pattern mining, a well-known problem in the field of data mining [22]. In the literature, several algorithms are available to mine frequent patterns from sequences of symbols. Depending on their output, we can distinguish two types of frequent pattern mining algorithms: those that discover only exact patterns [28,37] (hence vulnerable to noise), and those that allow frequent patterns to have gaps within the sequence of symbols [18,45] (hence noise-resilient).

Bosco et al. [12] address the problem of discovering routines that fulfill the "determinism" requirement. Specifically, this technique discovers sequences of actions such that the input(s) of each action in the sequence (except the first one) can be derived from the data observed in previous actions. However, this technique can only discover perfectly sequential routines and is hence not resilient to noise and variability in the order of the actions.

Leno et al. [29,31] combine techniques for discovering frequent routines, with techniques for discovering deterministic routines, thus addressing both of the above requirements. This latter proposal also addresses the problem of synthesizing an executable routine specification and that of detecting semantically equivalent routines, as discussed later in this chapter.

The discovery of automatable routines from sequences of actions is related to the problem of automated process discovery, discussed in [3,8] of this handbook. This relation is explored by Jiménez-Ramírez et al. [24], who apply process discovery techniques to extract process models from segmented UI logs. Importantly though, while it is possible to use automated process discovery algorithms to extract process models from segmented UI logs, the resulting process models cannot readily be used for automation (via RPA or other automation technology) for two reasons.

First, the process models discovered by process discovery techniques, such as those presented in [3,8], are control-flow models. They capture the occurrence and order of steps (tasks) in a process, but not the data taken as input and produced as output by each step in the process. Yet, in order to automate a routine, we need to know which data is used by each step in the routine and where these data comes from. We note that a subset of process discovery approaches can discover process models with data-driven branching conditions [13], or process models where some control-flow relations only hold under certain data-driven conditions [36], but they do not discover process models with data manipulation logic.

Second, the process models produced by automated process discovery techniques, typically contain traces that have not been observed (cf. the *generalization* property discussed in Chap. 2). However, when the purpose of a model is to serve as a blueprint for RPA, the generalization property is not desirable. Indeed, if a software bot executes such a model, it will sometimes produce sequences of action that might not correspond to a sequence of actions that a human worker would have performed. This, in turns, may lead to errors and these errors may later require time-consuming and costly corrective actions. Instead, routines for RPA must be 100% precise (cf. the definition of precision in Chap. 2), as a lack of precision may lead to potential errors when the routines are executed by an unattended RPA bot.

Discovering Executable Routine Specifications. Having identified a set of candidate routines for automation, the next step is that of *executable (sub-) routine discovery*. For each candidate routine, this step identifies the *activation condition* (UIs 2 and 19 in Table 1), which indicates when an instance of the routine should be triggered, and the *routine specification*, which specifies what UIs should be performed within that routine, what data is used by each UI in the routine, and how these data should be obtained.

The discovery of a routine specification involves identifying and synthesizing the transformation functions that have to be applied to the input data to convert it to the required format in the target application. In the running example, we can see that the web form requires a different date format than the one used in the spreadsheet (UIs 29 to 34). Hence, transferring the date of birth via simple copy and paste operations is insufficient, and the transformation function must be applied to achieve the desired result.

The problem of discovering executable routine specifications has been widely studied in the context of table auto-completion and data wrangling. For example, the Excel's Flash Fill feature detects string patterns in the values of the cells in a spreadsheet and uses these patterns for auto-completion [21]. Similarly, the authors in [9] propose an approach to extract structured relational data from semi-structured spreadsheets. However, such approaches can discover only the executable routines performed in one application and have a limited area of usage. In practice, the RPA routines often involve many of these applications.

Bosco et al. [12] suggest that the discovery of executable routine specifications can be tackled by applying methods for automated discovery of data transformations from examples [4,25]. However, these methods suffer from scalability issues when applied naively. Leno et al. [29] explore this approach and propose a series of optimizations to improve performance of the data transformation discovery techniques in the context of synthesis of routine specifications for RPA. This approach is further elaborated by the same authors in [31].

Gao et al. [19] extract rules from segmented UI logs to automatically fill in (web) forms. However, this approach only discovers branching conditions that specify whether a given activity has to be performed or not (e.g. check a box in a form) and only focuses on copy-paste operations without identifying more complex manipulations.

Agostinelly et al. [6] present an approach to discover routines from segmented UI logs and automate these routines via scripts. This approach, however, assumes that all the actions within a routine are automatable. In practice, it is possible that some actions have to be performed manually, and they can not be automated.

The output of the *executable (sub)routine discovery* step is a set of executable routine specifications of each automatable candidate routine. However, some of these specifications may produce identical effects, as they describe different variants of the same routine (e.g. filling in a web form in different orders). These variants are considered as duplicates and should be ignored, as their automation will not bring any benefits to the organization. Therefore, the next step in the RPM pipeline is *aggregation*. During this step, the discovered routine specifications leading to the same effects are replaced with one specification that captures the optimal way of performing the underlying routine. Several routine specifications may also be combined into a more complex specification that contains instructions on how to deal with different cases.

Once the script has been generated, it may be manually refined by an RPA developer, tested, and deployed into a production environment. The bot can be executed in *attended* or *unattended* settings. In attended settings, given an activation condition extracted from the routine specification, it can notify the user about its "readiness" to perform the routine when the condition is met and can be paused during execution, so that the user can make small corrections if needed and then resume the work. In unattended settings, the bot works independently without human involvement.

4 Outlook

There are a number of research challenges that need to be overcome to realize the vision of RPM, particularly in the areas of candidate routine discovery, extraction of automatable routines, and aggregation of equivalent routines (cf. Fig. 3).

In the area of candidate routine identification (and the related area of UI log segmentation), existing techniques assume that the routine instances are strictly separated in the UI log, i.e. there is no interleaving of user interactions belonging to one instance of one routine, and user interactions belonging to another instance of the same or of another routine. In practice, such interleaving may occur, for example, when a user is multi-tasking and thus alternating their attention between multiple routines.

In the area of automatable routine discovery, existing techniques are based on data transformation discovery, and as such they are limited to data transfer routines, where the goal is to take data from one system and transfer them to another system. Furthermore, these techniques are limited in scope to discovering routines where one record in one application, e.g. one row of a spreadsheet, is copied into one or more fields of another application (e.g. a web form). In reality, a single routine may involve complex iterations, for example, a routine may involve copying an invoice containing multiple invoice line-items from one application to another. In this case, the top-level routine (copying an invoice) contains a nested iterated sub-routine (copying multiple line items). These kind of structures cannot be discovered via existing data transformation discovery techniques. These latter techniques can discover that there is a routine consisting in copying an invoice line item, but they cannot reason holistically about the higher-level routine where the entire invoice is copied.

The area of routine aggregation is still a green field of research. A fundamental open problem in this space is the definition of notions of routine equivalence that would allow us to detect, for example, that a routine performed by one worker is the same as the one performed by another worker, even though these two workers perform the steps in their respective routines in completely different ways.

The RPM techniques discussed in this chapter focus on the discovery of routines that can be executed in an end-to-end manner by an RPA bot. This assumption is constraining. In reality, routines may be automated for a certain subset of cases, but not for all cases (i.e. automation may only be partially achievable). A key challenge, which goes beyond the scope of the proposed RPM pipeline, is how to discover partially deterministic routines. While a fully deterministic routine can be executed end-to-end in all cases, a partially deterministic routine can be stopped if the bot reaches a point where the routine cannot be deterministically continued given the input data and other data that the bot collects during the routine's execution. For example, while copying records of purchase orders from a spreadsheet or an enterprise system, a bot may detect that this order comes from China and then it may stop because it does not know how to handle such orders. Or, in a similar vein, a bot may find that a PO number is missing (the corresponding cell is empty), and hence it cannot proceed. Discovering conditions under which a routine cannot be deterministically continued (or started) is an open challenge in the field of RPM. Yet, this capability is a precondition to ensure that bots synthesized via RPM techniques can gracefully degrade and stop in order to hand off to human operators.

Finally, the vision of RPM exposed in this chapter, focuses on the problem of discovering automatable routines. Besides this problem, we envision that the field of RPM will encompass complementary problems and questions such as performance mining of RPA bots. This includes answering questions such as: "What is the success or defect rate of a bot when performing a given routine?", "What patterns are correlated with or are causal factors of bot failures?", and "Are there cases where the effects of a bot's actions are abnormal and warrant manual inspection?" In other words, over time, we envision that the scope of RPM will expand to cover the entire RPA lifecycle (cf. Fig. 1), rather than being purely focused on the development of RPA bots.

Acknowledgments. Work supported by the European Research Council (PIX project) and by the Australian Research Council (DP180102839).

References

1. van der Aalst, W.M.P., Bichler, M., Heinzl, A.: Robotic process automation. BISE **60**(4), 269–272 (2018)
2. van der Aalst, W.M.P.: Process mining: a 360 degrees overview. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 3–34. Springer, Cham (2022)
3. van der Aalst, W.M.P.: Foundations of process discovery. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 37–75. Springer, Cham (2022)
4. Abedjan, Z., Morcos, J., Ilyas, I.F., Ouzzani, M., Papotti, P., Stonebraker, M.: Dataxformer: a robust transformation discovery system. In 32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, 16–20 May 2016, pp. 1134–1145. IEEE Computer Society (2016)

5. Agostinelli, S.: Automated segmentation of user interface logs using trace alignment techniques (extended abstract). In: Di Ciccio, C., Depaire, B., De Weerdt, J., Di Francescomarino, C., Munoz-Gama, J., (eds.) Proceedings of the ICPM Doctoral Consortium and Tool Demonstration Track 2020, vol. 2703, CEUR Workshop Proceedings, pp. 13–14. CEUR-WS.org (2020)
6. Agostinelli, S., Lupia, M., Marrella, A., Mecella, M.: Automated generation of executable RPA scripts from user interface logs. In: Asatiani, A., et al. (eds.) BPM 2020. LNBIP, vol. 393, pp. 116–131. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58779-6_8
7. Agostinelli, S., Marrella, A., Mecella, M.: Automated segmentation of user interface logs. In: Czarnecki, C., Fettke, P., (eds.), Robotic Process Automation. De Gruyter (2021)
8. Augusto, A., Carmona, J., Verbeek, E.: Advanced process discovery techniques. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 76–107. Springer, Cham (2022)
9. Barowy, D.W., Gulwani, S., Hart, T., Zorn, B.G.: Flashrelate: extracting relational data from semi-structured spreadsheets using examples. In: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation 2015, pp. 218–228 (2015)
10. Bayomie, D., Awad, A., Ezat, E.: Correlating unlabeled events from cyclic business processes execution. In: Nurcan, S., Soffer, P., Bajec, M., Eder, J. (eds.) CAiSE 2016. LNCS, vol. 9694, pp. 274–289. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39696-5_17
11. Bayomie, D., Di Ciccio, C., La Rosa, M., Mendling, J.: A probabilistic approach to event-case correlation for process mining. In: Laender, A.H.F., Pernici, B., Lim, E.-P., de Oliveira, J.P.M. (eds.) ER 2019. LNCS, vol. 11788, pp. 136–152. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33223-5_12
12. Bosco, A., Augusto, A., Dumas, M., La Rosa, M., Fortino, G.: Discovering automatable routines from user interaction logs. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNBIP, vol. 360, pp. 144–162. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26643-1_9
13. de Leoni, M., Dumas, M., García-Bañuelos, L.: Discovering branching conditions from business process execution logs. In: Cortellessa, V., Varró, D. (eds.) FASE 2013. LNCS, vol. 7793, pp. 114–129. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37057-1_9
14. Dev, H., Liu, Z.: Identifying frequent user tasks from application logs. In: Proceedings of IUI 2017, pp. 263–273. Springer (2017)
15. Dragunov, A.N., Dietterich, T.G., Johnsrude, K., McLaughlin, M.R., Li, L., Herlocker, J.L.: Tasktracer: a desktop environment to support multi-tasking knowledge workers. In: IUI, ACM (2005)
16. Gonzalez, J., et al.: Robotic process automation: a scientific and industrial systematic mapping study. IEEE Access **8**, 39113–39129 (2020)
17. Ferreira, D.R., Gillblad, D.: Discovering process models from unlabelled event logs. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 143–158. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03848-8_11
18. Fumarola, F., Lanotte, P.F., Ceci, M., Malerba, D.: CloFAST: closed sequential pattern mining using sparse and vertical id-lists. Knowl. Inf. Syst. **48**(2), 429–463 (2016)

19. Gao, J., van Zelst, S.J., Lu, X., van der Aalst, W.M.P.: Automated robotic process automation: a self-learning approach. In: Panetto, H., Debruyne, C., Hepp, M., Lewis, D., Ardagna, C.A., Meersman, R. (eds.) OTM 2019. LNCS, vol. 11877, pp. 95–112. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33246-4_6
20. Geyer-Klingeberg, J., Nakladal, J., Baldauf, F., Veit, F.: Process mining and robotic process automation: a perfect match. In: Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018, pp. 124–131. CEUR-WS.org (2018)
21. Gulwani, S.: Automating string processing in spreadsheets using input-output examples. In: Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, pp. 317–330 (2011)
22. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data Mining Knowl. Disc. **15**(1), 55–86 (2007)
23. Intellipaat. RPA Lifecycle. https://intellipaat.com/blog/tutorial/rpa-tutorial/rpa-lifecycle/. Accessed 12 Sep 2021
24. Jimenez-Ramirez, A., Reijers, H.A., Barba, I., Del Valle, C.: A method to improve the early stages of the robotic process automation lifecycle. In: Giorgini, P., Weber, B. (eds.) CAiSE 2019. LNCS, vol. 11483, pp. 446–461. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21290-2_28
25. Jin, Z., Anderson, M.R., Cafarella, M.J., Jagadish, H.V.: Foofah: transforming data by example. In: SIGMOD, ACM (2017)
26. Kerremans, M., Srivastava, T.: Discover the differences and use cases of process mining versus task mining. Research Note G00723821, Gartner, April 2020
27. Lacity, M., Willcocks, L.P.: Robotic process automation at telefónica O2. MIS Q. Execut. **15**(1), 1–4 (2016)
28. Lee, S.D., De Raedt, L.: An efficient algorithm for mining string databases under constraints. In: Goethals, B., Siebes, A. (eds.) KDID 2004. LNCS, vol. 3377, pp. 108–129. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31841-5_7
29. Leno, V., Dumas, M., La Rosa, M., Maggi, F.M., Polyvyanyy, A.: Automated discovery of data transformations for robotic process automation. arXiv:abs/2001.01007 (2020)
30. Leno, V., Augusto, A., Dumas, M., La Rosa, M., Maggi, F.M., Polyvyanyy, A.: Identifying candidate routines for robotic process automation from unsegmented UI logs. In: 2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, 4–9 October 2020, pp. 153–160. IEEE (2020)
31. Leno, V., Augusto, A., Dumas, M., La Rosa, M., Maggi, F.M., Polyvyanyy, A.: Discovering data transfer routines from user interaction logs. Inf. Syst. **107**, 101916 (2022)
32. Leno, V., Polyvyanyy, A., Dumas, M., La Rosa, M., Maggi, F.M.: Robotic process mining: vision and challenges. Bus. Inf. Syst. Eng. **63**(3), 301–314 (2021)
33. Leno, V., Polyvyanyy, A., La Rosa, M., Dumas, M., Maggi, F.M.: Action logger: enabling process mining for robotic process automation. In Proceedings of the Dissertation Award, Doctoral Consortium, and Demonstration Track at BPM 2019, vol. 2420, CEUR Workshop Proceedings, pp. 124–128. CEUR-WS.org (2019)
34. Leopold, H., van der Aa, H., Reijers, H.A.: Identifying candidate tasks for robotic process automation in textual process descriptions. In: Gulden, J., Reinhartz-Berger, I., Schmidt, R., Guerreiro, S., Guédria, W., Bera, P. (eds.) BPMDS/EMMSAD -2018. LNBIP, vol. 318, pp. 67–81. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91704-7_5

35. Linn, C., Zimmermann, P., Werth, D.: Desktop activity mining - a new level of detail in mining business processes. In: Workshops der INFORMATIK 2018 - Architekturen, Prozesse, Sicherheit und Nachhaltigkeit, pp. 245–258 (2018)
36. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Data-driven process discovery - revealing conditional infrequent behavior from event logs. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 545–560. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_34
37. Ohlebusch, E., Beller, T.: Alphabet-independent algorithms for finding context-sensitive repeats in linear time. J. Disc. Algorithm **34**, 23–36 (2015)
38. Jimenez-Ramirez, A., Reijers, H.A., Barba, I., Del Valle, C.: A method to improve the early stages of the robotic process automation lifecycle. In: Giorgini, P., Weber, B. (eds.) CAiSE 2019. LNCS, vol. 11483, pp. 446–461. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21290-2_28
39. Shen, J., Li, L., Dietterich, T.G.: Real-time detection of task switches of desktop users. In: IJCAI (2007)
40. Spiliopoulou, M., Mobasher, B., Berendt, B., Nakagawa, M.: A framework for the evaluation of session reconstruction heuristics in web-usage analysis. Informs J. Comput. **15**(2), 171–190 (2003)
41. Syed, R., et al.: Robotic process automation: contemporary themes and challenges. Comput. Ind. **115**, 103162 (2020)
42. Tax, N., Sidorova, N., van der Aalst, W.M.P.: Discovering more precise process models from event logs by filtering out chaotic activities. J. Intell. Inf. Syst. **52**(1), 107–139 (2019)
43. Tornbohm, C.: Gartner market guide for robotic process automation software. Report G00319864, Gartner (2017)
44. van Zelst, S.J., Mannhardt, F., de Leoni, M., Koschmider, A.: Event abstraction in process mining: literature review and taxonomy. Granul. Comput. **6**(3), 719–736 (2020). https://doi.org/10.1007/s41066-020-00226-2
45. Wang, J., Han, J.: Bide: efficient mining of frequent closed sequences. In :Proceedings of the 20th International Conference on Data Engineering, pp. 79–90. IEEE (2004)

Closing

Scaling Process Mining to Turn Insights into Actions

Wil M. P. van der Aalst[1](✉) and Josep Carmona[2]

[1] Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany
wvdaalst@pads.rwth-aachen.de
[2] Universitat Politècnica de Catalunya, Barcelona, Spain
jcarmona@cs.upc.edu

Abstract. This final chapter reflects on the current status of the process mining discipline and provides an outlook on upcoming developments and challenges. The broader adoption of process mining will be a gradual process. Process mining is already used for high-volume processes in large organizations, but over time process mining will also become the "new normal" for smaller organizations and processes with fewer cases. To get the highest return on investment, organizations need to "scale" their process mining activities. Also, from a research point-of-view, there are many exciting challenges. On the one hand, many of the original problems (e.g., discovering high-quality process models and scaling conformance checking) remain (partly) unsolved, still allowing for significant improvements. On the other hand, the large-scale use of process mining provides many research opportunities and generates novel scientific questions.

Keywords: Process mining · Execution management · Process management

1 Process Mining: Overview and Summary

The chapters in this book illustrate the broadness of the process mining discipline. The interplay between data science and process science provides many challenges and opportunities [1]. In this book, we aim to provide a comprehensive overview. There are many dimensions to characterize the 16 earlier chapters.

- *Theory-driven* versus *application-driven*.
- *Backward-looking* (e.g., process discovery and conformance checking) versus *forward-looking* (e.g., simulation and predictive process analytics).
- *Simple* control-flow-oriented event logs versus *complex* object-centric event data considering *different types of objects* and attributes.

In the first chapter of this book [3], we started with Fig. 1 showing a 360 degrees overview of process mining. The subsequent chapters have been focusing on different parts of the pipeline depicted in Fig. 1. The initial chapters focused on *process discovery*, starting with creating a simple Directly-Follows Graph (DFG) followed by a range of alternative, more sophisticated, techniques. As shown, process discovery is an important topic, but also very difficult [1]. Event data do not contain negative examples and

W. M. P. van der Aalst and J. Carmona (Eds.): Process Mining Handbook, LNBIP 448, pp. 495–502, 2022.
https://doi.org/10.1007/978-3-031-08848-3_17

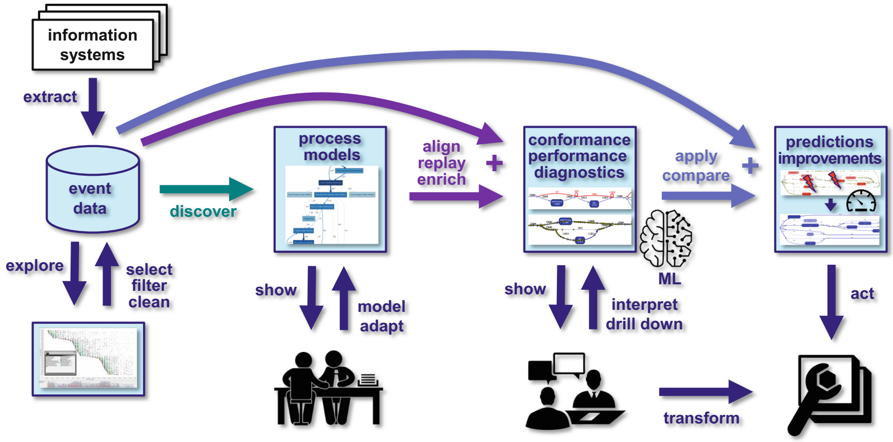


Fig. 1. Process mining uses event data extracted from information systems to provide insights and transparency that, ultimately, should lead to process improvements (i.e., process redesign, improved control, and automation).

the positive examples typically only cover a fraction of all possible behaviors. Mixtures of choice, concurrency, and loops make process discovery a *notoriously difficult task* with many *trade-offs*. Also, process models may be used for different purposes.

After discovery, the focus shifted to *conformance checking* [1,5]. Here the input consists of both modeled and observed behavior. For example, a multiset of traces is compared with an accepting Petri net. Surprisingly, state-of-the-art conformance checking techniques tend to be more demanding than discovery techniques (from a computational point of view). Computing alignments corresponds to solving optimization problems that grow exponentially in the size of the model and the length of traces in the event log.

Several chapters discussed the importance and complexity of data extraction and preprocessing. Later chapters focused on practical applications and more advanced topics such as model enhancement, streaming process mining, distributed process mining, and privacy-preserving process-mining techniques.

Figure 2 shows another overview of the building blocks of a successful process mining solution. The top of Fig. 2 shows examples of application domains where process mining can be used. In this book, we elaborated on applications in healthcare, auditing, sales, procurement, and IT services. However, process mining is a generic technology that can be used in any domain.

In the remainder of this concluding chapter, we take a step back and reflect on the developments in our discipline. Section 2 discusses the inevitability of process mining, but also stresses that concepts such as a Digital Twin of an Organization (DTO) are still far away from being a reality. Section 3 explains that it is important to scale process mining. Finally, Sect. 4 provides an outlook also listing research challenges.

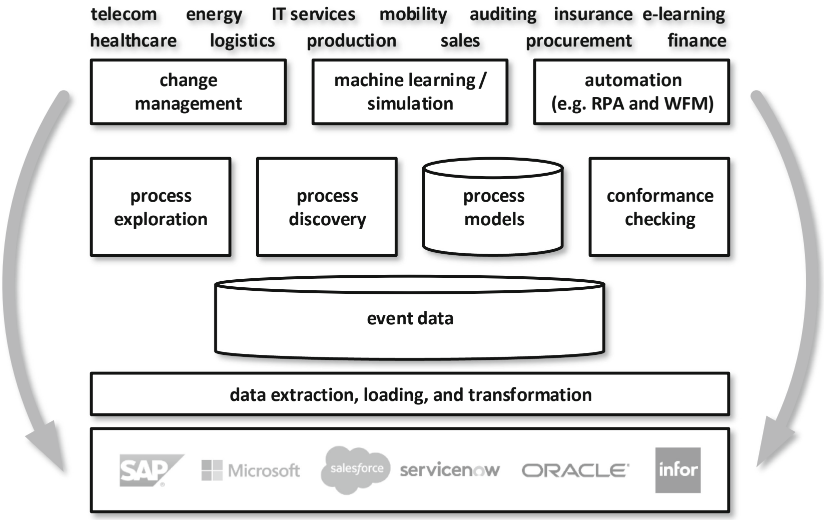


Fig. 2. Process mining can be used in any application domain. However, it may be non-trivial to extract accurate event data and turn process mining results into actions. Change management and automation play a key role in realizing sustained improvements (as indicated by the two arcs closing the loop).

2 Process Mining as the New Normal

Although process mining has proven its value in many organizations, it is not so easy to create a convincing *business case* to justify investments [1]. The reason is that process mining will most likely reveal performance and compliance problems, but this does not imply that these are automatically solved [8]. Financial and technical debts are well-known concepts. However, most organizations tend to ignore their *Operational Process Debts* (OPDs). OPDs cause operational friction, but are difficult to identify and address. Although process mining results are often surprising, they typically reveal OPDs that were known to some, but not addressed adequately. Making these OPDs visible and transparent helps to address them.

In [2], the first author coined the term *Process Hygiene* (PH) to stress that process mining should be the "new normal" not requiring a business case. Just like personal hygiene, one should not expect an immediate return on investment. We know that activities such as brushing our teeth, washing our hands after going to the toilet, and changing clothes are the right thing to do. The same applies to process mining activities, i.e., process hygiene serves a similar role as personal hygiene. People responsible for operational processes need to be willing to look at possible problems. Objectively monitoring and analyzing key processes is important for the overall health and well-being of an organization. Process mining helps to ensure process hygiene. Not using process mining reflects the inability or unwillingness to manage processes properly. Fortunately, an increasing number of organizations is aware of this.

Although process mining is slowly becoming the "new normal", most organizations will *not* be able to use the forward-looking forms of process mining. As long as the extraction of event data, process discovery, and conformance checking are challenging for an organization, it is unlikely that machine learning and other forward-looking techniques (including artificial intelligence and simulation) will be of help. Terms such as the *Digital Twin of an Organization* (DTO) illustrate the desire to autonomously manage, adapt, and improve processes. Gartner defines a DTO as "a dynamic software model of any organization that relies on operational and/or other data to understand how an organization operationalizes its business model, connects with its current state, responds to changes, deploys resources and delivers exceptional customer value". Creating a DTO can be seen as one of the grand challenges in information systems, just like autonomous driving in mobility. However, just like the development of self-driving cars, the process will be slow with many minor incremental improvements.

3 Scaling Process Mining

One of the main conclusions in [6] is that process mining needs *scale* to be most cost effective. Organizations need to aim for the *continuous* usage of process mining for *many processes* by *many people*. Initially, process mining was primarily used in process improvement projects. In such projects, a problematic process is analyzed to provide recommendations for improvement. Since data extraction is often the most problematic step, such projects often struggle to get results quickly. Moreover, the "end product" of such a project is often a just a report. To improve the process, change management and automation efforts are still needed. Therefore, traditional process mining projects struggle to realize a good Return on Investment (ROI).

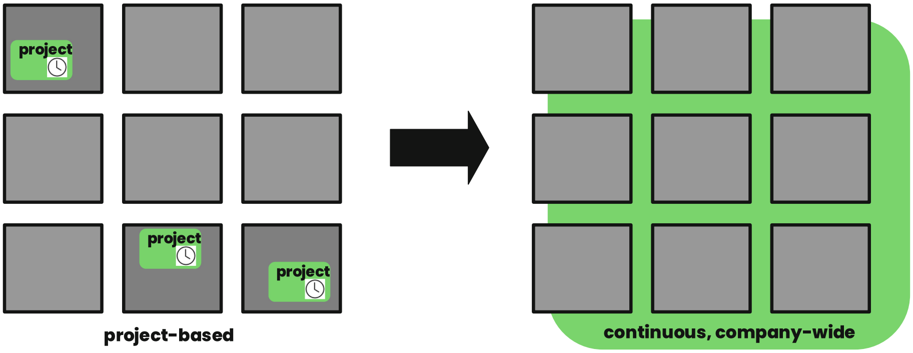


Fig. 3. Scaling process mining to maximize the benefits.

Therefore, process mining should not be seen as a project, but as a *continuous company-wide* activity as shown in Fig. 3. There are several reasons for this.

- If data extraction is done properly, the initial efforts are high, but this can be repeated without much extra work. Once the data extraction pipeline is realized, it is possible to *continuously* produce process mining results based on new event data.
- Process mining is a *generic* technology. Hence, investments in software and people should be spread over many processes and organizational units. For example, an insurance company that has multiple products (e.g., different types of insurance) and multiple offices (in different countries and cities) should not limit process mining to one product or one location.
- Organizational change often requires commitment from many stakeholders. Therefore, results should be *visible* for *all* involved in the process. If performance and compliance problems are only visible to a small group of experts, it is difficult to realize durable behavioral changes. Many improvement projects fail because people slip back into old ways of working after some time.

Compare process mining for an organization to creating weather forecasts for a country. It does not make any sense to create a weather forecast for just one city on a particular day. Investments only make sense if one is able to create a weather forecast for any city on any day. Similarly, process mining is most effective when applied to many processes continuously.

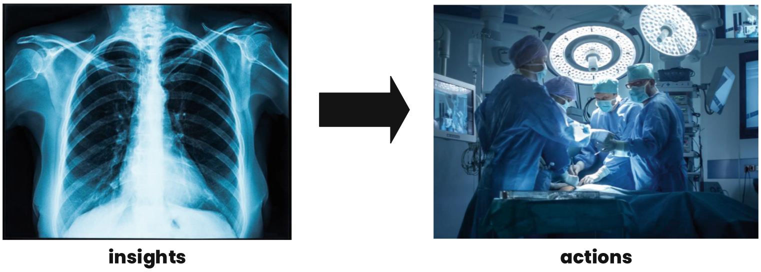


Fig. 4. Turning insights into actions.

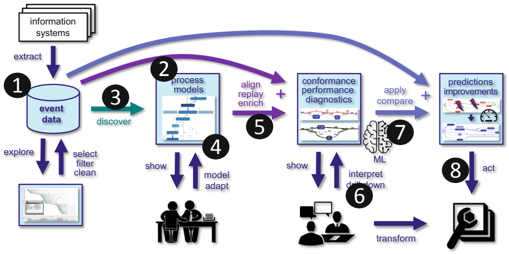
As part of scaling process mining, it is essential that insights are turned into concrete improvement actions. This is illustrated in Fig. 4. Process discovery and conformance checking can be seen as creating detailed X-ray images to detect problems and find root causes [1]. However, the value of an X-ray image is limited if it is not followed by interventions and treatment, e.g., surgery, chemotherapy, diet, and radiation therapy. Therefore, commercial process mining vendors are combining process mining with automation, e.g., Robotic Process Automation (RPA) and low-code automation platforms like Make.

4 Outlook

How will the process mining discipline and market evolve? Most analysts expect the usage of process mining to grow exponentially in the coming years. Given the growing availability of event data and mature tools, there is no reason to doubt this. To predict the evolution of methods, techniques, and software capabilities, it is good to take another look at the *process mining manifesto* [7] written by the *IEEE Task Force on Process Mining* in 2011. The process mining manifesto lists the following eleven challenges.

- *C1: Finding, Merging, and Cleaning Event Data*
- *C2: Dealing with Complex Event Logs Having Diverse Characteristics*
- *C3: Creating Representative Benchmarks*
- *C4: Dealing with Concept Drift*
- *C5: Improving the Representational Bias Used for Process Discovery*
- *C6: Balancing Between Quality Criteria such as Fitness, Simplicity, Precision, and Generalization*
- *C7: Cross-Organizational Mining*
- *C8: Providing Operational Support*
- *C9: Combining Process Mining With Other Types of Analysis*
- *C10: Improving Usability for Non-Experts*
- *C11: Improving Understandability for Non-Experts*

There has been substantial progress in the areas covered by these challenges posed over a decade ago. For example, we now have comprehensive sets of publicly available benchmarks (C3) and we much better understand the different quality criteria (C6). Thanks to the over 40 commercial process mining tools, it is now much easier to apply process mining (C10) and understand the diagnostics (C11). Due to the many approaches combining process mining and machine learning, there has been major progress with respect to C8 and C9. Nevertheless, most of the challenges are still relevant and even basic problems such as process discovery and conformance checking have not been completely solved.



1 Finding, extracting, and transforming event data is still taking up to 80% of the time.

2 Most techniques focus on a single case notion (i.e., a single process), whereas problems may be caused by interacting or competing processes.

3 Process discovery is not a solved problem despite powerful techniques like inductive mining. Concurrency is hard to discover from event data that provide only a sample.

4 There is a need to better integrate mining and modeling (e.g., user-guided discovery).

5 Conformance checking is time-consuming and diagnostics tend to be non-deterministic.

6 There is a need for techniques recommending process changes (i.e., moving beyond diagnostics).

7 Machine Learning (ML) techniques tend to perform poorly because essential aspects are missed (e.g., system load).

8 Process mining results need to trigger automated actions (e.g., start a corrective workflow).

Fig. 5. Process mining challenges in focus in the next five years.

Figure 5 annotates the overview diagram with some of the most relevant challenges for the coming years. There is quite some overlap with the eleven challenges in [7]. For example, finding, extracting and transforming input data is still one of the main challenges when applying process mining in practice. Approaches such as object-centric process mining [3,4] try to make this easier by storing information about multiple objects in a consistent manner and allowing for process models that are not limited to a single case notion. Figure 5 also shows that there are still many open problems when it comes to basic capabilities such as process discovery and conformance checking. Figure 5 also lists challenges that were not discussed in [7]. For example, how to better combine algorithms and domain knowledge to create better process models (*user-guided discovery*) and suggest improvements. There is also an increased emphasis on using process mining results to automatically trigger improvements (*action-oriented process mining*).

We hope that this chapter and book will inspire both academics and practitioners to work on these important challenges. The process mining discipline is rapidly developing and there is still room for original and significant contributions.

Acknowledgments. Funded by the Alexander von Humboldt (AvH) Stiftung and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2023 Internet of Production – 390621612.

References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer, Berlin (2016). https://doi.org/10.1007/978-3-662-49851-4
2. Aalst, W.: Academic view: development of the process mining discipline. In: Process Mining in Action, pp. 181–196. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-40172-6_21
3. van der Aalst, W.M.P.: Chapter 1 - process mining: a 360 degrees overview. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448. Springer, Berlin (2022). https://doi.org/10.1007/978-3-662-49851-4
4. van der Aalst, W.M.P., Berti, A.: Discovering object-centric Petri nets. Fundam. Informat. **175**(1–4), 1–40 (2020)
5. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking: Relating Processes and Models. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-99414-7
6. Galic, G., Wolf, M.: Global process mining survey 2021: delivering value with process analytics - adoption and success factors of process mining. Deloitte (2021). https://www2.deloitte.com/de/de/pages/finance/articles/global-process-mining-survey-2021.html
7. van der Aalst, et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28108-2_19
8. Reinkemeyer, L.: Process Mining in Action: Principles. Use Cases and Outlook. Springer-Verlag, Berlin (2020). https://doi.org/10.1007/978-3-030-40172-6

Author Index