Chapter 11
Process Monitoring

If you can't measure something, you can't understand it. If you can't understand it, you can't control it. If you can't control it, you can't improve it.
H. James Harrington (1929–)

Once we have implemented and deployed a redesigned business process, it may happen that the new process does not meet our expectations. For example, certain types of unforeseen exceptions may arise, the processing time of some tasks may be much higher than expected due to these exceptions, and queues may build up to the extent that process participants start taking shortcuts due to high pressure, while customers become unsatisfied due to long waiting times. A first step to address (and to preempt) these issues is to understand what is actually happening during the execution of the process. This is the overarching goal of the last phase of the BPM lifecycle, namely *process monitoring*.

We start this chapter by discussing the context of process monitoring, i.e., when process monitoring can be done, what classes of techniques exist, what data is required as input, and what output can be produced by these techniques. Next, we introduce common types of process performance dashboards, both for offline and online process monitoring. We then move to process mining techniques, which emphasize the use of process models for process monitoring. We conclude by showing how these techniques provide a bridge from the monitoring phase back to the discovery and analysis phases of the BPM lifecycle.

11.1 The Context of Process Monitoring

Process monitoring is about using the data generated by the execution of a business process in order to extract insights about the actual performance of the process and to verify its conformance with respect to norms, policies, or regulations. The data

M. Dumas et al., *Fundamentals of Business Process Management*,
https://doi.org/10.1007/978-3-662-56509-4_11

generated by business process executions generally takes the form of collections of event records. Each event record captures a state change in a process, such as the start or completion of a task, an incoming or outgoing message, or a timeout or escalation. Collections of such event records are called *event logs*.

Business process monitoring techniques take as input event logs and produce a number of artifacts to help process participants, analysts, process owners, and other managers to get a picture of the performance of the process at different levels of detail. Some performance monitoring techniques allow us to detect that the actual execution of the process deviates with respect to the intended execution captured in a process model, e.g., an invoice is paid out before it is approved by a financial officer, whereas normally this should not be the case. Other techniques help analysts to understand how and why the performance of a process varies across different cases or groups of cases, e.g., why some cases take too long to be completed. Or why some cases lead to customer complaints.

Process monitoring techniques can be broadly classified into those that provide a post-mortem view of a process, focused on already completed cases, and those that provide an on-the-fly view, focused on running cases. In this respect, we can classify process monitoring techniques into two categories:

Offline Process Monitoring is concerned with the analysis of historic process executions. The input for offline process monitoring are event logs covering a set of cases completed during a particular period of time, for instance a month, a quarter or a full year. Offline process monitoring techniques provide a picture of the performance of the process, the reasons for poor performance or for undesirable performance variations, and the conformance of the process with respect to certain rules or expected behaviors.

Online Process Monitoring is concerned with the assessment of the performance of currently running process instances. The main input for process monitoring are (incomplete) traces of ongoing cases. Online process monitoring techniques produce real-time pictures of the performance of ongoing cases, and generate alarms or trigger counteractions whenever certain performance objectives or compliance rules are not fulfilled, e.g., when a customer request remains un-replied beyond a given period of time.

Process monitoring techniques can also be classified into *statistics-based* and *model-based* techniques. The former category of techniques relies on statistical analysis of performance measures. This category encompasses descriptive analyses of the distribution of a given performance measure via aggregation functions such as mean, standard deviation, minimum and maximum cycle time, and processing time. It also encompasses visualization techniques allowing us for example to compare the distribution of the cycle time of ongoing cases of a process against that of all cases in the current semester or in the same month of the previous year. Process monitoring tools that emphasize statistics-based techniques are known as *Business Activity Monitoring* (BAM) or *Process Performance Measurement* (PPM) tools. The main output of these tools are *performance dashboards* displaying the performance

of a process via a combination of aggregate measures (e.g., mean cycle time, mean processing time) and various types of charts as discussed in the next section.

On the other hand, model-based techniques allow us to analyze the execution of cases based on process models. For example, given an event log containing events related to purchase orders, shipments, deliveries, and invoices, some model-based techniques allow us to discover a process model of an order-to-cash process and to visualize the performance of the process (e.g., waiting times) on top of this automatically discovered model. Other model-based monitoring techniques allow us to detect deviations between an event log and a process model manually constructed by an analyst. For example, the latter techniques allow us to detect that in some cases an invoice has been paid without a required approval. Process monitoring tools that emphasize the use of process models are known as *process mining* tools.

11.2 Process Performance Dashboards

A process performance dashboard is a graphical representation of one or more performance measures or other characteristics of a business process. Depending on their purpose and targeted users, process performance dashboards can be classified into three categories: *operational*, *tactical*, and *strategic*.

11.2.1 Operational Dashboards

Operational process dashboards target process participants and their operational managers (including the process owner). Their purpose is to display the performance of ongoing or recently completed cases in a way that allows process participants and their managers to plan their short-term work. For example, in the context of an order-to-cash process, an operational dashboard might target a financial officer involved in the business process or a dispatcher at a warehouse.

Typical measures displayed in an operational dashboard include the number of ongoing cases (work-in-process) classified by type of cases. For example, a dashboard may display the number of cases that are on time, overdue, and at risk of being overdue. This idea is illustrated in Figure 11.1, which shows a dashboard produced by Bizagi's BAM component. The left-hand side of this dashboard displays the work-in-process by means of a pie chart. The same dashboard contains a histogram (see bar chart on the right) showing the number of cases that will reach their expiry time (deadline) for different periods of time.

Sometimes, operational dashboards are defined at the level of resources or tasks rather than at the level of cases. For example, we saw in Figure 9.5 an operational performance dashboard generated by a BPMS (Perceptive), which displays the number of pending work items per resource.

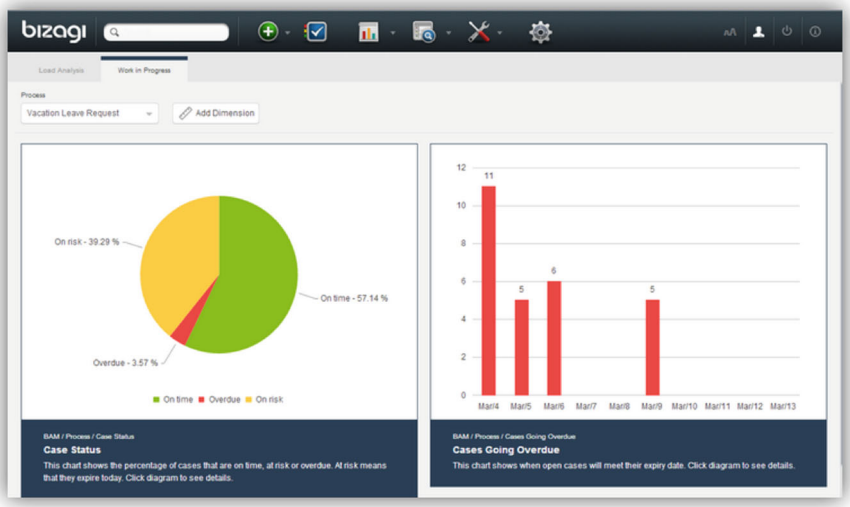


Fig. 11.1 Example of operational dashboard produced by Bizagi's BAM component

Exercise 11.1 We consider the prescription fulfillment process in Exercise 1.6 (page 30). Throughout a working day, the pharmacists and technicians at the pharmacy need to monitor the workload over the coming hours and detect situations where deadlines might be missed. Note that each case (i.e., each prescription) has a designated pick-up time (e.g., some prescriptions will be picked up at 5 p.m., others at 6 p.m., etc.). Also, we can assume that each pharmacist and each technician can handle a number of cases per hour. You can assume that pharmacists can do their part of the work on a given prescription in about 6 min of processing time, while pharmacists need 10 min of processing time per prescription to do their part of the work. You may also assume that a given prescription will always be in one of the following states: unprocessed, entered-and-verified, ready-to-deliver, and delivered. Describe two possible operational process dashboards to fulfill this need.

11.2.2 Tactical Dashboards

Tactical process dashboards target process owners, functional managers with oversight over parts of a process, and the analysts upon whom these managers rely. The purpose of tactical dashboards is to give a picture of the performance of a process over a relatively long period of time (e.g., a month or quarter or even a year) in order to put into evidence undesirable performance variations and their possible causes, long-term bottlenecks and deviations, or frequent sources of defects.

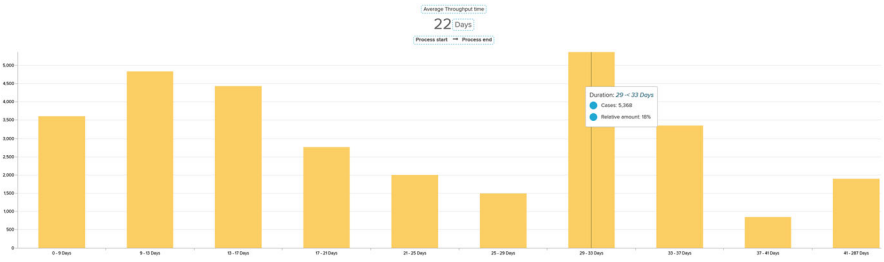


Fig. 11.2 Cycle time histogram of cases completed during a 1-year period

Typical measures displayed in a tactical process dashboard include number of completed cases, cycle time, waiting times, processing times, resource utilization, cost per case, and defect rate. For a given performance measure, a tactical dashboard may display statistics such as the minimum, maximum, mean, and standard deviation of this measure, or the entire distribution of the performance measure in the form of a histogram. For example, Figure 11.2 shows a cycle time (called throughput time) histogram of cases of a business process completed over a period of 1 year. For each bucket in the histogram, the dashboard allows us to inspect the number and the percentage of cases whose cycle time falls within that bucket. Some dashboards would allow us to drill drown and inspect the specific set of cases behind each bar.[1]

Since some of the above performance measures can also be defined at the level of tasks of a process, a tactical dashboard may provide a decomposed view of a performance measure on a per-task basis. For example, a tactical dashboard may display the mean waiting time of each task of a process. Such dashboards allow analysts to identify performance bottlenecks.

Tactical dashboards may also provide *longitudinal* or *cross-sectional* views of the performance of a process. A longitudinal view shows the variation of a given performance measure over time. For example, a longitudinal dashboard may show the mean cycle time of a process on a per-month basis. Such dashboards allow us to observe trends. A cross-sectional view is one that shows the distribution of a performance measure according to a given classification of cases. For example, a tactical dashboard may show the distribution of cases by geographical region (overlaid on a map) and within each region, it may also show the distribution of cases by type of product.

We can also use tactical process dashboards to comparatively analyze the performance of different teams. For example, if a lead-to-order process is performed by multiple sales teams in different geographical locations, a tactical dashboard may display the amount of time that each team spends in each stage of this lead-to-order

[1]This histogram was produced by Celonis—a tool we will mention again later in the context of process mining.

process. Such a comparative dashboard allows analysts and managers to understand why some teams perform better than others.

Exercise 11.2 Let us consider again the pharmacy prescription fulfillment process in Exercise 1.6 (page 30). The owner of this process oversees hundreds of pharmacies distributed across multiple regions. Specify what performance measures should be displayed and describe how could they be displayed in order to help the process owner to spot issues related to poor customer service.

11.2.3 Strategic Dashboards

Strategic process dashboards target executive managers. Their purpose is to provide a high-level picture of the performance of groups of processes along multiple performance dimensions. Accordingly, strategic dashboards are typically constructed by aggregating performance measures in two ways: (i) aggregating performance measures defined for individual processes into performance measures defined for entire groups of processes in a process architecture; and (ii) aggregating multiple performance measures related to the same performance dimension (e.g., multiple measures related to efficiency) into a single measure.

Example 11.1 We consider an insurance company with three core groups of processes: service development, sales, and claims handling. For each group of processes, there may be multiple measures related to efficiency, such as cycle time efficiency, resource utilization, and cost-to-revenue ratio per case.[2] Given these performance measures, and assuming that we give them equal weight, we can define a single measure of process efficiency using a weighted average. Let us assume that there are twelve processes related to claims handling (one process per type of service, for example claims handling for personal vehicle insurance, for commercial vehicle insurance, for home insurance, etc.). If we give equal weight to each of these twelve processes, we can define a measure of efficiency for the entire group of claims handling processes in the architecture. If we repeat this procedure for each type of performance dimension (efficiency being only one of them) and for each group of processes (claims handling being one of them), we get a set of performance measures for each group of processes in the architecture. Given this, we can construct a strategic performance dashboard that displays to us a set of aggregated performance measures for each group of processes in a process architecture. This can be achieved, for example, by using balanced scorecards as shown in Figure 2.10 (see page 63). This dashboard can then be used by executives to get a bird's-eye view of the performance of a given business area over a period

[2]The performance dimensions of an organization are usually defined in a *balanced scorecard* introduced in Section 2.1. In a balanced scorecard, efficiency is one of the performance dimensions of the *internal business process* perspective.

of time (a month, a quarter, or a year), and to compare it with the performance in previous periods, or in other business areas. □

Exercise 11.3 Consider the process landscape of Vienna's public transport operator Wiener Linien given in Figure 2.6 (page 48). The CFO of this organization has a long-term goal to improve efficiency, mainly by reducing costs. What dashboard could we offer to the CFO (updated every month) to give them an idea of where to focus the attention in terms of process improvement efforts? What data would we need to produce this dashboard?

11.2.4 Tools for Dashboard Creation

Operational and tactical process dashboards are often provided off-the-shelf by almost any BPMS. They can also be created using dedicated dashboarding tools available in other PAISs, e.g., CRM and ERP systems such as Microsoft Dynamics and SAP. Some types of process performance dashboards are also provided off-the-shelf by process mining tools, which we shall discuss next.

It is also a common practice to create process performance dashboards (at all three levels) using business analytics and business data visualization tools, such as Microsoft PowerBI,[3] QlikView,[4] and Tableau.[5] These tools allow one to design customized dashboards by dragging and dropping dashboard components using a visual editor and to import data from files or from databases.

One of the trends in the field of performance dashboards is to incorporate machine learning techniques in order to make predictions about future performance. At the time of writing this book, BPMS and process mining vendors are gradually incorporating predictive capabilities into their products. An example of an open-source tool to create predictive business process performance dashboards is Nirdizati.[6]

11.3 Introduction to Process Mining

Process mining is a family of techniques to analyze the performance and conformance of business processes based on event logs produced during their execution. Process mining techniques complement tactical process monitoring dashboards. Whereas tactical dashboards allow managers and analysts to get an aggregate picture

[3] http://powerbi.microsoft.com.

[4] http://www.qlik.com.

[5] https://www.tableau.com.

[6] http://nirdizati.org.

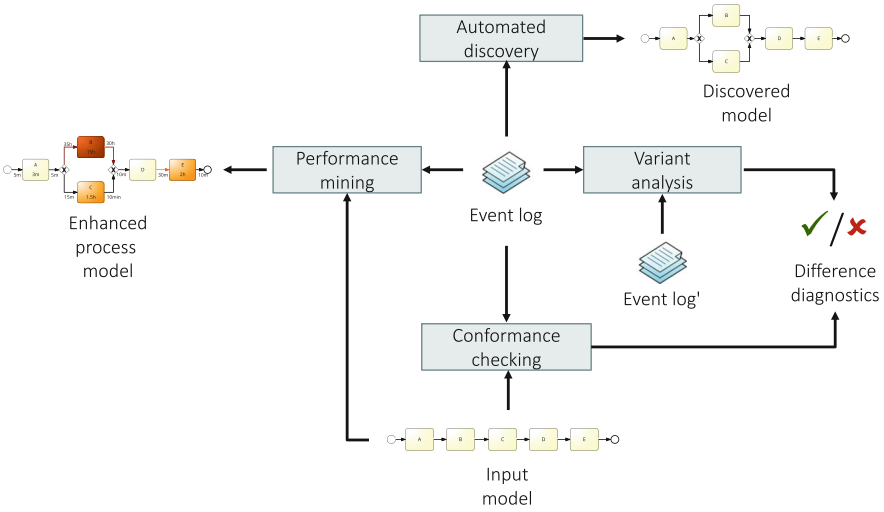


Fig. 11.3 Categories of process mining techniques and their inputs and output

of the health of a process, process mining techniques allow them to dig deeper. Specifically, these techniques allow us to understand how the process is actually being executed and to decorticate the performance of a process down to the level of individual tasks, resources, and handoffs.

In this section, we provide an overview of process mining techniques and we describe the structure of the event logs that are taken as input by these techniques.

11.3.1 Process Mining Techniques

As depicted in Figure 11.3, process mining techniques can be classified according to four use cases: (i) automated process discovery, (ii) conformance checking, (iii) performance mining, and (iv) variants analysis.

Automated process discovery techniques take as input an event log and produce a business process model that closely matches the behavior observed in the event log or implied by the traces in the event log. Automated process discovery can be used as part of a process discovery effort, in conjunction with other discovery methods presented in Chapter 5, or as part of tactical process performance monitoring efforts in conjunction with performance mining techniques as discussed below.

Conformance checking techniques take as input an event log and a process model, and they give us as output a list of differences between the process model and the event log. For example, the process model might tell us that after executing a task called A, we must execute a task called B. But maybe in the event log, sometimes after task A, we do not see task B. This might be because of an error, or more often

than not, it is because of an exception that is not captured in the process model. Some conformance checking techniques take as input an event log and a set of business rules instead of a process model. These techniques check if the event log fulfills the business rules, and if not, they show the violations of these rules.

Performance mining techniques take as input a process model and an event log and produce as output an *enhanced process model*. The enhanced process model contains additional elements (e.g., color-coded elements) that can answer questions like: "Why is the process slow?", or "Where do we waste the largest amount of time in a business process?" In other words, they can pinpoint the bottlenecks that slow down the process. Note that the process model used for performance mining might be a process model provided by the analyst or an automatically-discovered one.

Finally, *variants analysis* techniques take as input two event logs (corresponding to two variants of a process) and produce as output a list of differences. Typically, one of the event logs contains all the cases that end up in a positive outcome according to some criterion, while the other log contains all the cases that end up in a negative outcome. For example, the first log may contain all cases where the customer was satisfied, while the second one contains all cases that led to a complaint. Or the first log may contain all cases where the process completed on time, while the second one contains the delayed cases. Variants analysis techniques help their users to diagnose the reasons why an execution of a business process sometimes does not end up in a desirable outcome.

As Figure 11.3 shows, these four categories of process mining techniques take as input event logs. Accordingly, before discussing each category of techniques, we will present the structure of an event log and the challenges that need to be overcome to obtain those event logs in the first place.

11.3.2 Event Logs

When a process is executed on a BPMS, on an enterprise system (CRM, ERP), or on another supporting system, the system usually takes care of coordinating individual cases and informing participants about which tasks they need to work on. Participants often see only those tasks that they are directly responsible for, while the system hides the complexity of the overall process. Each participant typically has a personal *worklist* that shows the set of *work items* that need to be completed. If an explicit process model exists, each work item corresponds to a task in the process model. There might exist multiple work items corresponding to a single task if several cases are currently being worked on. For example, Chuck (as a process participant) might see that four work items are in his worklist, all relating to the "Confirm order" task of the order-to-cash process: one work item relates to an order from customer A, one from customer B, and two from customer C.

The structure of a work item is defined in the executable process model or directly implemented in the software. This means that participants see those data fields that have been declared as input for a task. For each work item they are working on,

they are supposed to document at least the completion. In this way, the system can keep track of the state of the process at any point in time. Among others, it is easy to record at what point in time somebody has started working on a work item, what input data was available, what output data was created, and who was the participant working on it. For example, when Chuck has confirmed the order of customer B, he enters the result in the system, and the system can decide if next the invoice should be emitted or the order confirmation should be escalated to someone above Chuck.

Most BPMSs and also other enterprise systems record events corresponding to the execution of work items and other relevant events such as the receipt of a message related to a given case of a process. These event records can be extracted from the database of the BPMS or enterprise system and represented as an *event log*.

An *event log* is a collection of timestamped event records. Each event record tells us something about the execution of a work item (and hence a task) of the process (e.g., that a task has started or has been completed), or it tells us that a given message event, escalation event, or other relevant event has occurred in the context of a given case in the process. For example, an event record in an event log may capture the fact that Chuck has confirmed a given purchase order at a given point in time.

Figure 11.4 illustrates what data is typically stored in an event log. We can see that a single event has a unique event ID. Furthermore, it refers to one individual case, it has a timestamp, and it shows which resources executed which task. These may be participants (e.g., Chuck and Susi) or software systems (SYS1, SYS2, DMS). For most mining techniques discussed in this chapter, it is a minimum requirement that for each event in the log we have three attributes: (i) in which case the event occurred (*case identifier*); (ii) which task the event refers to (herein called the *event class*); and (iii) when the event occurred (i.e., a *timestamp*). In practice, there may be other event attributes, such as for example the resource who performed a given task, the cost, or domain-specific data such as the loan amount in the case of a loan origination process.

The event log of Figure 11.4 is captured as a list in a tabular format.[7] Simple event logs such as this one are commonly represented as tables and stored in a Comma-Separated-Values (CSV) format. However, in more complex event logs, where the events have data attributes (e.g., the amount of a loan application, the shipping address of a purchase order), a flat CSV file is not a suitable representation.

A more versatile file format for storing and exchanging event logs is the *eXtensible Event Stream* (XES) format[8] standardized by the *IEEE Task Force on Process Mining*.[9] The majority of process mining tools can handle event logs in XES. The structure of an XES file is based on a data model, partially depicted in Figure 11.5. An XES file represents an event log. It contains multiple traces, and each trace can contain multiple events. All of them can contain different attributes.

[7] For simplicity, we only consider one supplier in this example.

[8] http://www.xes-standard.org.

[9] http://www.win.tue.nl/ieeetfpm.

CaseID	EventID	Timestamp	Activity	Resource
1	Ch-4680555556-1	2012-07-30 11:14	Check stock availability	SYS1
1	Re-5972222222-1	2012-07-30 14:20	Retrieve prod. from warehouse	Rick
1	Co-6319444444-1	2012-07-30 15:10	Confirm order	Chuck
1	Ge-6402777778-1	2012-07-30 15:22	Get shipping address	SYS2
1	Em-6555555556-1	2012-07-30 15:44	Emit invoice	SYS2
1	Re-4180555556-1	2012-08-04 10:02	Receive payment	SYS2
1	Sh-4659722222-1	2012-08-05 11:11	Ship product	Susi
1	Ar-3833333333-1	2012-08-06 09:12	Archive order	DMS
2	Ch-4055555556-2	2012-08-01 09:44	Check stock availability	SYS1
2	Ch-4208333333-2	2012-08-01 10:06	Check materials availability	SYS1
2	Re-4666666667-2	2012-08-01 11:12	Request raw materials	Ringo
2	Ob-3263888889-2	2012-08-03 07:50	Obtain raw materials	Olaf
2	Ma-6131944444-2	2012-08-04 14:43	Manufacture product	SYS1
2	Co-6187615741-2	2012-08-04 14:51	Confirm order	Conny
2	Em-6388888889-2	2012-08-04 15:20	Emit invoice	SYS2
2	Ge-6439814815-2	2012-08-04 15:27	Get shipping address	SYS2
2	Sh-7277777778-2	2012-08-04 17:28	Ship product	Sara
2	Re-3611111111-2	2012-08-07 08:40	Receive payment	SYS2
2	Ar-3680555556-2	2012-08-07 08:50	Archive order	DMS
3	Ch-4208333333-3	2012-08-02 10:06	Check stock availability	SYS1
3	Ch-4243055556-3	2012-08-02 10:11	Check materials availability	SYS1
3	Ma-6694444444-3	2012-08-02 16:04	Manufacture product	SYS1
3	Co-6751157407-3	2012-08-02 16:12	Confirm order	Chuck
3	Em-6895833333-3	2012-08-02 16:33	Emit invoice	SYS2
3	Sh-7013888889-3	2012-08-02 16:50	Get shipping address	SYS2
3	Ge-7069444444-3	2012-08-02 16:58	Ship product	Emil
3	Re-4305555556-3	2012-08-06 10:20	Receive payment	SYS2
3	Ar-4340277778-3	2012-08-06 10:25	Archive order	DMS
4	Ch-3409722222-4	2012-08-04 08:11	Check stock availability	SYS1
4	Re-5000115741-4	2012-08-04 12:00	Retrieve prod. from warehouse	SYS1
4	Co-5041898148-4	2012-08-04 12:06	Confirm order	Hans
4	Ge-5223148148-4	2012-08-04 12:32	Get shipping address	SYS2
4	Em-4034837963-4	2012-08-08 09:41	Emit invoice	SYS2
4	Re-4180555556-4	2012-08-08 10:02	Receive payment	SYS2
4	Sh-5715277778-4	2012-08-08 13:43	Ship product	Susi
4	Ar-5888888889-4	2012-08-08 14:08	Archive order	DMS

Fig. 11.4 Example of an event log for the order-to-cash process

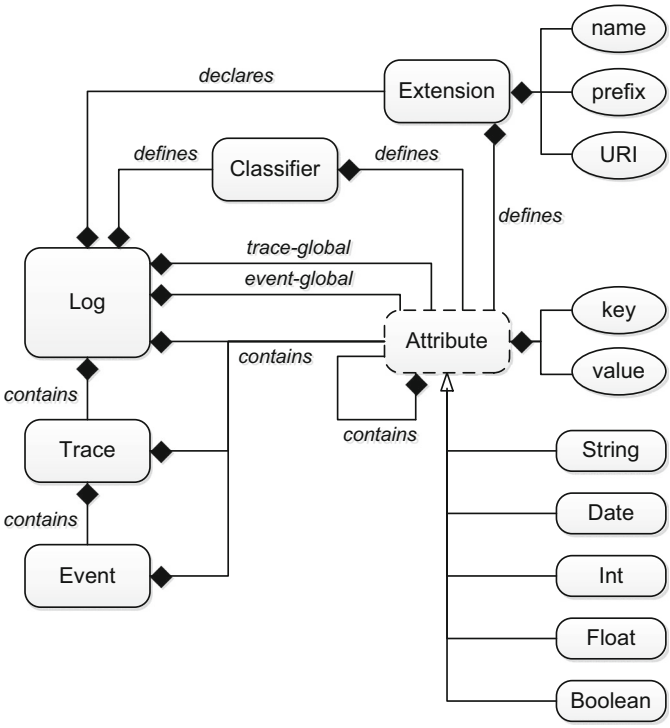


Fig. 11.5 Metamodel of the XES format

An attribute has to be a string, date, int, float, or boolean element as a key-value pair. Attributes have to refer to a global definition. There are two global elements in the XES file: one for defining trace attributes, the other for defining event attributes. Several classifiers can be defined in XES. A classifier maps one of more attributes of an event to a label that is used in the output of a process mining tool. In this way, for instance, events can be associated with tasks.

Figure 11.6 shows an extract of an XES log corresponding to the event log in Figure 11.4. The first global element (scope="trace") in this log tells us that every trace element should have a child element called "concept:name". This sub-element will be used to store the case identifier. In this example, there is one trace that has a value of 1 for this sub-element. At the level of individual events, there are three expected elements (i.e., "global" element with scope="event"): "concept:name", "time:timestamp", and "resource". The "concept:name" sub-element of an event will be used to store the name of the task to which the event refers. The "time:timestamp" and "resource" attributes give us the time of occurrence of the event and who performed it, respectively. The rest of the sample XES log represents one trace with two events. The first one refers to "Check stock availability", which

```
<log xes.version="1.0" xes.features="arbitrary-depth" xmlns="http://.../xes">
    <extension name="Concept" prefix="concept" uri="http://.../xes/concept.xesext"/>
    <extension name="Time" prefix="time" uri="http://.../xes/time.xesext"/>
    <global scope="trace">
        <string key="concept:name" value=""/>
    </global>
    <global scope="event">
        <string key="concept:name" value=""/>
        <date key="time:timestamp" value="1970-01-01T00:00:00.000+00:00"/>
        <string key="resource" value=""/>
    </global>
    <classifier name="Activity" keys="concept:name"/>
    <float key="log attribute" value="2335.23"/>
    <trace>
        <string key="concept:name" value="1"/>
        <event>
            <string key="concept:name" value="Check stock availability"/>
            <date key="time:timestamp" value="2012-07-30T11:14:00:000+01:00"/>
            <string key="resource" value=" SYS1 "/>
        </event>
        <event>
            <string key="concept:name" value="Retrieve product from warehouse"/>
            <date key="time:timestamp" value="2012-07-30T14:20:00:000+01:00"/>
            <string key="resource" value="Rick"/>
        </event>
    </trace>
</log>
```

Fig. 11.6 Example of a file in the XES format

was completed by SYS1 on the 30th of July 2012 at 11:14. The second event captures "Retrieve product from warehouse" conducted by Rick at 14:20.

Event data that is available in tabular format as in Figure 11.4 can be readily converted to XES. In many cases though, the data which is relevant for event logs is not directly accessible in the required format, but has to be extracted from different sources and integrated. This extraction and integration effort is generally not trivial. We can identify four major challenges for log data extraction, potentially requiring considerable effort in a project:

1. **Correlation challenge:** This refers to the problem of identifying the case an event belongs to. Many enterprise systems do not have an explicit notion of process defined. Therefore, we have to investigate which attribute of process-related entities might serve as a case identifier. Often, it is possible to utilize entity identifiers such as order number, invoice number, or shipment number.

2. **Timestamping challenge:** The challenge to work properly with timestamps stems from the fact that many systems do not consider logging as a primary task. This means that logging is often delayed until the system has idle time or little load. Therefore, we might find sequential events with the same timestamp in the log. This problem is worsened when logs from different systems potentially operating in different time zones have to be integrated. Partially, such problems can be resolved with domain knowledge, for example when events are known to always occur in a specific order.
3. **Longevity challenge:** For long running processes (e.g., processes with cycle times in the order of weeks or months), we might not yet have observed all cases that started during a recent time period (e.g., cases that started in the past 6 months) up to their completion. In other words, some cases might still be incomplete. Mixing these incomplete cases with completed cases can lead to a distorted picture of the process. For example, the cycle times or other performance measures calculated over incomplete cases should not be mixed with those of the completed ones, as they have different meaning. In order to avoid such distortions, we should consider the option of excluding incomplete cases from an event log. Also, we should ensure that the time period covered by an event log is significantly longer than the mean cycle time of the process, so that the event log contains samples of both short-lived and long-lived cases.
4. **Granularity challenge:** Typically, we are interested in conducting event log analysis on a conceptual level for which we have process models defined. In general, the granularity of event log recording is much finer such that each task of a process model might map to a set of events. For example, a task like "Retrieve product from warehouse" on the abstraction level of a process model maps to a series of events like "Work item #1,211 assigned", "Work item #1,211 started", "Purchase order form opened", "Product retrieved", and "Work item #1,211 completed". Often, fine-granular events many show up repeatedly in the logs while on an abstract level only a single task is executed. Therefore, it is difficult to define a precise mapping between the two levels of abstraction.

Exercise 11.4 Consider the final assembly process of Airbus for their A380 series. The final assembly of this aircraft series is located at the production site in Toulouse, France. Large parts are brought by ship to Bordeaux and brought to Toulouse by waterway and road transport. What is the challenge when log data of the A380 production process has to be integrated?

Once we have overcome the above data quality challenges, we are able to produce an event log consisting of *traces*, such that each trace contains the perfectly ordered sequence of events observed for a given case, each event refers to a task, and every task in the process is included in the event log. An event log with these properties can be re-written in a simplified representation, called a *workflow log*. A workflow log is a set of distinct traces, such that each trace consists of a sequence of symbols and each symbol represents an execution of a task. For example, given a set of tasks represented by symbols $\{a, b, c, d\}$, a possible workflow log is $\{\langle a, b, c, d\rangle, \langle a, c, b, d\rangle, \langle a, c, d\rangle\}$.

Case ID	Event ID	Timestamp	Activity
1	Ch-468	2012-07-30 11:14	Check stock availability
1	Re-597	2012-07-30 14:20	Retrieve product from warehouse
1	Co-631	2012-07-30 15:10	Confirm order
1	Ge-640	2012-07-30 15:22	Get shipping address
1	Em-655	2012-07-30 15:44	Emit invoice
1	Re-418	2012-08-04 10:02	Receive payment
1	Sh-465	2012-08-05 11:11	Ship product
1	Ar-383	2012-08-06 09:12	Archive order
2	Ch-405	2012-08-01 09:44	Check stock availability
2	Ch-420	2012-08-01 10:06	Check materials availability
2	Re-466	2012-08-01 11:12	Request raw materials
2	Ob-326	2012-08-03 07:50	Obtain raw materials
2	Ma-613	2012-08-04 14:43	Manufacture product
2	Co-618	2012-08-04 14:51	Confirm order
2	Em-638	2012-08-04 15:20	Emit invoice
2	Ge-643	2012-08-04 15:27	Get shipping address
2	Sh-727	2012-08-04 17:28	Ship product
2	Re-361	2012-08-07 08:40	Receive payment
2	Ar-368	2012-08-07 08:50	Archive order

Letter	Actvities
a	Check stock availability
b	Retrieve product from warehouse
c	Check materials availability
d	Request raw materials
e	Obtain raw materials
f	Manufacture product
g	Confirm order
h	Get shipping address
i	Ship product
j	Emit invoice
k	Receive payment
l	Archive order

Workflow Log
a,b,g,h,j,k,i,l
a,c,d,e,f,g,j,h,i,k,l
...

Fig. 11.7 Definition of a workflow log

Figure 11.7 sketches how a workflow log can be constructed from an event log.

Exercise 11.5 Have a look at Figure 11.4 and finish translating this event log into a workflow log by following the mapping rules sketched in Figure 11.7. Note that in this figure we have omitted the delimiters $\langle$ and $\rangle$ at the start and end of each trace.

Sometimes, it is useful to know how many times each *distinct trace* is observed in the event log. This is the case for example for robust process discovery techniques, discussed later in this chapter. When a trace occurs multiple times in a workflow log, we will write the number of occurrences before the contents of the trace. For example, we will write $10 \times \langle a, b, c, g, e, h \rangle$ to refer to a distinct trace $\langle a, b, c, g, e, h \rangle$ occurring 10 times in a log.

11.4 Automated Process Discovery

An automated process discovery technique takes as input an event log and produces as output a process model that captures the behavior of the log in a representative way. By *representative*, we mean that the constructed process model should be able to replay the traces of the event log and that it should not be able to replay traces that are neither in the event log nor implied by the traces found in the event log.

There is a wide range of automated process discovery techniques. Below we present a few of them. We start with a technique that produces a simple albeit rather incomplete representation of how tasks in the process follow each other, namely the so-called *dependency graph*. We then introduce a simple technique to generate

a BPMN process model from an event log under very strong assumptions, which often do not hold in practice. This technique gives us an idea of how difficult it is to automatically discover BPMN process models from event logs. Finally, we introduce more sophisticated and robust techniques that are able to generate relatively readable process models from large, real-life event logs.

11.4.1 Dependency Graphs

Dependency graphs are a popular technique to visualize event logs. A *dependency graph* (also known as a *directly-follows graph*) is a graph where each node represents one event class (i.e., a task) and each arc represents a "directly follows" relation. An arc exists between two event classes A and B if there is at least one trace in which B comes immediately after A. The arcs in a dependency graph may be annotated with an integer indicating the number of times that B directly follows A (hereby called the *absolute frequency*). In some process mining tools, it is also possible to annotate each arc with a temporal delay, e.g., the average time between an occurrence of task A and an occurrence of task B, among all occurrences of B that immediately follow an occurrence of A. This temporal delay gives an indication of the waiting time between a given pair of tasks in the process.

Example 11.2 Figure 11.8 shows an event log (left) and a corresponding dependency graph (right). We observe that the arc from task a to task c has an absolute frequency of 20. This is because the event log has two distinct traces where c occurs immediately before a and each of these distinct traces occurs 10 times. □

Exercise 11.6 Write down the dependency graph corresponding to the workflow log resulting from Exercise 11.5.

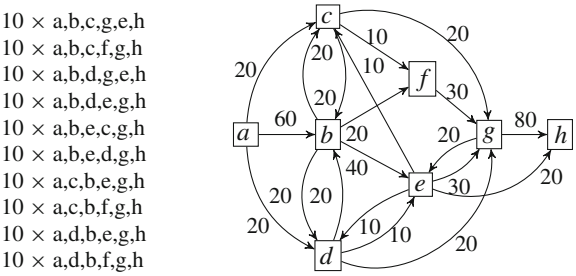


Fig. 11.8 Event log and corresponding dependency graph

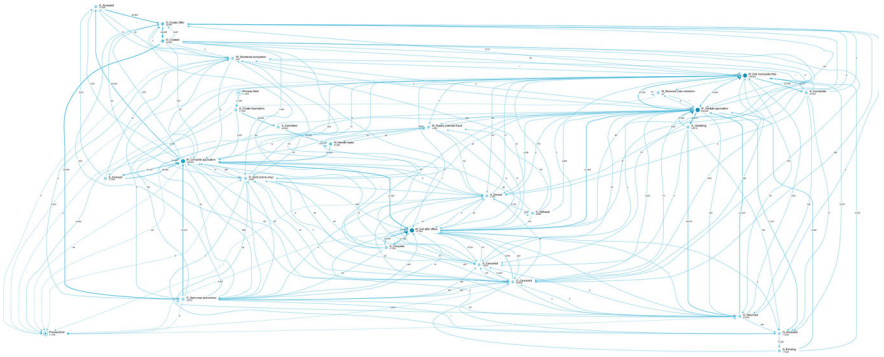
Owing to their simplicity, dependency graphs are supported by most commercial process mining tools, such as Celonis,[10] Disco,[11] Minit,[12] myInvenio,[13] and ProcessGold.[14] They are also supported by the so-called *fuzzy miner* plugin of ProM[15] and by Apromore,[16] two open-source process mining toolsets. All these tools provide visual cues to enhance the readability of dependency graphs. For example, the color and the thickness of the arcs may be used to encode how frequently a task in a pair directly follows the other (e.g., a thick and dark-blue arc may represent a frequent directly-follows relation relative to a thin, light-blue arc). These tools also offer visual zoom in operations that allow users to explore specific parts of a dependency graph in detail.

Besides their simplicity, one of the most appealing features of dependency graphs is that they are amenable to *abstraction* operations. In this context, abstraction refers to removing a subset of the nodes or arcs in a dependency graph in order to obtain a smaller dependency graph of a given event log. For example, process mining tools allow us to remove the most infrequent nodes or arcs from a dependency graph in order to obtain a simpler map that is easier to understand. Abstraction is an indispensable feature when we want to explore a real-life event log, as illustrated below.

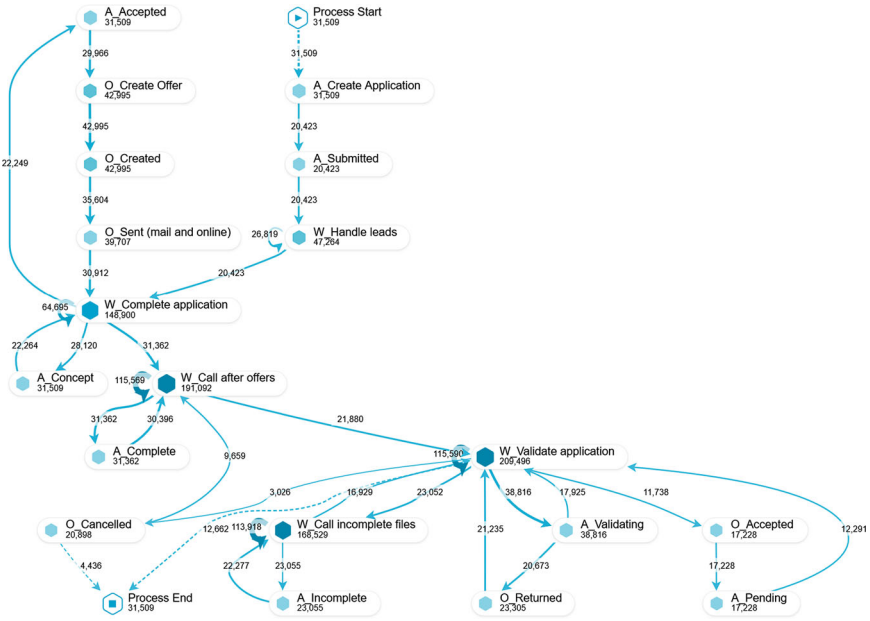
Example 11.3 We consider the event log of the Business Process Intelligence Challenge 2017 available at https://tinyurl.com/bpic2017. This is an event log of a loan origination process at a Dutch financial institution. The event log covers the application-to-approval and the offer-to-acceptance sub-processes of a loan origination process, meaning that it starts when a loan application is submitted by a customer and it ends when the customer accepts (or refuses) the corresponding loan offer. The full dependency graph produced by the Celonis Web-based tool on this event log is shown in Figure 11.9a. This map is too detailed to be understandable by a user. Even if we visually zoom-in to inspect specific parts of this map, it will be very difficult to understand what is going on in this process. For this reason, Celonis and other tools can abstract away the map by retaining only the most frequent arcs. Figure 11.9b shows the filtered map produced by Celonis when setting the task abstraction slider to 98% and the arc abstraction slider to 90%. □

While abstraction is a useful mechanism when visualizing large event logs, it is not sufficient to handle the full complexity of real-life event logs. Accordingly, process mining tools also offer a second type of simplification operation called *event log filtering*. Filtering an event log means removing a subset of the traces, events or

[10] http://www.celonis.com.
[11] https://fluxicon.com/disco.
[12] http://minitlabs.com.
[13] http://www.my-invenio.com.
[14] https://processgold.com/en.
[15] http://www.promtools.org.
[16] http://apromore.org.



(a)



(b)

Fig. 11.9 Example of a full dependency graph and an abstracted version thereof. (**a**) Full dependency graph. (**b**) Filtered dependency graph

pairs of events in order to obtain a simpler log. Note that filtering and abstraction have different inputs and outputs. Whereas filtering transforms an event log into a smaller log, abstraction only affects the dependency graph, not the log.

In general, process mining tools provide three types of filters:

- *Event filters*, which allow us to remove all events in a log that fulfill a condition or, conversely, to retain only those events that fulfill a condition. For example, in an event log of a procure-to-pay process, we can define a filter that removes all events that contain an event of event class "Amend purchase requisition". This operation returns a filtered event log that has the same set of traces as the original log, but some of the traces are slightly shorter because any occurrence of "Amend purchase requisition" is removed. The conditions used in a filter can involve multiple sub-conditions. For example, we can define an event filter to remove all occurrences of event "Amend purchase requisition" such that the value of attribute "reason" is equal to "SupplierChanged", meaning all purchase requisition amendments corresponding to a change of supplier. Similarly, we can define a filter that removes all the "Amend purchase requisition" events performed by a given resource (e.g., Rick).
- *Event pair filters*, which allow us to remove all pairs of events in a log that fulfill a condition, or to retain only those pairs of events that fulfill a condition. For example, we can define a filter that only retains event pairs (e1, e2) in each trace, such that e2 appears after e1, and such that e1 corresponds to task "Amend purchase requisition" and e2 corresponds to task "Approve purchase requisition amendment".
- *Trace filters*, which allow us to remove all traces in a log that fulfill a condition, or retain only those traces that fulfill a condition. For example, we can remove all traces where an event corresponding to a given task occurs (e.g., remove all traces where event "Amend purchase requisition" occurs). We can also use temporal conditions here, such as for example removing all traces that have a cycle time of less than 20 days. Note that here we are removing entire traces and not just individual events in a trace.

Exercise 11.7 Using a process mining tool of your choice, answer the following questions with reference to the event log of the Business Process Intelligence Challenge 2017 (https://tinyurl.com/bpic2017):

A. How many cases are represented in this event log in total?
B. What is the mean cycle time of the cases captured in this event log?
C. Let us say that a case is successful if its trace contains at least one occurrence of event O_Accepted, meaning that the customer accepted the loan offer. How many cases are successful? What is the mean cycle time of the successful cases?
D. Let us say that a case is unsuccessful if its trace contains at least one occurrence of event O_Refused. How many cases are unsuccessful? What is the mean cycle time of the unsuccessful cases?
E. Let us say that a case is canceled if the last event in its trace is O_Cancelled. Note that a trace may contain O_Cancelled, but if it contains other events after

O_Cancelled then we will not treat it as canceled because it means that the case continued despite the occurrence of a cancelation event. How many cases were canceled? What is the mean cycle time of the canceled cases?

While dependency graphs are a useful log visualization approach, especially when used in combination with filtering and abstraction operations, they do not allow us to understand in detail what is happening at each point in the process. In particular, they do not allow us to easily distinguish whether two tasks are in parallel, in a loop, or in some other relation, as a BPMN process model would do. If our goal is to understand in detail how the process is executed, we are much better off with a BPMN process model. Fortunately, there are a number of techniques to automatically discover a BPMN process model from an event log. Below, we present first a simple but limited technique, namely the α-algorithm, and we then discuss other more sophisticated techniques that can discover readable process models from large event logs.

11.4.2 The α-Algorithm

The *α-algorithm* is a basic algorithm for discovering process models from event logs. The algorithm is simple enough that we can easily grasp how it works in detail. However, it makes a strong assumption about the event log, namely *behavioral completeness*. An event log is behaviorally complete if whenever a task a can be directly followed by a task b in the actual process, then there is at least one case in the event log where we observe ab. In practice, we can hardly assume that we find the complete set of behavioral options in a log. Advanced techniques, which we will discuss later, try to lift this assumption by trying to guess which traces the process might have, which are not present in the event log.

The α-algorithm constructs a process model from a behaviorally complete event log in two phases. In the first phase, a set of *order relations* between pairs of events are extracted from the workflow log. In the second phase, the process model is constructed in a stepwise fashion from these identified relations. The *order* relations refer to pairs of tasks which directly follow one another in the log. They provide the basis for the definition of three more specific relations that refer to *causality*, to potential *parallelism* and to *no-direct-succession*. We refer to this set of relations as the α relations.

- The basic directly-follows relation $a > b$ holds if we can observe in the workflow log L that a task a is directly followed by b. This is the same relation as that captured in a dependency graph.
- The causality relation $a \rightarrow b$ is derived from the directly-follows relation. It holds if we observe in L that $a > b$ and that $b \not> a$.
- The relation of potential parallelism $a||b$ holds if both $a > b$ and $b > a$ are observed in the workflow log L.
- The relation of no-direct-succession $a\#b$ holds if $a \not> b$ and $b \not> a$.

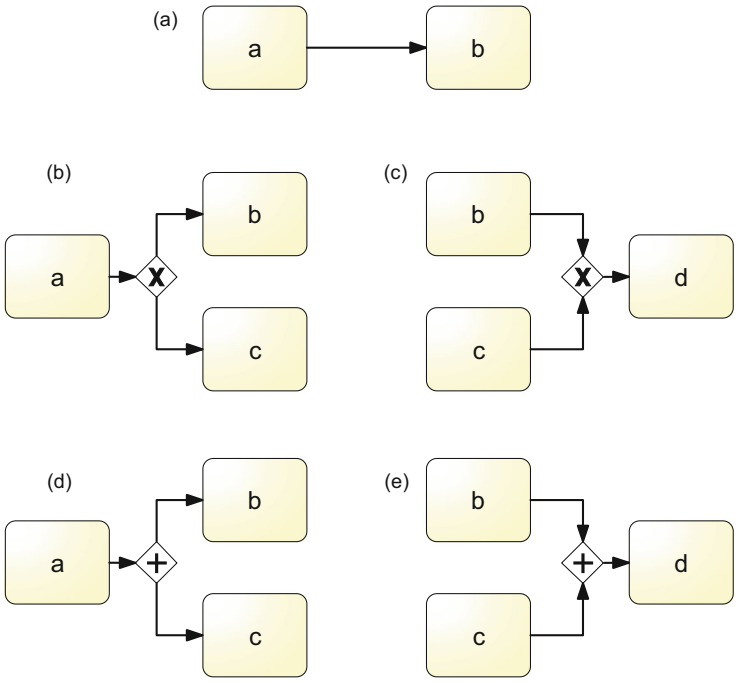


Fig. 11.10 Simple control flow patterns

The reason why exactly these relations are used is illustrated in Figure 11.10. There exist five characteristic combinations of relations between the tasks in a workflow log that can be mapped to simple control flow patterns.

Pattern (a) depicts a sequence of tasks a and b. If we model them in this way, it should be guaranteed that in a workflow log we will find a followed by b, i.e., $a > b$, but never b followed by a, i.e., $b \not> a$. This means that the causality relation $a \rightarrow b$ should hold.

Pattern (b) also relates to a characteristic combination of relations. The workflow log should show that $a \rightarrow b$ and $a \rightarrow c$ hold, and that b and c would not be mutual successors, i.e., $b \# c$.

Pattern (c) also requires that b and c are not mutual successors, i.e., $b \# c$, while both $b \rightarrow d$ and $c \rightarrow d$ have to hold.

Pattern (d) demands that $a \rightarrow b$ and $a \rightarrow c$ hold, and that b and c show potential parallelism, i.e., $b || c$.

Pattern (e) refers to $b \rightarrow d$ and $c \rightarrow d$ while b and c show potential parallelism, i.e., $b || c$.

The idea of the α-algorithm is to identify the relations between all pairs of tasks from the workflow log in order to reconstruct a process model based on Patterns (a) to (e). Therefore, before applying the α-algorithm, we must extract all basic

order relations from the workflow log L. Consider the workflow log depicted in Figure 11.7 containing the two cases $\langle a, b, g, h, j, k, i, l\rangle$ and $\langle a, c, d, e, f, g, j, h, i, k, l\rangle$. From this workflow log, we can derive the following relations.

- The basic order relations $>$ refer to pairs of tasks in which one task directly follows the another. These relations can be directly read from the log:

$a > b$	$h > j$	$i > l$	$d > e$	$g > j$	$i > k$
$b > g$	$j > k$	$a > c$	$e > f$	$j > h$	$k > l$
$g > h$	$k > i$	$c > d$	$f > g$	$h > i$	

- The causal relations can be found when an order relation does not hold in the opposite direction. This is the case for all pairs except (h, j) and (i, k), and their opposites. We get:

$a \to b$	$j \to k$	$a \to c$	$d \to e$	$f \to g$	$h \to i$
$b \to g$	$i \to l$	$c \to d$	$e \to f$	$g \to j$	$k \to l$
$g \to h$					

- The potential parallelism relation holds true for $h||j$ as well as for $k||i$ (and the corresponding symmetric cases).
- The remaining relation of no-direct-succession can be found for all pairs that do not belong to $\to$ and $||$. It can be easily derived when we write down the relations in a matrix as shown in Figure 11.11. This matrix is also referred to as the *footprint matrix* of the log.

Exercise 11.8 Have a look at the workflow log you constructed in Exercise 11.5 (page 427). Define the relations $>, \to, ||, \#$, as well as the footprint matrix of this workflow log.

The α-algorithm takes an event log L and its α relations as a starting point. The essential idea of the algorithm is that whenever a task directly follows another one in the log, the two tasks should be directly connected in the process model. Furthermore, if there is more than one task that can follow another, we have to determine whether the set of succeeding tasks is partially exclusive or concurrent. An exception from the principle that tasks should be connected in the process model is when two tasks are potentially parallel, i.e., those pairs included in $||$. The details of the α-algorithm are defined according to the following eight steps.[17]

1. Let T_L be the set of all tasks in the log.
2. Let T_I be the set of tasks that appear at least once as first task of a case.

[17]Note that the α-algorithm was originally defined for constructing Petri nets. The version shown here is a simplification based on the five simple control-flow patterns of Figure 11.10, in order to construct BPMN models.

	a	b	c	d	e	f	g	h	i	j	k	l
a	#	→	→	#	#	#	#	#	#	#	#	#
b	←	#	#	#	#	#	→	#	#	#	#	#
c	←	#	#	→	#	#	#	#	#	#	#	#
d	#	#	←	#	→	#	#	#	#	#	#	#
e	#	#	#	←	#	→	#	#	#	#	#	#
f	#	#	#	#	←	#	→	#	#	#	#	#
g	#	←	#	#	#	←	#	→	#	→	#	#
h	#	#	#	#	#	#	←	#	→	‖	#	#
i	#	#	#	#	#	#	#	←	#	#	‖	→
j	#	#	#	#	#	#	←	‖	#	#	→	#
k	#	#	#	#	#	#	#	#	‖	←	#	→
l	#	#	#	#	#	#	#	#	←	#	←	#

Fig. 11.11 Footprint represented as a matrix of the workflow log $L = [\langle a, b, g, h, j, k, i, l\rangle, \langle a, c, d, e, f, g, j, h, i, k, l\rangle]$

3. Let T_O be the set of tasks that appear at least once as last task in a case.
4. Let X_L be the set of potential task connections. X_L is composed of:
   a. Pattern (a): all pairs for which $a \rightarrow b$ holds.
   b. Pattern (b): all triples for which $a \rightarrow (b \# c)$ holds.
   c. Pattern (c): all triples for which $(b \# c) \rightarrow d$ holds.

   Note that triples for which Pattern (d) $a \rightarrow (b||c)$ or Pattern (e) $(b||c) \rightarrow d$ hold are not included in X_L.
5. Construct the set Y_L as a subset of X_L by:
   a. Eliminating $a \rightarrow b$ and $a \rightarrow c$ if there exists some $a \rightarrow (b \# c)$.
   b. Eliminating $b \rightarrow c$ and $b \rightarrow d$ if there exists some $(b \# c) \rightarrow d$.
6. Connect start and end events in the following way:
   a. If there are multiple tasks in the set T_I of first tasks, then draw a start event leading to an XOR-split, which connects to every task in T_I. Otherwise, directly connect the start event with the first task.
   b. For each task in the set T_O of last tasks, add an end event and draw an arc from the task to the end event.
7. Construct the flow arcs in the following way:
   a. Pattern (a): for each $a \rightarrow b$ in Y_L, draw an arc a to b.
   b. Pattern (b): for each $a \rightarrow (b \# c)$ in Y_L, draw an arc from a to an XOR-split, and from there to b and c.

c. Pattern (c): for each $(b\#c) \rightarrow d$ in Y_L, draw an arc from b and c to an XOR-join, and from there to d.
d. Pattern (d) and (e): if a task in the so-constructed process model has multiple incoming or multiple outgoing arcs, bundle these arcs with an AND-join or AND-split, respectively.

8. Return the newly constructed process model.

Let us apply the α-algorithm to $L = [\langle a, b, g, h, j, k, i, l\rangle, \langle a, c, d, e, f, g, j, h, i, k, l\rangle]$. Steps 1–3 identify $T_L = \{a, b, c, d, e, f, g, h, i, j, k, l\}$, $T_I = \{a\}$, and $T_O = \{l\}$. In Step 4a all causal relations are added to X_L including $a \rightarrow b$,$a \rightarrow c$, etc. In Step 4b, we work row by row through the footprint matrix of Figure 11.11 and check if there are cells sharing a $\rightarrow$ relation while relating to tasks that are pairwise in $\#$. In the row a, we observe both $a \rightarrow b$ and $a \rightarrow c$. Also, $b\#c$ holds. Therefore, we add $a \rightarrow (b\#c)$ to X_L. We also consider row g and its relation to h and j. However, as $h||j$ holds, we do not add them. In Step 4c, we progress column by column through the footprint matrix and see if there are cells sharing a $\rightarrow$ relation while relating to tasks that are mutually in $\#$. In column g, we observe two $\rightarrow$ relations to b and f. Also, $b\#f$ holds. Accordingly, we add $(b\#f) \rightarrow g$ to X_L. We also check i and k that share the same relation to l. However, as $i||k$ holds, we do not add them. There are no further complex combinations found in Step 4d.

In Step 5, we eliminate the basic elements in X_L that are covered by the complex patterns found in Steps 4b and 4c. Accordingly, we delete $a \rightarrow b$, $a \rightarrow c$, $b \rightarrow g$, and $f \rightarrow g$. In Step 6a we introduce a start event and connect it with a; in 6b, task l is connected with an end event. In Step 7, arcs and gateways are added for the elements of Y_L. Finally, in Step 8 the resulting process model is returned, as shown in Figure 11.12.

Exercise 11.9 Consider the workflow log and the footprint you constructed in Exercises 11.5 (page 427) and 11.8 (page 434). Show step-by-step how the α-algorithm works on this workflow log, and draw the resulting process model.

11.4.3 Robust Process Discovery

The α-algorithm can reconstruct a process model from a behaviorally complete event log if that log has been generated from a structured process model. There are also limitations to be noted, though. The α-algorithm is not able to distinguish so-called *short loops* from true parallelism. As can be seen in Figure 11.13, all three models can produce a workflow log that yields $b||c$ in the corresponding footprint. Several extensions to the α-algorithm have been proposed. The idea of the $\alpha+$-algorithm is to define the relation $||$ in a stricter way such that $b||c$ is only included if there is no sequence bcb in the log. In this way, models (b) and (c) in Figure 11.13 can be distinguished from model (a) in their generated logs. Furthermore, we can use pre-processing to extract direct repetitions like aa or bb from a log, note down

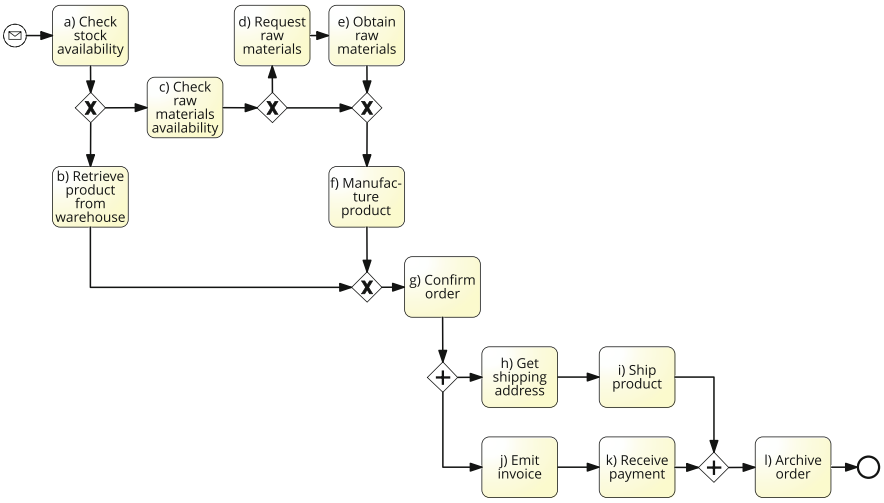


Fig. 11.12 Process model constructed by the α-algorithm from log $L = [\langle a, b, g, h, j, k, i, l\rangle, \langle a, c, d, e, f, g, j, h, i, k, l\rangle]$

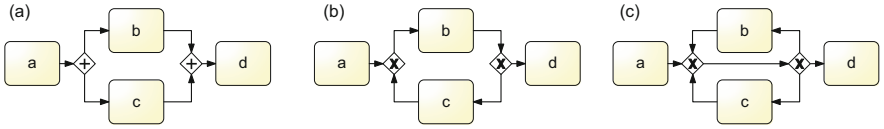


Fig. 11.13 Examples of two short loops, (**b**) and (**c**), that cannot be distinguished from model (**a**) by the α-algorithm

the corresponding tasks, and continue with a log from which such repeated behavior is mapped to a single execution.

Another limitation of the α-algorithm is its inability to deal with *incompleteness* in an event log. The α-algorithm assumes that the $>$ relation (from which the other relations are derived) is complete, meaning that if the process allows task a to be directly followed by task b, then the relation $a > b$ must be observed in at least one trace of the log. This assumption is too strong for processes that have a lot of parallelism. For example, if in a process there is a block of ten concurrent tasks $a_1, \ldots a_{10}$, we need to observe each relation $a_i > a_j$ for each $i, j \in [1..10]$, meaning 100 possible combinations. It suffices that one of these combinations has not been observed in the event log for the α-algorithm to discover the wrong model. This example shows that we need more robust process discovery algorithms, which can smartly infer relations that are not explicitly observed in the event log, but that are somehow implied by what can be observed in the log.

A related limitation is the α-algorithm's inability to deal with *noise*. Event logs often include cases with a missing head, a missing tail, or a missing intermediate episode because some events in a case have not been recorded. For example, a worker might have forgotten to mark an invoice as *paid*, and hence the "Payment received" event is missing in the corresponding trace. Furthermore, there may be

logging errors leading to events being recorded in the wrong order (or with wrong timestamps) or recorded twice. Ideally, such noise should not distort the process model produced by a process discovery technique.

Fortunately, there are other more robust algorithms for automated process discovery, such as the *heuristics miner* [192], the *structured heuristics miner* [11], the *inductive miner* [86] and the *split miner* [10]. These techniques generally work as follows. First, they construct the dependency graph from the event log. Second, they remove some of the nodes and arcs in the process dependency graph in order to deal with noise (infrequent behavior). Finally, they apply a set of heuristic rules to discover split gateways and join gateways in order to turn the filtered dependency graph into a (BPMN) process model.

For example, to handle noise, the heuristics miner [192] uses a relative frequency metric between pairs of event labels, defined as $a \Rightarrow b = \left(\frac{|a>b|-|b>a|}{|a>b|+|b>a|+1}\right)$, where $|a > b|$ is the number of times when task a is directly followed by task b in the log. This metric has a value close to +1 when task a is directly followed by task b sometimes and task b is (almost) never directly followed by a. This means that there is a clear direct-precedence relation from a to b. When the value of $a \Rightarrow b$ is not close to 1 (for example when it is less than 0.8), it means that the direct precedence relation is not very clear (it might be noise) and hence it is deleted. Other similar frequency metrics are used to detect self loops and short loops in order to avoid the limitations of the α-algorithm.

Once the dependency graph has been filtered and self and short loops have been identified, the heuristics miner identifies split and join gateways by analyzing both the filtered dependency graph and the traces in the log. Specifically, if task a is directly followed by b and by c, but not by both (i.e., after b we do not observe c or we very rarely observe it), the heuristics miner will put an XOR-split between task a on the one hand, and tasks b and c on the other hand. Meanwhile, if a is generally followed by both b and by c, the heuristics miner will put an AND-split after task a and before b and c. The term *generally* here means that the heuristics miner tolerates some cases where a is not followed by b or by c if this happens infrequently. Join gateways are discovered similarly: An XOR-join is placed between tasks b and c on the one hand, and task d on the other hand, if d is generally preceded by either task b or c but not both, while an AND-join is placed there if d is generally preceded by both b and c.

Unlike the α-algorithm, the heuristics miner is able to detect self-loops and short-loops, and to handle incomplete and noisy event logs. However, when applied to large real-life event logs, the heuristics miner often produces process models that are too large, spaghetti-like and not sound (see Section 5.4.1 for a definition of soundness of process models).

As discussed in Section 5.4.3, a desirable property of process models is that they should be block-structured. Block-structured process models are always sound, and in addition, they are generally easier to understand than unstructured ones. Accordingly, some of the most robust process discovery techniques try to produce block-structured process models. This is the case for example of the inductive

miner and the structured heuristics miner. Like the heuristics miner, the inductive miner starts by constructing a dependency graph and filtering out infrequent arcs. But instead of immediately proceeding to identifying gateways from the filtered dependency graph, it first analyzes the filtered dependency graph in order to identify arcs that (if removed) would bisect the dependency graph into two separate parts. By doing this several times, it ends up dividing the dependency graph into blocks. Finally, it discovers the gateways locally within each block, in such a way that each split gateway has a corresponding join gateway of the same type, which is the key characteristic of block-structured process models.

The structured heuristics miner implements an alternative approach. It first applies the heuristics miner in order to discover a process model. This process model might be relatively complex (high number of gateways) and highly unstructured. To improve this model, the structured heuristics miner then applies an algorithm that converts unstructured process models into block-structured ones. In doing so, the algorithm also ensures that the resulting model is sound.

The split miner goes beyond the inductive miner and the structured heuristics miner by discovering process models that are sound, but not always perfectly block-structured. By allowing itself to discover non-block-structured models, the split miner technique discovers higher-quality models as discussed below.

The heuristics and inductive miners can be both found in ProM, while these two algorithms, as well as the structured heuristics miner and the split miner, can all be found in Apromore.

11.4.4 Quality Measures for Automated Process Discovery

Given the availability of several algorithms for automated discovery of process models, one may wonder which algorithm to choose for a given event log. A series of experiments reported in [12] suggests that the inductive miner and the split miner are among the most robust algorithms for automated process discovery. However, their relative performance on a given event log can vary and some tuning is needed. By *tuning*, we mean that the algorithm needs to be applied with different parameter settings. Each parameter setting leads to a different process model. Hence, we then need to select the best of them.

This raises the question of how to assess the quality of a process model discovered from an event log. The quality of automatically-discovered process models is usually evaluated using four criteria: *fitness*, *precision*, *generalization*, and *complexity* [1].

Fitness is the ability of the discovered process model to replay the behavior contained in the log. A fitness equal to 1 means that the model can replay every trace in the log. We will discuss later how traces are replayed against a log in order to compute a fitness measure.

Precision refers to the extent to which the discovered process model generates only the traces found in the log. A precision equal to 1 means that any trace produced

by the process model also appears in the log. If a process model can produce a trace that is not in the log, this reduces the precision. A precision of 0 would mean that none of the traces that the model produces is observed in the log.

Generalization refers to the extent to which the discovered process model captures traces that are not present in the log because the log is incomplete, but that are likely to be allowed by the underlying business process. It is tricky to measure generalization, because it is hard to know if a trace T belongs to a process, if all we have is an incomplete event log and trace T does not appear in this log. A standard trick to measure generalization is called k-fold cross-validation. The idea is to divide the log into k parts (for example five parts), to discover the model from $k-1$ parts (i.e., we hold out one part), and to measure the fitness of the discovered model against the hold-out part, and the precision of the discovered model against the entire log. This operation is repeated for every possible hold-out part, and the measures of fitness and precision obtained for every hold-out operation are averaged, leading to two measures called *k-fold fitness* and *k-fold precision* respectively. A k-fold fitness equal to 1 means that the discovered model produces all traces that are part of the observed process, even if those traces are not in the event log. Similarly, a k-fold precision close to 1 means that the discovered model does not over-generalize the process, i.e., it does not produce traces that are not part of the process.

Finally, the *complexity* of a process model quantifies how difficult it is to understand the model. Complexity can be measured simply in terms of size (number of nodes in a BPMN model). Usually, larger models are more complex and hence difficult to understand. However, two models can have the same size and one of them might be more difficult to understand because it contains too many gateways. Hence, it is common to also measure complexity of a BPMN process model using a metric called *Coefficient of Network Connectivity* (CNC), which is the number of flows in a BPMN model divided by the number of nodes. The higher the CNC is, the higher is the number of gateways relative to nodes, which makes the model more difficult to understand. Also, as discussed in Section 5.4.3, block-structured process models are easier to comprehend than unstructured ones. Accordingly, it is also common to measure the complexity of a process model via a measure called *structuredness*. The structuredness of a process model is the percentage of nodes located directly inside a well-structured single-entry single-exit fragment. A perfectly block-structured process model has a structuredness equal to 1, whereas a spaghetti-like process model (with arcs going all over the place) would have a low structuredness.

Example 11.4 We consider the event log available at: http://tinyurl.com/sampleSAPLog. The process model discovered using the heuristics miner implementation in ProM version 6 is shown in Figure 11.14a. This process model is not sound. For example, if task D is executed, the next task according to the model should be N. After task N is executed, a token is placed in the flow between task N and the AND gateway. There is no token anywhere else in the model. Hence, the other incoming flow of this AND gateway will never receive a token thereafter.

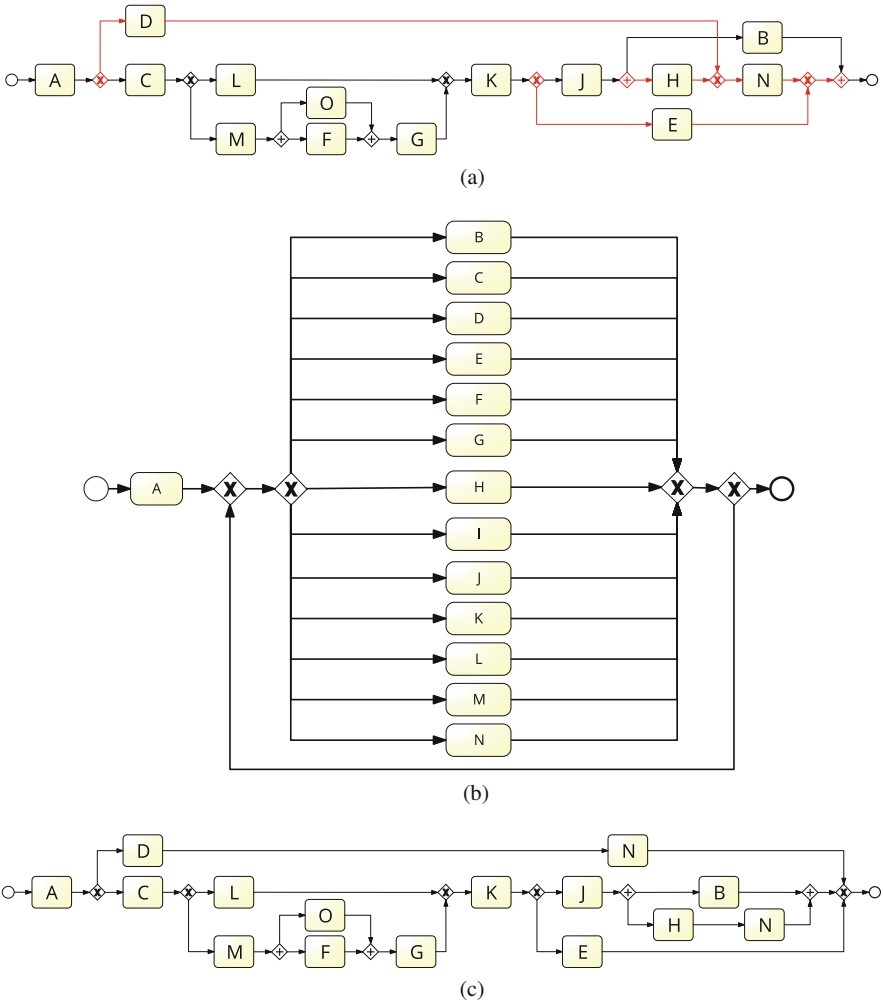


Fig. 11.14 Models discovered from a sample log using three discovery techniques . (**a**) Heuristics miner (ProM v6). (**b**) Inductive miner (ProM v6). (**c**) Structured heuristics miner (Apromore)

The process is hence in a deadlock. Similarly, observe that if after task K, task E is executed, the case ends up in a deadlock for the same reason as above.

The model discovered using the inductive miner (in ProM v6) for this same log is shown in Figure 11.14b. This model is block-structured and sound. However, the model is very unconstrained—it can produce too many traces. After executing task A, from the first XOR-split we can execute any task between B and N, after which the second XOR-split is reached (the one before the end event). Once this gateway has been reached, any of the tasks in the process can be executed, potentially again, via the loopback arc, or the case may end immediately. Therefore, this process

model allows tasks B-N to be executed in any order and any number of times. This type of control-flow structure is called a *flower pattern*. Because of this flower pattern, this process model can produce many traces that are not in the event log. Hence, the model has a low precision. On the other hand, the process model has perfect fitness. Indeed, given that it can accept almost any sequence of tasks A-N, it can in particular accept any of the traces that appear in the event log.

Finally, the model discovered using the structured heuristics miner is shown in Figure 11.14c.[18] This model is also block-structured and sound. We note that in this model there are two tasks labelled N. This is because the structured miner transforms the process model produced by the heuristics miner into a structured process model. And when an unstructured process model is transformed into an equivalent block-structured one, it is sometimes necessary to duplicate some of the tasks in the model. This process model allows less behavior than the one discovered by the inductive miner. It turns out that the fitness of this model with respect to the log is almost 1, the precision is 1, and the generalization is close to 1. □

Exercise 11.10 We consider an event log containing pathways of patients suffering from sepsis infection at a hospital: http://tinyurl.com/SepsisLog. Using one or more process mining tools, discover at least two process models from this event log using different automated process discovery techniques. Compare the discovered models in terms of complexity, fitness and precision.

11.5 Process Performance Mining

Event logs provide us with detailed data to quantify and visualize the performance of a process. In Section 8.1.3, we discussed the four performance dimensions that form the so-called Devil's Quadrangle: time, cost, quality, and flexibility. In this section, we will show how to use event logs to assess the performance of a process according to each of these dimensions.

11.5.1 Time Dimension

Time and its more specific measures *cycle time* and *waiting time* are important general performance measures. Event logs typically show timestamps such that they can be used for time analysis. Time analysis is concerned with the temporal occurrence and probabilities of different types of events. The event logs of a process generally relate each event to the point in time of its occurrence. Therefore, it is straightforward to plot events on the time axis. Furthermore, we can utilize

[18] The split miner produces this same model when applied to this log (with a parallelism threshold of 40%).

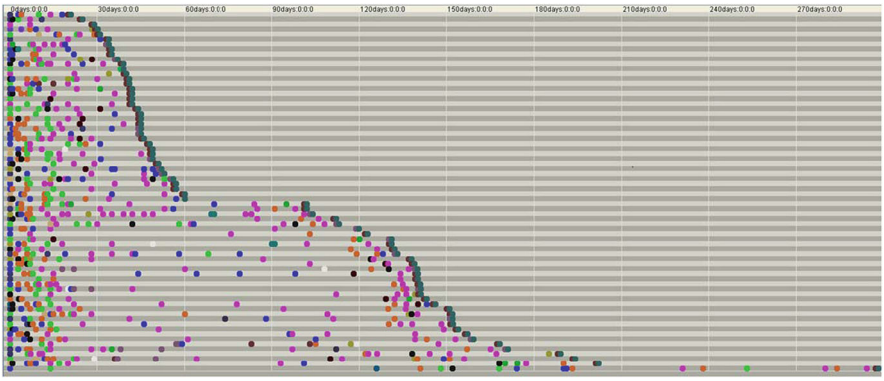


Fig. 11.15 Dotted chart of log data

classifiers to group events on a second axis. A classifier typically refers to one of the attributes of an event, like case ID or participant ID. There are two levels of detail for plotting events in a diagram: *dotted charts* using the timestamp to plot an event and *timeline chart* showing the duration (processing time) of a task and its waiting time.

The dotted chart is a simple, yet powerful visualization tool for event logs. Each event is plotted on a two-dimensional canvas with the first axis representing its occurrence in time and the second axis as its association with a classifier like a case ID. There are different options to organize the first axis. Time can be represented either *relative*, such that the first event is counted as zero, or *absolute* such that later cases with a later start event are further right in comparison to cases that started earlier. The second axis can be sorted according to different criteria. For instance, cases can be shown according to their historical order or their overall cycle time.

Figure 11.15 shows the dotted chart of the event log of a healthcare process. This chart has been produced using the ProM tool. The events are plotted according to their relative time and sorted according to their overall cycle time. It can be seen that there is a considerable variation in terms of cycle time. Furthermore, the chart suggests that there might be three distinct classes of cases: those that take no more than 60 days, those taking 60 to 210 days, and a small class of cases taking longer than 210 days. Such an explorative inspection can provide a good basis for a more detailed analysis of the factors influencing the cycle time.

Exercise 11.11 Draw a dotted chart of the event log in Figure 11.4 (page 423) showing relative cycle time and being sorted by cycle time.

If the event log contains timestamps capturing both the start and the end of each task, we can plot tasks not as dots, but as bars, as shown in Figure 11.16. This type of a visualization is called a *timeline chart*. A timeline chart shows the waiting time (from activation until starting) and the processing time (from starting until completion) for each task. The timeline chart is more informative than the dotted chart, since it shows waiting times and processing times separately, and hence it allows us to pinpoint bottlenecks in the process.

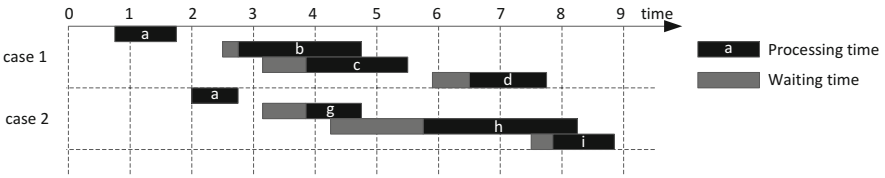


Fig. 11.16 Example of timeline chart

We saw in Section 11.4.1 that dependency graphs are a versatile approach to visualize event logs. In particular, we saw that dependency graphs can give us a picture of the most frequent paths in a process. Another useful property of dependency graphs is that they can be used to visualize temporal delays (e.g., waiting times or processing times) via color-coding or thickness-coding. Concretely, arcs in the dependency graph may be color-coded by the corresponding temporal delay between the source event and the target event of the arc. This visualization allows us to identify the longest waiting times in the process—also known as *bottlenecks*. Meanwhile, the nodes in a dependency graph can be color-coded with the processing times of each task. The latter requires that each task captured in the log have both a start timestamp and an end timestamp so that the processing time can be computed. If the log contains only the completion timestamp of each task, it is only possible to analyze the cycle times of tasks and not the processing times.

Example 11.5 Figure 11.17 shows the *performance view* produced by Disco when applied to the BPI Challenge 2017 log. The most infrequent arcs have been removed to avoid clutter. The task and the arc with the highest times are indicated in red. If we visually zoom in, in the tool, we are able to see that the task with the slowest processing time is "W_Assess Potential Fraud" (3.1 days) and the highest inter-event time is the one between "A_Complete" and "A_Cancelled" (27.4 days). □

Exercise 11.12 Using a process mining tool, analyze the following event log of a telephone repair process: http://tinyurl.com/repairLogs with the aim of identifying the bottlenecks in this process. Which task has the longest waiting time and which one has the longest processing time?

Dependency graphs can also be used to analyze delays resulting from handoffs between process participants—in addition to delays between consecutive tasks as discussed above. When discussing the structure of event logs, we pointed out that an event record in an event log should have at least a case identifier, a timestamp, and an event class (i.e., a reference to a task in the process), but there can be other event attributes too, like for example the *resource* (process participant) who performed the task in question. The event class is the attribute that is displayed in each node of a dependency graph. When importing an event log into a tool that supports dependency graphs, the tool gives an option to specify which event attribute should

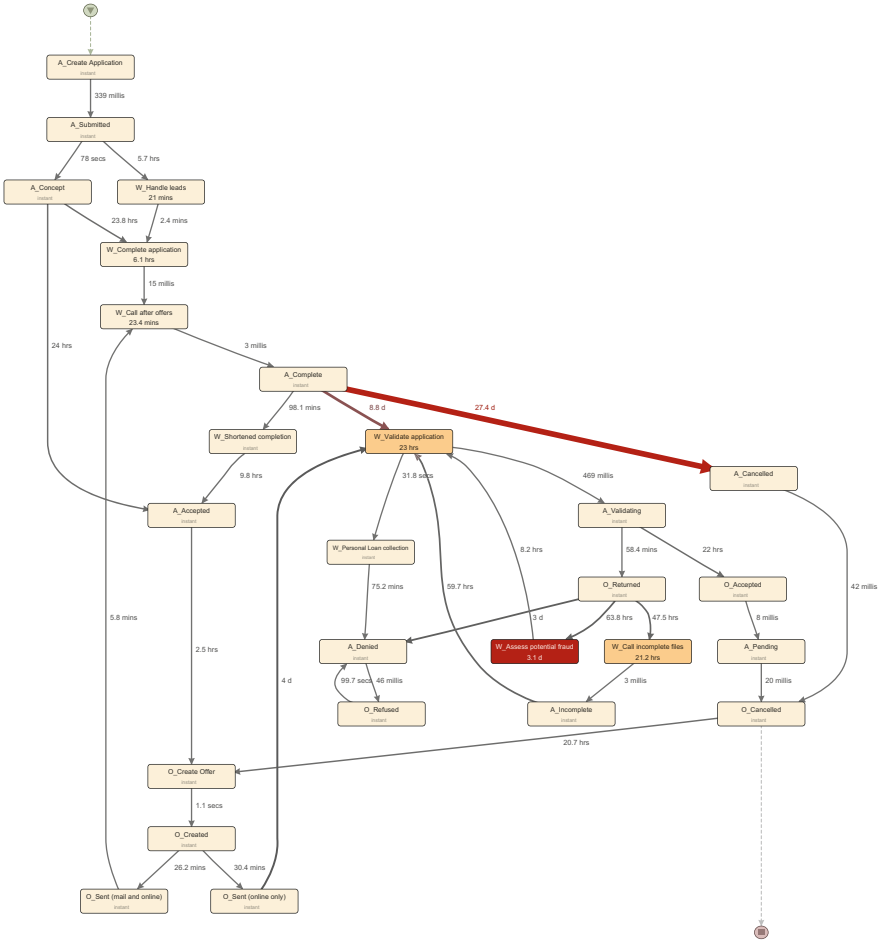


Fig. 11.17 Performance view of the BPI Challenge 2017 event log in Disco

be used as the event class.[19] We can set the tool so as to use the *resource* attribute as the event class.[20] If we do so, the resulting dependency graph will contain one node per resource (process participant) and there will be one arc between nodes A and B if resource B has performed a task immediately after resource A in at least one trace in the log. This map captures all the handoffs that have occurred in the process. We can then apply filters and abstraction operations to this map of handoffs, in the same way as we can apply filters and abstraction operations to dependency graphs where the nodes correspond to tasks. The handoff map can also be color-coded, with the

[19] In some tools, the event class is called the "Activity".

[20] In Disco, this option is only available when importing logs in CSV format. In Celonis, Minit and myInvenio it is available also when importing XES files.

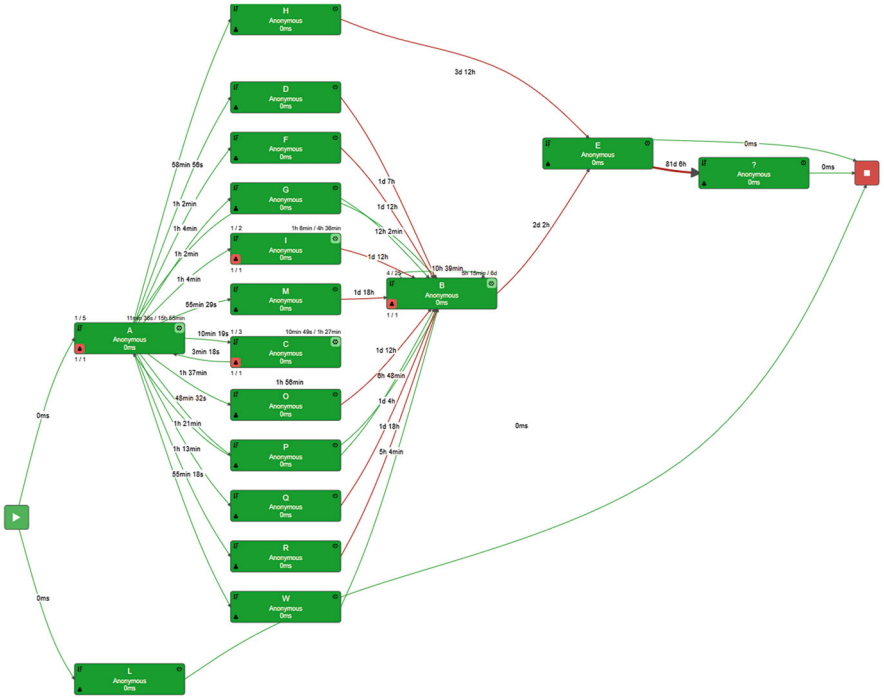


Fig. 11.18 Handoff view of the Sepsis event log in myInvenio

colors corresponding either to the frequency of each handoff, or the waiting time between handoffs.

Example 11.6 Figure 11.18 shows a fragment of the *average duration view* produced by myInvenio when applied to the previously mentioned sepsis treatment log.[21] The map shows that when a case starts (leftmost node), it is first handled by resource A. After a few minutes or about an hour, resource A hands off to resource H or to resources D, F, . . . , R. Resource H hands off to E, while F, . . . , R hand over to B who later hands off to E. Handoffs that take longer than 1 day are shown in red (this threshold can be configured). For example, the handoff from H to E and from B to E take 2–3 days. Resource E hands off to a resource marked as "?". This last handoff takes the largest amount of time (81 days and 6 h), most likely because

[21] We imported the log in CSV format, marked the "org:resource" column as the "Activity" in the import wizard, the "Case ID" column as the "Process Id", and the "Complete timestamp" as the "Start timestamp" because MyInvenio requires a start timestamp column. We filtered the graph so that it shows the 60% most frequent nodes and we placed the *arcs filter* at its minimum value. Finally, we moved to the average duration view, turned on the KPI palette, and set the upper-bound of "activity wait queue threshold" to 1 day so that all arcs with duration of more than a day are shown in red.

cases are left in a pending state until (on average) 81 days and 6 h after the last task, once it is clear that the patient does not need further treatment. In this map, the nodes have a duration of zero, because this event log only has end timestamps (no start timestamps) and hence the processing time cannot be computed. □

11.5.2 Cost Dimension

When measuring the cost of a business process, we need to take into account both direct and indirect costs. Direct costs, like the purchasing costs of four wheels which are assembled on a car, can be more directly determined on a per-case basis. Indirect labor costs or machine and infrastructure depreciation costs are more difficult to handle. In accounting, the concept of *activity-based costing* (ABC) was developed to assign indirect costs to products and services, and to individual customers. The motivation of ABC is that human resources and machinery are often shared by different products and services, and they are used to serve different customers. For instance, the depot of BuildIT rents out expensive machinery such as bulldozers to different construction sites. On the one hand, that involves costs in terms of working hours of the persons working at the depot. On the other hand, machines like bulldozers lose in value over time and require maintenance. The idea of ABC is to use tasks for distributing the indirect costs, e.g., associated with the depot.

Example 11.7 According to Figure 1.6 in Chapter 1 (see page 19), the rental process of BuildIT contains five major tasks. We observe the following durations in the event logs for the case of a bulldozer rental requested on 21st August:

- "Submit equipment rental request" is conducted by the site engineer. It takes the engineer 20 min to fill in the form on paper. The production of each paper form costs € 1. The site engineer gets an annual salary of € 60,000.
- The clerk receives the form, selects a suitable piece of equipment, and checks its availability. These tasks take altogether 15 min. The clerk has an annual salary of € 40,000.
- The works engineer reviews the rental request (annual salary of € 50,000). This review takes 10 mins.
- The clerk is also responsible for sending a confirmation including a purchase order for renting the equipment, which takes 30 min.

In order to work with these numbers we have to make some assumptions. First, at BuildIT the actual working year contains 250 working days of 8 h. Furthermore, all employees receive health insurance and pension contributions of 20% on top of their salary. Finally, people take on average 10 days of sick leave per year. Given the above, the labor cost of each participant per minute is $\frac{salary \times 120\%}{(250-10)\times 8\times 60}$. This is for the site engineer € 0.63 per minute, for the clerk € 0.42 per minute, and for the works engineer € 0.52 per minute. Altogether, this case created costs of 20 min × € 0.63

per minute + (15+30) min × € 0.42 per minute + 10 min × € 0.52 per minute, which sums up to € 36.70.

Now consider a case of a street construction process. We observe from the event log the following durations (processing times) for these two tasks:

- "Prepare the foundation" is conducted by four workers. It took one week at a specific construction case. The used excavator costs € 100,000. It is written off in 5 years and has annual maintenance costs of € 5,000. A worker gets an annual salary of € 35,000.
- "Tar the road" is conducted by six workers. It took 2 days in this case. The tarring machine costs € 200,000, is also written off in 5 years and costs € 10,000 in annual maintenance.

For this case, we can also take the costs of the machinery into account. The labor cost per day is € 175 for one worker. The excavator costs € 20,000 + 5,000 per annum for write-off and maintenance, which is € 104.17 per working day. For the preparation of the foundation, this amounts to 4 × 5 × € 175 + 5 × € 104.17 = € 4,020.85. For the taring of the road, the costs are 6 × 2 × € 175 + 2 × € 208.34 = € 2,516.68. □

Exercise 11.13 Consider that the paper form is printed by the site engineer in the rental process, that the printer costs € 300 written off in 3 years, and that a pile of 500 pieces of paper costs € 10. Why would it make sense to include these costs in the calculation? Or why not?

An inherent problem of ABC is the need to keep track of the duration of tasks like renting out equipment or approving rental requests. Event data stored in process-aware information systems can help to provide such data. Some systems only keep track of task completion. However, ABC also requires the start of tasks to be kept. This means that we need to keep track of the timestamps of the point in time when a resource starts working on a specific task. What is important to take into account here is the cost of achieving additional transparency. There is a trade-off, and once it becomes overly expensive to gain more transparency, it is good to not include those costs in the calculation.

11.5.3 Quality Dimension

The quality of a product created in a process is often not directly visible from execution logs. However, a good indication is to check whether there are repetitions in the execution logs, because they typically occur when a task has not been completed successfully. Repetitions can be found in sequences of task. In Chapter 7, we saw that the loop of a rework pattern increases the cycle time of a task to $CT = \frac{T}{1-r}$ in comparison to T being the time to execute the task only once. The question is now how to determine the repetition probability r from a series of event logs.

The first part of the answer to this question can be given by reformulating the equation such that it is solved for r. By multiplication with $1 - r$, we get $CT - r \times CT = T$. Subtraction of CT yields $-r \times CT = T - CT$, which can be divided by $-CT$ resulting in

$$r = 1 - \frac{T}{CT}.$$

Both CT and T can now be determined using the data of the event logs. Consider the five cases in which we observe the following execution times for task a:

1. 5 min, 10 min;
2. 10 min;
3. 20 min, 6 min, 10 min;
4. 5 min;
5. 10 min, 10 min.

The cycle time CT of a can now be calculated as the average execution time of a per case, while the average execution time T is the average execution time of a per instantiation. Both can be determined based on the sum of all executions of a, which is 86 min here. We have five cases, such that $CT = 86/5 = 17.2$. Altogether, a is executed nine times yielding $T = 86/9 = 9.56$. Hence, we get $r = 1 - \frac{86/9}{86/5} = 1 - \frac{5}{9} = 0.44$. Of course, this calculation is only an approximation of the real value for r. It builds on the assumption that the duration of a task always follows the same distribution, no matter if it is the first, the second or another iteration.

Exercise 11.14 Determine the probability of repetition r for the following execution times of task b:

1. 20 min, 10 min;
2. 30 min;
3. 30 min, 5 min;
4. 20 min;
5. 20 min, 5 min;
6. 25 min.

Also explain why the value is misleading for these logs.

In some software systems it might be easier to track repetition based on the assignment of tasks to resources. One example are *ticketing systems* that record which resource is working on a case. Also, the logs of these systems offer insights into repetition. A typical process supported with ticketing systems is incident resolution. For example, an incident might be a call by a customer who complains that the online banking system does not work. Such an incident is recorded by a dedicated participant, e.g., a call center agent. Then, it is forwarded to a first-level support team that tries to solve the problem. In case the problem turns out to be too specific, it is forwarded to a second-level support team with specialized knowledge in the problem domain. In the best case, the problem is solved and the customer

is notified. In the undesirable case, the team identifies that the problem is within the competence area of another team. This has the consequence that the problem is rooted back to the first-level team. Similar to the repetition of tasks, we now see that there is a repeated assignment of the problem to the same team. By analyzing the event log, we can determine how often this repeated assignment occurs and to what extent it is affecting the mean cycle time of the process.

Using a process mining tool, we can apply a filter to an event log so as to retain only those cases where the same task appears (at least) twice. In Disco this is achieved using a so-called "follower" filter while in Celonis it is called a "process flow selection" filter.

Exercise 11.15 With reference to the event log of the Business Process Intelligence Challenge 2017 (https://tinyurl.com/bpic2017), in how many cases was "W_Assess Potential Fraud" completed at least twice in the same case?

11.5.4 Flexibility Dimension

Flexibility refers to the degree of variation that a process permits. This flexibility can be discussed in relation to the event logs the process produces. For the company owning the process, this is important information in order to compare the desired level of flexibility with the actual flexibility. It might turn out that the process is more flexible than what is demanded from a business perspective. This is the case when flexibility can be equated with lack of standardization. Often, the performance of processes suffers when too many options are allowed. Consider again the process for renting equipment at BuildIT. The process requires an equipment rental request form to be sent by email. Some engineers however prefer to call the depot directly instead of filling the form. Since these engineers are highly distinguished, it is not easy for the clerk to turn down these calls. As a result, the clerk fills out the form while being on the phone. Not only does this procedure take more time, but, due to noise at the construction site, it also increases the likelihood of errors. In practice, this means that the rental process has two options to submit a request: by form (the standard procedure) and via the phone.

Partially, such flexibility as described above can be directly observed in an event log. We have seen that the workflow log of a process plays an important role for automated process discovery. It can also be used to assess the flexibility of the process. The workflow log summarizes the essential behavior of the process. As it defines each execution as a sequence of tasks, it abstracts from the temporal distance between them. In this way, the workflow log contains a set of traces that have a unique sequence. This means that if two executions contain the same sequence of tasks, then they only result in a single trace to be included in the workflow log. This abstraction of process executions in terms of a workflow log makes it a good starting

point for discussing flexibility. Accordingly, the number of *distinct executions* DE can be defined based on a workflow log L as

$$DE = |L|.$$

Exercise 11.16 Consider the event log of the order-to-cash process in Figure 11.4 (page 423). What is the number of distinct executions DE?

The question arises whether the number of distinct executions always gives a good indication for flexibility. At times, the number of distinct executions might give an overly high quantification of flexibility. This might be the case for processes with concurrency. Such processes can be highly structured, but having only a small set of concurrent tasks results in a rich set of potential execution sequences. Consider the process model constructed by the α-algorithm in Figure 11.12 (page 437). The tasks i and h are concurrent to j and k. Indeed, there are six options to execute them:

1. i, h, j, k,
2. j, k, i, h,
3. i, j, k, h,
4. j, i, h, k,
5. i, j, h, k and
6. j, i, k, h.

While the order is not strict, all of them must be executed. Therefore, it might be a good idea to additionally consider whether a task is optional. If T refers to the number of tasks that appear in the workflow log, then the set T_{opt} contains those tasks that are optional. Optionality according to the log means that for a particular task there exists at least one trace in which the task does not occur. For the log of Figure 11.4, we observe that the tasks b to f depend upon the availability of raw materials. We can quantify the degree of *optionality* OPT as

$$OPT = \frac{T_{opt}}{T}.$$

11.6 Conformance Checking

Conformance checking is concerned with the question whether or not the execution of a process follows predefined rules or constraints. This question can be answered by inspecting the event log. If a particular constraint does not hold, we speak of a *violation*. In particular, conformance checking is concerned with identifying these violations and making statements about the extent of them altogether. Violations might relate to the three process perspectives of control flow, data, and resources, in isolation or in combination. In the following, we describe how they can be specified.

11.6.1 Conformance of Control Flow

From the control-flow perspective, conformance can be analyzed in two ways: based on *constraints* or based on a *normative process model*. Below, we discuss these two approaches in turn.

There are three recurrent types of control-flow constraints: mandatoriness, exclusiveness, and ordering. These three constraint types define how two tasks are allowed to be related in a process. A company might want to define a *mandatoriness* constraint for certain tasks, because these tasks are required from a control perspective. Consider again the case of BuildIT and its equipment rental process. A works engineer is supposed to review the rental request. This task serves as a control for ensuring that only appropriate equipment is rented. This measure might help to keep the rental costs in line. This verification task is a likely candidate for being mandatory. At the level of the event log, violations of mandatory task constraints can be found by searching for traces in which the task in question is not executed.

An *exclusiveness* constraint states that two or more tasks cannot co-occur in the same case. In the case of the equipment rental process, for example, an exclusiveness constraint might state that a rejection of a rental request cannot be followed by its approval. This exclusiveness can be checked by searching for traces in which both tasks appear.

An *ordering* constraint states that two tasks must always occur in a given order. In the equipment rental process, an ordering constraint might state that the availability of the requested equipment must be checked before the request is reviewed. Obviously, it is a waste of effort to review requests that cannot be met because the equipment is not available. Violations to order constraints can be found by searching for traces with the tasks appearing in the wrong order.

Exercise 11.17 Consider the event log in Figure 11.4 (page 423). Which tasks can be considered mandatory, or exclusive with respect to one another?

Conformance of control flow can also be checked by comparing the behavior observed in the log against a process model. In this setting, the purpose of conformance checking is to identify two types of discrepancies:

- Unfitting log behavior: behavior observed in the log that is not allowed by the normative process model.
- Additional model behavior: behavior allowed in the normative process model but never observed in the log.

If some discrepancies are found, a conformance checking technique will tell us where each discrepancy occurs, and what exactly is different between the event log and the process model.

There are broadly three families of techniques for conformance checking between an event log and a process model: replay, trace alignment, and behavioral alignment. In replay techniques [152], each trace is replayed against the process

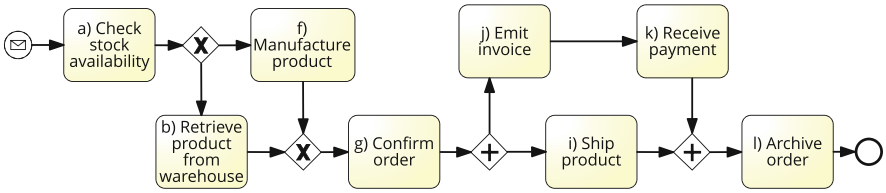


Fig. 11.19 BPMN model with a token on the start event for replaying the case $\langle a, b, g, i, j, k, l \rangle$

model one event at a time starting from the initial state of the process model (i.e., the state where there is a token in the start event). At each step of the replay, we determine if the next event in the event log was allowed to be executed according to the process model. If the next event can be replayed, we advance to the next state in the process model by moving tokens forward in the model. If the next event cannot be replayed, we record a *replay error* and a local correction is made to resume the replay procedure. The local correction may be for example to skip/ignore a task in the process model or to skip an event in the log.

Example 11.8 Using the token rules of BPMN, we can replay the case $\langle a, b, g, i, j, k, l \rangle$ on the model shown in Figure 11.19. In the initial state, the process has a token on the start event. Once the case is started, this token moves to the output arc of the start event. This arc leads to task a ("Check stock availability"), which means that the token enables this task to be executed. The token moves to this task while it is executed, and it is forwarded to the output arc once the task is completed. Now, the XOR-split is activated, which means a decision has to be taken to continue either with b ("Retrieve product from warehouse") or with f ("Manufacture product"). For the considered case, we continue with b. After g ("Confirm order") has been completed, we arrive at an AND-split. An AND-split consumes a token from its input arc and creates one token on each of its output arcs. As a result, we have two tokens afterwards: one enabling i ("Ship product") and one enabling j ("Emit invoice"). In this state we can proceed either with i or j. These tasks are concurrent. In order to replay the case, we first execute i and j afterwards. Once i and later k are completed, the AND-join is allowed to proceed. One token on each of its input arcs is required for that. Both these tokens are consumed and a single token is created on its output arc. As a result, l ("Archive order") can be executed. When a replay error is detected, it is reported and a local correction is made to resume the replay procedure. The local correction may be for example to skip a task in the process model or to skip an event in the log. □

Using token replay, we can also measure the conformance of a trace to a process model by comparing at each step the number of tokens that are required to replay an event versus the tokens that are actually available. Specifically, at each step we can keep track of the following values:

- c: the number of tokens that are correctly consumed,
- p: the number of tokens that are correctly produced,

- m: the number of tokens that are missing for executing the next event in the trace, and
- r: the number of tokens remaining unconsumed after executing the final event in the trace.

Example 11.9 Consider the case $\langle a, b, i, j, k, l \rangle$ in which the confirmation of the order has been omitted. Figure 11.20 first shows the state before replaying b of the case $\langle a, b, i, j, k, l \rangle$. After replaying b, there is a token available to execute g, but it is not consumed. Instead, there is a missing token for firing the AND-split, which would activate i and j. The figure also shows the number of correctly produced and consumed tokens for each step until completion. Using the four values of c, p, m, and r, we can calculate the fitness measure for a complete trace as an indicator of conformance. It is defined based on the fraction of missing tokens to correctly consumed tokens ($\frac{m}{c}$) and the fraction of remaining tokens to produced tokens ($\frac{r}{p}$) as

$$fitness = \frac{1}{2}(1 - \frac{m}{c}) + \frac{1}{2}(1 - \frac{r}{p}).$$

Summing all our values of c, all values of p, of m, and of r, we obtain an overall $c = 12, p = 12, m = 1$, and $r = 1$. From these, we get a fitness of $\frac{1}{2}(1-\frac{1}{12})+\frac{1}{2}(1-\frac{1}{12}) = 0.9166$. When we consider a set of cases, not just a single case, we can easily calculate the fitness in the same way. The idea is to simply continue counting c, p, m, and r by replaying the next case in the process model. Once we have replayed all cases, we get the resulting overall fitness of this set of cases. □

The results of conformance analysis can be interpreted in two ways. First, we can use the overall fitness measure to get an idea of how accurately the process model matches the actually observed behavior as reflected by the set of cases. While the fitness as an overall measure is useful to this end, it does not help us to analyze the deviations in more detail. Therefore, and secondly, we can inspect at which arcs of the process model we have encountered missing or remaining tokens. Figure 11.21 shows the corresponding numbers from replaying several cases in the process model. It can be seen that apparently most deviations relate to task g and some to task l. This information can be utilized to ask process participants why g has been omitted for some cases. The goal of such an inquiry would be to find out whether this omission is desirable. The question is whether the process participants have found a more efficient way to handle the confirmation, or whether an omission has to be considered bad practice and as such be discouraged. For the case of the archival (task l), the omission is likely to be bad practice.

In replay techniques, the recovery from an error is done locally. Hence, these techniques might not identify the minimum number of errors that can explain the unfitting log behavior. This limitation is addressed by *trace alignment* techniques [4]. These techniques identify, for each trace in the log, the closest corresponding trace that can be parsed by the model. Trace alignment techniques also compute an alignment showing the points of divergence between these two traces.

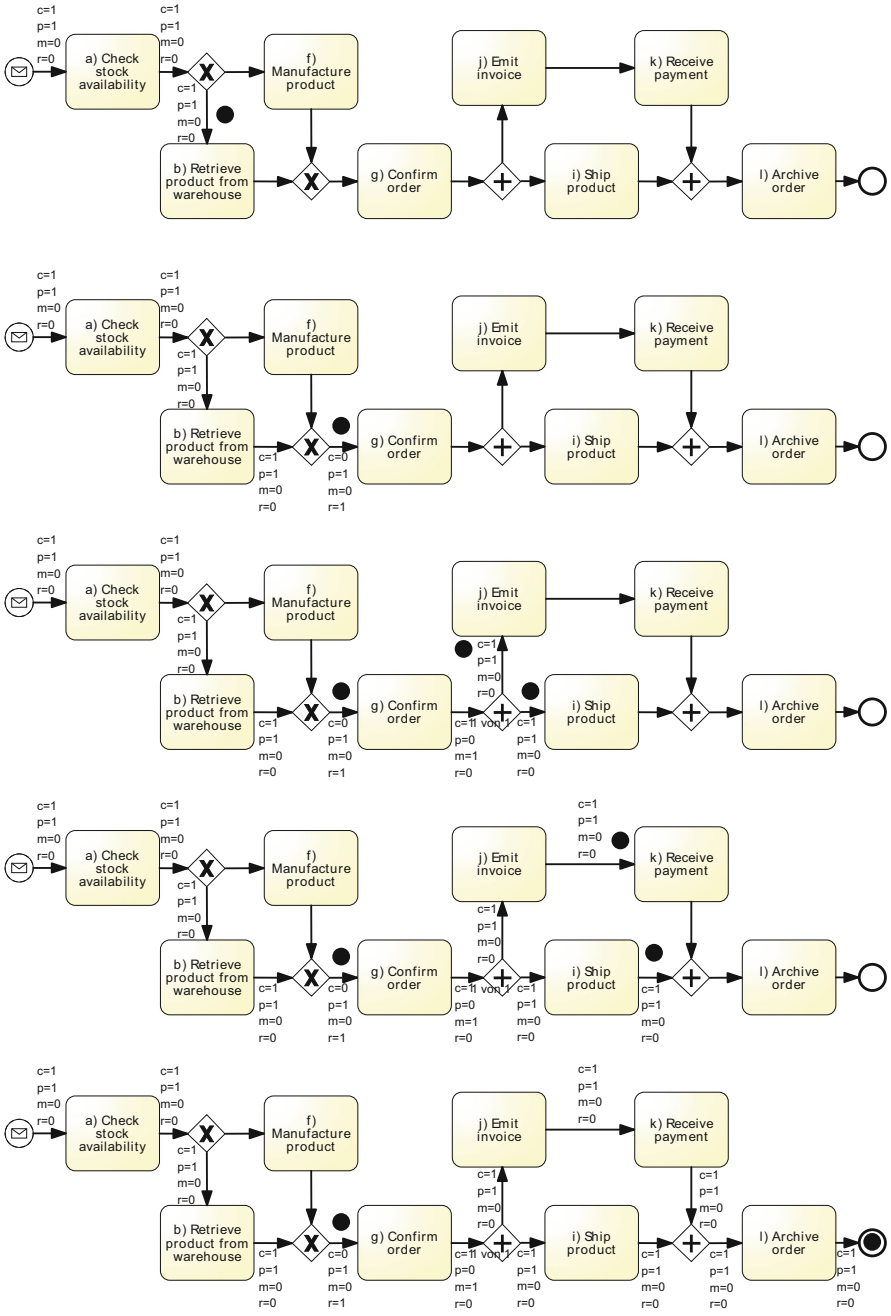


Fig. 11.20 Replaying the non-conforming case $\langle a, b, i, j, k, l \rangle$

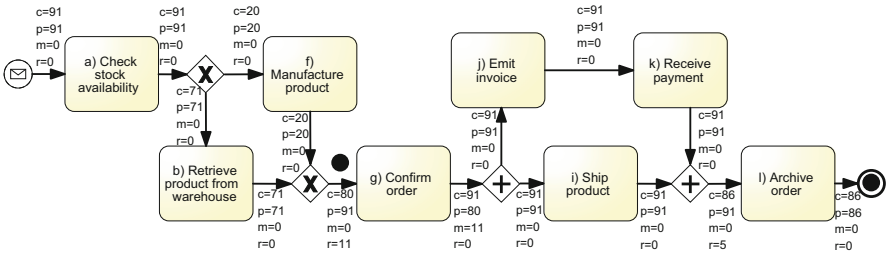


Fig. 11.21 Result of replaying cases in the process model

The output is a set of pairs of aligned traces. Each pair shows a trace in the log that does not match exactly a trace in the model, together with the corresponding closest trace produced by the model.

Trace alignment techniques do not explicitly handle concurrent tasks nor cyclic behavior (repetition of tasks). If for example four tasks can occur only in a fixed order in the process model (e.g., $\langle a, b, c, d \rangle$), but they can occur concurrently in the log (i.e., in any order), this difference cannot be directly detected by trace alignment, because it cannot be observed at the level of individual traces. This limitation is addressed by *behavioral alignment* [54]. Unlike trace alignment, behavioral alignment does not align one trace at a time, but it aligns the entire event log against the process model at once, so that we can observe differences that cannot be observed at the level of individual traces.

The output of a behavioral alignment technique is a data structure known as a *Partially Synchronized Product* (PSP), which captures every state where a task or behavioral dependency occurs in the process model, but does not occur in the event log, or vice versa. Behavioral alignment techniques can therefore detect both unfitting behavior and additional behavior.

The PSP is a rather unreadable structure, but it can be used to generate a set of *difference statements* in natural language. For example, a possible difference statement is the following: "In the model, after task a, task b always occurs before c, while in the log, tasks b and c are concurrent.". Each such discrepancy can be displayed visually in addition to textually. For example, Figure 11.22 shows a discrepancy detected by Apromore's conformance checker. Similar visualizations of model-log discrepancies are provided by Signavio Process Intelligence,[22] LANA,[23] Celonis, and MyInvenio.

Once a discrepancy such as the one above has been identified, the user may either decide to ignore it—for example because the discrepancy occurs rarely and it is not worth capturing such rare exceptions in the model—or decide to fix it. The process of fixing a model so that it fits better an event log is called *model repair*. Apromore

[22] https://www.signavio.com.

[23] https://lana-labs.com/en.

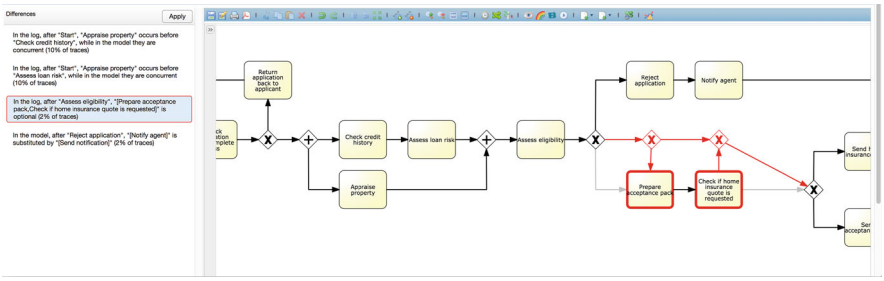


Fig. 11.22 Visualization of a model-log discrepancy in Apromore

provides features for fixing each identified discrepancy automatically. In other tools, this repair can be done manually by updating the model according to the detected difference, and then re-running the conformance checker.

11.6.2 Conformance of Data and Resources

In the above discussion, we have focused on checking conformance along the control-flow perspective, that is, checking that the occurrence of tasks and their relative order in the event log are consistent with a given process model. Equally important though is to check conformance along the data and the resource perspectives of a business process. Consider the situation where an expensive caterpillar is requested for a construction site of BuildIT. Many companies have extra rules for high expenses or risky commitments. In the case of BuildIT, the rental of the caterpillar requires the additional signature of a manager. Similar cases can be found in banks where a loan of more than € 1,000,000 might require the additional sign-off from a director. There might also be a rule stipulating that a loan should not be granted to a black-listed applicant. Such constraints can be checked by searching for cases in the log where a given data field takes a forbidden value.

The example of the additionally required signature already points to a combination of data and resource constraints. If a certain amount is exceeded, then there is a dedicated resource who is required to approve it. However, there are also constraints that purely refer to the resource perspective. Participants usually require *permissions* to execute certain tasks. For instance, the person signing off the caterpillar rent must be a manager, and it is not allowed that this person is a construction worker. Typically, permissions are bundled for specific *roles*. For example, this can be done by explicitly listing what a manager and a construction worker are allowed to do. Violations of permissions can be checked by searching for each task performed by a participant, and verifying if the appropriate role or permission existed. A control rule that requires two different persons to approve a business transaction is called a *separation of duties* constraint. These rules do not necessarily involve supervisors.

For example, it might be ok with a particular bank if a loan of € 100,000 is signed by two bank clerks, while a € 1,000,000 loan requires the signature of a clerk and a director. This separation-of-duties policy can be checked by searching for cases in the log where the same participant or two participants with the same role approved the same transaction.

Exercise 11.18 With reference to the event log of the Business Process Intelligence Challenge 2017 (https://tinyurl.com/bpic2017), in how many cases was "W_Assess Potential Fraud" completed at least twice in the same case by the same resource? Hint: This exercise is similar to Exercise 11.15 (page 450), but it involves an additional constraint: the same resource should be involved in both task occurrences.

11.7 Variants Analysis

The performance of a business process can vary considerably over time (e.g., the performance this month is lower than last month's), geographically (the performance of a prescription fulfillment process across several pharmacies is different), or across business units, product types, or customer types. And even within a given period of time, a given place, a given business unit, type of product, and type of customer, we will often observe significant differences between the cases with the best and with the worst performance. Some cases go smoothly and lead to a positive outcome in a timely manner, while others lead to negative outcomes or they are delayed, leading to customer dissatisfaction. A frequent question in the mouth of a process owner is: "Why is the performance for some cases so much worse than for others?"

Variants analysis techniques allow us to analyze differences across subsets of cases in a process (i.e., two variants). Given two subsets of cases L1 and L2, these techniques allow us to identify characteristics that are commonly found in the cases in L1, and are rare or non-existent in the cases in L2. There might be many such characteristics, and of course, we are interested in finding those that can help us to explain why the cases in L1 might exhibit better or worst performance than those in L2.

We can use any criterion to split the cases in a log into L1 and L2. This criterion could be based on the time when the case started (e.g., all cases that started in the year 2015 go to L1, all those that started in 2016 go to L2). Or it could be based on product type (e.g., all car insurance claims for sports cars go into L1, all other car insurance claims go into L2). It is also common to split the log into cases that have a normal (or satisfactory) performance (L1) vs. those with undesirable performance (L2). This type of analysis is called *deviance mining*, because what we want is to understand why some cases are deviant (i.e., those in L2). Note that deviance can also be seen from a positive perspective. Indeed, we could split the log into those cases that have normal or below-normal performance (L1) and those

that have above-normal performance (L2). The cases in log L2 are hence positively deviant.

Variants analysis can be done manually or automatically. In the manual approach, we typically start by discovering a process model for L1, another one for L2, and we compare these two models visually. For example, we can discover a dependency graph for L1 and another for L2, put them side-by-side, and compare them using the frequency view (nodes and arcs are annotated by frequency) and the performance view (nodes and arcs are annotated with times). In this way, we can spot tasks or paths that are more frequent in L1 than L2 or vice versa, or we can spot differences in the cycles times of tasks. We can go further and inspect the data perspective, for example to determine if some data attribute values are more common in L1 than in L2, or if some resources are more active in L1 than in L2.

An alternative is to apply automated *log delta analysis* techniques [175]. These techniques take as input two event logs and produce a set of *patterns* that are common in L1 and uncommon in L2 or vice versa. We then analyze these patterns and determine which ones might be able to explain the observed differences in performance. For example, the log delta analysis technique implemented in the Apromore tool takes as input two event logs and produces a set of *difference statements* similar to those we saw in the context of conformance checking, e.g., "In log L1, task a is sometimes skipped, whereas in log L2, task a is always executed", or "In log L1, task a is sometimes repeated, whereas in log L2, task a is almost never repeated".

Other log delta analysis techniques are based on so-called *association rules*, which are commonly used in the field of data mining. These techniques take as input a log (which they see as a set of sequences) and produce as output patterns of the form "after a, we eventually observe b". Some of these techniques (called *discriminative sequence mining* techniques) take as input two logs and identify association rules such as the one above, which are common in one log but not in the other.

In general, manual comparison is suitable for relatively small event logs, or in cases where we can filter the event log or abstract the dependency graph sufficiently so that the key differences are still visible, yet the dependency graph is not overwhelmingly complex. For more complex scenarios, we can use automated log delta analysis to identify a set of patterns, select a subset of those patterns, and analyze them further by manually inspecting a few cases where these patterns occur, and determining if the occurrence of these patterns might help to explain why those cases exhibit higher or lower performance.

Example 11.10 We consider again the event log of the Business Process Intelligence Challenge 2017 (https://tinyurl.com/bpic2017), used in Example 11.3 (see page 429). We observe that the distribution of the cycle time across cases has a long tail. Specifically, 96% of cases take less than 45 days to complete, but the remaining 4% of cases have cycle times going to 169 days in some cases. One may wonder why do these handful of cases take so long? To answer this question, we split the log into two sub-logs: one log contains the cases that take less than 45 days (we call

these the *fast cases*) and the other contains all cases that take 45 days or more (*slow cases*).[24] We then discover a dependency graph for each of these two sub-logs.

By doing a side-by-side inspection of these dependency graphs under the frequency view, we observe that the offer cancelation (event type "O_Cancelled") is very frequent among the slow cases, while not so frequent in the fast cases. Specifically, among the 1,299 slow cases, this event type occurs 2,061 times in 1,035 cases (80% of slow cases), hence about twice per case where it occurs. Meanwile, among the 30,209 fast cases, this event type occurs 18,834 times in 14,466 cases (48% of slow cases), hence about 1.37 times per case.[25]

Also, 994 cases out of 1,299 slow cases end with an "O_Cancelled" event (again about 80% of cases).[26]

Similar remarks can be made regarding event "A_Cancelled" (application cancelation), which occurs in about half of slow cases, but only in about a third of fast cases.

We also observe that the situation where the customer is called due to incomplete files is much more frequent in slow cases. Indeed, task "W_Call Incomplete files" occurs 1,725 times in 793 slow cases (61% of slow cases), while it occurs 21,493 times in 14,210 cases of the fast cases (47% of fast cases). If we look at the performance view, we see that this task is a bottleneck in the slow cases (5.9 days of processing time), but less so in the fast cases (11.4 h), suggesting that in the slow cases, the customer turns out to be more difficult to reach.

In summary, we conclude that cancelations and incomplete files are possible factors explaining why some cases take over 45 days to complete. □

Exercise 11.19 We consider a process for handling health insurance claims, for which we have extracted two event logs, namely L1 and L2. Log L1 contains all the cases executed in 2011, while L2 contains all cases executed in 2012. The logs are available in the book's companion website or directly at: http://tinyurl.com/InsuranceLogs.

Based on these logs, answer the following questions using a process mining tool:

1. What is the cycle time of each log?
2. Describe the differences between the frequency of tasks and the order in which tasks are executed in 2011 (L1) versus 2012 (L2). Hint: If you are using dependency graphs, you should consider using the abstraction slider in your tool to hide some of the most infrequent arcs so as to make the maps more readable.

[24] This split into sub-logs can be achieved using the filtering operations supported by Celonis, Disco, Minit, myInvenio, and other process mining tools.

[25] These numbers are calculated using Disco. The numbers can slightly differ in other tools. For example, Celonis finds 1,378 slow cases, out of which 1,088 contain "O_Cancelled".

[26] To determine which cases end with a given event type in each log, we can add a second filter to each of them, specifically a filter of type *endpoint* in Disco, an *activity selection* filter with a *case ends with* constraint in Celonis, or a *match at the end of case* filter in myInvenio.

3. Where are the bottlenecks (highest waiting times) in each of the two logs and how do these bottlenecks differ?

11.8 Putting It All Together: Process Mining in Practice

We have seen that the term *process mining* refers to a broad collection of techniques to extract insights from event logs generated during the execution of a business process. Some of these techniques are focused on discovering a model of the process, while others allow us to analyze the process from different perspectives (conformance, performance, variants).

Faced with a business problem, an analyst is likely to use several of these techniques together. In this respect, there are two approaches to apply process mining to a given business process:

1. The *exploratory* approach, which is driven by the curiosity to understand how a business process is executed without any specific question at hand. In this approach, we typically start by discovering a process model or by doing conformance checking against an existing model. This analysis is typically complemented with a general analysis of bottlenecks and by a longitudinal analysis (e.g., comparing the performance of the process in the latest month versus the month before) or a cross-sectional analysis (e.g., comparing the performance of the process for different customer or product types).
2. The *question-driven* approach is driven by a business problem. For example, we may want to find out why certain insurance claims take too long to be resolved. In this case we start by identifying and scoping the problem, formulating questions and then applying process mining techniques to answer these questions.

These two approaches are complementary. Oftentimes we start with an exploratory approach, and as we get a clearer picture of the process and its bottlenecks, we continue the analysis with concrete business-driven questions in mind.

In a question-driven approach, there are typically five phases involved: (1) framing the problem; (2) collecting the data; (3) analyzing the data; (4) interpreting the results; and (5) formulating a process improvement proposal.

Example 11.11 We discuss each of these phases using an example of a process mining project reported in [171].

> With its nine million customers and 16,000 employees, Suncorp is the largest insurer in Australia and second largest in New Zealand. Besides insurance services, it provides wealth management services, banking and superannuation services. As of 2012, the typical value chain for insurance would consist of four core processes: insurance service development, sales, after-sale services, and claims handling, covering all in all around 500 tasks.
> Suncorp provides a wide range of insurance services, for example home insurance, commercial insurance, and motor insurance. The company has acquired over time a number of insurance brands targeting various market segments. As of 2012, Suncorp managed

nine different brands and 30 core value chains. This high degree of variation manifested itself within the organisation into diverging performances in their claim handling processes, particularly the one for commercial insurance. The owner of the process noticed that frequently, claims with low payout (i.e., low-value claims), which should have been resolved in a matter of days, were taking unexpectedly long. In fact, some of these low-value claims were taking longer to be resolved than some of the high-value claims.
By talking to different stakeholders, the process owner formulated various possible reasons for these unexpected delays. For example, she hypothesized that these delays came from the fact that a new system had been recently put in place to manage claims, and maybe the resources using this new system did not have appropriate training. However, acting on this or other hypotheses had not led to any significant process improvement.
It is in this context that a question-driven process mining project was started with the aim of shedding light into the causes behind the observed delays.

□

In the first phase of the question-driven approach, we need to formulate a top-level problem or question for the process mining project. For example, why does the process perform poorly, for example, in terms of cycle time? Or why do we have frequent defects (high error rates)?

In the Suncorp case study, the top-level problem was that oftentimes simple claims take an unexpectedly long time to complete. This problem leads to the following questions: What distinguishes the processing of simple claims completed on time, from simple claims that are not completed on time? And what early predictors can be used to determine that a given simple claim will not be completed on time? The next step was to define what is a *slow simple claim*. After some analysis of the distribution of cycle times, it was decided to define it as a claim with a payout of less then X (for an undisclosed X) that takes more than 5 days to be resolved.

Having framed the problem in this way, the project got on board two part-time business analysts with prior expertise in this business process, a database administrator, and a process mining expert. The process owner acted as the business sponsor of the project. The project took 4 months.

The second phase of the question-driven approach is *data collection*. Here, we need to find relevant data sources, which are typically the databases of the enterprise systems over which the process is executed. As part of this extraction, we need to identify the process-related entities and their identifiers (in particular the case identifier) so as to be able to group different events into traces, which is necessary in order to produce an event log in the XES format. In the Suncorp case, the data was extracted from different tables of a claims handling system.

A time-consuming step within this phase is that of data pre-processing. Once we have the event log, we need to clean the data by filtering irrelevant events, cover gaps that may exist due to recording errors, and combine equivalent events from different tables if the data is scattered across tables (potentially originating from different systems, e.g., a CRM and an ERP system)—recall that we are interested in tracing the end-to-end business process. Often, for example, the control-flow data (events and their timestamps) are available in one table, while the resource data is in another table, and the business objects (containing the data attributes) are in another

set of tables. It also happens that different tables may record different variants of the same process, for example one table per product.

The third phase is that of *analysis*. In the Suncorp case, a data analyst took the log consisting of traces of low-value claims and divided it into two sub-logs: fast simple claims and slow simple claims, as defined above. The analyst applied a manual variants analysis technique. Specifically, the log was split into two by applying filtering operations (less than 5 days, more than 5 days) and two dependency graphs were discovered and visually compared. After some effort, it became apparent that there were some relevant differences, specifically two types of unwanted repetitions were occurring frequently in the slow cases, but rarely in the fast cases. These results were backed up by a statistical analysis of the repetitions.

The fourth phase is dedicated to extracting insights from the analysis results and to validating these insights with the domain experts and other stakeholders. In the Suncorp case study, the analysts discussed the findings with several process participants and other stakeholders knowledgeable of the process. They took concrete instances where unwanted repetitions occurred, and showed them to these stakeholders. Based on these focused discussions, they managed to determine the key reasons for the unwanted repetitions.

Using these insights, they put up a business case for process improvement (last phase), which proposed a set of changes to the process in question. The latter project led to a reduction in cycle time (from 30–60 days to 5 days) and a reduction of 30% in resource utilization.

11.9 Recap

We discussed two families of process monitoring techniques: statistics-based techniques and model-based techniques. Given a set of performance measures, statistics-based techniques allow us to analyze the performance of a process by means of aggregation functions (e.g., mean and standard deviation) or by means of data visualization techniques. These statistical aggregates and visualizations are typically grouped into dashboards. We saw three types of dashboards: strategic dashboards (targeted at the top management), tactical dashboards (targeted at analysts, process owners and other managers), and operational dashboards (targeted at process participants and operational managers).

On the other hand, model-based techniques (also known as process mining techniques) use process models as a central instrument. Given an event log capturing a number of executions of the process, these techniques allow us to discover a process model (automated process discovery), to compare a given process model to the event log (conformance checking), to analyze the performance of the process with respect to a process model (performance mining), and to compare the performance of different subsets of cases (variants analysis).

We discussed several automated process discovery techniques, starting with dependency graphs, which give us a zoomable view of an event log, all the way

to techniques to automatically discover BPMN process models, such as the split miner.

With respect to conformance checking, we discussed how to check different types of control-flow constraints using an event log. We also saw how to compare a process model and an event log in order to determine if, how, and to what extent the execution of cases recorded in the log deviates with respect to the process model.

We then saw how performance mining techniques allow us to use event logs and process models as a basis to analyze the performance of a process with respect to each of the four dimensions of the Devil's Quadrangle. In particular, we saw how dependency graphs can be enhanced with temporal measures (waiting times, processing times) in order to analyze the performance of a process.

With respect to variants analysis, we saw how a visual comparison of dependency graphs allows us to understand the differences between two variants of a process (e.g., normal and deviant executions). We also discussed an alternative technique known as log delta analysis, which automatically computes a list of differences between two event logs.

11.10 Solutions to Exercises

Solution 11.1 The simplest (but insufficient) solution is to display the histogram of due times of pending cases, i.e., a bar chart showing how many pending prescriptions are due at 3 p.m., 4 p.m., etc., as in the right-hand side of Figure 11.1. This histogram gives us an idea of the work-in-process, but it says nothing about our capacity to handle it nor about the progress of partially-completed cases.

A more complete alternative is a segmented bar chart where each bar shows the amount of prescriptions due in a given 1-h-period (as above) and each bar is decomposed into segments corresponding to states. As illustrated in Figure 11.23a, this bar chart tells us how much work has to be done by each hour of the day. But it does not tell us anything about of our capacity to handle it.

Another approach is to estimate, for each hour of the day h, the amount of *cumulative processing time* required in order to fulfill all prescriptions that are due before h. For example, the 5 p.m. bar in this chart would represent how much processing time is needed to fulfill all prescriptions due before 5 p.m. The *cumulative processing time* can be displayed as a bar chart. Next to each bar, we can then display the *cumulative capacity* available up to each hour of the day, i.e., how much effort the employees of the pharmacy can collectively deliver before each hour of the day. For example, if there are three technicians and one pharmacist, then they can collectively deliver 4 person-hours per calendar hour. So if it is currently 10 a.m. in the morning, they can collectively deliver 4 person-hours by 11 a.m., 8 person-hours by 12 p.m., etc. When the cumulative processing time due at a given hour is close to or exceeds the capacity available before that hour, it means that deadlines are likely to be missed. For example, Figure 11.23b shows a situation where the cumulative processing time due by 6 p.m. is about the same as the cumulative

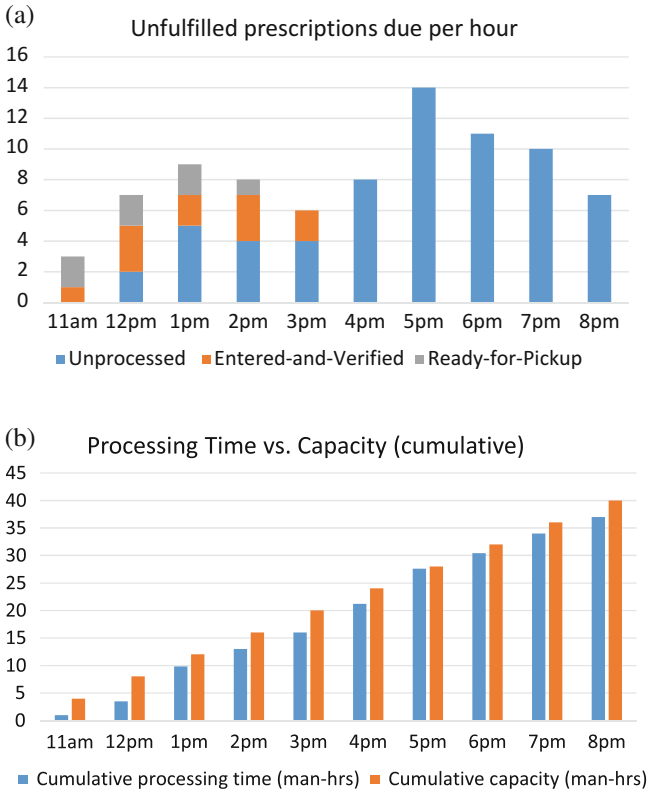


Fig. 11.23 Operational dashboard for pharmacy prescription process . (**a**) Segmented bar chart of unfulfilled prescriptions. (**b**) Bar chart of demand (required processing time) vs. capacity

capacity. Hence, it is possible that deadlines will be missed at 5 p.m. due to the small amount of slack. Note that this chart can be drawn separately for pharmacists and for technicians.

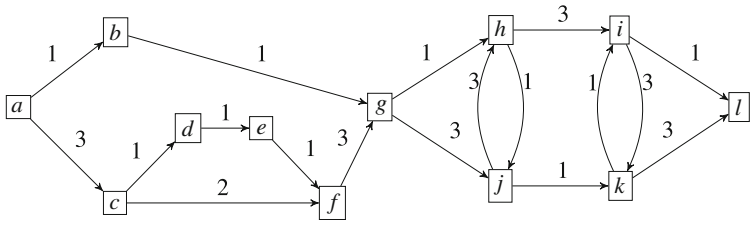
Solution 11.2 Given that the focus is on customer service, a natural performance measure could be the defect rate for different types of defects, e.g., percentage of prescriptions not delivered on time, percentage of prescriptions delivered incorrectly, and percentage of prescriptions canceled due to OOS. Since the pharmacies are geographically distributed, we can display these measures on top of a map. For each region, we can display three circles: one per performance measure. The value of a performance measure for a given region can be encoded by the radius of the circle and also the color of the circle. This way, the process owner can visually identify which regions are under-performing with respect to each of the chosen measures.

Solution 11.3 We can use the process landscape as a starting point and assign a color to each process group to encode the change in efficiency with respect to the previous month (blue = efficiency improved, yellow = neutral, red = efficiency increased). To do so, we need to have a measure of efficiency for groups of processes. Since the focus is on cost reduction, one approach would be to define a measure of aggregate cost per month for each process (the cost required to execute the instances of the process during a month). Then, we can aggregate this measure to an entire group of processes via summation. Finally, we can compute the difference between the cost generated by each process group on a given month and calculate the delta between the current month and the previous one (or current month versus same month the previous year to adjust for seasonality). Ideally, it should be possible to drill down in this dashboard and open each process group to see how the cost is decomposed within each process group.

Solution 11.4 It might be the case that parts of the production process are administered using different information systems. Accordingly, the event logs have to be integrated. In terms of correlation, this means that case identifiers from different systems have to be matched. If the timestamps of the events are recorded in different time zones, they have to be harmonized. The shipment might not be arranged by Airbus. Therefore, it might be the case that different snapshots of transportation might not be accessible. Also, information systems might not be directly producing case-related event logs. The data would then have to be extracted from the databases of these systems. Finally, the events might be recorded at diverging levels of granularity, ranging from a detailed record of production steps to coarse-granular records of transportation stages.

Solution 11.5 The workflow log considers the order of events for each case. We use letters a to l for referring to the tasks. The workflow log L contains three traces as the first and the fourth trace are the same. Therefore, we get $L = [\langle a, b, g, h, j, k, i, l\rangle, \langle a, c, d, e, f, g, j, h, i, k, l\rangle, 2 \times \langle a, c, f, g, j, h, i, k, l\rangle]$.

Solution 11.6



Solution 11.7

A. 31,509 cases.
B. 21.9 days.
C. 17,228 cases, 18.1 days.

D. 3,720 cases, 16.8 days.
E. 4,436 cases, 23 days.

Note that to answer questions C and D, we need to apply a filter that retains only cases where O_Accepted and O_Refused occur, whereas to answer question E we need to apply a filter that retains only those cases that finish with O_Cancelled. The latter can be achieved with an "endpoint" filter in Disco or an *activity selection* filter with a *case ends with* constraint in Celonis.

Solution 11.8 The following basic relations can be observed:

$a > b$	$h > j$	$i > l$	$d > e$	$g > j$	$i > k$
$b > g$	$j > k$	$a > c$	$e > f$	$j > h$	$k > l$
$g > h$	$k > i$	$c > d$	$f > g$	$h > i$	$c > f$

The following matrix shows the resulting relations.

	a	b	c	d	e	f	g	h	i	j	k	l
a	#	$\rightarrow$	$\rightarrow$	#	#	#	#	#	#	#	#	#
b	$\leftarrow$	#	#	#	#	#	$\rightarrow$	#	#	#	#	#
c	$\leftarrow$	#	#	$\rightarrow$	#	$\rightarrow$	#	#	#	#	#	#
d	#	#	$\leftarrow$	#	$\rightarrow$	#	#	#	#	#	#	#
e	#	#	#	$\leftarrow$	#	$\rightarrow$	#	#	#	#	#	#
f	#	#	$\leftarrow$	#	$\leftarrow$	#	$\rightarrow$	#	#	#	#	#
g	#	$\leftarrow$	#	#	#	$\leftarrow$	#	$\rightarrow$	#	$\rightarrow$	#	#
h	#	#	#	#	#	#	$\leftarrow$	#	$\rightarrow$	$\|$	#	#
i	#	#	#	#	#	#	#	$\leftarrow$	#	#	$\|$	$\rightarrow$
j	#	#	#	#	#	#	$\leftarrow$	$\|$	#	#	$\rightarrow$	#
k	#	#	#	#	#	#	#	#	$\|$	$\leftarrow$	#	$\rightarrow$
l	#	#	#	#	#	#	#	#	$\leftarrow$	#	$\leftarrow$	#

Solution 11.9 The α-algorithm stepwise yields the following sets:

1. $T_L = \{a, b, c, d, e, f, g, h, i, j, k, l\}$.
2. $T_I = \{a\}$.
3. $T_O = \{l\}$.
4. $X_L = Z_1 \cup Z_2$ with
   $Z_1 = \{a \rightarrow b, a \rightarrow c, b \rightarrow g, c \rightarrow d, c \rightarrow f, d \rightarrow e, e \rightarrow f, f \rightarrow g, g \rightarrow h, g \rightarrow j, h \rightarrow i, i \rightarrow l, j \rightarrow k, k \rightarrow l\}$ and
   $Z_2 = \{a \rightarrow (b\#c), c \rightarrow (d\#f), (c\#e) \rightarrow f, (b\#f) \rightarrow g\}$
5. $Y_L = Z_2 \cup \{d \rightarrow e, g \rightarrow h, g \rightarrow j, h \rightarrow i, j \rightarrow k, i \rightarrow l, k \rightarrow l\}$.
6. Add start event pointing to a and end event following after l.
7. Construct process model based on Y_L with XOR and AND gateways.

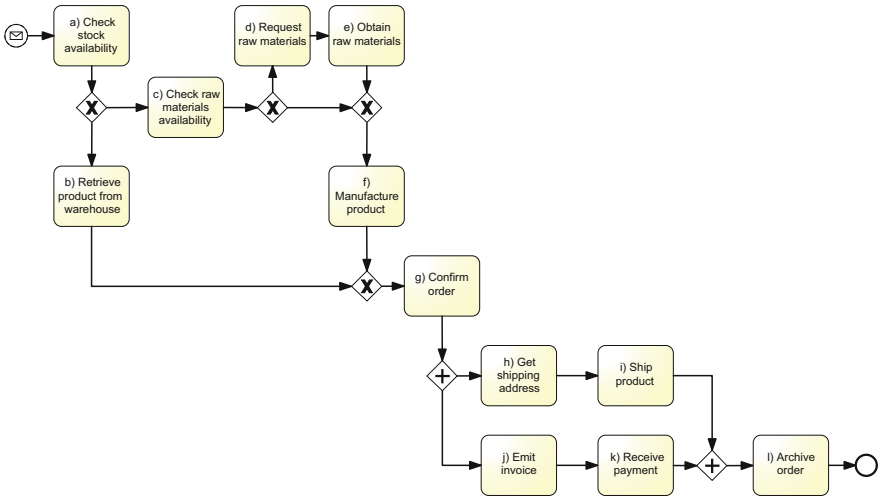


Fig. 11.24 Process model constructed by the α-algorithm

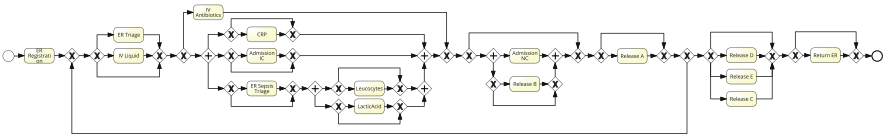


Fig. 11.25 Model discovered by the inductive miner from the Sepsis log

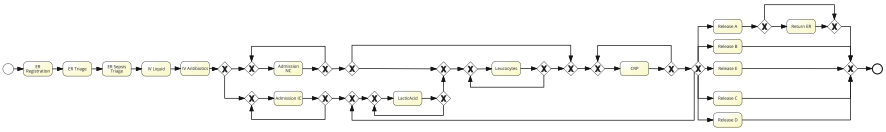


Fig. 11.26 Model discovered by Apromore's split miner from the Sepsis log

8. Return process model, see Figure 11.24.

Solution 11.10 Figures 11.25 and 11.26 show the models discovered from this log using the inductive miner (available in ProM and Apromore) and the split miner (available in Apromore). The model produced by the inductive miner has 50 nodes and is perfectly block-structured as expected. The one produced by the split miner has 31 nodes and it is block-structured except for a cycle with two entry points.

The model produced by the inductive miner exhibits the flower pattern—all but the first task can be skipped or repeated any number of times. Accordingly, it achieves almost perfect fitness (0.99 according to ProM),[27] but at the expense of

[27]The fitness was calculated by translating the BPMN model to a Petri net and checking for log conformance using the "Replay a Log on Petri Net for Conformance Analysis" in ProM v6.5.

low precision (0.48). The model produced by the split miner has a fitness of 0.73 and a precision of 0.86.

The structured heuristics miner gives a very large model when applied on this event log (over 200 nodes).

Solution 11.11 First, we have to determine the cycle time. Case 1 takes roughly 2 h less than 7 days, case 2 1 h less than 6 days, case 3 takes 4 days and 20 min, and case 4 takes 4 days and 6 h. Therefore, the relative order must be case 3, case 4, case 2 and case 1. Each event has to be visualized according to the time elapsed since the first event of the case.

Solution 11.12 The longest waiting time is the one between "Analyze defect" and "Inform user" (9.7 days on average). The longest processing time is that of "Repair (complex)" (11.6 days). Note: It might be, however, that in reality this processing time includes some "waiting time" in the sense that a worker might start working on a work item W of this task, then stop and move on to other work items, and then come back to item W.

Solution 11.13 In general, it is preferable to include all expenses in the cost calculation as it provides transparency on what is spent where. In this case, however, this might not be a good idea since the cost per paper is relatively low (€ 0.02). It has to be kept in mind though that the decision whether to record certain costs should not be governed by the unit piece, but by the relative impact on cost calculation. A cost of paper of € 0.02 is small as compared to overall process costs of several thousand Euros. However, it can be relatively high if the overall process produces € 0.30 costs per instance and millions of instances are processed per day. Think of the situation of processing bank transactions using paper forms versus using online banking. The high amount of transactions per day might result in considerable costs per year.

Solution 11.14 The formula yields $r = 1 - \frac{T}{CT} = 1 - \frac{18.3}{27.5} = 0.333$. This result is apparently misleading because every second task required rework. However, the rework time is in general much smaller than the first-try duration. Hence, T appears to be relatively small in comparison to CT, which results in a low value for r.

Solution 11.15 In Celonis, this question can be answered by applying a *rework* filter that retains only cases where event "W_Assess Potential Fraud - complete" occurs at least twice in the same case. The answer given by Celonis is 13 cases. In Disco, this question can be answered by applying a *follower* filter that retains only those cases where an event of type "W_Assess Potential Fraud" is eventually followed by another event of type "W_Assess Potential Fraud". In the version of Disco we tested (version 2.0), this filter retrieved 20 cases. This is because Disco also counts cases where "W_Assess Potential Fraud" was scheduled or started twice or more, but was not completed twice or more (it counts cases where this task was aborted).

Solution 11.16 We have to consider four different cases. However, as the first and the fourth case show the same sequence of tasks, there are three distinct executions.

Solution 11.17 There are several tasks that can be observed in all cases. These mandatory tasks are a, g, h, i, j, k, l. The other tasks b, c, d, e, f are not mandatory. Exclusiveness relationships hold between b and c, d, e, f pairwise.

Solution 11.18 In Disco, this question can be answered by applying an event pair filter ("follower" filter) that retains only those cases where an event of type "W_Assess Potential Fraud" occurs after another event of type "W_Assess Potential Fraud" and such that both events refer to the same resource. In our test using Disco v2.0, the answer is 8 cases. In Celonis, this question can be answered by opening an analysis of type "Case Explorer" on this event log, applying a filter that retains all cases where "W_Assess Potential Fraud - complete" occurs twice, and inspecting each of the 13 cases one by one, to determine which resource performed "W_Assess Potential Fraud - complete". The result is 3 cases. It is also possible in Celonis to textually specify a filter that extracts all cases where at least two distinct resources have performed an event of type "W_Assess potential fraud - complete". This latter approach requires familiarity with the filter specification language of Celonis.

Solution 11.19

1. L1: 54.8 days; L2: 25 days.
2. In log L2, task "Contact Hospital" is executed four times less often than in log L1 (130 times versus 518 times). Also, in log L1, task "Contact Hospital" is executed in parallel with "High Medical History" whereas in log L2, "Contact Hospital" is executed always after "High Medical History".
3. In log L1, there are waiting times of over 30 days between sending the notification and sending the questionnaire. These waiting times are of around 16–19 h in log L2. Also, in L2, there are waiting times of around 21 days before "Receive Questionnaire" and "Archive" (and similar waiting times between "Skip questionnaire" and "Achive"). These waiting times are in the order of 10 days in L2. On the other hand, in log L2, there is a waiting time of around 28 days between "Register" and "Create questionnaire", whereas this waiting time is around 58 min in L1. This bottleneck however only affects about a third of the cases in L2.

11.11 Further Exercises

Exercise 11.20 Consider the following event log. Write the corresponding workflow log using the same approach as in Figure 11.7 (page 427). You may use abbreviations CF, SB, SR, and CC to denote the four tasks in this process.

Exercise 11.21 Draw the dependency graph of the workflow log obtained in Exercise 11.20.

Exercise 11.22 With reference to the event log of the Business Process Intelligence Challenge 2017 (https://tinyurl.com/bpic2017), does it sometimes happen that a

Case ID	Task Label	Timestamp
1	Create Fine	19-04-2017 14:00:00
2	Create Fine	19-04-2017 15:00:00
1	Send Bill	19-04-2017 15:05:00
2	Send Bill	19-04-2017 15:07:00
3	Create Fine	20-04-2017 10:00:00
3	Send Bill	20-04-2017 14:00:00
4	Create Fine	21-04-2017 11:00:00
4	Send Bill	21-04-2017 11:10:00
1	Process Payment	24-04-2017 14:30:00
1	Close Case	24-04-2017 14:32:00
2	Send Reminder	19-04-2017 10:00:00
3	Send Reminder	20-05-2017 10:00:00
2	Process Payment	22-05-2017 09:05:00
2	Close Case	22-05-2017 09:06:00
4	Send Reminder	21-05-2017 15:10:00
4	Send Reminder	21-05-2017 17:10:00
4	Process Payment	26-05-2017 14:30:00
4	Close Case	26-05-2017 14:31:00
3	Send Reminder	20-06-2017 10:00:00
3	Send Reminder	20-07-2017 10:00:00
3	Process Payment	25-07-2017 14:00:00
3	Close Case	25-07-2017 14:01:00

loan offer is canceled (meaning that event "O_Cancelled" occurs), but later the offer is accepted (i.e., "O_Accepted")? If yes, in what percentage of cases does this happen?

Exercise 11.23 Consider the workflow log $[\langle a, b, c, d, e\rangle, \langle a, b, d, c, e\rangle, \langle a, c, d, b, e\rangle, \langle a, d, c, b, e\rangle, \langle a, d, b, c, e\rangle, \langle a, d, c, b, e\rangle]$. Show step-by-step how the α-algorithm works on this workflow log, and draw the resulting process model.

Exercise 11.24 Consider the log in Exercise 11.20 (page 470). Show step-by-step how the α-algorithm works on this log, and draw the resulting process model.

Exercise 11.25 Draw a model in BPMN that can produce every possible execution sequence that includes tasks $\{a, b, c, d, e\}$. Discuss the fitness and precision of this model with respect to the workflow log $\{\langle a, b, c, d, e\rangle, \langle a, c, b, d, e\rangle\}$. Draw a process model that would have perfect fitness and perfect precision with respect to this workflow log (i.e., a model that can produce exactly this log). Would the α-algorithm produce this model with perfect fitness and precision?

Exercise 11.26 Using a process mining tool, discover a BPMN process model from the following event log of an authentication process: http://tinyurl.com/simpleEventLog (also available in the book's companion website).

Note: This log is sufficiently simple that it is possible to manually derive the BPMN process model from a dependency graph.

Exercise 11.27 Using a process mining tool, discover a BPMN process model from the following event log of a telephone repair process: http://tinyurl.com/repairLogs (also available in the book's companion website).
Note: This log is more complex and difficult to manually understand using a dependency graph. Consider using a process mining tool that can discover BPMN process models (e.g., ProM or Apromore).

Exercise 11.28 Consider there is an AND-split in a process model with two subsequent tasks a and b. What kind of pattern do these tasks show on the timeline chart?

Exercise 11.29 Consider the workflow log that you created for Exercise 11.5 (page 427) from the cases shown in Figure 11.4. Replay these logs in the process model of Figure 11.21 (page 456). Note down consumed, produced, missing and remaining tokens for each arc, and calculate the fitness measure. Assume that tasks not shown in the process model do not change the distribution of tokens in the replay.

Exercise 11.30 Do the models you discovered in Exercises 11.26 and 11.27 perfectly fit the corresponding event log? If not, describe to what extent and how the event log differs with respect to the discovered model.

Exercise 11.31 We consider again the health insurance process examined in Exercise 11.19 on page 460 (available at http://tinyurl.com/InsuranceLogs) and specifically the log containing the 2011 cases (log L1). Based on this log, complete the following tasks using a process mining tool:

1. Extract from the log all the cases containing at least one occurrence of high insurance check. The filtered log will be called FilteredHigh.
2. Extract from the log all the cases containing at least one occurrence of low insurance check. The filtered low will be called FilteredLow.
3. Compare the mean cycle time of cases of FilteredHigh and FilteredLow.
4. Describe the main differences between FilteredHigh and FilteredLow in terms of the frequency and relative order of tasks. Hint: You may use a log delta analysis tool or you may derive a dependency graph for FilteredHigh and another for FilteredLow, and abstract away all infrequent behaviors in the dependency graph (i.e., set the task and the arc abstraction sliders to their minimum).

11.12 Further Readings

The question of selecting process performance measures is crucial when setting up a process monitoring system. Leyer et al. [88] propose a procedure for defining performance measures for a given business process. They also show how to use process mining techniques to calculate performance measures from event logs.

Harmon's book [64] gives a broad perspective on how to define process performance measures at the level of an entire organization, and how to set up a governance framework to maintain these performance measures.

There are two complementary questions when designing performance dashboards. The first one is how to go from the definition of the performance objectives down to a sketch of a dashboard. This question is thoroughly covered in Eckerson's book [41]. The second question is how to design a performance dashboard that is understandable and compelling. To this end, we recommend following the guidelines on storytelling with data given in [80] and other data visualization guidebooks.

A comprehensive overview of existing techniques and research challenges in the field of process mining is given in Van der Aalst's book on process mining [1]. A more concise overview can be found in the process mining manifesto [70] composed by the IEEE Task Force on Process Mining. The website of the IEEE Task Force on Process Mining provides datasets (event logs) and case studies that illustrate how process mining is used in a range of industries.[28] A survey and comparative evaluation of automated process discovery techniques is provided in [12], while an overview of conformance checking techniques is given in [118].

In Section 11.8, we discussed a case study in which an insurance company applied a question-driven process mining approach. When applying process mining in a large organization, it is desirable to adopt a systematic approach (i.e., a method). Two methods for conducting process mining projects are presented in [180] and [5].

[28] http://www.win.tue.nl/ieeetfpm.