Chapter 5
Getting the Data

Process mining is impossible without proper event logs. This chapter describes the information that should be present in such event logs. Depending on the process mining technique used, these requirements may vary. The challenge is to extract such data from a variety of data sources, e.g., databases, flat files, message logs, transaction logs, ERP systems, and document management systems. When merging and extracting data, both syntax and semantics play an important role. Moreover, depending on the questions one seeks to answer, different views on the available data are needed. Process mining, like any other data-driven analysis approach, needs to deal with data quality problems. We discuss typical data quality challenges encountered in reality. The insights provided in this chapter help to get the event data assumed to be present in later chapters.

5.1 Data Sources

In Chap. 2, we introduced the concept of process mining. The idea is to analyze event data from a process-oriented perspective. The goal of process mining is to answer questions about operational processes. Examples are:

- What *really* happened in the past?
- Why did it happen?
- What is likely to happen in the future?
- When and why do organizations and people deviate?
- How to control a process better?
- How to redesign a process to improve its performance?

In subsequent chapters, we will discuss various techniques to answer the preceding questions. However, first we focus on the event data needed.

Figure 5.1 shows the overall "process mining workflow" emphasizing the role of event data. Starting point is the "raw" data hidden in all kinds of data sources. A data source may be a simple flat file, an Excel spreadsheet, a transaction log,

W. van der Aalst, *Process Mining*, DOI 10.1007/978-3-662-49851-4_5

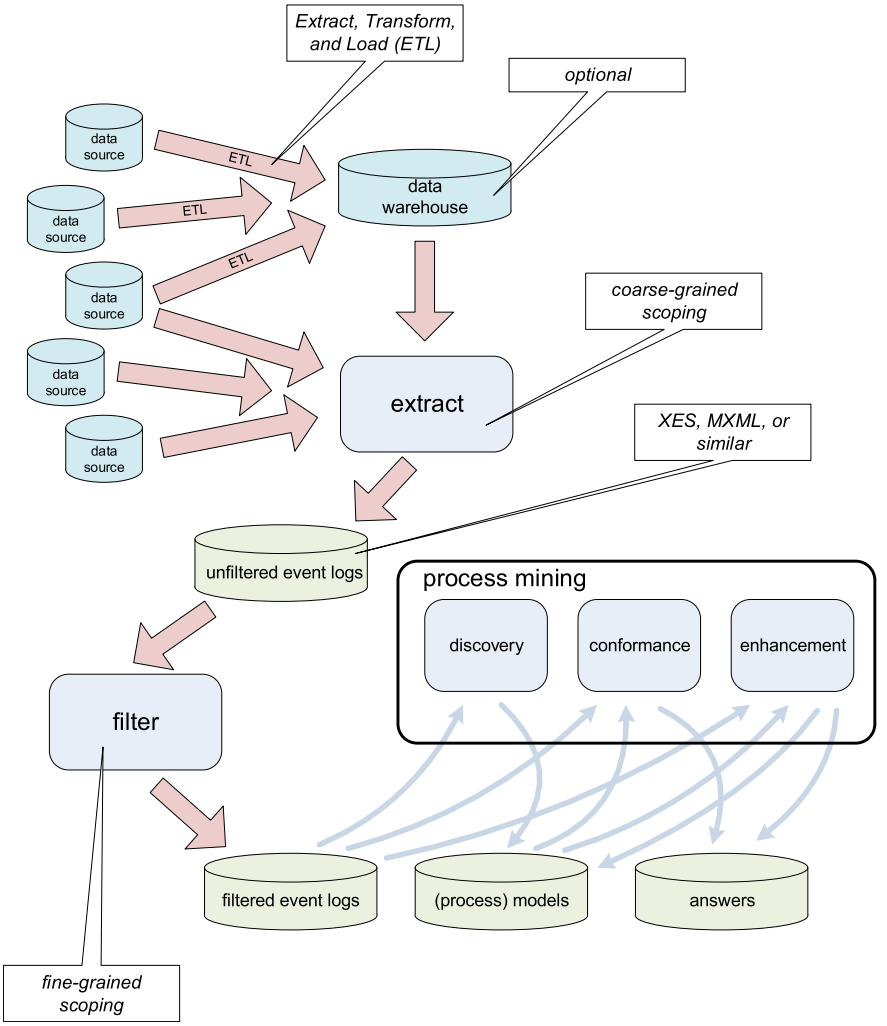

Fig. 5.1 Overview describing the workflow of getting from heterogeneous data sources to process mining results

or a database table. However, one should not expect all the data to be in a single well-structured data source. The reality is that event data is typically scattered over different data sources and often quite some efforts are needed to collect the relevant data. Consider, for example, a full SAP implementation that typically has more than 10,000 tables. Data may be scattered due to technical or organizational reasons. For example, there may be legacy systems holding crucial data or information systems used only at the departmental level. For cross-organizational process mining, e.g., to analyze supply chains, data may even be scattered over multiple organizations. Events can also be captured by tapping of message exchanges [161] (e.g., SOAP

messages) and recording read and write actions [47]. Data sources may be structured and well-described by meta data. Unfortunately, in many situations the data is unstructured or important meta data is missing. Data may originate from web pages, emails, PDF documents, scanned text, screen scraping, etc. Even if data is structured and described by meta data, the sheer complexity of enterprise information systems may be overwhelming, There is no point in trying to exhaustively extract event logs from thousands of tables and other data sources. Data extraction should be driven by questions rather than the availability of lots of data.

In the context of BI and data mining, the phrase *"Extract, Transform, and Load"* (ETL) is used to describe the process that involves: (a) *extracting* data from outside sources, (b) *transforming* it to fit operational needs (dealing with syntactical and semantical issues while ensuring predefined quality levels), and (c) *loading* it into the target system, e.g., a data warehouse or relational database. A *data warehouse* is a single logical repository of an organization's transactional and operational data. The data warehouse does not produce data but simply taps off data from operational systems. The goal is to unify information such that it can be used for reporting, analysis, forecasting, etc. Figure 5.1 shows that ETL activities can be used to populate a data warehouse. It may require quite some efforts to create the common view required for a data warehouse. Different data sources may use different keys, formatting conventions, etc. For example, one data source may identify a patient by her last name and birth date while another data source uses her social security number. One data source may use the date format "31-12-2010" whereas another uses the format "2010/12/31".

If a data warehouse already exists, it most likely holds valuable input for process mining. However, many organizations do not have a good data warehouse. The warehouse may contain only a subset of the information needed for end-to-end process mining, e.g., only data related to customers is stored. Moreover, if a data warehouse is present, it does not need to be process oriented. For example, the typical warehouse data used for *Online Analytical Processing* (OLAP) does not provide much process-related information. OLAP tools are excellent for viewing multidimensional data from different angles, drilling down, and for creating all kinds of reports (see Sect. 12.4). However, OLAP tools do not require the storage of business events and their ordering. The data sets used by the mainstream data mining approaches described in Chap. 4 also do not store such information. For example, a decision tree learner can be applied to any table consisting of rows (instances) and columns (variables). As will be shown in the next section, process mining requires information on relevant events and their order.

Whether there is a data warehouse or not, data needs to be extracted and converted into event logs. Here, *scoping* is of the utmost importance. Often the problem is not the syntactical conversion but the selection of suitable data. Questions like "Which of the more than 10,000 SAP tables to convert?" need to be answered first. Typical formats to store event logs are *XES* (eXtensible Event Stream) and *MXML* (Mining eXtensible Markup Language). These will be discussed in Sect. 5.3. For the moment, we assume that *one event log corresponds to one process*, i.e., when scoping the data in the extraction step, only events relevant for the process to be analyzed

should be included. In Sect. 5.5, we discuss the problem of converting "3-D data" into "2-D event logs", i.e., events are projected onto the desired process model.

Depending on the questions and viewpoint chosen, different event logs may be extracted from the same data set. Consider for example the data in a hospital. One may be interested in the discovery of patient flows, i.e., typical diagnosis and treatment paths. However, one may also be interested in optimizing the workflow within the radiology department. Both questions require different event logs, although some events may be shared among the two required event logs. Once an event log is created, it is typically *filtered*. Filtering is an iterative process. *Coarse-grained scoping* was done when extracting the data into an event log. Filtering corresponds to *fine-grained scoping* based on initial analysis results. For example, for process discovery one can decide to focus on the 10 most frequent activities to keep the model manageable.

Based on the filtered log, the different types of process mining described in Sect. 2.2 can be applied: *discovery*, *conformance*, and *enhancement*.

Although Fig. 5.1 does not reflect the iterative nature of the whole process well, it should be noted that process mining results most likely trigger new questions and these questions may lead to the exploration of new data sources and more detailed data extractions. Typically, several iterations of the extraction, filtering, and mining phases are needed.

5.2 Event Logs

Table 5.1 shows a fragment of the event log already discussed in Chap. 2. This table illustrates the typical information present in an event log used for process mining. The table shows events related to the handling of requests for compensation. We assume that an event log contains data related to a *single process*, i.e., the first coarse-grained scoping step in Fig. 5.1 should make sure that all events can be related to this process. Moreover, each event in the log needs to refer to a *single process instance*, often referred to as *case*. In Table 5.1, each request corresponds to a case, e.g., case 1. We also assume that events can be related to some *activity*. In Table 5.1, events refer to activities like *register request*, *check ticket*, and *reject*. These assumptions are quite natural in the context of process mining. All mainstream process modeling notations, including the ones discussed in Chap. 3, specify a process as a collection of activities such that the life-cycle of a single instance is described. Hence, the "case id" and "activity" columns in Table 5.1 represent the bare minimum for process mining. Moreover, events within a case need to be ordered. For example, event 35654423 (the execution of activity *register request* for Case 1) occurs before event 35654424 (the execution of activity *examine thoroughly* for the same case). Without ordering information it is of course impossible to discover causal dependencies in process models.

Table 5.1 also shows additional information per event. For example, all events have a *timestamp* (i.e., date and time information such as "30-12-2010:11.02"). This

Table 5.1 A fragment of some event log: each line corresponds to an event

Case id	Event id	Properties				
		Timestamp	Activity	Resource	Cost	...
1	35654423	30-12-2010:11.02	register request	Pete	50	...
	35654424	31-12-2010:10.06	examine thoroughly	Sue	400	...
	35654425	05-01-2011:15.12	check ticket	Mike	100	...
	35654426	06-01-2011:11.18	decide	Sara	200	...
	35654427	07-01-2011:14.24	reject request	Pete	200	...
2	35654483	30-12-2010:11.32	register request	Mike	50	...
	35654485	30-12-2010:12.12	check ticket	Mike	100	...
	35654487	30-12-2010:14.16	examine casually	Pete	400	...
	35654488	05-01-2011:11.22	decide	Sara	200	...
	35654489	08-01-2011:12.05	pay compensation	Ellen	200	...
3	35654521	30-12-2010:14.32	register request	Pete	50	...
	35654522	30-12-2010:15.06	examine casually	Mike	400	...
	35654524	30-12-2010:16.34	check ticket	Ellen	100	...
	35654525	06-01-2011:09.18	decide	Sara	200	...
	35654526	06-01-2011:12.18	reinitiate request	Sara	200	...
	35654527	06-01-2011:13.06	examine thoroughly	Sean	400	...
	35654530	08-01-2011:11.43	check ticket	Pete	100	...
	35654531	09-01-2011:09.55	decide	Sara	200	...
	35654533	15-01-2011:10.45	pay compensation	Ellen	200	...
4	35654641	06-01-2011:15.02	register request	Pete	50	...
	35654643	07-01-2011:12.06	check ticket	Mike	100	...
	35654644	08-01-2011:14.43	examine thoroughly	Sean	400	...
	35654645	09-01-2011:12.02	decide	Sara	200	...
	35654647	12-01-2011:15.44	reject request	Ellen	200	...
...	...	...	...	...	...	...

information is useful when analyzing performance related properties, e.g., the waiting time between two activities. The events in Table 5.1 also refer to *resources*, i.e., the persons executing the activities. Also *costs* are associated to events. In the context of process mining, these properties are referred to as *attributes*. These attributes are similar to the notion of variables in Chap. 4.

Figure 5.2 shows the tree structure of an event log. Using this figure we can list our assumptions about event logs.

- A *process* consists of *cases*.
- A case consists of *events* such that each event relates to precisely one case.
- Events within a case are *ordered*.

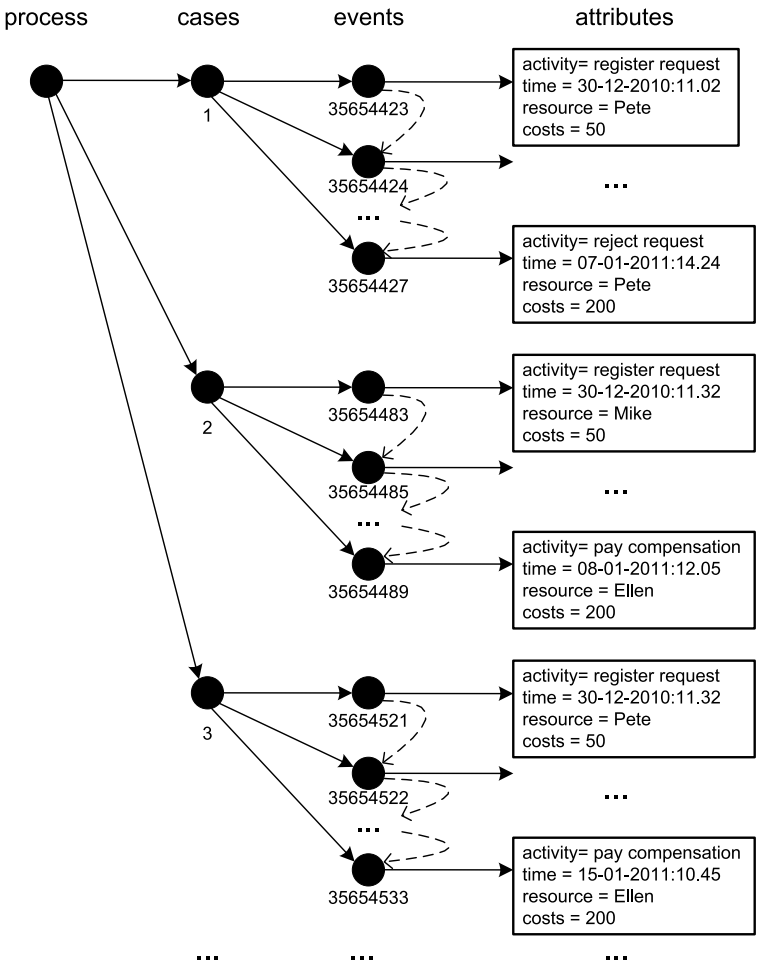

Fig. 5.2 Structure of event logs

- Events can have *attributes*. Examples of typical attribute names are activity, time, costs, and resource.

Not all events need to have the same set of attributes. However, typically, events referring to the same activity have the same set of attributes.

To be able to reason about logs and to precisely specify the requirements for event logs, we formalize the various notions.

Definition 5.1 (Event, attribute) Let $\mathscr{E}$ be the *event universe*, i.e., the set of all possible event identifiers. Events may be characterized by various *attributes*, e.g., an event may have a timestamp, correspond to an activity, is executed by a particular person, has associated costs, etc. Let AN be a set of attribute names. For any event

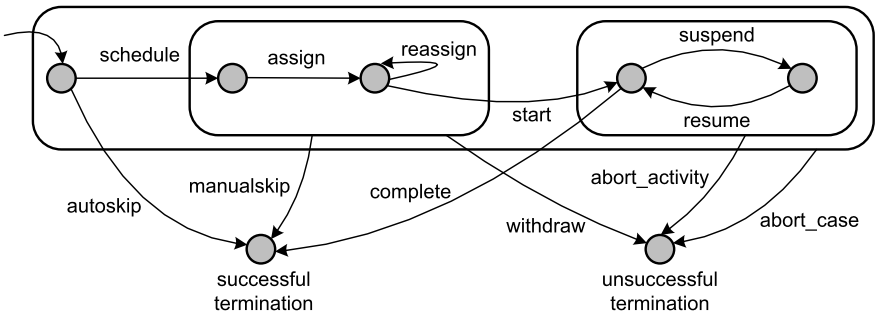

Fig. 5.3 Standard transactional life-cycle model

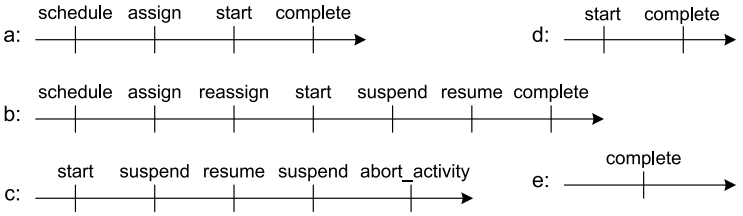

Fig. 5.4 Transactional events for five activity instances

$e \in \mathscr{E}$ and name $n \in AN$, $\#_n(e)$ is the value of attribute n for event e. If event e does not have an attribute named n, then $\#_n(e) = \perp$ (null value).

For convenience we assume the following standard attributes:

- $\#_{activity}(e)$ is the *activity* associated to event e.
- $\#_{time}(e)$ is the *timestamp* of event e.
- $\#_{resource}(e)$ is the *resource* associated to event e.
- $\#_{trans}(e)$ is the *transaction type* associated to event e, examples are schedule, start, complete, and suspend.

These are just examples. None of these attributes is mandatory. However, for these standard attributes we will assume some conventions. For example, timestamps should be non-descending in the event log. Moreover, we assume a time domain $\mathscr{T}$, i.e., $\#_{time}(e) \in \mathscr{T}$ for any $e \in \mathscr{E}$. The transaction type attribute $\#_{trans}(e)$ refers to the life-cycle of activities. In most situations, activities take time. Therefore, events may point out for example the start or completion of activities. In this book, we assume the *transactional life-cycle model* shown in Fig. 5.3.

Figure 5.4 shows some examples to explain the life-cycle model. The life-cycles of five activity instances are shown: a, b, c, d, and e. a is first scheduled for execution (i.e., an event e_1 with $\#_{trans}(e_1) = schedule$ and $\#_{activity}(e_1) = a$ occurs), then the activity is assigned to a resource (i.e., an event e_2 with $\#_{trans}(e_2) = assign$ and $\#_{activity}(e_2) = a$ occurs). Later the activity is started by this resource, and finally the

activity completes. Note that four events were recorded for this activity instance. Activity instance *b* has seven events associated to it. Compared to *a* the activity is reassigned (i.e., the resource that is supposed to execute the activity is changed), suspended (temporarily halted), and resumed. Of course it is possible to skip stages in the transactional life-cycle model, because events are not recorded or because certain steps are not necessary. Activity instance *d* in Fig. 5.4 has just two events; *e* just one, i.e., for *e* only the completion of the activity instance is recorded. Transaction type "autoskip" refers to an action by the system bypassing the activity. Transaction type "manualskip" refers to resource initiated skipping. Transaction types "abort_activity" and "abort_case" correspond to aborting the activity or the whole case. A "withdraw" event signals the situation in which the activity is canceled before it was started. Figure 5.3 shows all transaction types, their enabling, and their effect. For example, according to the transactional life-cycle model, "abort_activity" is only possible when the activity instance is running (i.e., started, suspended, or resumed).

Events can have many attributes. We often refer to the event by its activity name. Technically this is not correct. There may be many events that refer to the same activity name. Within a case these events may refer to the same activity instance (e.g., start and complete events) or different activity instances (e.g., in a loop). This distinction is particularly important when measuring service times, waiting times, etc. Consider, for example, the scenario in which the same activity is started twice for the same case, i.e., two activity instances are running in parallel, and then one of them completes. Did the activity that was started first complete or the second one? Fig. 5.5 illustrates the dilemma. Given the footprint of two starts followed by two completes of the same activity, there are two possible scenarios. In one scenario the durations of the two activity instances are 5 and 6. In the other scenario the durations of the activity instances are 9 and 2. Yet they leave the same footprint in the event log.

This problem can be addressed by adding information to the log or by using heuristics. This can be seen as a "secondary correlation problem", i.e., relating two events within the same case. The primary correlation problem is to relate events to cases, i.e., process instances [53]. Figure 5.5 shows that even within one case there may be the need to correlate events because they belong to the same activity instance. When implementing systems, such information can easily be added to the logs; just provide an activity instance attribute to keep track of this. When dealing with existing systems this is not as simple as it seems. For example, when correlating messages between organizations there may be the need to scan the content of the message to find a suitable identifier (e.g., address or name). It is also possible to use heuristics to resolve most problems, e.g., in Fig. 5.5 one could just assume a first-in-first-out order and pick the first scenario. Moreover, one may introduce timeouts when the time between a start event and complete event is too long. For example, start events that are not followed by a corresponding complete event within 45 minutes are removed from the log.

Process mining techniques can be used to automatically discover process models. In these process models activities play a central role. These correspond to transitions

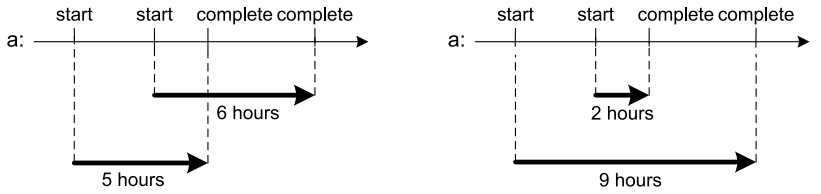

Fig. 5.5 Two scenarios involving two activity instance leaving the same footprint in the log

in Petri nets, tasks in YAWL, functions in EPCs, state transitions in transition systems, and tasks in BPMN. However, the transactional life-cycle model in Fig. 5.3 shows that there may be multiple events referring to the same activity. Some process mining techniques take into account the transactional model whereas others just consider atomic events. Moreover, sometimes we just want to focus on complete events whereas at other times the focus may be on withdrawals. This can be supported by filtering (e.g., removing events of a particular type) and by the concept of a *classifier*. A *classifier is a function that maps the attributes of an event onto a label* used in the resulting process model. This can be seen as the "name" of the event. In principle there can be many classifiers. However, only one is used at a time. Therefore, we can use the notation $\underline{e}$ to refer to the name used in the process model.

Definition 5.2 (Classifier) For any event $e \in \mathscr{E}$, $\underline{e}$ is the *name* of the event.

If events are simply identified by their activity name, then $\underline{e} = \#_{activity}(e)$. This means that activity instance a in Fig. 5.4 would be mapped onto $\langle a, a, a, a \rangle$. In this case the basic α-algorithm (not using transactional information) would create just one a transition. If events are identified by their activity name and transaction type, then $\underline{e} = (\#_{activity}(e), \#_{trans}(e))$. Now activity instance a would be mapped onto $\langle (a, schedule), (a, assign), (a, start), (a, complete) \rangle$ and the basic α-algorithm would create four transitions referring to a's life-cycle. As shown in Sect. 6.2.4, transaction type attributes such as start, complete, etc. can be exploited to create a two-level process model that hides the transactional life-cycles of individual activities in subprocesses. It is also possible to use a completely different classifier, e.g., $\underline{e} = \#_{resource}(e)$. In this case events are named after the resources executing them. In this book we assume the classifier $\underline{e} = \#_{activity}(e)$ as the *default classifier*. This is why we considered the activity attribute to be mandatory in our initial examples. From now on, we only require a classifier.

Sequences
Sequences are the most natural way to present traces in an event log. When describing the operational semantics of Petri nets and transition systems, we also modeled behavior in terms of sequences. Given their importance, we introduce some useful operators on sequences.

For a given set A, A^* is the set of all finite sequences over A. A finite sequence over A of length n is a mapping $\sigma \in \{1, \ldots, n\} \rightarrow A$. Such a sequence is represented by a string, i.e., $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ where $a_i = \sigma(i)$ for $1 \leq i \leq n$. $|\sigma|$ denotes the length of the sequence, i.e., $|\sigma| = n$. $\sigma \oplus a' = \langle a_1, \ldots, a_n, a' \rangle$ is the sequence with element a' appended at the end. Similarly, $\sigma_1 \oplus \sigma_2$ appends sequence σ_2 to σ_1 resulting a sequence of length $|\sigma_1| + |\sigma_2|$.

$hd^k(\sigma) = \langle a_1, a_2, \ldots, a_{k\,min\,n} \rangle$, i.e., the "head" of the sequence consisting of the first k elements (if possible). Note that $hd^0(\sigma)$ is the empty sequence and for $k \geq n$: $hd^k(\sigma) = \sigma$. $pref(\sigma) = \{hd^k(\sigma) \mid 0 \leq k \leq n\}$ is the set of prefixes of σ.

$tl^k(\sigma) = \langle a_{(n-k+1)\,max\,1}, a_{k+2}, \ldots, a_n \rangle$, i.e., the "tail" of the sequence composed of the last k elements (if possible). Note that $tl^0(\sigma)$ is the empty sequence and for $k \geq n$: $tl^k(\sigma) = \sigma$.

$\sigma \uparrow X$ is the projection of σ onto some subset $X \subseteq A$, e.g., $\langle a, b, c, a, b, c, d \rangle \uparrow \{a, b\} = \langle a, b, a, b \rangle$ and $\langle d, a, a, a, a, a, a, d \rangle \uparrow \{d\} = \langle d, d \rangle$.

For any sequence $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ over A, $\partial_{set}(\sigma) = \{a_1, a_2, \ldots, a_n\}$ and $\partial_{multiset}(\sigma) = [a_1, a_2, \ldots, a_n]$. ∂_{set} converts a sequence into a set, e.g., $\partial_{set}(\langle d, a, a, a, a, a, a, d \rangle) = \{a, d\}$. a is an element of σ, denoted as $a \in \sigma$, if and only if $a \in \partial_{set}(\sigma)$. $\partial_{multiset}$ converts a sequence into a multi-set, e.g., $\partial_{multiset}(\langle d, a, a, a, a, a, a, d \rangle) = [a^6, d^2]$. $\partial_{multiset}(\sigma)$ is also known as the *Parikh vector* of σ. These conversions allow us to treat sequences as sets or bags when needed.

An event log consists of cases and cases consist of events. The events for a case are represented in the form of a *trace*, i.e., a sequence of unique events. Moreover, cases, like events, can have attributes.

Definition 5.3 (Case, trace, event log) Let $\mathscr{C}$ be the *case universe*, i.e., the set of all possible case identifiers. Cases, like events, have attributes. For any case $c \in \mathscr{C}$ and name $n \in AN$: $\#_n(c)$ is the value of attribute n for case c ($\#_n(c) = \perp$ if case c has no attribute named n). Each case has a special mandatory attribute *trace*, $\#_{trace}(c) \in \mathscr{E}^*$.[1] $\hat{c} = \#_{trace}(c)$ is a shorthand for referring to the trace of a case.

A *trace* is a finite sequence of events $\sigma \in \mathscr{E}^*$ such that each event appears only once, i.e., for $1 \leq i < j \leq |\sigma|$: $\sigma(i) \neq \sigma(j)$.

An *event log* is a set of cases $L \subseteq \mathscr{C}$ such that each event appears at most once in the entire log, i.e., for any $c_1, c_2 \in L$ such that $c_1 \neq c_2$: $\partial_{set}(\hat{c_1}) \cap \partial_{set}(\hat{c_2}) = \emptyset$.

If an event log contains timestamps, then the ordering in a trace should respect these timestamps, i.e., for any $c \in L$, i and j such that $1 \leq i < j \leq |\hat{c}|$: $\#_{time}(\hat{c}(i)) \leq \#_{time}(\hat{c}(j))$.

[1] In the remainder, we assume $\#_{trace}(c) \neq \langle\,\rangle$, i.e., traces in a log contain at least one event.

Events and cases are represented using *unique* identifiers. An identifier $e \in \mathscr{E}$ refers to an event and an identifier $c \in \mathscr{C}$ refers to a case. This mechanism allows us to point to a specific event or a specific case. This is important as there may be many events having identical attributes, e.g., start events of some activity a may have been recorded for different cases and even within a case there may be multiple of such events. Similarly, there may be different cases that followed the same path in the process. These identifiers are just a technicality that helps us to point to particular events and cases. Therefore, they do not need to exist in the original data source and may be generated when extracting the data from different data sources.

Events and cases may have any number of attributes. Using the classifier mechanism, each event gets a name. Therefore, we often require events to have an activity attribute. Cases always have a trace attribute; $\hat{c} = \#_{trace}(c)$ is the sequence of events that have been recorded for c.

By formalizing event logs in this way, we *precisely* formulate the *requirements* we impose on event logs *without* discussing a concrete *syntax*. Moreover, we can use this formal representation to query the event log and use it as a starting point for analysis and reasoning. Some examples:

- $\{\#_{activity}(e) \mid c \in L \ \wedge \ e \in \hat{c}\}$ is the set of all activities appearing in log L.
- $\{\#_{resource}(e) \mid c \in L \ \wedge \ e \in \hat{c} \ \wedge \ \#_{trans}(e) = manualskip\}$ is the set of all resources that skipped an activity.
- $\{a \in \mathscr{A} \mid c \in L \ \wedge \ a = \#_{activity}(\hat{c}(1)) \ \wedge \ a = \#_{activity}(\hat{c}(|\hat{c}|))\}$ is the set of all activities that served as start and end activity for the same case.

Table 5.1 defines an event log in the sense of Definition 5.3. $L = \{1, 2, 3, 4, \ldots\}$ is the set of cases shown in Table 5.1. $\hat{1} = \#_{trace}(1) = \langle 35654423, 35654424, 35654425, 35654426, 35654427\rangle$ is the trace of case 1. $\#_{activity}(35654423) =$ *register request* is the activity associated to event 35654423. $\#_{time}(35654423) =$ *30-12-2010:11.02* is the timestamp associated to this event. $\#_{resource}(35654423) =$ *Pete* is the resource doing the registration. $\#_{costs}(35654423) = 50$ are the costs associated to event 35654423. $\#_{activity}(35654424) =$ *examine thoroughly* is the activity associated to second event of case 1. Etc.

Depending on the attributes in the log, different types of analysis are possible. Figure 5.6 sketches possible results. The Petri net can be discovered by just using the activity attribute ($\#_{activity}(e)$). To measure durations of activities, one needs to have a transactional attribute ($\#_{trans}(e)$) to distinguish start from completion, and timestamps ($\#_{time}(e)$). To measure costs, the costs attribute is used ($\#_{costs}(e)$). Figure 5.6 also shows a *role* per activity and a *social network*. These have been discovered using the resource attribute ($\#_{resource}(e)$). For example, activities *decide* and *reinitiate request* require the role *manager* and Sara is the only one having this role. The social network in Fig. 5.6 shows how work is flowing through the organization, e.g., activities done by Sara are often followed by activities of Ellen. The thicker the connecting arc is, the more work is handed over from one person to another.

Table 5.1 happens to show unique id's for both events and cases, i.e., elements of the sets $\mathscr{E} = \{35654423, 35654424, 35654425, 35654426, 35654427, \ldots\}$ (event universe) and $\mathscr{C} = \{1, 2, 3, 4, \ldots\}$ (case universe) are shown *explicitly* in the table.

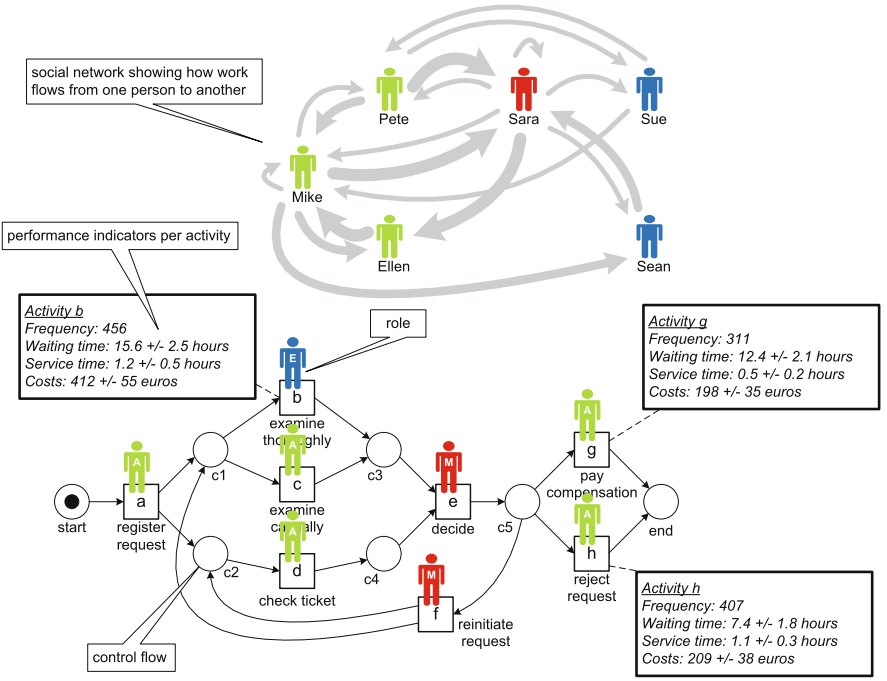

Fig. 5.6 Various types of process mining results based on the attributes in the event log

This is not mandatory; these identities are just used for mathematical convenience and have no further meaning. One can think of them as a symbolic key in a table or a position in an XML document. The reason for adding them is that this way it becomes easy to refer to a particular case or event. In fact, for simple algorithms like the α-algorithm, Definition 5.3 is a bit of overkill. See, for example, Table 2.2 in Sect. 2.3 showing the essential information used to construct a Petri net. If one is just interested in activity names (or some other classifier), the definition can be simplified drastically as is shown next.

Definition 5.4 (Simple event log) Let $\mathscr{A}$ be a set of activity names. A *simple* trace σ is a sequence of activities, i.e., $\sigma \in \mathscr{A}^*$. A *simple* event log L is a multi-set of traces over $\mathscr{A}$, i.e., $L \in \mathbb{B}(\mathscr{A}^*)$.[2]

A simple event log is just a multi-set of traces over some set $\mathscr{A}$. For example $[\langle a,b,c,d\rangle^3, \langle a,c,b,d\rangle^2, \langle a,e,d\rangle]$ defines a log containing 6 cases. In total there are $3 \times 4 + 2 \times 4 + 1 \times 3 = 23$ events. All cases start with a and end with d. In a simple log there are no attributes, e.g., timestamps and resource information are abstracted from. Moreover, cases and events are no longer uniquely identifiable. For

[2]Note that we still assume that each trace contains at least one element, i.e., $\sigma \in L$ implies $\sigma \neq \langle\,\rangle$.

example, the three cases following the sequence $\langle a, b, c, d\rangle$ in the simple event log $[\langle a, b, c, d\rangle^3, \langle a, c, b, d\rangle^2, \langle a, e, d\rangle]$ cannot be distinguished.

Definition 5.5 (Transforming an event log into a simple event log) Let $L \subseteq \mathscr{C}$ be an event log as defined in Definition 5.3. Assume that a classifier has been defined: $\underline{e}$ is the name of event $e \in \mathscr{E}$. This classifier can also be applied to sequences, i.e., $\underline{\langle e_1, e_2, \ldots, e_n\rangle} = \langle \underline{e_1}, \underline{e_2}, \ldots, \underline{e_n}\rangle$. $\underline{L} = [\ \underline{(\hat{c})} \mid c \in L\ ]$ is the simple event log corresponding to L.

All cases in L are converted into sequences of (activity) names using the classifier. A case $c \in L$ is an identifier from the case universe $\mathscr{C}$. $\hat{c} = \#_{trace}(c) = \langle e_1, e_2, \ldots, e_n\rangle \in \mathscr{E}^*$ is the sequence of events executed for c. $\underline{(\hat{c})} = \langle \underline{e_1}, \underline{e_2}, \ldots, \underline{e_n}\rangle$ maps these events onto (activity) names using the classifier.

If we apply this transformation to the event log shown in Table 5.1 while assuming the default classifier ($\underline{e} = \#_{activity}(e)$), then we obtain the event log

$$\begin{aligned}\underline{L} = [&\langle \textit{register request}, \textit{examine thoroughly}, \textit{check ticket}, \textit{decide}, \textit{reject request}\rangle,\\ &\langle \textit{register request}, \textit{check ticket}, \textit{examine casually}, \textit{decide}, \textit{pay compensation}\rangle,\\ &\langle \textit{register request}, \textit{examine casually}, \textit{check ticket}, \textit{decide}, \textit{reinitiate request},\\ &\ \textit{examine thoroughly}, \textit{check ticket}, \textit{decide}, \textit{pay compensation}\rangle,\\ &\langle \textit{register request}, \textit{check ticket}, \textit{examine thoroughly}, \textit{decide}, \textit{reject request}\rangle,\\ &\ldots\ ]\end{aligned}$$

Another classifier could have been used to create a simple log. For example, when using the classifier $\underline{e} = \#_{resource}(e)$, the following log is obtained:

$$\begin{aligned}\underline{L} = [&\langle \textit{Pete}, \textit{Sue}, \textit{Mike}, \textit{Sara}, \textit{Pete}\rangle,\\ &\langle \textit{Mike}, \textit{Mike}, \textit{Pete}, \textit{Sara}, \textit{Ellen}\rangle,\\ &\langle \textit{Pete}, \textit{Mike}, \textit{Ellen}, \textit{Sara}, \textit{Sara}, \textit{Sean}, \textit{Pete}, \textit{Sara}, \textit{Ellen}\rangle,\\ &\langle \textit{Pete}, \textit{Mike}, \textit{Sean}, \textit{Sara}, \textit{Ellen}\rangle,\\ &\ldots\ ]\end{aligned}$$

In this event log, the activity names have been replaced by the names of the people executing the activities. Such projections are used when constructing a social network.

In the remainder, we will use whatever notation is most suitable. Definition 5.3 specifies a precise but very generic description of an event log that can be used for various purposes. Definition 5.4 describes a very simple format without any attributes. This format is useful for explaining simple process discovery algorithms that are not using the information stored in additional attributes. For simple event logs we focus on a single attribute (typically the activity name). As shown, any event log L can be easily converted into a simple event log $\underline{L}$.

5.3 XES

Until 2010 the de facto standard for storing and exchanging event logs was *MXML* (Mining eXtensible Markup Language). MXML emerged in 2003 and was later adopted by the process mining tool ProM. Using MXML it is possible to store event logs such as the one shown in Table 5.1 using an XML-based syntax. *ProMimport* is a tool supporting the conversion of different data sources to MXML, e.g., MS Access, Aris PPM, CSV, Apache, Adept, PeopleSoft, Subversion, SAP R/3, Protos, CPN Tools, Cognos, and Staffware. MXML has a standard notation for storing timestamps, resources, and transaction types. Moreover, one can add arbitrary data elements to events and cases. The latter resulted in ad-hoc extensions of MXML where certain data attributes were interpreted in a specific manner. For example, *SA-MXML* (Semantically Annotated Mining eXtensible Markup Language) is a semantic annotated version of the MXML format used by the ProM framework. SA-MXML incorporates references between elements in logs and concepts in ontologies. For example, a resource can have a reference to a concept in an ontology describing a hierarchy of roles, organizational entities, and positions. To realize these semantic annotations, existing XML elements were interpreted in a new manner. Other extensions were realized in a similar manner. Although this approach worked quite well in practice, the various ad-hoc extensions also revealed shortcomings of the MXML format. This triggered the development of *XES* (eXtensible Event Stream) [64].

XES is the successor of MXML. Based on many practical experiences with MXML, the XES format has been made less restrictive and truly extendible. In September 2010, the format was adopted by the *IEEE Task Force on Process Mining* and became the de facto exchange format for process mining. The IEEE Standards Organization is currently evaluating XES with the aim to turn XES into an official IEEE standard. The format is supported by tools such as ProM (as of version 6), see www.xes-standard.org for detailed information about the standard.

Figure 5.7 shows the XES meta model expressed in terms of a UML class diagram. A XES document (i.e., XML file) contains one log consisting of any number of traces. Each trace describes a sequential list of events corresponding to a particular case. The log, its traces, and its events may have any number of attributes. Attributes may be nested. There are five core types: *String*, *Date*, *Int*, *Float*, and *Boolean*. These correspond to the standard XML types: *xs:string*, *xs:dateTime*, *xs:long*, *xs:double*, and *xs:boolean*. For example, *2011-12-17T21:00:00.000+02:00* is a value of type *xs:dateTime* representing nine o'clock in the evening of December 17th, 2011 in timezone GMT+2.

XES does not prescribe a fixed set of mandatory attributes for each element (log, trace, and event); an event can have any number of attributes. However, to provide semantics for such attributes, the log refers to so-called *extensions*. An extension gives semantics to particular attributes. For example the *Time extension* defines a timestamp attribute of type *xs:dateTime*. This corresponds to the $\#_{time}(e)$ attribute used in Sect. 5.2. The *Organizational extension* defines a resource attribute of type *xs:string*. This corresponds to the $\#_{resource}(e)$ attribute used in Sect. 5.2. Users can

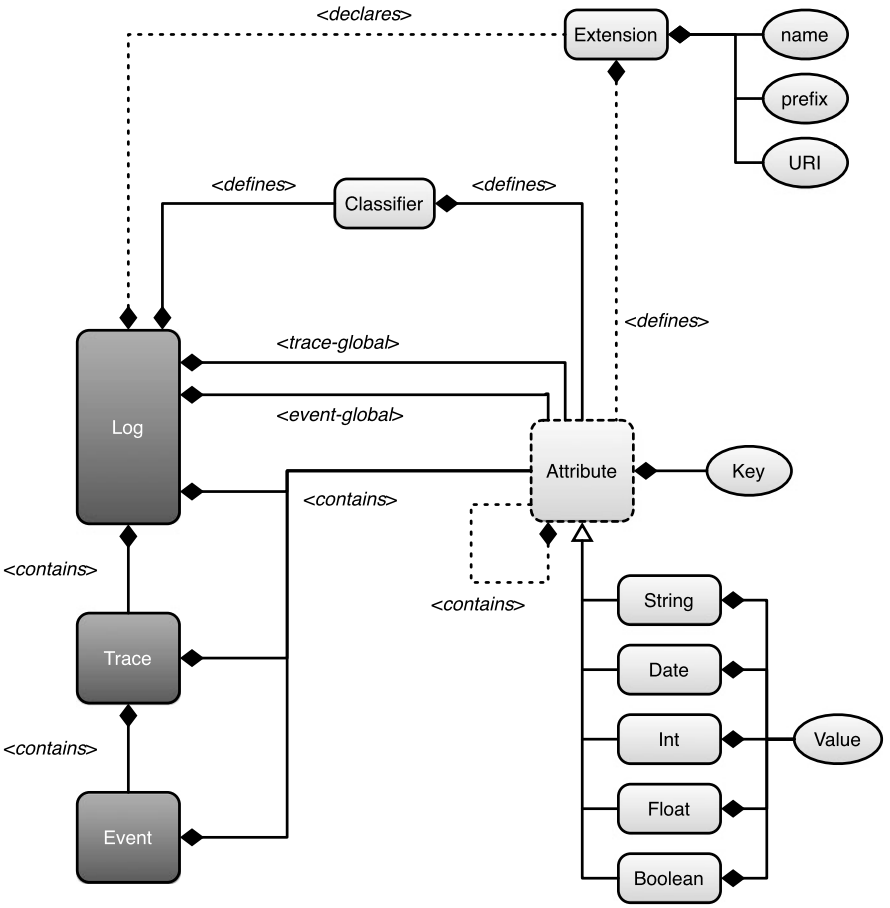

Fig. 5.7 Meta model of XES [64]. A log contains traces and each trace contains events. Logs, traces, and events have attributes. Extensions may define new attributes and a log should declare the extensions used in it. Global attributes are attributes that are declared to be mandatory. Such attributes reside at the trace or event level. Attributes may be nested. Event classifiers are defined for the log and assign a "label" (e.g., activity name) to each event. There may be multiple classifiers

define their own extensions. For example, it is possible to develop domain-specific or even organization-specific extensions. Figure 5.7 shows that a log declares the set of extensions to be used. Each extension may define attributes that are considered to be standard when the extension is used.

In Sect. 5.2, we used $\mathscr{C}$ and $\mathscr{E}$ to denote the case respectively event universe. This was used to be able to refer to a case and event. In XES such unique identifiers are not necessary. In fact, one can think of the position in the log as the identifier of an event or case.

XES may declare particular attributes to be *mandatory*. For example, it may be stated that any trace should have a name or that any event should have a timestamp.

For this purpose a log holds two lists of *global attributes*: one for the traces and one for the events.

XES supports the *classifier* concept described earlier (Definition 5.2). A XES log defines an arbitrary number of classifiers. Each classifier is specified by a list of attributes. Any two events that have the identical values with respect to these attributes are considered to be equal for that classifier. These attributes should be mandatory event attributes. For example, if a classifier is specified by both a name attribute and a resource attribute, then two events are mapped onto the same class if their name and resource attributes coincide.

The XES meta model shown in Fig. 5.7 does not prescribe a concrete syntax. In principle many serializations are possible. However, to exchange XES documents, a standard XML serialization is used. Figure 5.8 shows a fragment of the XES XML serialization of the event log of Table 5.1. In the example XES log three extensions are declared: *Concept*, *Time*, and *Organizational*. For each of these extensions a shorter prefix is given. These prefixes are used in the attribute names. For example, the *Time* extension defines an attribute *timestamp*. As shown in Fig. 5.8, this extension uses prefix *time*, therefore the timestamp of an event is stored using the key *time:timestamp*.

The example log in Fig. 5.8 specifies two lists of global attributes. Traces have one global attribute: attribute *concept:name* is mandatory for all traces. Events have three global attributes: attributes *time:timestamp*, *concept:name* and *org:resource* are mandatory for all events.

Three classifiers are defined in the XES log shown in Fig. 5.8. Classifier *Activity* classifies events based on the *concept:name* attribute. Classifier *Resource* classifies events based on the *org:resource* attribute. Classifier *Both* classifies events based on two attributes: *concept:name* and *org:resource*. Recall that Definition 5.2 already introduced the concept of a classifier: an event $e \in \mathcal{E}$ is classified as $\underline{e}$. For example, $\underline{e} = \#_{resource}(e)$ classifies events based on the resource executing the event.

For more information about the concrete syntax of XES we refer to www.xes-standard.org. However, the fragment shown in Fig. 5.8 already demonstrates that XES indeed operationalizes the concept of an event log as described in Definition 5.3. Moreover, the extension mechanism makes the format extendible while at the same time providing semantics for commonly used attributes. In the context of XES, five *standard extensions* have been defined. These extensions are described in the so-called *XESEXT* XML format [64]. Here we only mention a subset of standard attributes defined by these extensions.

- The *concept extension* defines the *name* attribute for traces and events. Note that the example XES file indeed uses *concept:name* attributes for traces and events. For traces, the attribute typically represents some identifier for the case. For events, the attribute typically represents the activity name. The concept extension also defines the *instance* attribute for events. This is used to distinguish different activity instances in the same trace. This extension can be used to resolve the dilemma shown in Fig. 5.5.

```
<?xml version="1.0" encoding="UTF-8" ?>
   <extension name="Concept" prefix="concept" uri="http://.../concept.xesext"/>
   <extension name="Time" prefix="time" uri="http://.../time.xesext"/>
   <extension name="Organizational" prefix="org" uri="http://.../org.xesext"/>
   <global scope="trace">
      <string key="concept:name" value="name"/>
   </global>
   <global scope="event">
      <date key="time:timestamp" value="2010-12-17T20:01:02.229+02:00"/>
      <string key="concept:name" value="name"/>
      <string key="org:resource" value="resource"/>
   </global>
   <classifier name="Activity" keys="concept:name"/>
   <classifier name="Resource" keys="org:resource"/>
   <classifier name="Both" keys="concept:name org:resource"/>
   <trace>
      <string key="concept:name" value="1"/>
      <event>
         <string key="concept:name" value="register request"/>
         <string key="org:resource" value="Pete"/>
         <date key="time:timestamp" value="2010-12-30T11:02:00.000+01:00"/>
         <string key="Event_ID" value="35654423"/>
         <string key="Costs" value="50"/>
      </event>
      <event>
         <string key="concept:name" value="examine thoroughly"/>
         <string key="org:resource" value="Sue"/>
         <date key="time:timestamp" value="2010-12-31T10:06:00.000+01:00"/>
         <string key="Event_ID" value="35654424"/>
         <string key="Costs" value="400"/>
      </event>
      <event>
         <string key="concept:name" value="check ticket"/>
         <string key="org:resource" value="Mike"/>
         <date key="time:timestamp" value="2011-01-05T15:12:00.000+01:00"/>
         <string key="Event_ID" value="35654425"/>
         <string key="Costs" value="100"/>
      </event>
      <event>
         <string key="concept:name" value="decide"/>
         <string key="org:resource" value="Sara"/>
         <date key="time:timestamp" value="2011-01-06T11:18:00.000+01:00"/>
         <string key="Event_ID" value="35654426"/>
         <string key="Costs" value="200"/>
      </event>
      <event>
         <string key="concept:name" value="reject request"/>
         <string key="org:resource" value="Pete"/>
         <date key="time:timestamp" value="2011-01-07T14:24:00.000+01:00"/>
         <string key="Event_ID" value="35654427"/>
         <string key="Costs" value="200"/>
      </event>
   </trace>
   <trace>
      <string key="concept:name" value="2"/>
      <event>
         <string key="concept:name" value="register request"/>
         <string key="org:resource" value="Mike"/>
         <date key="time:timestamp" value="2010-12-30T11:32:00.000+01:00"/>
         <string key="Event_ID" value="35654483"/>
         <string key="Costs" value="50"/>
      </event>
      ...
   </trace>
   ...
</log>
```

Fig. 5.8 Fragment of a XES file

- The *life-cycle extension* defines the *transition* attribute for events. When using the standard *transactional life-cycle model* shown in Fig. 5.3, possible values of this attribute are "schedule", "start", "complete", "autoskip", etc.
- The *organizational extension* defines three standard attributes for events: *resource*, *role*, and *group*. The resource attribute refers to the resource that triggered or executed the event. The role and group attributes characterize the (required) capabilities of the resource and the resource's position in the organization. For example, an event executed by a sales manager may have role "manager" and group "sales department" associated to it.
- The *time extension* defines the *timestamp* attribute for events. Since such a timestamp is of type *xs:dateTime*, both a date and time are recorded.
- The *semantic extension* defines the *modelReference* attribute for all elements in the log. This extension is inspired by *SA-MXML*. The references in the log point to concepts in an ontology. For example, there may be an ontology describing different kinds of customers, e.g., Silver, Gold, and Platinum customers. Using the *modelReference* attribute a trace can point to this ontology thus classifying the customer.

Users and organizations can add new extensions and share these with others. For example, general extensions referring to costs, risks, context, etc. can be added. However, extensions may also be domain-specific (e.g., healthcare, customs, or retail) or organization-specific.

Currently, XES is supported by tools such as ProM, Nitro, XESame, Disco, Celonis, Minit, SNP Business Process Analysis, Rialto Process, and the reference implementation OpenXES. ProM is probably the most widely used process mining tool (see Sect. 11.3) providing a wide variety of process mining techniques. ProM 6 can load both MXML and XES files. Tools like Disco (www.fluxicon.com) can be used to quickly convert event logs into the XES (or MXML) format. XESame (www.processmining.org) generates XES files from collections of database tables. Here, the idea is that given a set of tables there may be different views possible (see also Sect. 5.5). Therefore, XES files serve as a view on the event data. OpenXES (www.openxes.org) is the XES reference implementation, an open source java library for reading, storing, and writing XES logs. OpenXES can easily be embedded in other tools and is able to efficiently (de)serialize large event logs into/from XML files. This frees software developers from developing tedious code to import and export event data.

Challenges when extracting event logs

Definition 5.3 provides a succinct formal definition of the requirements an event log needs to satisfy. XES operationalizes these requirements and provides a concrete syntax. Hence, the target format is well-defined. Nonetheless, extracting event logs may be very challenging. Here, we list the five most important challenges.

- **Challenge 1: Correlation**
  Events in an event log are grouped per case. This simple requirement can be quite challenging as it requires *event correlation*, i.e., events need to be related to each other. Consider, for example, event data scattered over multiple tables or even multiple systems. How to identify events and their corresponding cases? Also consider messages exchanged with other organizations. How to relate responses to the original requests? When designing logging functionality from scratch, it is quite easy to address this problem. However, when dealing with legacy and a variety or interconnected systems, additional efforts are needed to correlate events; see [53] for an example of an approach to correlate events without any a-priori information.
- **Challenge 2: Timestamps**
  Events need to be ordered per case. In principle, such ordering does not require timestamps. However, when merging data from different sources, one typically needs to depend on timestamps to sort events (in order of occurrence). This may be problematic because of multiple clocks and delayed recording. For example, in an X-ray machine the different components have local clocks and events are often queueing before being recorded. Therefore, there may be significant differences between the actual time an event takes place and its timestamp in the log. As a result the ordering of events is unreliable, e.g., cause and effect may be reversed. In other applications, timestamps may be too coarse. In fact, many information systems only record a date and not a timestamp. For example, most events in a hospital are recorded in the hospital information system based on a patient id and a date, without storing the actual time of the test or visit. As a result it is impossible to reconstruct the order of events on a given day. One way to address this problem is to assume only a partial ordering of events (i.e., not a total order) and subsequently use dedicated process mining algorithms for this. Another way to (partially) address the problem is to "guess" the order based on domain knowledge or frequent patterns across days.
- **Challenge 3: Snapshots**
  Cases may have a lifetime extending beyond the recorded period, e.g., a case was started before the beginning of the event log or was still running when the recording stopped. Therefore, it is important to realize that event logs typically just provide a *snapshot* of a longer running process. When the average duration of a case is short compared to the length of the recording, it is best to solve this problem by removing incomplete cases. In many cases, the initial and final activities are known, thus making it easy to filter the event log: simply remove all cases with a missing "head" or "tail". However, when the average duration of a case is of the same order of magnitude as the length of the recording, it becomes difficult to discover end-to-end processes.

- **Challenge 4: Scoping**
  The fourth problem is the scoping of the event log. Enterprise information systems may have thousands of tables with business-relevant data (cf. a typical SAP installation). How to decide which tables to incorporate? Domain knowledge is needed to locate the required data and to scope it. Obviously, the desired scope depends on both the available data and the questions that need to be answered.
- **Challenge 5: Granularity**
  In many applications, the events in the event log are at a different level of granularity than the activities relevant for end users. Some systems produce low-level events that are too detailed to be presented to stakeholders interested in managing or improving the process. Fortunately, there are several approaches to preprocess low-level event logs. For example, in [77] it is shown that frequently appearing low-level patterns can be abstracted into events representing activities.

The availability of high-quality event logs is essential for process mining. Moreover, good event logs can serve many other purposes. Sometimes the term *business process provenance* is used to refer to the systematic collection of the information needed to reconstruct what has actually happened in the business process. From an auditing point of view the systematic, reliable, and trustworthy recording of events is essential. The term "provenance" originates from scientific computing [39]. Here, provenance information is recorded to ensure that scientific experiments are reproducible. High-quality event logs that cannot be tampered with make sure that "history cannot be rewritten or obscured" and serve as a solid basis for process improvement and auditing. Therefore, XES should be seen in a provenance context that extends beyond process discovery and includes topics such as conformance checking. We will elaborate on this topic in Part IV.

5.4 Data Quality

From a practical point of view *data quality* is of the utmost importance for the success of process mining. If event data is missing or cannot be trusted, then the results of process mining are less valuable. To be able to discuss the quality of event data, we first conceptualize event data using the definitions in Sect. 5.2 and the XES meta model presented in Sect. 5.3. The conceptualization is used to systematic classify data quality problems. After the problems have been identified, 12 guidelines for logging are given [147].

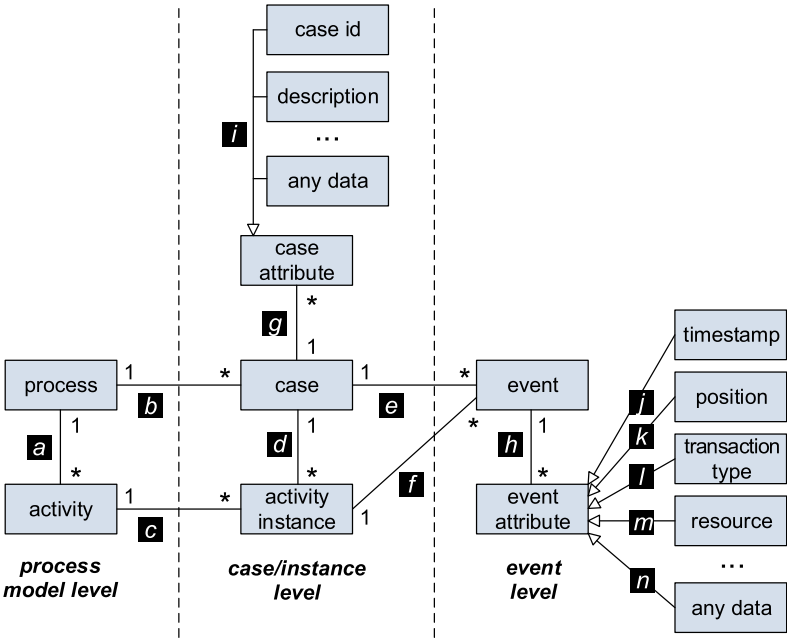

Fig. 5.9 Basic logging concepts conceptualized using a class diagram

5.4.1 Conceptualizing Event Logs

Section 5.2 introduced various event-log-related notions: events, event attribute names and values, activities, timestamps, transaction types, resources, cases, traces, and case attribute names and values. Also subtle notions such as classifiers and activity instances were discussed. More or less the same constructs were discussed in the context of XES in Sect. 5.3. XES is extendible and allows for the definition of domain-specific logs; however, to discuss data quality issues, we consolidate things in a simple class diagram.

Figure 5.9 aims to list the key ingredients of an event log. Anything shown in this class model can be captured using XES or the formal definitions provided before. However, using Fig. 5.9 we can discuss the key concepts without being distracted by the formalities of Sect. 5.2 and the technicalities of XES described in Sect. 5.3.

In Fig. 5.9, three levels are identified: *process model level*, *case/instance level*, and *event level*. The case/instance level shown in Fig. 5.9 consists of *cases* and *activity instances* that connect *processes* and *activities* in the model to *events* in the event log. When modeling, cases and activity instances only exist in abstract form. Only when a process model is instantiated, we can point to concrete cases, activity instances, and events. When observing a real process we are confronted with concrete instances of the process (i.e., cases). The same holds for activities and activity instances. Within the same case (i.e., process instance) there may be

multiple instances of the same activity. For instance, some check activity may be performed multiple times for the same customer request.

The class diagram shown in Fig. 5.9 shows the following associations and cardinalities:

- *a* Each process may have an arbitrary number of activities, but each activity belongs to precisely one process.
- *b* Each case belongs to precisely one process.
- *c* Each activity instance refers to precisely one activity.
- *d* Each activity instance belongs to precisely one case; there may be several activity instances for each activity/case combination.
- *e* Each event refers to precisely one case.
- *f* Each event corresponds to one activity instance; for the same activity instance there may be multiple events.
- *g* Each case attribute refers to one case; each attribute has a name and a value, e.g., "*(birthdate, 29-01-1966)*".
- *h* Each event attribute refers to one event and is characterized by a name and a corresponding value, e.g., "*(costs, $199.99)*".
- *i* There are different subclasses of case attributes, e.g., the description of a case, case identifier, start time of case, etc. Case attributes are invariant, i.e., they do not change while the corresponding events of the case occur.
- *j–n* There are different subclasses of event attributes, e.g., the time of occurrence of the event (*j*), the position in trace (*k*), the transaction type (*l*), the resource causing the event (*m*), or any other type of attribute data (costs, risk, age, etc.).

The attributes attached to events provide valuable information that can be aggregated and mapped onto the process model level. For instance, timestamps can be used to compute the mean waiting time for an activity. Resource attributes attached to events can be used to learn working patterns and allocation rules. Cost information can be projected onto process models to see inefficiencies.

In most cases, event data are provided as a table, a CSV (Comma Separated Values) file, or a spreadsheet (e.g., Excel file) where each row corresponds to an event. This is illustrated by Fig. 5.10. The "dashed boxes" refer to attributes that are derivable from the relations between events, cases, activity instances, activities and processes. For example, the *process* attribute of an event can be derived by following relations *e* and *b*. The *activity* attribute of an event can be derived via relations *f* and *c*. Note that we abstract from case attributes in the latter classification of quality problems. We will focus on the key entities *case*, *activity instance*, and *event* as highlighted in Fig. 5.10.

Per event attribute, it is indicated whether the attribute is mandatory. The *process* attribute of an event is optional. If the attribute is missing, we assume there is just one process.

The *activity instance* attribute is also optional. In many data sets this information will be missing. For example, only complete events are recorded, making activity

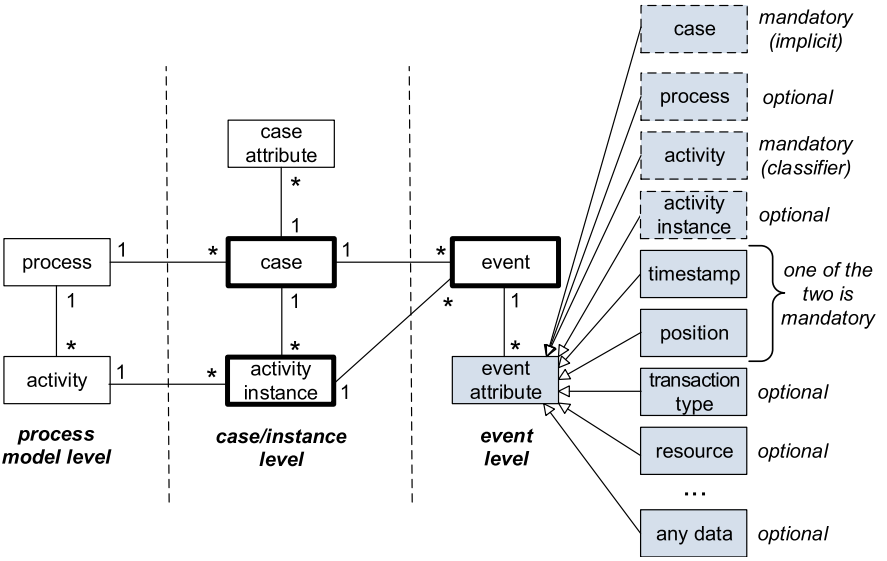

Fig. 5.10 Class diagram zooming in on event attributes and later used to classify data quality problems

instances singletons. If there are start and complete events but not explicit activity instances, then one may use heuristics to derive activity instances. Under the condition that start and complete events alternate, it is possible to deterministically derive activity instances. However, Fig. 5.5 shows an example where alternative explanations exist. In case of overlapping activity instances, transactional information is not sufficient. If start or complete events are missing, similar problems emerge. Heuristics may be used to solve these problems (see Sect. 5.2). Moreover, it is always possible to consider each event as a singleton activity instance. Obviously, such solutions may introduce data quality problems. When start events cannot be related to complete events, it is impossible to measure service times and resource utilization accurately.

In most cases, events have *timestamps* determining the *position* in a trace. Most control-flow discovery algorithms only use the ordering of events within a case as input. Moreover, multiple events may have the same timestamp. In some cases, timestamps are rather coarse (minutes or even days). If timestamps are not fine-grained enough, then the precision of the results is impacted (e.g., mean waiting time). Some process mining algorithms may be based on partial ordered traces rather than totally ordered traces. The lack of a total order may be caused by a lack of precision or by explicit information on causalities. As indicated in Fig. 5.10, we assume that at least the timestamp or position in the trace is known.

The other two standard event attributes—*transaction type* and *resource*—are also optional (see Sect. 5.2). Moreover, there may be additional data attributes related to costs, volume, risks, context, etc. (named *any data* in Fig. 5.10).

5.4.2 Classification of Data Quality Issues

Figure 5.10 summarizes the key concepts related to event data. These are used to discuss *data quality* issues. We consider three main entities (*case*, *activity instance*, and *event*) and nine event attributes (*case*, *process*, *activity*, *activity instance*, *timestamp*, *position*, *transaction type*, *resource*, and *any data*). This allows us to create a *classification* of data quality problems. This classification is related to the challenges mentioned in the context of XES (Sect. 5.3) and is inspired by [80, 98].

First, we consider the main entities (case, activity instance, and event) and not the attributes. At the entity level there are three potential problems:

- (*Missing in log*) The entity exists (or existed) in reality, but was not recorded. For example, an event (e.g., taking a blood sample) occurred but it was not captured by the information system.
- (*Missing in reality*) The entity does not exist and never existed in reality, but was recorded. For example, a scheduled doctor's appointment never took place due to an emergency, but it was recorded by the information system anyway.
- (*Concealed in log*) The entity was recorded and exists (or existed) in reality, but it is hidden in a larger less structured data set. For example, the same entity may appear multiple times in the event log. The scope of the data set may also be much larger than needed for analysis. The event log may be a "mashup" of different data sources creating such challenges. It may be far from trivial to select, identify, and deduplicate entities.

Table 5.2 provides an overview of data quality problems at the entity level. For example, the cell *EV-MIL* refers to missing events.

Table 5.3 classifies problems related to *event attributes*. There are three potential problems related to such attributes:

- (*Missing attribute*) The attribute has not been recorded for a particular event. For example, the timestamp of an event is missing.
- (*Incorrect attribute*) The recorded value of the event attribute is wrong. For example, an event is related to another case.
- (*Imprecise attribute*) The value of the event attribute is too imprecise. For example, the value of a timestamp is too coarse-grained or the address is incomplete.

Table 5.3 combines the above problem types with the different types of event attributes identified in Fig. 5.10. For example, the cell *CASE-MIS* refers to events that cannot be related to a case.

Cell *TS-INC* refers to incorrect timestamps. For example, a patient in a hospital gets his medication at noon, but the doctor enters this in the hospital information system later in the afternoon. As a result, the timestamp of recording is different from the actual time of the event thus potentially creating a data quality problem.

Cell *TS-IMP* refers to timestamps that are too coarse-grained. In a hospital some events may only have a date ("24-3-2016"). Hence, the ordering of events on the same day is lost. The desired precision depends on the process to be analyzed. For example, when analyzing software systems millisecond precision may still be too

Table 5.2 Type of problem (*MIL*, *MIR*, or *CIL*) versus the entity (*CASE*, *AI*, or *EV*) affected

Entity	Type of problem		
	Missing in log (*MIL*)	Missing in reality (*MIR*)	Concealed in log (*CIL*)
Case (*CASE*)	A case is missing in the event log, e.g., a customer order got flushed.	A case that never existed was added to the log, e.g., by inadvertently entering an improper identifier a fictive case is created.	A case is "hidden" in larger data set, e.g., customer orders, order lines, and deliveries with overlapping identifiers are intermingled in a single log.
Activity instance (*AI*)	An activity instance is missing in the event log, e.g., the start and complete of a production step are not related through an activity instance.	An activity instance that never existed was added to the log.	An activity instance is "hidden" in larger data set.
Event (*EV*)	An event is missing in the event log, e.g., a medical test was not recorded in the event log.	An event that never occurred was inadvertently added to the log, e.g., a check that was never conducted was recorded to feign compliance.	An event is "hidden" in larger data set, e.g., identifying a security breach from transactional data having a much broader scope.

coarse-grained. Nanosecond precision may be required to analyze delays in automated processes.

Cell *RES-IMP* refers to events that do not refer to a specific resource, but a group of resources having the same role or working in the same department. This makes it impossible to analyze queueing and workload at the level of individuals.

Table 5.3 focuses on event attributes. When attributes are related to cases, the same problems may appear (missing, incorrect, or imprecise), but such data quality problems are not listed here.

Table 5.4 shows another quality dimension orthogonal to the classification given thus far. Data quality problems may persist and occur *continuously* throughout the event log that is analyzed. Problems may seem irregular and occur *intermittent*. Unknown intermittent data quality problems may be more problematic than known problems that occur continuously. For example, if a check is recorded in 80% of cases, one may derive incorrect conclusions assuming that all checks were logged. Data quality problems are classified as *changing* if clear patterns can be identified. For example, approvals are never recorded on Sunday or the granularity of timestamps improved after the installation of the new system.

The three recurrence categories in Table 5.4 (*CONT*, *INT*, and *CHNG*) can be combined with Tables 5.2 and 5.3. Some examples:

Table 5.3 Type of problem (*MIS*, *INC*, *IMP*) versus the event attribute affected, e.g., cell *TS-MIS* refers to missing timestamps

Attribute	Type of problem		
	Missing attribute (*MIS*)	Incorrect attribute (*INC*)	Imprecise attribute (*IMP*)
Case (*CASE*)	The event does not refer to a case.	The event refers to the wrong case.	The event may be related to multiple cases due to ambiguity.
Process (*PROC*)	The event cannot be related to a specific process.	The event refers to an unrelated process.	The event may be associated to multiple processes in an ambiguous manner.
Activity (*ACT*)	The event does not refer to an activity.	The event refers to another activity.	The event may be related to multiple activities due to imprecise labels.
Activity instance (*AI*)	The event does not refer to an activity instance.	The event refers to the wrong activity instance.	The event may be inadvertently related to multiple activity instances.
Timestamp (*TS*)	The event has no timestamp.	The event has an incorrect timestamp, e.g., a wrong date was entered or the event was recorded at a later point in time.	The event has a timestamp that is too coarse-grained, e.g., only a date was recorded.
Position (*POS*)	Ordering information for the event is missing (may be reconstructed using timestamps).	The event appears at the wrong position in the event log (e.g., the ordering of events is not consistent with the timestamps).	Ordering information for the event is partially lost.
Transaction type (*TT*)	The event has no transaction type (start, complete, etc.).	The transaction information is wrong.	The event may be inadvertently related to multiple transaction types.
Resource (*RES*)	The event is not related to any resource.	The event is related to the wrong resource.	The event may be related to multiple resources, e.g., only the role or department of the resources is recorded.
Any data (*ANY*)	A data attribute (e.g., costs) is missing for the event.	A data attribute has the wrong value (e.g., wrong amount).	The event has an attribute value that is too coarse-grained (e.g., street name is given but number is missing).

Table 5.4 Recurrence of data quality problems: A particular problem (e.g., a missing attribute) may persist, repeat in an unpredictable manner, or return periodically

Recurrence	Examples
Continuous (*CONT*)	A precise timestamp is missing for all medical examination events. All cases handled by the Eindhoven branch are missing in the log. The name of the responsible doctor is never recorded.
Intermittent (*INT*)	Some events have precise timestamps whereas for other events only the date is known. Blood pressure measurements that have been performed are not always recorded (e.g., depending on workload). Some nurses repeatedly forget to enter the name of the responsible doctor.
Changing (*CHNG*)	In the second week of January, timestamps were missing due to software problems. In weekends, the resource attribute is not recorded due to understaffing. In the second semester, intermediate tests were not recorded.

- *RES-MIS-CONT* refers to the problem that the resource attribute is never recorded.
- *EV-MIL-INT* refers to the problem that events are sometimes missing from the log.
- *TS-IMP-CHNG* refers to the problem that events have imprecise timestamps in certain periods, e.g., before the new software system was installed only dates were recorded.

In total $((3 \times 3) + (9 \times 3)) \times 3 = 108$ data quality problems can be identified using the three tables. These include the 17 problems identified in [80] and the 27 problems identified in [98]. The empirical investigation reported in [98] shows that *EV-MIL-**, *TS-IMP-**, and *RES-IMP-** are among the most frequent data quality problems in hospitals.

5.4.3 Guidelines for Logging

The data quality problems just described illustrate that the input side of data analysis is often neglected. Event data are often seen as a by-product. For example, the data is there for financial reasons or simply because a programmer decided to put a write statement in the code. Since the "input side of process mining" is vital, we now discuss the *12 guidelines for logging* introduced in [147]. These guidelines make no assumptions on the underlying technology used to record event data.

In this section, we use a rather loose definition of event data: events simply refer to "things that happen" and are described by *references* and *attributes*. *References* have a *reference name* and an *identifier* that refers to some object (person, case, ticket, machine, room, etc.) in the universe of discourse. *Attributes* have a *name* and a *value*, e.g., *age* = *49* or *time* = "*19-12-2015 03:14:00*". In Fig. 5.10, we did not make a distinction between references and attributes. However, it is easy to add

this dimension. Based on these concepts, we define our 12 *Guidelines for Logging* (**GL1**–**GL12**) [147].

To create an event log from such "raw events", (1) we need to select the events relevant for the process at hand, (2) events need to be correlated to form process instances (cases), (3) events need to be ordered using timestamp information (or have an explicit order), and (4) event attributes need to be selected or computed based on the raw data (resource, cost, etc.). The guidelines for logging refer both to the availability of raw event data and the transformation process. The aim is to improve data quality and avoid the issues listed in Sect. 5.4.2. Moreover, the guidelines also emphasize the correct interpretation of event data.

GL1 *Reference and attribute names should have clear semantics, i.e., they should have the same meaning for all people involved in creating and analyzing event data.* Different stakeholders should interpret event data in the same way.

GL2 *There should be a structured and managed collection of reference and attribute names.* Ideally, names are grouped hierarchically (like a taxonomy or ontology). A new reference or attribute name can only be added after there is consensus on its value and meaning. Also consider adding domain or organization-specific extensions (see, for example, the extension mechanism of XES described in Sect. 5.3).

GL3 *References should be stable (e.g., identifiers should not be reused or rely on the context).* For example, references should not be time, region, or language dependent. Some systems create different logs depending on the language settings. This is unnecessarily complicating analysis.

GL4 *Attribute values should be as precise as possible. If the value does not have the desired precision, then this should be indicated explicitly (e.g., through a qualifier).* For example, if for some events only the date is known but not the exact timestamp, then this should be stated explicitly.

GL5 *Uncertainty with respect to the occurrence of the event or its references or attributes should be captured through appropriate qualifiers.* For example, due to communication errors, some values may be less reliable than usual. Note that uncertainty is different from imprecision.

GL6 *Events should be at least partially ordered. The ordering of events may be stored explicitly (e.g., using a list) or implicitly through an attribute denoting the event's timestamp.* If the recording of timestamps is unreliable or imprecise, there may still be ways to order events based on observed causalities (e.g., usage of data).

GL7 *If possible, also store transactional information about the event (start, complete, abort, schedule, assign, suspend, resume, withdraw, etc.).* Having start and complete events allows for the computation of activity durations. It is recommended to explicitly link events to activity instances to be able to relate events belonging to the same activity occurrence. Without references to activity instances it may not always be clear which events belong together, e.g., which start event corresponds to which complete event.

GL8 *Perform regularly automated consistency and correctness checks to ensure the syntactical correctness of the event log.* Check for missing references or attributes, and reference/attribute names not agreed upon. Event quality assurance is a continuous process (to avoid degradation of log quality over time).

GL9 *Ensure comparability of event logs over time and different groups of cases or process variants.* The logging itself should not change over time (without being reported). For comparative process mining, it is vital that the same logging principles are used. If for some groups of cases, some events are not recorded even though they occur, then this may suggest differences that do not actually exist.

GL10 *Do not aggregate events in the event log used as input for the analysis process.* Aggregation should be done during analysis and not before (since it cannot be undone). Event data should be as "raw" as possible.

GL11 *Do not remove events and ensure provenance. Reproducibility is key for process mining.* For example, do not remove a student from the database after he dropped out since this may lead to misleading analysis results. Mark objects as not relevant (a so-called "soft delete") rather than deleting them: concerts are not deleted—they are canceled; employees are not deleted—they are fired, etc.

GL12 *Ensure privacy without losing meaningful correlations.* Sensitive or private data should be removed as early as possible (i.e., before analysis). However, if possible, one should avoid removing correlations. For example, it is often not useful to know the name of a student, but it may be important to still be able to use his high school marks and know what other courses he failed. Hashing can be a powerful tool in the trade-off between privacy and analysis.

The guidelines and classification aim to make the reader aware of data quality problems directly influencing the results of process mining.

5.5 Flattening Reality into Event Logs

In order to do process mining, events need to be related to cases. As indicated before, this is natural as a process model describes the life-cycle of a case of a particular type. All activities in a conventional process model (independent of the notation used) correspond to status changes of such a case. We will refer to such process models as *flat models*. In this book, we adopt this (often hidden) assumption associated to all mainstream process modeling notations. However, it is important to realize that *real-life processes are not flat*. We use a simple example to illustrate this.

Consider the class diagram shown in Fig. 5.11 describing a database consisting of four tables. Table *Order* contains information about orders. For example, each record in the *Order* table has a unique order number, refers to a customer, and has an associated amount. Multiple products can be ordered in one order. Therefore,

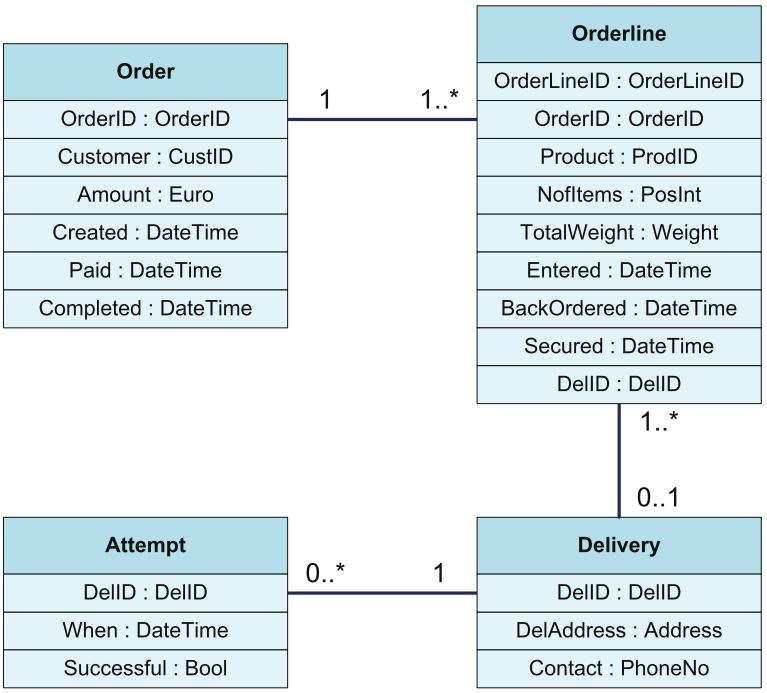

Fig. 5.11 Class diagram showing the relations between orders, order lines, deliveries, and delivery attempts

Table *Orderline* holds information about individual order lines. Records in the *Orderline* table refer to orders in the *Order* table. Figure 5.11 shows that each order line corresponds to one order and each order corresponds to one or more order lines. One order line describes which type of product is ordered, the quantity, and weight. Table *Delivery* holds information about deliveries. Each delivery corresponds to a collection of order lines. These order lines are delivered to a particular address. To deliver the corresponding collection of products, multiple attempts may be needed. Table *Attempts* stores information about these attempted deliveries. An attempt may be successful or not. If not, another attempt is made at a later point in time. Figure 5.11 shows that each delivery corresponds to zero or more attempts and one or more order lines. Each order line corresponds to at most one delivery.

Table 5.5 shows a small fragment of a larger *Order* table. For each order, up to three timestamps are recorded. The timestamp in the *Created* column denotes the time at which the order was created. The *Paid* column denotes the time at which the order was paid and the *Completed* column denotes the time at which the order was completed. Table 5.5 shows several *null* timestamps. This indicates that the corresponding events did not take place yet.

Table 5.6 shows some example records of the *Orderline* table. Each line refers to a particular product. For example, order line 112346 corresponds to two iPod nanos that are part of order 91245. Each order line refers to an order and to a delivery

Table 5.5 Some records of the *Order* table

Order					
OrderID	Customer	Amount	Created	Paid	Completed
91245	John	100	28-11-2011:08.12	02-12-2011:13.45	05-12-2011:11.33
91561	Mike	530	28-11-2011:12.22	03-12-2011:14.34	05-12-2011:09.32
91812	Mary	234	29-11-2011:09.45	02-12-2011:09.44	04-12-2011:13.33
92233	Sue	110	29-11-2011:10.12	null	null
92345	Kirsten	195	29-11-2011:14.45	02-12-2011:13.45	null
92355	Pete	320	29-11-2011:16.32	null	null
...	...	...	...	...	...

(if already created). For each order line, up to three timestamps are recorded: the time of entering the order line (column *Entered*), the time of back ordering (column *BackOrdered*), and the time of securing the item (column *Secured*). A *null* value indicates that the corresponding event did not take place (yet). Typically, only few order lines will be back-ordered (i.e., most rows will have a null value in the *Back-Ordered* column). A backorder is an order line that cannot be delivered because of a lack of sufficient inventory. Therefore, the inventory needs to be replenished before the backorder can be delivered. Since only few order lines become backorders, column *BackOrdered* has many null values. Once the products are available and reserved for a particular order line, the corresponding timestamp is added in column *Secured*.

Information about deliveries is stored in the *Delivery* table shown in Table 5.7. For each delivery, an address and a phone number are recorded. Each delivery refers to a collection of order lines and may require multiple attempts.

Attempts to deliver products are recorded in the *Attempt* table. Table 5.8 shows some example attempts. An attempt has a timestamp and refers to a delivery (column *DellID*). Delivery 882345 required three attempts before the corresponding set of order lines could be delivered successfully. Delivery 882346 required only one attempt.

The four tables show only a snapshot of the available data. Orders that have not yet been fully handled may have many null values.

The database consisting of tables *Order*, *Orderline*, *Delivery*, and *Attempts* is a bit artificial and its design could be improved. For example, the tables with multiple timestamps could have been split into multiple tables. Moreover, in an ERP system like SAP much more detailed information is stored. Hence, the four tables are an oversimplification of reality and only serve as a means to explain the problem of *flattening reality for process mining*.

Clearly the timestamps in the four tables correspond to events related to the "overall ordering and delivery" process. However, when creating an event log, each event needs to be associated to a particular case. Therefore, we need to flatten the four tables into one table with a "case id" column. However, one can choose from four types of cases: orders, order lines, deliveries, and attempts. Any record in one of the four tables potentially corresponds to a case. Which one to choose?

Table 5.6 Part of the *Orderline* table: each record corresponds to an order line. An order line refers to an order (column *OrderID*) and a delivery (column *DellID*)

Orderline								
OrderLineID	OrderID	Product	NofItems	TotalWeight	Entered	BackOrdered	Secured	DellID
112345	91245	iPhone 4G	1	0.250	28-11-2011:08.13	null	28-11-2011:08.55	882345
112346	91245	iPod nano	2	0.300	28-11-2011:08.14	28-11-2011:08.55	30-11-2011:09.06	882346
112347	91245	iPod classic	1	0.200	28-11-2011:08.15	null	29-11-2011:10.06	882345
112448	91561	iPhone 4G	1	0.250	28-11-2011:12.23	null	28-11-2011:12.59	882345
112449	91561	iPod classic	1	0.200	28-11-2011:12.24	28-11-2011:16.22	null	null
112452	91812	iPhone 4G	5	1.250	29-11-2011:09.46	null	29-11-2011:10.58	882346
...	...	...	...	...	...	...	...	...

Table 5.7 Some records of the *Delivery* table

Delivery		
DellID	DelAddress	Contact
882345	5513VJ-22a	0497-2553660
882346	5513XG-45	040-2298761
...	...	...

Table 5.8 Part of the *Attempt* table

Attempt		
DellID	When	Successful
882345	05-12-2011:08.55	false
882345	06-12-2011:09.12	false
882345	07-12-2011:08.56	true
882346	05-12-2011:08.43	true
...	...	...

Let us assume that we are mainly interested in orders. Therefore, we let each case correspond to a record in table *Order*. Table *Order* has up to three timestamps per record. Hence, only three events per case can be found if only the *Order* table is considered and information about order lines and deliveries related to an order remains unused. When using only records from the *Order* table, control-flow discovery will most likely return a sequential process consisting of three steps: *create*, *pay*, and *complete*. To obtain a process model containing more activities, we need to consider the other tables. By using the references in the tables, orders can be related to order lines. In turn, order lines can be related to deliveries and the corresponding attempts. For example, order lines 112345, 112346 and 112347 refer to order 91245. Figure 5.12 shows all events that can be found by searching for all records that can be related to order 91245. The rectangles refer to concrete records in the four tables. The rounded rectangles refer to possible events and their attributes. All events in the figure refer to case 91245. As Fig. 5.12 shows, order 91245 is related to order lines 112345, 112346 and 112347, and deliveries 882345 and 882346. For delivery 882345 there are three corresponding attempt records and for delivery 882346 only one.

The top three events in Fig. 5.12 (the events directly connected to the root node) would have been the only events if only the *Order* table would have been considered. There are also various intermediate selections possible, resulting in subsets of the events shown in Fig. 5.12. For example, only the top ten events remain if only the *Order* and *Orderline* tables are considered, thus abstracting from deliveries.

Table 5.9 shows an event log using the selection illustrated by Fig. 5.12. As before, each line corresponds to an event. The *case id* column shows how events are correlated, i.e., each event refers to an order. The *activity* column names events as already shown Fig. 5.12. The *timestamp* column shows the date and time associated to the event. The *other attributes* column shows additional attributes. Depending on the type of activity, different attributes are recorded. Table 5.9 shows that there is quite some redundancy in the event log. This is partly unavoidable due to the

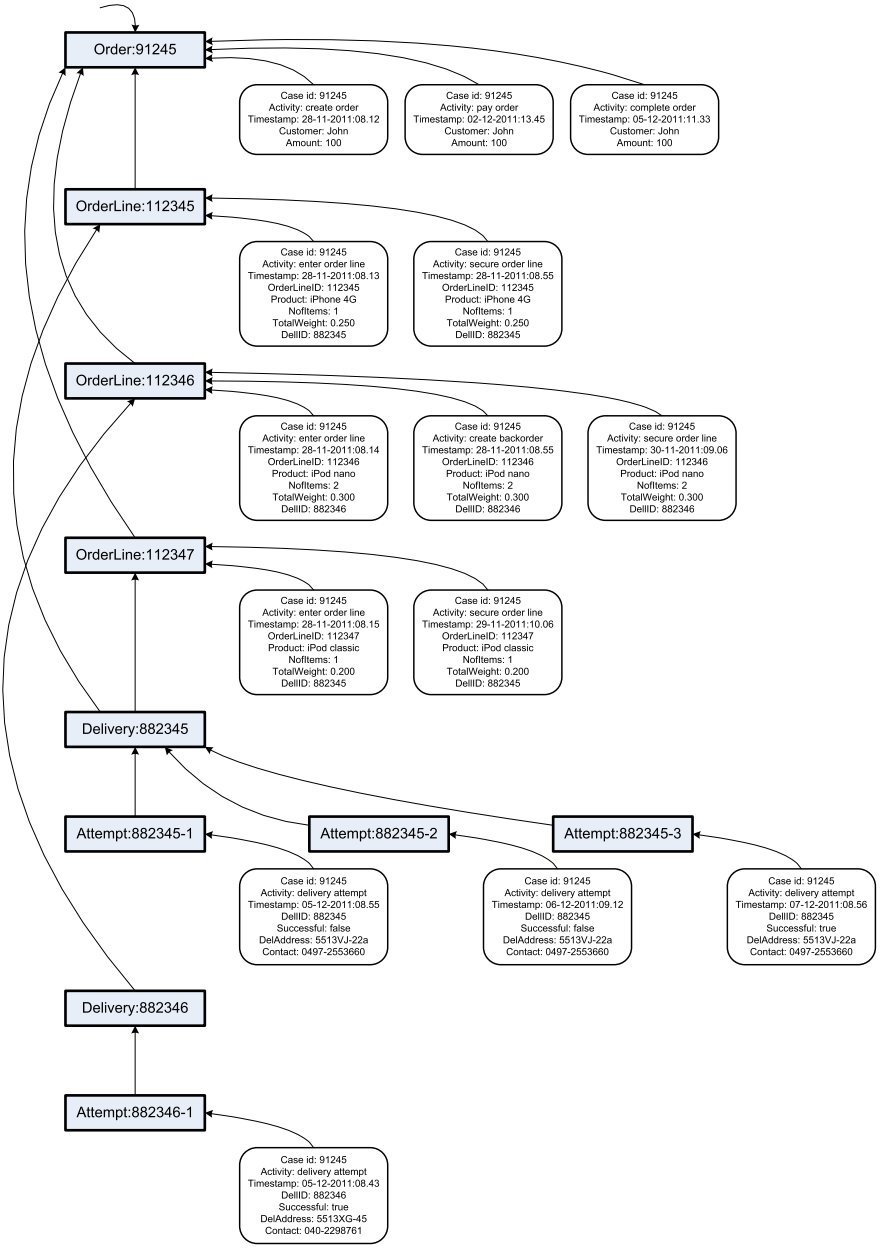

Fig. 5.12 All events that can be related to order 91245. The 14 rounded rectangles correspond to events associated to case 91245. The squared rectangles represent records in one of the four tables

Table 5.9 Events extracted from all four tables using order records from the *Order* table as a starting point

Attempt			
Case id	Activity	Timestamp	Other attributes
91245	create order	28-11-2011:08.12	Customer: John, Amount: 100
91245	enter order line	28-11-2011:08.13	OrderLineID: 112345, Product: iPhone 4G, NofItems: 1, TotalWeight: 0.250, DellID: 882345
91245	enter order line	28-11-2011:08.14	OrderLineID: 112346, Product: iPod nano, NofItems: 2, TotalWeight: 0.300, DellID: 882346
91245	enter order line	28-11-2011:08.15	OrderLineID: 112347, Product: iPod classic, NofItems: 1, TotalWeight: 0.200, DellID: 882345
91245	secure order line	28-11-2011:08.55	OrderLineID: 112345, Product: iPhone 4G, NofItems: 1, TotalWeight: 0.250, DellID: 882345
91245	create backorder	28-11-2011:08.55	OrderLineID: 112346, Product: iPod nano, NofItems: 2, TotalWeight: 0.300, DellID: 882346
91245	secure order line	29-11-2011:10.06	OrderLineID: 112347, Product: iPod classic, NofItems: 1, TotalWeight: 0.200, DellID: 882345
91245	secure order line	30-11-2011:09.06	OrderLineID: 112346, Product: iPod nano, NofItems: 2, TotalWeight: 0.300, DellID: 882346
91245	pay order	02-12-2011:13.45	Customer: John, Amount: 100
91245	delivery attempt	05-12-2011:08.43	DellID: 882346, Successful: true, DelAddress: 5513XG-45, Contact: 040-2298761
91245	delivery attempt	05-12-2011:08.55	DellID: 882345, Successful: false, DelAddress: 5513VJ-22a, Contact: 0497-2553660
91245	complete order	05-12-2011:11.33	Customer: John, Amount: 100
91245	delivery attempt	06-12-2011:09.12	DellID: 882345, Successful: false, DelAddress: 5513VJ-22a, Contact: 0497-2553660
91245	delivery attempt	07-12-2011:08.56	DellID: 882345, Successful: true, DelAddress: 5513VJ-22a, Contact: 0497-2553660
91561	create order	28-11-2011:12.22	Customer: Mike, Amount: 530
91561	enter order line	28-11-2011:12.23	OrderLineID: 112448, Product: iPhone 4G, NofItems: 1, TotalWeight: 0.250, DellID: 882345
...	...	...	...
...	...	...	...

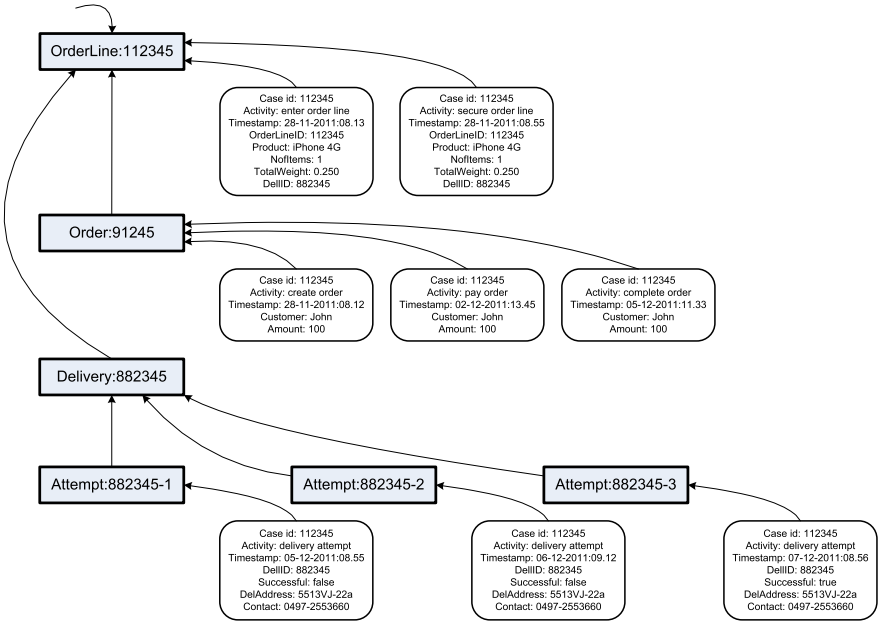

Fig. 5.13 All events that can be related to order line 112345

structure that logs should have. However, the case attributes *Customer* and *John* do not need to be repeated for each event; one could have used case attributes rather than event attributes.

Table 5.9 *flattens* the original database consisting of four tables. The flattened event log is like a *view* on the complete data set. Alternative views are possible. For example, Fig. 5.13 shows another way to flatten the original database. Now cases correspond to order lines rather than orders. Hence, the root node is order line 112345. This is an order line of order 91245 and three attempts were needed to deliver the iPhone 4G. The timestamps in the *Order* table have been used to create events associated to order line cases rather than orders. Based on the view sketched in Fig. 5.13, one can generate another event log.

In Fig. 5.12, the root node is an order. In Fig. 5.13, the root node is an order line. Similarly, it is possible to take a delivery or delivery attempts as root node. Moreover, various selections of events can be used, e.g., the three order-related events or the three delivery-related events in Fig. 5.13 could have been left out from the selection. This shows that many views on the original data set are possible, i.e., there are many ways to flatten reality as recorded into a single event log.

Flattening a data set into an event log can be compared to aggregating multidimensional data in Online Analytical Processing (OLAP) (see Sect. 12.4). For example, using a typical OLAP tool, sales data can be viewed by product categories, by region, and/or by quarter. Depending on the type of question, a different view on the data can be chosen. One important difference is that in process mining we analyze

processes rather than a simple OLAP cube. Therefore, we need to correlate events and order them, thus making the extraction process more complex.

Proclets: Seeing in 3-D

Process mining shows that the assumptions made by classical process modeling languages such as BPMN, UML ADs, Statecharts, BPEL, YAWL, WF-nets, and EPCs are somewhat artificial. They only provide *one monolithic view* on the real process of interest. The process is flattened to allow for a diagram that describes the life-cycle of one case in isolation. The application of process mining to real-life processes shows that squeezing a process into such a single monolithic flat model is problematic. Like in physics, where experiments help to (in)validate models, process discovery also helps to reveal the limitations of oversimplified models. The empirical nature of process mining helps managers, consultants, and process analysts to better understand the "fabric of real business processes" and, thus, also see the limitations of conventional process modeling languages.

Proclets [153] are one of the few business process modeling languages allowing for *3-D process models*. Rather than describing the whole process in terms of one monolithic 2-D process model, the process is modeled as a collection of interacting Proclets. For example, when modeling orders and deliveries, the class diagram of Fig. 5.11 can be used as a starting point. Based on this class diagram four classes of Proclets are identified: orders, order lines, deliveries, and delivery attempts. Each Proclet class is modeled separately. The Proclets interact and are related by following the real anatomy of the process.

See [153] for more examples illustrating that classical notations force the modeler to *straightjacket* processes into one monolithic model. Unfortunately, hierarchy concepts in conventional languages do not support one-to-many or many-to-many relationships. In Fig. 5.11, orders and deliveries are in a many-to-many relationship: one order may result in multiple deliveries and one delivery many involve order lines of different orders. This cannot be handled by the refinement of activities; order and delivery Proclets need to coexist independent of one another.

Object-oriented modeling and artifact-centric modeling use ideas related to Proclets. However, mainstream process modeling notations and BPM systems still use conventional 2-D notations. The ACSI project [1] aims to promote the use of Proclets and develop new process mining techniques for non-monolithic processes.

Although it is important to view business processes in 3-D, we often need to resort to 2-D models for a variety of reasons. Here we mention three of them. First of all, the data sources provided may only allow for a 2-D view, e.g., only one table is provided as input. Second, users expect process models in terms of classical

2-D process modeling languages such as BPMN, UML ADs, Statecharts, BPEL, YAWL, WF-nets, and EPCs. Last but not least, most process mining techniques require flattening the data. Therefore, we advocate the following approach.

- Create a *process-oriented data warehouse* containing information about relevant events. The data warehouse should avoid storing aggregated data, and gather the raw business events instead. In traditional data warehouses, events are aggregated into quantitative data, thus hampering process analysis.
- Depending on the questions, define an appropriate *view*. Based on the chosen view, *flatten* the required data to produce an event log (e.g., in XES format). This corresponds to taking a *2-D slice* from the 3-D data.
- Use the 2-D slice to apply a variety of *process mining* techniques. If needed, *filter* the event log further (e.g., removing infrequent activities). Continue extracting, filtering, and mining until the questions are answered.

Depending on the questions, it may be the case that multiple 2-D slices need to be taken to create a 3-D view on the overall process. This view is consistent with Fig. 5.1; by extracting the event data the scope of the process is determined.