Chapter 10
Operational Support

Most process-mining techniques work on "post mortem" event data, i.e., they analyze events that belong to cases that have already completed. Obviously, it is not possible to influence the execution of "post mortem" cases. Moreover, cases that are still in the pipeline cannot be guided on the basis of "post mortem" event data only. Today, however, many data sources are updated in (near) real-time and sufficient computing power is available to analyze events when they occur. Therefore, process mining should not be restricted to off-line analysis and can also be used for online operational support. This chapter broadens the scope of process mining to include online decision support. For example, for a running case the remaining flow time can be predicted and suitable actions can be recommended to minimize costs.

10.1 Refined Process Mining Framework

Thus far we identified three main types of process mining: *discovery*, *conformance*, and *enhancement* (cf. Figs. 2.5 and 9.1). Orthogonal to these types of process mining we identified several perspectives including: the *control-flow perspective* ("How?"), the *organizational perspective* ("Who?"), and the *case/data perspective* ("What?"). The classification of process mining techniques into discovery, conformance, and enhancement does reflect that analysis can be done *online* or *off-line*. Moreover, Figs. 2.5 and 9.1 do not acknowledge that there are essentially two types of models ("de jure models" and "de facto models") and two types of data ("pre mortem" and "post mortem" event data) [139].

Figure 10.1 shows our *refined process mining framework*. As before, we assume some external "world" consisting of business processes, people, organizations, etc. and supported by some information system. The information system records information about this "world" in such a way that event logs can be extracted as described in Chap. 5.

Figure 10.1 emphasizes the systematic, reliable, and trustworthy recording of events by using the term *provenance*. This term originates from scientific comput-

W. van der Aalst, *Process Mining*, DOI 10.1007/978-3-662-49851-4_10

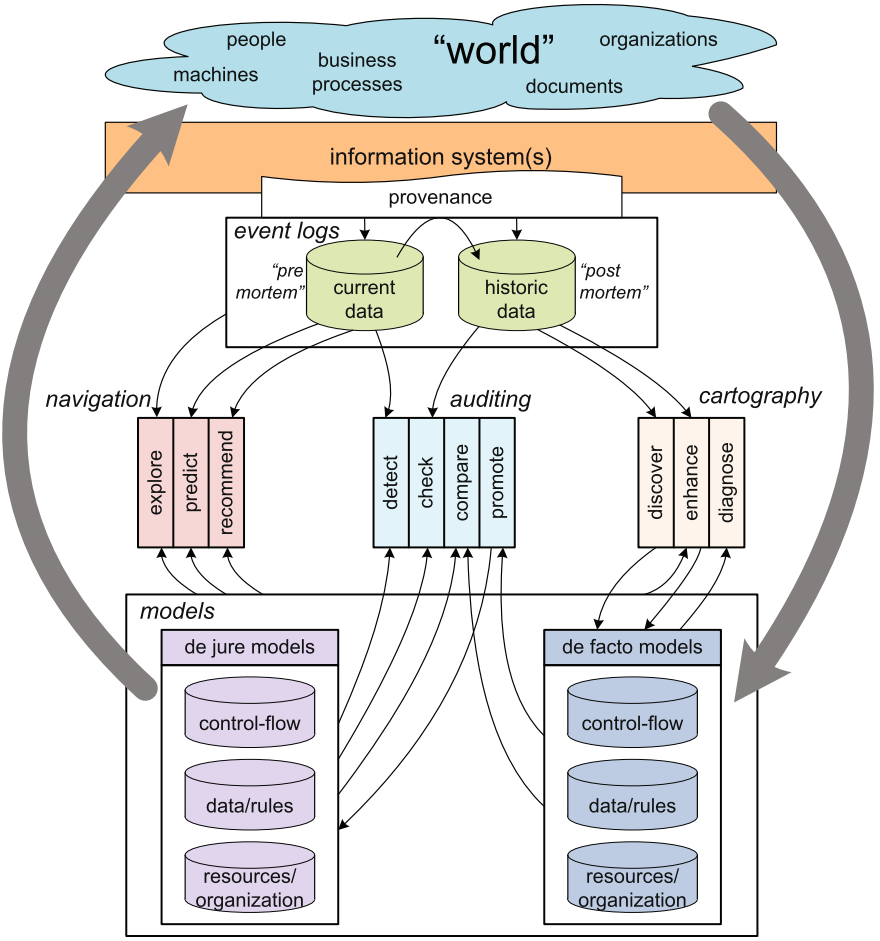


Fig. 10.1 Refined process mining framework

ing, where it refers to the data that is needed to be able to reproduce an experiment [39]. *Business process provenance* aims to systematically collect the information needed to reconstruct what has actually happened in a process or organization. When organizations base their decisions on event data it is essential to make sure that these describe history well. Moreover, from an auditing point of view it is necessary to ensure that event logs cannot be tampered with. Business process provenance refers to the set of activities needed to ensure that history, as captured in event logs, "cannot be rewritten or obscured" such that it can serve as a reliable basis for process improvement and auditing.

Data in event logs are partitioned into *"pre mortem"* and *"post mortem" event data* in the refined process mining framework depicted in Fig. 10.1. "Post mortem" event data refer to information about cases that have completed, i.e., these data can

be used for process improvement and auditing, but not for influencing the cases they refer to. Most event logs considered thus far contained only historic, i.e., "post mortem", event data. "Pre mortem" event data refer to cases that have not yet completed. If a case is still running, i.e., the case is still "alive" (pre mortem), then it may be possible that information in the event log about this case (i.e., current data) can be exploited to ensure the correct or efficient handling of this case.

"Post mortem" event data is most relevant for *off-line process mining*, e.g., discovering the control-flow of a process based on one year of event data. For *online process mining* a mixture of "pre mortem" (current) and "post mortem" (historic) data is needed. For example, historic information can be used to learn a predictive model. Subsequently, information about a running case is combined with the predictive model to provide an estimate for the remaining flow time of the case.

The refined process mining framework also distinguishes between two types of models: *"de jure models"* and *"de facto models"*. *A de jure model is normative*, i.e., it specifies how things should be done or handled. For example, a process model used to configure a BPM system is normative and forces people to work in a particular way. *A de facto model is descriptive* and its goal is not to steer or control reality. Instead, de facto models aim to capture reality. The techniques presented in Chaps. 6 and 7 aim to produce de facto models. Figure 10.1 also highlights that models can cover different perspectives, i.e., process mining is not limited to control-flow and is also concerned with resources, data, organizational entities, decision points, costs, etc. The two large arrows in Fig. 10.1 illustrate that de facto models are derived from reality (right downward arrow) and that de jure models aim to influence reality (left upward arrow).

After refining event logs into "pre mortem" and "post mortem" and partitioning models into "de jure" and "de facto", we can identify ten process mining related activities as shown in Fig. 10.1. These ten activities are grouped into three categories: *cartography*, *auditing*, and *navigation*.

10.1.1 Cartography

Process models can be seen as the "maps" describing the operational processes of organizations, i.e., just like geographic maps, process models aim to describe reality. In order to do this, *abstractions* are needed. For example, on a roadmap a highway may be denoted by an orange line having a thickness of four millimeters. In reality the highway will not be orange; the orange coloring is just used to emphasize the importance of highways. If the scale of the map is 1 : 500000, then the thickness of the line corresponds to a highway of 2 kilometers wide. In reality, the highway will not be so broad. If the thickness of the line would correspond to reality (assuming the same scale), it would be approximately 0.05 millimeter (for a highway of 25 meters wide). Hence, the highway would be (close to) invisible. Therefore, the scale is modified to make the map more readable and useful. When making process models, we need to use similar abstractions. In Chap. 15, we will elaborate on the relationships between process maps and geographic maps. Also note that in Sect. 6.4.4 we

already used the metaphor of a "process view" to argue that a discovered process model views reality from a particular "angle", is "framed", and shown using a particular "resolution". Metaphors such as "maps" and "views" help in understanding the role of process models in BPM.

Figure 10.1 shows that three activities are grouped under cartography: *discover*, *enhance*, and *diagnose*.

- *Discover.* This activity is concerned with the extraction of (process) models as discussed in Chaps. 6 and 7.
- *Enhance.* When existing process models (either discovered or hand-made) can be related to event logs, it is possible to enhance these models. The connection can be used to repair models or to extend them. In Sect. 8.5.1, we showed that models can be made more faithful using the diagnostics provided by conformance checking techniques. Chap. 9 illustrated how attributes in event logs can be used to add additional perspectives to a model.
- *Diagnose.* This activity does not directly use event logs and focuses on classical model-based process analysis as discussed in Sect. 3.3, e.g., process models can be checked for the absence of deadlocks or alternative models can be simulated to estimate the effect of various redesigns on average cycle times.

10.1.2 Auditing

In Sect. 8.1, we defined *auditing* as the set of activities used to check whether business processes are executed within certain boundaries set by managers, governments, and other stakeholders [166]. In Fig. 10.1, the auditing category groups all activities that are concerned with the comparison of behaviors, e.g., two process models or a process model and an event log are put side by side.

- *Detect.* This activity compares de jure models with current "pre mortem" data (events of running process instances) with the goal to detect deviations at runtime. The moment a predefined rule is violated, an alert is generated.
- *Check.* As demonstrated in Chap. 8, historic "post mortem" data can be cross-checked with de jure models. The goal of this activity is to pinpoint deviations and quantify the level of compliance.
- *Compare.* De facto models can be compared with de jure models to see in what way reality deviates from what was planned or expected. Unlike for the previous two activities, no event log is used directly. However, the de facto model may have been discovered using historic data; this way event data are used indirectly for the comparison. In Sect. 8.4, we showed that footprints can be used for model-to-model (and log-to-model) comparisons.
- *Promote.* Based on an analysis of the differences between a de facto model and a de jure model, it is possible to promote parts of the de facto model to a new de jure model. By promoting proven "best practices" to the de jure model, existing processes can be improved.

Note that the *detect* and *check* activities are similar except for the event data used. The former activity uses "pre mortem" data and aims at online analysis to be able to react immediately when a discrepancy is detected. The latter activity uses "post mortem" data and is done off-line.

10.1.3 Navigation

The last category of process mining activities aim at business process *navigation*. Unlike the cartography and auditing activities, navigation activities are forward-looking. For example, process mining techniques can be used to make predictions about the future of a particular case and guide the user in selecting suitable actions. When comparing this with a car navigation system from TomTom or Garmin, this corresponds to functionalities such predicting the arrival time and guiding the driver using spoken instructions. In Chap. 15, we elaborate on the similarities between car navigation and process mining.

Figure 10.1 lists three navigation activities: *explore*, *predict*, and *recommend*.

- *Explore.* The combination of event data and models can be used to explore business processes at run-time. Running cases can be visualized and compared with similar cases that were handled earlier.
- *Predict.* By combining information about running cases with models (discovered or hand-made), it is possible to make predictions about the future, e.g., the remaining flow time and the probability of success.
- *Recommend.* The information used for predicting the future can also be used to recommend suitable actions (e.g. to minimize costs or time). The goal is to enable functionality similar to the guidance given by car navigation systems.

In earlier chapters, we focused on activities using historic ("post mortem") data only, i.e., activities *discover*, *enhance*, and *check* in Fig. 10.1. In the remainder of this chapter, we shift our attention to online analysis also using "pre mortem" data.

10.2 Online Process Mining

Traditionally, process mining has been used in an off-line fashion using only "post mortem" data. This means that only completed cases are being considered, i.e., the traces in the event log are *complete* traces corresponding to cases that were fully handled in the past. For *operational support* we also consider "pre mortem" event data and respond to such data in an online fashion. Now only running cases are considered as these can, potentially, still be influenced. A running case may still generate events. Therefore, it is described by a *partial* trace.

Figure 10.2 shows the essence of operational support. Consider a case for which activities a and b have been executed. Partial trace $\sigma_p = \langle a, b \rangle$ describes the known

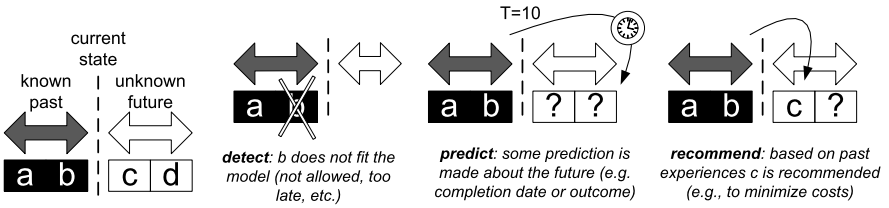


Fig. 10.2 Three process mining activities related to operational support: *detect*, *predict*, and *recommend*

Table 10.1 Fragment of event log with timestamps and transactional information. For instance, event a^{12}_{start} denotes the start of activity a at time 12

Case id	Trace
1	$\langle a^{12}_{start}, a^{19}_{complete}, b^{25}_{start}, d^{26}_{start}, b^{32}_{complete}, d^{33}_{complete}, e^{35}_{start}, e^{40}_{complete}, h^{50}_{start}, h^{54}_{complete}\rangle$
2	$\langle a^{17}_{start}, a^{23}_{complete}, d^{28}_{start}, c^{30}_{start}, d^{32}_{complete}, c^{38}_{complete}, e^{50}_{start}, e^{59}_{complete}, g^{70}_{start}, g^{73}_{complete}\rangle$
3	$\langle a^{25}_{start}, a^{30}_{complete}, c^{32}_{start}, c^{35}_{complete}, d^{35}_{start}, d^{40}_{complete}, e^{45}_{start}, e^{50}_{complete}, f^{50}_{start}, f^{55}_{complete}, b^{60}_{start}, d^{62}_{start}, b^{65}_{complete}, d^{67}_{complete}, e^{80}_{start}, e^{87}_{complete}, g^{90}_{start}, g^{98}_{complete}\rangle$
...	...

past of the case. Note that the two events may have all kinds of attributes (e.g., timestamps and associated resources), but these are not shown here. In the state after observing σ_p, the future of the case is not known yet. One possible future could be that c and d will be executed resulting in a complete trace $\sigma_c = \langle a, b, c, d\rangle$. Figure 10.2 shows three operational support activities: *detect*, *predict*, and *recommend*. These correspond to the activities already mentioned in the context of Fig. 10.1.

- *Detect*. This activity compares the partial trace σ_p with some normative model, e.g., a process model or an LTL constraint. Such a check could reveal a violation as shown in Fig. 10.2. If b was not allowed after a, an alert would be generated.
- *Predict*. This activity makes statements about the events following σ_p. For example, the expected completion time could be predicted by comparing the current case to similar cases that were handled in the past.
- *Recommend*. Recommendations guide the user in selecting the next activity after σ_p. For example, it could be that, based on historic information, it is recommended to execute activity c next (e.g., to minimize costs or flow time).

Note that all three activities assume some model, e.g., predictions and recommendations could be based on a regression model or obtained using simulation. Besides the three operational support activities illustrated by Fig. 10.2, it is also possible to simply explore partial traces. For example, dotted chart visualization and other visual analytics techniques can also be applied to running cases.

In the remainder, we show how some of the process mining techniques presented earlier can be modified to provide operational support. In order to do this, we use the event log shown in Table 10.1. This log was also used in earlier chapters and is based

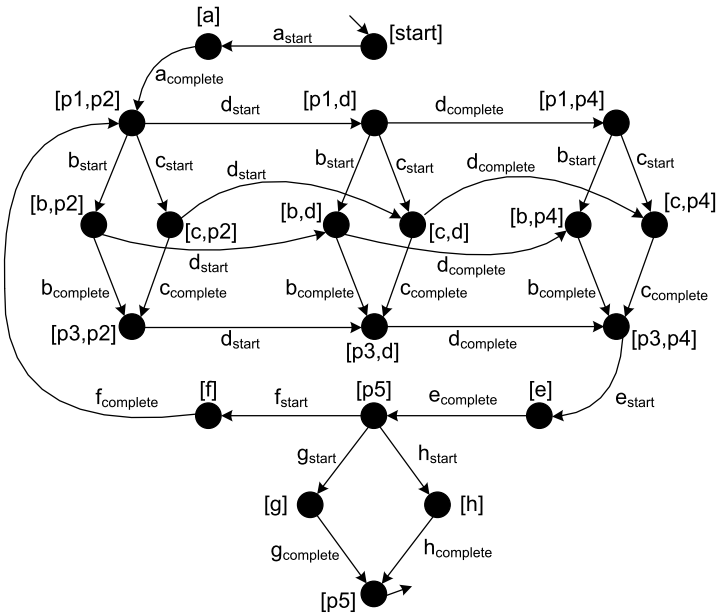


Fig. 10.3 Transition system modeling the process that generated the event log shown in Table 10.1. This process was already modeled in terms of a WF-net (Fig. 2.2) and in terms of BPMN (Fig. 2.3). However, in this transition system we model the start and completion of an activity explicitly. In terms of the WF-net in Fig. 2.2, this means that transition a is split into transitions a_{start} and $a_{complete}$ connected by a place named a; etc.

on the running example introduced in Chap. 2. The WF-net shown in Fig. 2.2 models the process for which events have been recorded in Table 10.1. Figure 2.3 models the same process in terms of BPMN. Independent of the notation used, we can also derive a transition system modeling the same process as is shown in Fig. 10.3. The transition system labels the nodes with markings of the corresponding Petri net in which each activity is modeled by a start and complete transition. Transition a_{start} consumes a token from place *start* and produces a token for place a. The token in place a models that activity a is being executed. Transition $a_{complete}$ consumes a token from place a and produces a token for each of the places $p1$ and $p2$ (state $[p1, p2]$ in Fig. 10.3). The state labeled $[b, d]$ in Fig. 10.3 corresponds to the marking with tokens in b and d, i.e., activities b and d are being executed in parallel. The state space of the BPMN model shown in Fig. 2.3 is isomorphic to the transition system shown in Fig. 10.3.

10.3 Detect

The first operational support activity we elaborate on is *detecting* deviations at runtime. This can be seen as conformance checking "on-the-fly". Compared to confor-

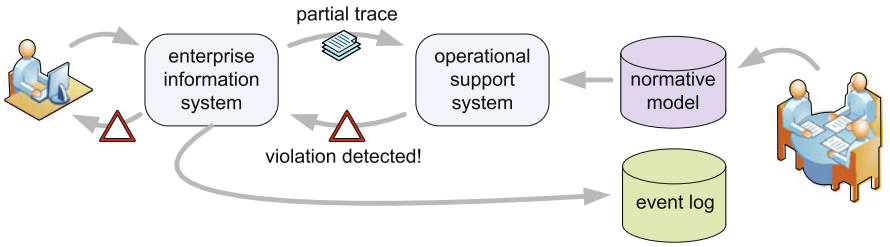


Fig. 10.4 Detecting violations at run-time: the moment a deviation is detected, an alert is generated

mance checking as described in Chap. 8 there are two important differences: (a) we do no consider the log as a whole but focus on the *partial trace of a particular case*, and (b) in case of a deviation there should be an *immediate response* when the deviation occurs. Figure 10.4 illustrates this type of operational support. Users are interacting with some enterprise information system. Based on their actions, events are recorded. The partial trace of each case is continuously checked by the operational support system, i.e., each time an event occurs, the partial trace of the corresponding case is sent to the operational support system. The operational support system immediately generates an alert if a deviation is detected. The enterprise information system and its users can take appropriate actions based on this alert, e.g., a manager is notified such that corrective actions can be taken.

All cases in the event log shown in Table 10.1 conform to the transition system of Fig. 10.3, the WF-net shown in Fig. 2.2, and the BPMN model shown in Fig. 2.3. Therefore, when these cases were executing, no deviations could be detected with respect to these models. Assume now that the more restrictive WF-net shown in Fig. 10.5 describes the desired normative behavior. Compared to the original model activity d (i.e., checking the ticket) should occur after b or c (i.e., one of the examinations).[1]

Let us now consider the first case, $\sigma_1 = \langle a^{12}_{start}, a^{19}_{complete}, b^{25}_{start}, d^{26}_{start}, b^{32}_{complete}, d^{33}_{complete}, e^{35}_{start}, e^{40}_{complete}, h^{50}_{start}, h^{54}_{complete} \rangle$. After each event it is checked whether there is a deviation or not. At time 12, after executing the first event a^{12}_{start} no deviation is found, because trace $\langle a^{12}_{start} \rangle$ can be replayed in Fig. 10.5 without missing tokens.[2] The next two events can also be replayed, i.e., $\langle a^{12}_{start}, a^{19}_{complete}, b^{25}_{start} \rangle$ is a possible firing sequence of the WF-net in which each activity is refined into a start

[1]Note that this diagram can be simplified by removing place $c2$, the arc from $c3$ to e, and the arc from d to $c3$ (i.e., N_2 in Fig. 8.2). The simplified model has the same behavior, i.e., both are bisimilar.

[2]The WF-net Fig. 10.5 has only one transition per activity while the log contains start and complete events. As described in Sect. 6.2.4, each activity can be described by a small subprocess. Assume that all transitions in Fig. 10.5 are split into a start transition and complete transition connected through a place named after the activity. For example, transition a is refined into transitions a_{start} and $a_{complete}$ connected by a place a. Note that the transition system in Fig. 10.3 used the same naming convention.

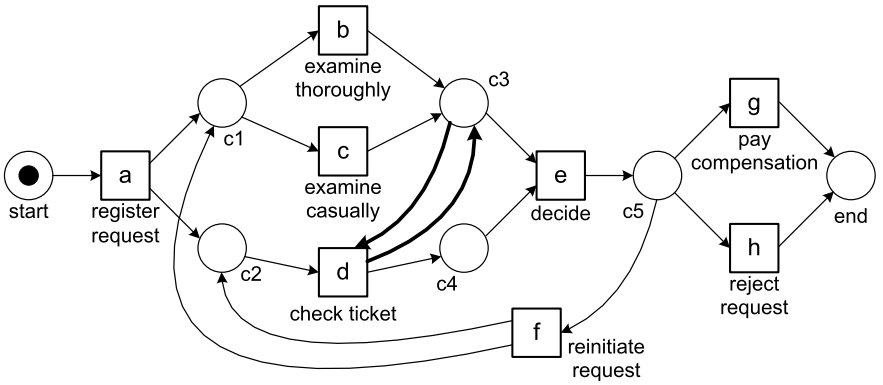
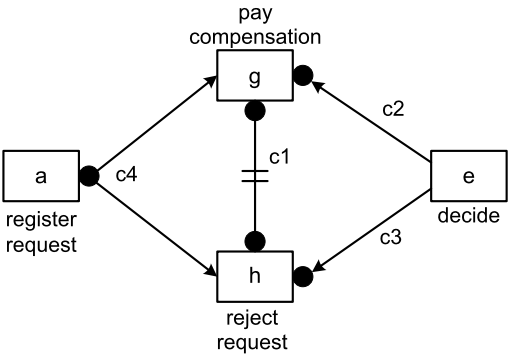


Fig. 10.5 WF-net modeling an additional constraint: d can only be started once b or c has completed

Fig. 10.6 Declare specification composed of four constraints: $c1$, $c2$, $c3$, and $c4$



and complete transition. The state after replaying the three events is $[c2, b]$. The next event, i.e., d^{26}_{start} is not possible in this state. Hence, an *alert* is generated at time 26 based on partial trace $\langle a^{12}_{start}, a^{19}_{complete}, b^{25}_{start}, d^{26}_{start}\rangle$. The alert signals that activity d was started without being enabled. For the second case a deviation is detected at time 28; based on the partial trace $\langle a^{17}_{start}, a^{23}_{complete}, d^{28}_{start}\rangle$ an alert is generated stating that d was started before it was enabled. For the third case a deviation is detected at time 62. The prefix $\langle a^{25}_{start}, a^{30}_{complete}, c^{32}_{start}, c^{35}_{complete}, d^{35}_{start}, d^{40}_{complete}, e^{45}_{start}, e^{50}_{complete}, f^{50}_{start}, f^{55}_{complete}, b^{60}_{start}, d^{62}_{start}\rangle$ cannot be replayed properly because the second instance of d is started without being enabled. These examples show that the replay approach from Chap. 8 can also be used at run-time for detecting deviations the moment they happen.

In Chap. 8, we introduced *Declare* as an example of a constraint-based language. We used the Declare model shown in Fig. 10.6 to explain some of basic concepts. Each of the four constraints shown can be specified in terms of LTL. Constraint $c1$ is a non-coexistence constraint stating that g and h should not both happen. The LTL expression for this constraint is $!((\Diamond g) \wedge (\Diamond h))$. Constraint $c2$ is a precedence

constraint $((!g)\ W\ e)$ modeling the requirement that g should not happen before e has happened. Constraint $c3$ is a similar precedence constraint, but now referring to h rather than g. Constraint $c4$ is a branched response constraint stating that every occurrence of a should eventually be followed by g or h, i.e., $\Box(a \Rightarrow (\Diamond(g \vee h)))$.

Consider some case having a partial trace σ_p listing the events that have happened thus far. Each constraint c in Fig. 10.6 is in one of the following states for partial trace σ_p:

- *Satisfied*. The LTL formula corresponding to c evaluates to true for the partial trace σ_p.
- *Temporarily violated*. The LTL formula corresponding to c evaluates to false for σ_p, however, there is a longer trace σ'_p that has σ_p as a prefix and for which the LTL formula corresponding to c evaluates to true.
- *Permanently violated*. The LTL formula corresponding to c evaluates to false for σ_p and all its extensions, i.e., there is no σ'_p that has σ_p as a prefix and for which the LTL formula evaluates to true.

These three notions can be lifted from the level of a *single constraint* to the level of a *complete Declare specification*. A Declare specification is *satisfied* for a case if all of its constraints are satisfied. A Declare specification is *temporarily violated* by a case if for the current partial trace at least one of the constraints is violated, however, there is a possible future in which all constraints are satisfied. A Declare specification is *permanently violated* by a case if no such future exists.

None of the cases shown in Table 10.1 violates any of the constraints shown in Fig. 10.6, i.e., for each trace, at the *end*, all constraints are satisfied. Let us now consider a scenario in which for a case the trace $\sigma = \langle a, b, d, g \rangle$ is executed. For simplicity, we removed timestamps and transactional information. Initially, i.e., for trace $\sigma_0 = \langle\ \rangle$, all constraints are satisfied. After executing a, i.e., for prefix $\sigma_1 = \langle a \rangle$, constraint $c4$ is temporarily violated. Because there is a possible future in which all constraints are satisfied, there is no need to generate an alert. However, diagnostic information stating that constraint $c4$ is temporarily violated could be provided. Executing b and d does not change the situation, i.e., both partial traces $\sigma_2 = \langle a, b \rangle$ and $\sigma_3 = \langle a, b, d \rangle$ temporarily violate $c4$. However, after executing g the situation changes. Partial trace $\sigma_4 = \langle a, b, d, g \rangle$ satisfies constraint $c4$. However, constraint $c2$ is permanently violated by σ_4 as there is no "possible future" in which e occurs before g. Therefore, a deviation is detected and reported.

Figure 10.7 shows another Declare model. Constraint $c1$ is the same non-coexistence constraint as before. Constraint $c2$ is a response constraint stating that every occurrence of activity e should eventually be followed by g, i.e., $\Box(e \Rightarrow (\Diamond g))$ in LTL terms. Constraint $c3$ is a similar response constraint (every occurrence of activity e should eventually be followed by h). Constraint $c4$ is a precedence constraint $((!g)\ W\ a)$ modeling the requirement that g should not happen before a has happened. Constraint $c5$ is also a precedence constraint $((!h)\ W\ a)$. Assume that Fig. 10.7 is the normative model. Let us first consider a scenario in which for a case the trace $\sigma = \langle a, b, d, g \rangle$ is executed. For all prefixes, all of the constraints are satisfied, i.e., no alerts need to be executed during the lifetime of this case. Let us

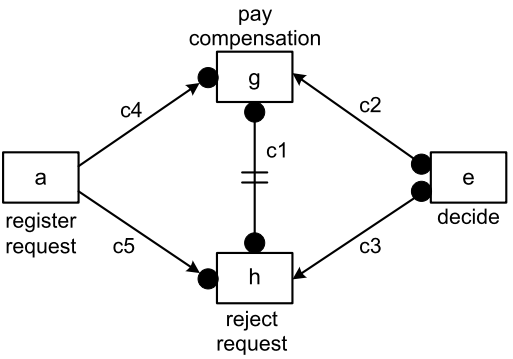


Fig. 10.7 Another Declare specification. Note that $c1$, $c2$, and $c3$ imply that e cannot be executed without permanently violating the specification

now consider the scenario $\sigma = \langle a, b, d, e, g \rangle$. No alerts need to be generated for the first three events. In fact at any stage all five constraints are satisfied. However, after executing e constraints $c2$ and $c3$ are temporarily violated. To remove these temporary violations, both g and h need to be executed after $\langle a, b, d, e \rangle$. However, the execution of both g and h results in a permanent violation of $c1$. Because there is no possible future in which *all* constraints are satisfied, the Declare specification is permanently violated by prefix $\langle a, b, d, e \rangle$ and an alert is generated directly after e occurs. Note that in the latter scenario, there are only temporarily violated constraints whereas the whole specification is permanently violated. Therefore, advanced reasoning is required to determine whether an event signifies a deviation or not. As shown in [103, 162], one can use model checking or abductive logic programming to detect such deviations and provide informative alerts.

10.4 Predict

The second operational support activity we consider is *prediction*. As shown in Fig. 10.8, we again consider the setting in which users are interacting with some enterprise information system. The events recorded for cases can be sent to the operational support system in the form of partial traces. Based on such a partial trace and some predictive model, a prediction is generated. Examples of predictions are:

- The predicted remaining flow time is 14 days;
- The predicted probability of meeting the legal deadline is 0.72;
- The predicted total cost of this case is € 4500;
- The predicted probability that activity a will occur is 0.34;
- The predicted probability that person r will work on this case is 0.57;
- The predicted probability that a case will be rejected is 0.67; and
- The predicted total service time is 98 minutes.

In the fictive example shown in Fig. 10.8, the operational support system predicts that the completion date will be April 25th, 2011.

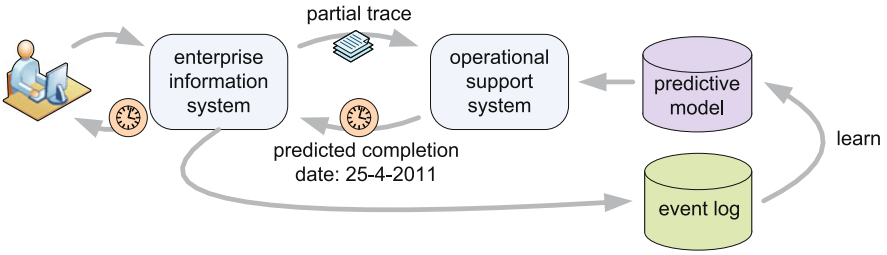


Fig. 10.8 Both the partial trace of a running case and some predictive model are used to provide a prediction (e.g., remaining flow time, expected total costs, or probability of success)

Various techniques can be used to generate predictions. For example, the supervised learning techniques discussed in Sect. 4.1.2 can be used to answer some of these questions. Using feature extraction, relevant properties of the partial trace need to be mapped onto predictor variables. Moreover, the feature we would like to predict is mapped onto a response variable. The response variable is often a performance indicator, e.g., remaining flow time or total costs. If the response variable is numeric, typically regression analysis is used. For a categorical response variable, classification techniques such as decision tree learning can be used. The predictive model is based on historic "post mortem" event data, but can be used to make predictions for the cases that are still running.

Given the variety of approaches and the broad spectrum of possible questions, we cannot provide a comprehensive overview of prediction techniques. Therefore, as an example, we select one particular technique answering a specific question. In the remainder, we show how to *predict the remaining flow time using an annotated transition system* [164, 167]. Starting point for this approach is an event log with timestamps as shown in Table 10.1 and a transition system such as the one shown in Fig. 10.3. The transition system can be obtained by computing the state-space of a process model expressed in another language (WF-nets, BPMN, YAWL, EPCs, etc.). For example, the transition system in Fig. 10.3 can be obtained from the WF-net in Fig. 2.2 or the BPMN model in Fig. 2.3. The transition system can also be obtained using the technique described in Sect. 7.4.1, i.e., using an event log L and a state representation function $l^{state}()$, one can automatically generate a transition system able to replay the event log.

Assuming that the event log fits the transition system, one can replay the events on the model and collect timing information. Non-fitting events and/or cases can be simply ignored or handled as described in Sect. 8.2. Figure 10.9 shows the timed replay of the first two traces in Table 10.1.

Let us consider the first case, $\langle a^{12}_{start}, a^{19}_{complete}, b^{25}_{start}, d^{26}_{start}, b^{32}_{complete}, d^{33}_{complete}, e^{35}_{start}, e^{40}_{complete}, h^{50}_{start}, h^{54}_{complete}\rangle$. This case started at time 12 and ended at time 54. Hence, its flow time was 42 time units. States visited by this case are annotated with a tag (t, e, r, s) where t is the *time* the state is visited, e is the *elapsed time* since the start when visiting the state, r is the *remaining flow time*, and s is the *sojourn time*. State $[a]$ is tagged with the annotation $(t = 12, e = 0, r = 42, s = 7)$ because

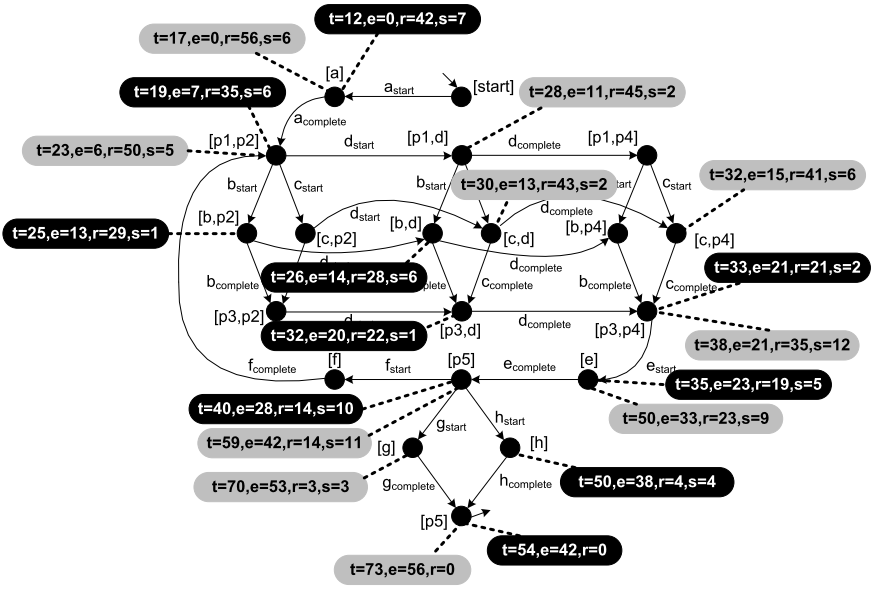


Fig. 10.9 Statistics collected while replaying the first two cases: t is the time the state is visited, e is the elapsed time since the start when visiting the state, r is the remaining flow time, and s is the sojourn time

this state was visited by the case directly after the first event a^{12}_{start} occurred. $t = 12$ because event a^{12}_{start} occurred at time 12. $e = 12 - 12 = 0$ because no time elapsed after executing just one event. $r = 54 - 12 = 42$ is the remaining time until the end of the case after a was started at time 12. $s = 19 - 12 = 7$ because the next event occurred 7 time units later. State $[p1, p2]$ is tagged with annotation $(t = 19, e = 7, r = 35, s = 6)$ because a completed at time $t = 19$. $e = 19 - 12 = 7$ because a completed 7 time units after the case started. $r = 54 - 19 = 35$ because the case ended at time 54. $s = 25 - 19 = 6$ because the next event occurred 6 time units later. Figure 10.9 shows all annotations related to the first two cases. For example, state $[p3, p4]$ was visited once by each of the two cases resulting in annotations $(t = 33, e = 21, r = 21, s = 2)$ and $(t = 38, e = 21, r = 35, s = 12)$. The initial state $[start]$ has no annotations since no events have occurred when visiting this state. The final state $[p5]$ has no sojourn time because there is no next event when visiting this state.

Table 10.1 shows only a fragment of the whole event log. However, it is obvious that the other cases in the log can be replayed in a similar fashion to gather more annotations. For example, the third case visited state $[p3, p4]$ twice, after event $d^{40}_{complete}$ and after event $d^{67}_{complete}$. The first visit resulted in annotation $(t = 40, e = 15, r = 58, s = 5)$ and the second visit resulted in annotation $(t = 67, e = 42, r = 31, s = 13)$. Assuming a large event log, there may be hundreds or even thousands of annotations per state. For each state x it is possible to create

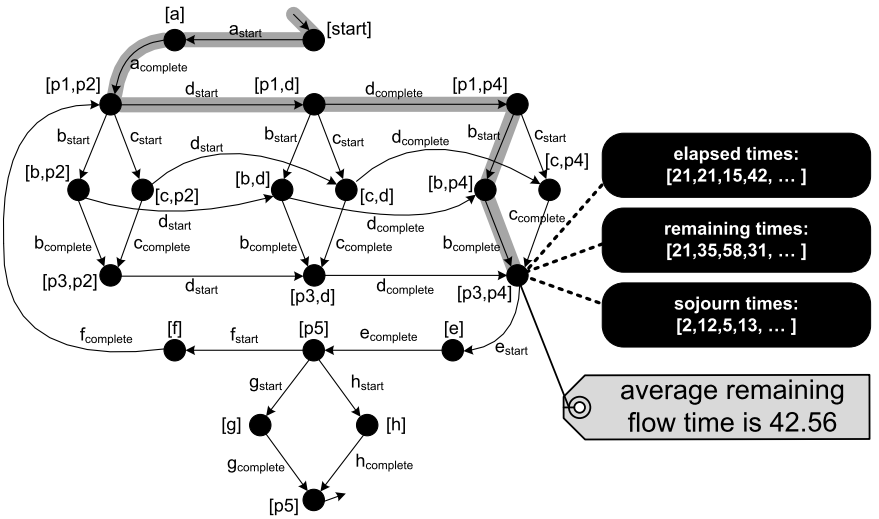


Fig. 10.10 Each state has a multi-set of remaining flow times (one element for each visit). This is the basis for predicting the remaining flow time of future cases. For a case with partial trace $\langle a^{512}_{start}, a^{518}_{complete}, d^{525}_{start}, d^{526}_{complete}, b^{532}_{start}, b^{533}_{complete}\rangle$, the predicted remaining flow time is 42.56. This is the mean remaining flow time of cases in state $[p3, p4]$

a multi-set $Q^{remaining}_x$ of remaining flow times based on these annotations. For state $[p3, p4]$ this multi-set is $Q^{remaining}_{[p3,p4]} = [21, 35, 58, 31, \ldots]$: the first case visited state $[p3, p4]$ once (21 time units before completion), the second case visited $[p3, p4]$ once (35 time units before completion), the third case visited $[p3, p4]$ twice (58 and 31 time units before completion), etc. Similar multi-sets exist for elapsed times ($Q^{elapsed}_{[p3,p4]} = [21, 21, 15, 42, \ldots]$) and sojourn times ($Q^{sojourn}_{[p3,p4]} = [2, 12, 5, 13, \ldots]$). Based on these multi-sets all kinds of statistics can be computed. For example, the *mean remaining flow time* in state $[p3, p4]$ is $\sum_{q \in Q} \frac{Q(q) \times q}{|Q|}$ with $Q = Q^{remaining}_{[p3,p4]}$. Like in Sect. 9.4, it is possible to compute other standard statistics such as standard deviation, minimum, and maximum. One can also fit a distribution on the sample data using standard statistical software. For example, based on the samples $Q^{remaining}_{[p3,p4]} = [21, 35, 58, 31, \ldots]$ one could find that these remaining flow times are best described by a Gamma distribution with parameters $r = 8.0502$ and $\lambda = 0.18915$. This distribution has a mean of 42.56 and a standard deviation of 15.0. As shown in Chap. 9, such insights can be used to extend models with the time information. Moreover, the annotated transition system can also be used actively, and predict the remaining time for a running case.

Figure 10.10 shows the transition system with annotations for state $[p3, p4]$. Moreover, the path of a partial trace of a case that is still running is highlighted in the figure. The partial trace of this case is $\langle a^{512}_{start}, a^{518}_{complete}, d^{525}_{start}, d^{526}_{complete}, b^{532}_{start}, b^{533}_{complete}\rangle$. At time 533 we are interested in the remaining flow time of this case. An obvious predictor for the remaining flow time of the running case is the mean

remaining flow time of all earlier cases in the same state, i.e., 42.56. Hence, the case is expected to complete around time 575.56. This illustrates that for any running case, at any point in time, one can predict the remaining flow time.

The annotated transition system can be used to make more refined statements about the predicted remaining flow time. For example, it is clear that the size of multi-set $Q^{remaining}_{[p3,p4]}$ and the standard deviation of the historic samples in this multi-set have impact on the reliability of the prediction. Rather than giving a single prediction value, it is also possible to produce predictions like "With 90% confidence the remaining flow time is predicted to be between 40 and 45 days" or "78% of similar cases were handled within 50 days". Moreover, as shown in [167], it is possible to use cross-validation to determine the quality of predictions.

The approach based on an annotated transition system is not restricted to predicting the remaining flow time. Obviously, one could predict the sojourn time in a similar fashion. Moreover, also non-time related predictions can be made using the same approach. For example, suppose that we are interested in whether the request is accepted (activity g occurs) or rejected (activity h occurs). To make such predictions, we annotate states with information about known outcomes for "post mortem" cases. For example, $Q^{accepted}_{[p3,p4]} = [0, 1, 1, 1, \ldots]$. For state $[p3, p4]$, a "0" is added to this multi-set for each visit of a case that will be rejected and "1" is added for each visit of a case that will be accepted. The average value of $Q^{accepted}_{[p3,p4]}$ is a predictor for the probability that a case visiting state $[p3, p4]$ will be accepted. This example shows that a *wide variety of predictions* can be generated using a suitable annotated transition system. It is important to note that process-related information is taken into account, i.e., the prediction is based on the *state* of the running case rather than some static attribute. Classical data mining approaches (e.g., based on regression or decision trees) typically use static attributes of a case rather than state information.

The transition system shown in Fig. 10.10 happens to coincide with the states of the WF-net and BPMN model provided earlier. However, as discussed in Sect. 7.4.1, different transition systems can be constructed based on an event log. The event log L and the state representation function $l^{state}()$ determine the level of detail and the aspects considered. For example, it is possible to abstract from irrelevant activities resulting in a more coarse-grained transition system. However, it is also possible to include information about resources and data in the state, thus resulting in a more fine-grained transition system. There should be sufficient visits to all states to make reliable predictions. The transition system is too fine-grained if many states are rarely visited when replaying log L. The level of abstraction should be consistent with the size of the log and the response variable that needs to be predicted. For supervised learning this is generally referred to as the problem of feature extraction, i.e., determining the predictor variables that are most relevant for predicting the response variable. See [167] for more details and examples.

The approach based on annotated transition systems is just one of many approaches that could be used for prediction. For example, *short-term simulation* could be used to explore the possible futures of a particular case in a particular state (see Sect. 9.6). The simulation model learned based on historic data is initialized with the

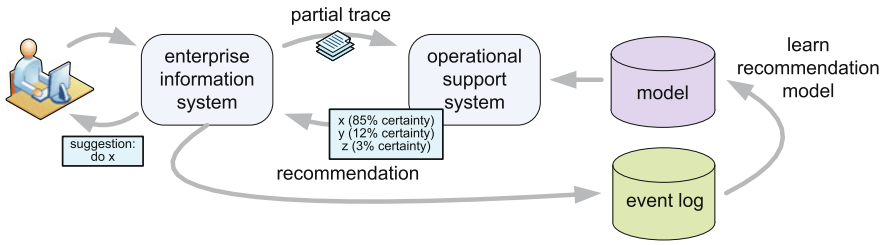


Fig. 10.11 A model based on historic data is used to provide recommendations for running cases. Recommendations are not enforced and may have quality attributes attached, e.g., in 85% of similar cases, x is the activity that minimizes flow time

current state of the running case. Subsequently, the remaining lifetime of the case is simulated repeatedly to obtain sample measurements for the performance indicator to be predicted.

10.5 Recommend

The third operational support activity we consider in this chapter is *recommendation*. As Fig. 10.11 shows, the setting is similar to prediction, i.e., a partial trace is sent to the operational support system followed by a response. However, the response is not a prediction but a recommendation about what to do next. To provide such a recommendation, a model is learned from "post mortem" event data. Moreover, the operational support system should know what the *decision space* is, i.e., what are the possible actions from which to choose one. Based on the recommendation model these actions are ordered. For example, in Fig. 10.11 the operational support system recommends to do action x with 85% certainty. The other two possible actions have a "lower" recommendation: y is recommended with 12% certainty and z is recommended with 3% certainty. In most cases it is impossible to give a recommendation that is guaranteed to be optimal; the best choice for the next step may depend on the occurrence of unknown external events in the future. For example, in Fig. 10.11 there may be cases for which z turns out to be the best choice.

A recommendation is always given *with respect to a specific goal*. Examples of goals are:

- Minimize the remaining flow time;
- Minimize the total costs;
- Maximize the fraction of cases handled within 4 weeks;
- Maximize the fraction of cases that is accepted; and
- Minimize resource usage.

These goals can also be aggregated and combined, e.g., to balance between cost reduction and flow time reduction. To operationalize such a goal, a performance indicator needs to be defined, e.g., remaining flow time or total costs. This performance indicator corresponds to the response variable in supervised learning.

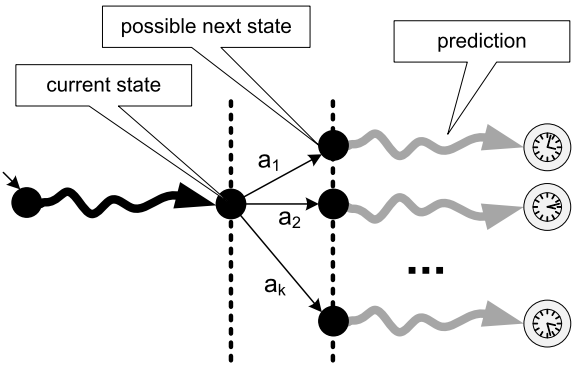


Fig. 10.12 Recommendations can be based on predictions. For every possible choice, simply predict the performance indicator of interest. Then, recommend the best one(s)

A recommendation makes statements about a set of possible actions, i.e., the decision space. The decision space may be a set of activities, e.g., $\{f, g, h\}$. This means that in the current state activities f, g, and h are possible candidates and the question to be answered by the operational support system is "Which candidate is best given the goal selected?". However, the decision space may also consist of a set of resources and the goal is then to recommend the best resource to execute a given activity. For example, the operational support system could recommend allocating activity h to Mike to minimize the flow time. This example shows that recommendations are not limited to control-flow and can also refer to other perspectives. Therefore, we use the term "action" rather than activity. The decision space for a running case may be part of the message sent from the enterprise information system to the operational support system. Otherwise, the recommendation model should be able to derive the decision space based on the partial trace.

As shown in Fig. 10.12, recommending an action to achieve a goal is closely related to predicting the corresponding performance indicator. Suppose that for a case having a partial trace σ_p we need to recommend some action from a set of possible actions $\{a_1, a_2, \ldots, a_k\}$. The existing partial trace can be extended by assuming that action a_1 is selected (although it did not happen yet). σ_1 is the resulting extended partial trace, i.e., $\sigma_1 = \sigma_p \oplus a_1$. (Here we assume that a_1 is an activity and we use simple traces.) The same can be done for all other actions resulting in a set of partial traces $D = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$. Now a prediction is made for the selected performance indicator and each element of D. The resulting predictions are compared and ranked. If σ_2 has the best predicted value (e.g., shortest remaining flow time), then a_2 is recommended first.

Depending on the prediction technique used, the recommendation can also include information about its reliability/quality, e.g., the confidence or certainty that a particular selection is optimal with respect to the goal. For example, in Fig. 10.11 the recommendation attaches a confidence to each of the three possible actions. How to interpret such confidence values depends on the underlying prediction method. For example, if short-term simulation is used, then the 85% certainty of x mentioned in Fig. 10.11 (i.e., the confidence attached to recommendation x) would mean that in 85% of the simulation experiments action x resulted in the shortest remaining flow time.

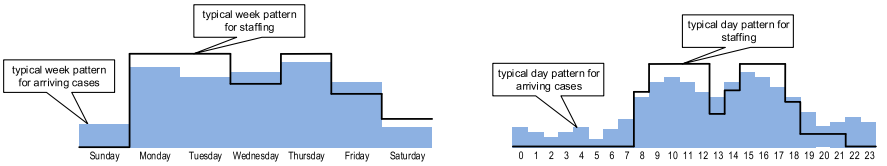


Fig. 10.13 Recurring weekly and daily patterns that are highly predictable but seldom used in analysis

10.6 Processes Are Not in Steady State!

In Sect. 3.1, we argued that models are often an idealized view on reality. As an example, we mentioned the so-called Yerkes–Dodson law suggesting that a person's speed of working increases when workload increases (until a point where performance degrades due to stress). Such phenomena are typically not captured in simulation models [139, 163]. Obviously, this limits the predictive value of such models. Also models learned from event data may be blind to such phenomena. The main complication is that *processes are not in steady state*. Processes are influenced by working hours, weekends, contextual factors, and drifts. Understanding these phenomena is important, not only for operational support, but for interpreting process mining results in real-life settings.

10.6.1 Daily, Weekly and Seasonal Patterns in Processes

Figure 10.13 shows examples of typical weekly and daily patterns. Less work arrives during weekends and at night. Resource availability also fluctuates in a predictable manner due to office hours, lunch breaks and weekends. Staffing may be adapted to such patterns. However, there may still be congestion causing delays (often at foreseeable points in times).

Daily, weekly and seasonal patterns can be learned from event data. This is highly relevant for operational support. Assume that a person works only on Fridays or that she is only in the office in the morning. Using averages over historic data, we may predict that an activity will be completed by this person on Wednesday in the late afternoon. Such predictions are wrong and can easily be detected using cross-validation, however, the corresponding insights easily get lost in standard performance measures. The challenge is to incorporate recurring patterns when making predictions and suggesting improvements.

10.6.2 Contextual Factors

Processes are executed in a particular *context*, but this context is often neglected during analysis [118, 148]. The approach based on an annotated transition system

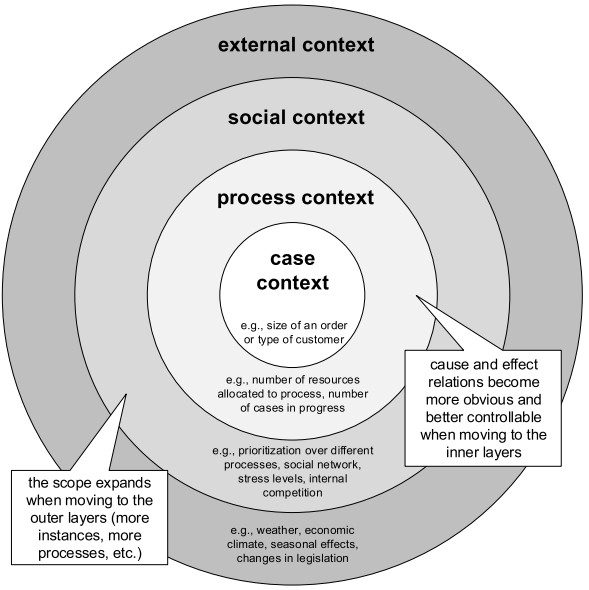


Fig. 10.14 The *context* in which events occur and processes unfold should be taken into account

presented in Sect. 10.4 reduces the context to the average remaining flow time of cases that visited the same state before (see Fig. 10.10). Let us compare predictions in operational processes to predicting driving times by a navigation system. Suppose we would like to know the time it takes to drive from Eindhoven to Amsterdam. Such a prediction could be based on properties of the driver and car. How long did people of the same age category driving the same brand of car take in the past? Such type of prediction is comparable to the analysis of the remaining flow time of cases in state $[p3, p4]$ in Fig. 10.10. However, such analysis neglects *important contextual factors*. The driving time may depend on the weather and the time of the day. More important, the driving time strongly depends on the other cars currently driving in the same direction! This illustrates the relevance of context in process mining.

In Fig. 10.14, we distinguish four types of context relevant for process mining [148]:

- *Case Context.* Process instances (i.e., cases) may have various properties that influence their execution. Consider, for example, the way a customer order is handled. The type of customer placing the order may influence the path the case follows in the process. The size of the order may influence the type of shipping selected or may influence the transportation time. These properties can be directly related to the individual process instance and we refer to them as the *case context*. Typically, it is not difficult to discover relationships between the case context and the observed behavior of the case. For example, one could discover that an activity is typically skipped for gold customers.
- *Process Context.* A process may be instantiated many times, e.g., thousands of customer orders are handled by the same process per year. Yet, the corresponding

process model typically describes the life-cycle of one order in isolation. Although interactions among process instances are not made explicit in such models, cases may influence each other. For example, instances may compete for the same resources. An order may be delayed by too much work-in-progress. Looking at one instance in isolation like in Fig. 10.10 is not sufficient for understanding the observed behavior. Process mining techniques should also consider the *process context*, e.g., the number of instances being handled and the number of resources available for the process. For example, when predicting the expected remaining flow time for a particular case one should not only consider the case context (e.g., the status of the order) but also the process context (e.g., workload and resource availability).

- *Social Context.* The process context considers all factors that can be directly related to a process and its instances. However, people and organizations are typically not allocated to a single process and may be involved in many additional processes. Moreover, activities are executed by people that operate in a social network. Friction between individuals may delay process instances and the speed at which people work may vary due to circumstances that cannot be fully attributed to the process being analyzed. All of these factors are referred to as the *social context*. This context characterizes the way in which people work together *within a particular organization*. Today's process mining techniques tend to neglect the social context even though it is clear that this context directly impacts the way that cases are handled.
- *External Context.* The external context captures all factors that are part of an even wider ecosystem that extends beyond the control sphere of the organization. For example, the weather, the economic climate, and changing regulations may influence the way that cases are being handled. The weather may influence the workload, e.g., a storm or flooding may lead to increased volumes of insurance claims. Changing oil prices may influence the number of customer orders (e.g., the demand for heating oil increases when prices drop). More stringent identity checks may influence the order in which social security related activities are being executed. Although the external context can have a dramatic impact on the process being analyzed, it is difficult to select the relevant variables.

The factors closely related to a case are often easy to identify. However the social and external contexts are more difficult to capture in a few variables that can be used by process mining algorithms. Moreover, analysis (e.g., predictions) may suffer from the so-called "curse of dimensionality" (see Sect. 4.6.3). In high-dimensional feature spaces, enormous amounts of event data are required to reliably learn the effect of contextual factors.

10.6.3 Concept Drift in Processes

The term *concept drift* refers to the situation in which the process is changing while being analyzed [81]. For instance, in the beginning of the event log two activities

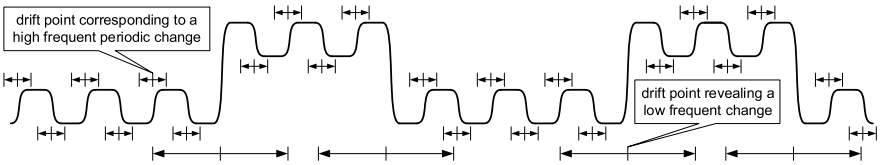


Fig. 10.15 A periodically changing process with two types of drift at different time scales. Shorter sliding windows are used to detect short-term drifts. The left half of the sliding window is compared with right half to spot statistically relevant changes. The larger sliding windows at the bottom are used for long-term drifts

may be concurrent whereas later in the log these activities become sequential. Processes may change due to periodic/seasonal changes (e.g., "in December there is more demand" or "on Friday afternoon there are fewer employees available") or due to changing conditions (e.g., "the market is getting more competitive"). Such changes impact processes and it is vital to detect and analyze them.

There are three challenges when dealing with concept drift [81]:

- *Change point detection*. Did the process change? If so, when did it change?
- *Change localization and characterization*. What has changed?
- *Change process discovery*. How to capture and predict "second-order" dynamics?

Once a point of change has been identified, the next step is to characterize the nature of change, and to identify the region(s) of change in a process. Concept drift is challenging because of the dynamic nature of processes. Processes in steady-state are not static ("first-order" dynamics), making the detection of irregular behavior ("second-order" dynamics) challenging. Most approaches use a sliding window approach [81, 177, 188]. Different windows with events are compared using statistical methods to detect significant changes. To precisely localize change points and to detect drift at different time scales, the lengths of such windows need to be varied. Different types of drifts may be intertwined as illustrated by Fig. 10.15.

10.7 Process Mining Spectrum

The refined process mining framework shown in Fig. 10.1 illustrates the broadness of the *process mining spectrum*. We identified 10 process mining activities ranging from discovery and conformance checking to the three operational support activities described in this chapter. These activities may be concerned with "de jure" or "de facto" models, "pre mortem" or "post mortem" event data, and one or more perspectives (control-flow perspective, organizational perspective, case/data perspective, etc.). In this chapter, we showed that process mining techniques originally intended for off-line analysis can be adapted for operational support. For example, replay techniques originally developed for conformance checking can be used to *detect* policy violations, *predict* remaining flow times, and *recommend* activities in an online setting.