# Foundations of Process Enhancement

Massimiliano de Leoni(✉)

Department of Mathematics, University of Padua, Padua, Italy
massimiliano.deleoni@unipd.it

**Abstract.** Process models are among the milestones for Business Process Management and Mining, and used to describe a business process or to prescribe how its instances should be carried out. It follows that they need to fulfill certain properties to be useful. If they aim to represent how the process is currently being executed, they need to be precise and recall the behavior observed in reality. If the goal is to ensure that the process is executed according to laws and regulations, its model should only allow the behavior that is valid from a domain viewpoint and provides some guarantee to ensure good performance level. Process enhancement is the type of Process Mining that aims at models that fulfill these properties, and the literature further splits it into two subfields: process extension and process improvement. *Process extension* aims to incorporate the process perspectives on data, decision, resources and time into the model: their inclusion in process models enable designers to fine-tune the model specifications, thus obtaining models with higher levels of precision. Process improvement passes through an "improved" process model. If the model contains portions of behavior that lead to unsatisfactory outcomes (high costs, low customer satisfactions, etc.) or that violate norms and regulations, one would like those portions to be disallowed by the model. In case some executions are observed in reality and are not allowed by the model, they should be incorporated into the model if they are observed to generally yield good performances. This chapter discusses these two types of process enhancement, and illustrates some basic and some advanced techniques to tackle it, highlighting the pros and cons, and the underlaying assumptions.

**Keywords:** Process improvement · Process extension · Decision discovery · Role discovery · Bottleneck analyses · Model repair

A process model is one of the main milestones for Business Process Management and Mining, and may be of two natures. A first nature of process models is descriptive: they are used by process analysts to engage process stakeholders (e.g., actors, managers, chief officers) into discussions on how the instances of the process have typically been executed, or how they should be. A second nature is prescriptive, and that is the case when the models are used as input for Process-aware Information System to automate processes and enforce how they must be carried out [10]. In both of scenarios, desirable models need to fulfill certain properties to be of fruitful use:

1. Models need to be precise and only allow legitimate behavior (high precision). This is especially relevant for models with a prescriptive nature: one wants to ensure that the information systems enforce how process instances must be executed, and also how they must not be.

2. Models should enable the executions that have been observed (high fitness), and are valid from a business viewpoint.
3. If certain executions are proven to lead to poor performance levels, they should be disallowed even if observed. Similarly, if certain executions have proven to reach good performance levels, they should be allowed.

This chapter introduces a number of techniques for process enhancement, which is the type of process mining that aims to create models that fulfill one or more of the properties mentioned above. Process enhancement starts with the provision of a process model for which these properties are relevant. This model can be mined from data, or designed by hand on the basis of process documentations, and/or stakeholder input. The literature proposes two types of process enhancement [25]: *process extension* and *process improvement*.

*Process extension* focuses on the first property (high precision) and aims to incorporate different perspectives. The model often only defines the control-flow perspective, which is certainly the process-model backbone, but it is insufficient to precisely encode the behavior that a model must explicitly allow or disallow. Processes manipulate, read and produce data (objects), their activities are performed by resources within due deadlines, and they take time to be carried out. In literature, these aspects are named perspectives: data, resource and time perspectives. Their inclusion in process models enable designers to fine-tune the model specifications, thus obtaining models with higher levels of precision.

*Process improvement* focuses on the other properties, and starts from the belief that, if a model has a prescriptive nature, process improvement passes through an "improved" process model. Improvement can be regarded as ensuring process models *(i)* to better reflect reality, and/or *(ii)* to only allow executions that are valid from a domain viewpoint and/or are correlated to better performances.

## 1 Process Extension: Basic Techniques

The extension of process models to incorporate multiple perspectives relies on the presence of attributes associated with the events. The definition of simplified event log introduced in [1] can be extended accordingly:

**Definition 1 (Simplified Multi-Perspective Event Log).** *Let $\mathcal{U}_{ev}$ be the universe of events. A simplified event log $L \subset 2^{\mathcal{U}_{ev}^*}$ is a set of traces, sequences of events, with the constraint that an event can only belong to one trace: $\forall \sigma', \sigma'' \in L.\ e \in \sigma', e \in \sigma'' \Rightarrow \sigma' = \sigma''$.*

Sections 1.2, 1.3 and 1.4 illustrates basic techniques to extend the models to incorporate the data, resource and time perspective, respectively.

### 1.1 Model-Aligned Event Logs

Several techniques for process enhancement requires that the event-log traces can be replayed on the process model (cf. [5]). This requires events to be univocally mapped

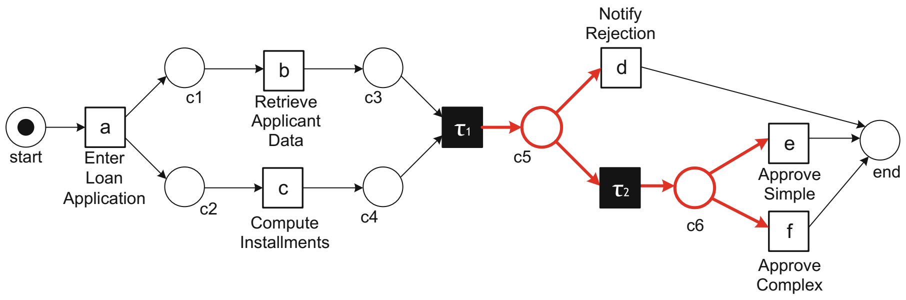


**Fig. 1.** The Petri net of the working example used in this chapter. The letters inside the transitions identify the transition names, while the script underneath indicates the transition label, namely the activity name. The thicker, red-coloured place and arcs identify a decision point, namely places with outgoing arcs to multiple transitions. (Color figure online)

onto process activities; in case of Petri-net models, events must be mapped onto transitions. However, multiple process activities (e.g., Petri-net transitions) can have the same label, and the choice of the activity to which to map each event is not necessarily local, but it depends on the entire sequence of activities that are executed. Furthermore, when processes are modelled via Petri nets, the model may include invisible transitions, namely transitions with no associated labels that, by definition, leave no trail in event logs. To further complicate the matter, log traces might not be compliant with the process model: certain activities might have been executed when not expected, or not executed when expected.

The situations above can be tackled by solving the following problem: given a log trace $\sigma_L = \langle e_1, \ldots, e_m \rangle$ and an accepting Petri net $AN = (N, M_{init}, M_{final})$ where $N = (P, T, F, l)$, we need to find the **model-aligned trace** $\sigma_P = \langle f_1, \ldots, f_n \rangle$ such that

1. the activity attribute of the events in $\sigma_P$ is defined over the domain of transition names, namely for each $1 \leq i \leq n$, $act(f_i) \in T$;
2. $\sigma_P$ is allowed by $AN$, namely $M_{init} \xrightarrow{\langle \#_{act}(f_1), \ldots, \#_{act}(f_m) \rangle} M_{final}$;
3. an event $f_i \in \sigma_P$ can be mapped to an event $e_j \in \sigma_L$ if the activity attribute of $f_i$ takes on a value equal to the transition of the activity of $e_j$, namely $l(\#_{act}(f_i)) = \#_{act}(e_j)$.
4. $\sigma_P$ is the best match to $\sigma_L$, namely that minimizes the number of events in $\sigma_L$ and in $\sigma_P$ that cannot be mapped.

The computation of a **closest model-aligned trace** can be achieved through alignments (cf. [5]), as explained through an following example: let us consider the log trace $\langle e_a, e_b, e_e \rangle$ where $e_a$, $e_b$, and $e_e$ are respectively the events for activities *Enter Loan Application*, *Retrieve Applicant Data*, and *Approve Simple* the subscript indicates the event activity (e.g. $\#_{act}(e_a) = a$). The model is depicted in Fig. 1, where transitions $\tau_1$ and $\tau_2$ are invisible, and the label of each transition is shown under the respective transition. The alignment between the model and the trace is as follows

$$\gamma = \begin{array}{|c|c|c|c|c|c|} e_a & e_b & \gg & \gg & \gg & e_e \\ \hline a & b & c & \tau_1 & \tau_2 & e \end{array}$$

The top row and bottom row respectively identify the log component of the alignments (namely the events), and the process/model component (the Petri-net transitions). To create a model-aligned trace, we need to synthesize the sequence of events. For each synchronous move between an event $e$ and a transition $t$, we create an event $e'$ such that $\#_{act}(e') = t$, and for any other event attribute $a$, $\#_a(e') = \#_a(e)$. For each model move for a transition $t$, we create an event $e_t$ such that only the activity attribute is populated $\#_{act}(e') = t$. These events are then ordered according to the order of the moves in the alignments. This means that, for the trace in question, the model-aligned trace is $\langle e'_a, e'_b, e'_c, e'_{\tau_1}, e'_{\tau_2}, e'_e \rangle$ where the subscript indicates the activity associated with the event, i.e. $\#_{act}(e'_x) = x$. Events $e'_a$, $e'_b$ and $e'_e$ are also populated with the additional attributes and values that are present for $e_a$, $e_b$, and $e_e$, respectively: for instance, for each attribute $v$ of $e_a$ different from $act$, $\#_v(e'_a) = \#_v(e_a)$.

Hereafter, the event log that originates from the information systems (i.e. with activity labels) is referred to as **event log**, while the event log defined over model transitions is named **model-aligned event log**. The events of model-aligned traces that stem from synchronous moves take on the attributes and their values from the mapped events of the real event log, including resource and timestamp. The events that come from model moves do not have any attribute but the activity. Note that, strictly speaking, a model-aligned trace is not a repaired trace as discussed in [5]: the activities of model-aligned traces are transitions names, where log traces refer to transition labels.

## 1.2 Data-Perspective Discovery

The data perspective focuses on how data objects are manipulated by the activities during the execution of process instances. The study of this perspective is of high relevance because the process-instance execution routing is affected by the characteristics of the specific process instance, such as the amount requested for a loan or the profile of the loan requestor, and also by the outcomes of previous steps in the process, such as the results of a verification activity. As an example, let us consider golden and silver profiles of potential loan requestors: a financial institute might decide to treat golden customers via a different procedure than that for other customers. Since the data perspective affects how decisions are made in the process, this perspective is also often referred to as **decision perspective**.

It is nowadays gaining momentum to represent this perspective in an integrated model, e.g., extending a BPMN model, or as a set of separate tables, also known as decision tables. This is also testified by the continous refinement of the Decision Model and Notation (DMN), a standard by the Object Management Group to describe and model decision tables [21].

Historically, the discovery of the data perspective is called *Decision Mining*, a name that was introduced by the seminal work by Rozinat et al. [23]. However, this work could not be applied on Petri nets containing invisible transitions or multiple transitions associated to the same label. This limitation has been lifted in [8] through the use of alignments and the construction of model-aligned tables.

**Table 1.** A fragment of a model-complaint event log for the model in Fig. 1. The gray events have been introduced as result of alignment model move for invisible transitions. Their case identifier is inherited from the other trace events.

| CaseId | Activity | Loan Amount | Loan Length | Age | Income | Verification | Instalment Amount | Instalment DivIncome | Resource |
|---|---|---|---|---|---|---|---|---|---|
| 1 | a | 400000 | 30 | 30 | 2500 | | | | Sue |
| 1 | b | | | | | TRUE | | | Max |
| 1 | c | | | | | | 1225 | 0.49 | John |
| 1 | $\tau_1$ | | | | | | | | |
| 1 | $\tau_2$ | | | | | | | | |
| 1 | f | | | | | | | | Jennifer |
| 2 | a | 450000 | 30 | 30 | 3000 | | | | Mark |
| 2 | c | | | | | | 1380 | 0.46 | Max |
| 2 | b | | | | | FALSE | | | John |
| 2 | $\tau_1$ | | | | | | | | |
| 2 | d | | | | | | | | Sue |
| 3 | a | 10000 | 30 | 71 | 2500 | | | | Mark |
| 3 | c | | | | | | 25 | 0.01 | Max |
| 3 | b | | | | | TRUE | | | John |
| 3 | $\tau_1$ | | | | | | | | |
| 3 | d | | | | | | | | Mark |
| 4 | a | 400000 | 25 | 30 | 2700 | | | | Mark |
| 4 | b | | | | | TRUE | | | Max |
| 4 | c | | | | | | 1458 | 0.54 | John |
| 4 | $\tau_1$ | | | | | | | | |
| 4 | d | | | | | | | | Sue |
| 5 | a | 30000 | 3 | 30 | 2500 | | | | Sue |
| 5 | c | | | | | | 925 | 0.37 | John |
| 5 | b | | | | | TRUE | | | Max |
| 5 | $\tau_1$ | | | | | | | | |
| 5 | $\tau_2$ | | | | | | | | |
| 5 | e | | | | | | | | Anne |

The simplest representation of the data perspective is to attach decision rules to decision points. When processes are modelled via Petri nets, decision points are places with arcs to multiple transitions (see, e.g., the red place with thick border and the outgoing arcs in Fig. 1). The rules explaining the choices are driven by additional process data, which are generated by activities/transitions preceding the split. In the remainder, this additional data is abstracted as a set of process attributes, where each attribute can take on a value within the respective attribute domain. For these Petri nets extended with data, a guard over these process attributes is attached to each transition, which can possibly be identically true. A transition is enabled if every incoming place has a token as for classical Petri nets, but also the associated guard needs to evaluatr true wrt. the current value assignment to process attributes. As for classical Petri net, a transition is enabled if every input place has a token; however, in this case, that is only a necessary

**Table 2.** The observation instances for the model in Fig. 1 and the event log in Table 1 to discover the guards at decision point for place $c5$. The last column indicates the class feature.

| Loan Amount | Loan Length | Age | Income | Verification | Instalment Amount | Instalment DivIncome | Transition |
|---|---|---|---|---|---|---|---|
| 400000 | 30 | 30 | 2500 | TRUE | 1225 | 0.49 | $\tau_2$ |
| 450000 | 30 | 30 | 3000 | FALSE | 1380 | 0.46 | $d$ |
| 10000 | 30 | 71 | 2500 | TRUE | 25 | 0.01 | $d$ |
| 400000 | 25 | 30 | 2700 | TRUE | 1458 | 0.54 | $d$ |
| 400000 | 25 | 30 | 2700 | TRUE | 925 | 0.37 | $\tau_2$ |

condition: the guard also needs to evaluate true. Other process-modelling notations have equivalent constructs to represent this: for instance, BPMN models use XOR-split gateways, depicted as a diamond with a X symbol inside, and conditions are represented on the arcs going out the gateway.

The basic algorithm for guard discovery assumes that the decisions are mutually exclusive: when a process instance reaches a decision point, one and exactly one branch is enabled for any assignment of values to attributes.

In [8, 23], the decision-mining problem is transformed into a classification problem: which transition is expected to fire (namely is enabled) for each valid assignment of values to process attributes. This problem can be tackled through decision-tree learning: decision trees have the remarkable advantage to explicitly indicate the classification criteria, namely which transition is enabled for each assignments of values to attributes.

The intuition can be given through an example related to a process modelled via the Petri net in Fig. 1. A corresponding model-aligned event log is represented conveniently in tabular form in Table 1. Loan lengths are measured in years, attribute *Income* is the monthly salary, *InstallmentAmount* is the amount of each monthly installment, and *InstalmentDivIncome* is the ratio between *InstallmentAmount* and *Income*. Last, *Verification* is a boolean process attribute to which a value is assigned as result of executing the activity *Retrieve Applicant Data* (Petri-net transition $b$): if the retrieval of applicant data confirms the information provided by applicants through the first activity, attribute *Verification* takes on a false value, and the loan request is going to be rejected. Let us focus on decision point $c5$, which is input place of transitions $d$ or $\tau_2$. It follows that $d$ and $\tau_2$ are mutually exclusive. We need to train a decision tree that define the conditions that discriminate when $d$ or $\tau_2$ is expected to occur, which are going to become the guards of $d$ and $\tau_2$. Table 2 shows the instances to be used to train the decision-tree model. The last column holds the class values of the learning instances, whereas the others columns refer to the independent variables. Since the model does not contain loops, exactly one token is produced in place $c5$, for each process instance. The first row refers to the case with identifier 1: in this case, the transition $\tau_2$ is observed when the following values were assigned to the process variable through the execution of given activities (i.e. model transitions): $LoanAmount = 400000$, $LoanLength = 30$, $Age = 30$, $Income = 2500$, $InstalmentAmount = 1225$, $Verification = TRUE$ and $InstalmentDivIncome = 0.49$. The second row refers

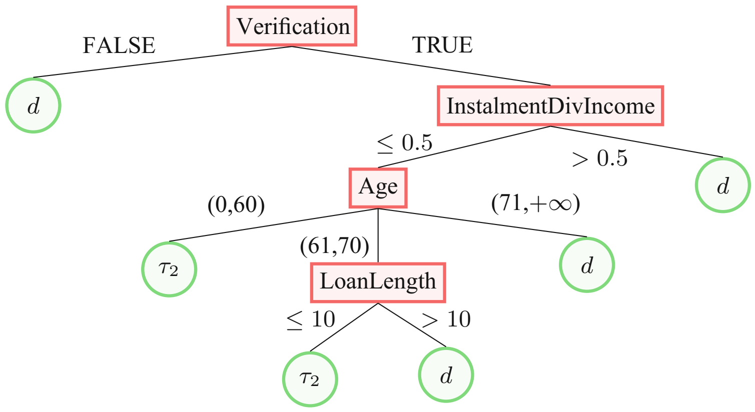


**Fig. 2.** A possible decision tree that is learned from the observation instances in Table 2.

to the second trace (id 2): in this case, $d$ was observed with $LoanAmount = 450000$, $LoanLength = 30$, $Age = 30$, $InstalmentAmount = 1380$, $Income = 3000$, $Verification = FALSE$ and $InstalmentDivIncome = 0.46$. In case values are assigned to process attributes by different transitions, the latest observed value is considered. Figure 2 shows a possible tree that can be learned from the observation instances in Table 2. The guards can be extracted from the decision tree, traversing the paths from the room to leaves. The guard for any transition $t$ is in the form of $expr_1 \vee \ldots \vee expr_n$ where $expr_i$ refers to the $i$-th path that leads to a leaf labeled as $t$, and is a conjunction of atoms $variable\ operator\ constant$ (e.g., $age \leq 60$ or $Verification = FALSE$) of the nodes and arcs part of the path.

As an example, the guard of transition $d$ (activity *Notify Rejection*)is an expression with four subexpression: $expr^d_1 \vee \ldots \vee expr^d_4$. Sub-expression $expr^d_1$ refers to the path for the left-most node, which includes the root node $Verification$ and the edge associated with label $FALSE$, lead to expression $Verification = FALSE$; $expr^d_2$ refers to the second left-most node with label $d$:

$$Verification = TRUE \wedge InstalmentDivIncome \leq 0.5 \wedge 61 \leq Age \leq 70 \wedge LoanLength > 10$$

and etc. Considering the four paths in the decision tree, the guard for $d$ is as follows:

$$\begin{array}{l} Verification = FALSE \\ \vee(Verification = TRUE \wedge InstalmentDivIncome \leq 0.5 \wedge 61 \leq Age \leq 70 \wedge LoanLength > 10) \\ \vee(Verification = TRUE \wedge InstalmentDivIncome \leq 0.5 \wedge Age \geq 70) \\ \vee(Verification = TRUE \wedge InstalmentDivIncome > 0.5) \end{array}$$

That is a disjunction of conjunction of terms related to paths from the root to the leaves labeled with $d$. One can similarly obtain the guard of $\tau_2$, which is the disjunction of two expressions:

$$\begin{array}{l} (Verification = TRUE \wedge InstalmentDivIncome \leq 0.5 \wedge Age \leq 60) \\ \vee(Verification = TRUE \wedge InstalmentDivIncome \leq 0.5 \wedge 61 \leq Age \leq 70 \wedge LoanLength \leq 10) \end{array}$$

**Missing Values**
Let us consider again the model-aligned event log in Fig. 1, and suppose that the event for transition $b$ in the trace with case identifier 1 comes from a model move. This means that, in the real event log before computing alignment, an event for $b$ in the first trace was not observed. In such a case, the fact that transition $b$ assigns a TRUE value to *Verification* is lacking. As a consequence, the first decision-tree training instance has a missing value for attribute *Verification*. Several techniques exist for the management of missing values of a given variable $f$, such as:

1. assigning the most common (if categorical) or the average value (if numerical) among those observed for $f$ in the other training instances,
2. similarly as above but restricting to the instances with the same class value (i.e. the same transition in our setting),
3. creating one training instance for each of the $n$ observed values - say value $v$ - and assigning value $v$ to the corresponding instance, with a weight that is equal to the number of instance with value $v$ for $f$ divided by the number of instance with some value for feature $f$.

Several implementations of decision-tree learning algorithms (e.g., of C4.5 [22]) are already equipped with the missing-value managements. However, it is important to think carefully about the meaning of missing values. It might be - as many schemes implicitly assume - that the value was produced but, for quality issues, was not recorded in the dataset. However, it could also mean that the transition did not produce that value, due to, e.g., some concept drift or impossibility to find a suitable value for the specific instance in question. When this is true, *the missing value conveys important information*, and the learning instance should carry the information that the value was missing via an additional boolean feature, instead of injecting random values. This additional feature can increase the discriminative power of the guards, differentiating the situations in which the information was provided from those in which the information was missing.

## 1.3 Organizational Mining

Organizational Mining focuses on the resources, which refer to anyone or anything involved in performing activities, such as a human process participant, a software system (e.g., a server) or an equipment (e.g., a production machine). The organization perspective, also referred to as *resource perspective*, aims to model how resources are grouped, ad how they interact to each other.

Among different goals, organizational mining aims at how resources collaborated to carry on individual process instances. Typically, the resource collaborations can be represented as *social networks*, which are graphs where nodes are the resources and arcs, direct or indirect, indicate some form of collaboration between pairs of resources [25,26]. Arcs can be also given weights, which is proportional to the frequency/intensity of the collaborations. One of the most studied social networks in Business Process Management relates to the hand-over of work between resources. A work hand-over between two resources $a$ and $b$ exists in a process instance $p$, if $a$ has executed

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| Anne | | | | | 1 | |
| Jennifer | | | | | | 1 |
| John | | 0.4 | 0.6 | | | |
| Mark | 0.6 | | | 0.2 | | |
| Max | | 0.6 | 0.4 | | | |
| Sue | 0.4 | | | 0.4 | | |

(a)

| Role | Resources | Activities |
|---|---|---|
| R1 | Anne, Jennifer | e, f |
| R2 | John, Max | b, c |
| R3 | Mark, Sue | a, d |

(b)

**Fig. 3.** An example of application of role discovery for the process referring to the log in Fig. 1. Table (a) is the resource-activity matrix, where colors are used to define a reasonable grouping of rows, i.e. resources. When no value is depicted for a cell, it should be intended as zero. Table (b) details the discovered roles. Note that the role name cannot be automatically derived.

an activity for $p$, which is directly followed by a second activity that is executed by $b$. This implies that $a$ has handed over the progression of the execution of $p$ to $b$, which, in turn, can later hand it over to another resource. Among the different goals, organizational mining aims to discover these social networks, and later to analyze them. Social network analysis is very interesting in Organizational Mining because it can unveil relevant information about resources. Notably, it can discover *cliques* of resources that tend to work together, or critical resources that are less "replaceable". Less replaceable resources are characterized by a large degree of incoming and outgoing arcs, and the removal of the corresponding nodes in the graph may create longer paths between pairs of resources, or even yield disconnected components.

Space consideration prevents us from further discussing social-network analysis, and forces us to rather focus on analyzing the event logs to discover *roles*, groups of resources that work on the same activities. Clustering techniques are simple techniques to discover roles within organizations, especially under the assumption that a resource plays one single role. The starting point is to build a resource-activity matrix, such as that in Fig. 3(a). Rows refer to different resources and columns to different activities. The value for the row $r$ and column $a$ indicates the average number of times that $r$ executes $a$ in a process execution. For instance, Mark executes activity $a$ 0.6 times per case, on average. Note that, if an activity is executed exactly once per process instance, the sum of the values of the cells of the corresponding column is one. A sum lower or higher than one indicates an activity to be optional or be involved in a loop.

In a resource-activity matrix, each row is a different resource and can be regarded as a vector with as many dimensions as the number of process activities: the value of the dimension for a given activity is equal to that of the corresponding cell in the matrix. Rows are thus vectors, points of a cartesian space, that can be clustered, e.g., via well-known clustering algorithms, such as K-Means or DBScan [19]). The row colors in Fig. 3(a) illustrates a reasonable clustering for the matrix in question. As an example, Mark and Sue belong to the same cluster and, hence, play the same role: their role allows them to perform activities *a* and *d*. The same is for John and Max, who can perform *b* and *c*. Anne and Jennifer form the third role that enables them to perform *e* and *f*. Note that it would be equally reasonable to split the cluster with Anne and Jennifer into two, although a simpler solution with fewer role is possibly preferable when equivalent.

## 1.4 Time Perspective

A process-instance execution can takes a considerable amount of time to be carried out. Depending on the domain, it might even take months or years to conclude: consider, e.g., a health-care process to follow up cancer diagnoses, or that to give monthly unemployment benefits, or even a process to reintegrate workers who have suffered physical issues that prevent them from going back to their original employment. It follows that process activities are not instantaneous as we have so far considered, but they take some time to be executed. In fact, certain activities require external inputs (e.g., the production of documents, the arrival of materials and other goods), and the availability of necessary machines and suitable human resources. If these requirements are not met at the moment when the activities are ready to be started, their execution is forcibly delayed. These delays can have a cascading effect on other activities that follow in the process.

Within the realm of Process Mining, the time perspective focuses on the timing of events that carry timestamp information. The time-perspective analysis can notably be used to discover process bottlenecks, and monitor the service levels: their analysis enables verifying whether executions are carried on within a reasonable amount of time (e.g., a complaint is addressed within the same day in which it is filed), or whether the temporal process constraint are fulfilled (e.g., the second shot of the COVID-19's Pzifer vaccine is given within 21 days from the first). The analysis of the perspectives on time and resource is also partly overlapping: thanks to the time information, process analysts can assess, for instance, whether resources are fairly, overly, or scantily utilized.

The verification of the satisfaction of time-related constraints is related to conformance checking (see [5]) As an example, Mannhardt and Blinde illustrates an interesting case study to check the conformance of the treatment of patients who suffered from Sepsis [18]. The conformance checking of time perspective is not covered in this chapter, which conversely focuses on extending and annotating a process model to unveil potential time-related issues, especially process' bottlenecks.

The presence or absence of bottlenecks can be related to *(i)* waiting time, namely the difference between the timestamp of the actual start of an activity instance and the earliest moment in which the instance could have started (cf. above discussion of delays caused by lack of resources), or *(ii)* service time, i.e. the duration of an activity-instance execution.

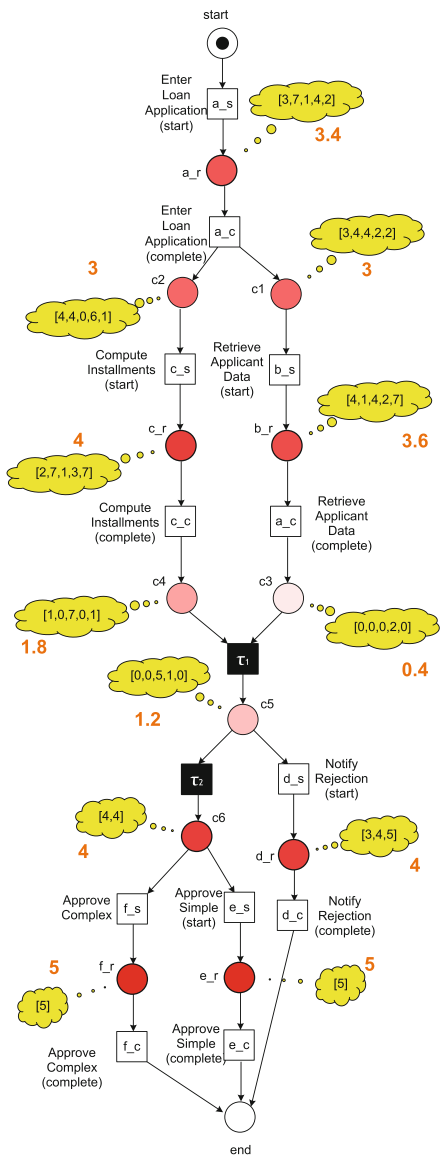
Several ways exists to analyse the service and waiting times of the activities of a process model, e.g. modelled via Petri nets. The performances at the different points of the model can be analyzed through *queue mining* [24]. For instance, queue mining can be employed to estimate how long a token typically remains unconsumed in a Petri-net place. This estimation is far from being easy because it requires to consider several factors: the average length of the token queues in places, the policy of consumptions of tokens (FIFO or according to some priorities), the relationships between places (e.g., connected to the same transition), etc. Queue mining considers the process model as a queuing network, whose characteristics are determined after analysing the information stored in event logs. A queuing network is used to determine the activity execution policies. When a queue network is created, several off-the-shelf techniques can be employed for its analysis.

Space limitation forces this chapter to only focus on a simple technique based on the Petri-net token-replay game: real-log traces are transformed in model-aligned traces that are replayed on the Petri-net model to collect waiting and service times. The transformation to model-aligned traces ensures that they are replayable on the model. However, the firings of Petri-net transitions are atomic by definition, and hence their execution take no time. This is clearly not realistic, and requires to explicitly model the starting and completion of activity instances are two separate Petri-net transitions. This explicit modelling can be simply explained through our working example of the process modelled as in Fig. 1.

Each visible transition is split into the sequence of two transitions that model the starting and completion of activity instances, yielding the Petri net in Fig. 4(a). For instance, activity *Enter Loan Application* is now represented through two transitions, named *a_s* and *a_c*, which respectively fire when instances of that activity starts or completes. We aim to play the token game: this means that transition *a_s* fires upon a start event for activity *Enter Loan Application*, and *a_c* upon a complete event for the same activity. This means that, when a token is present in the places named $a\_r, \ldots, f\_r$, it indicates that the activity associated with transitions $a\_s, \ldots, f\_s$ are being executed, respectively. Note that there is no need to split invisible transitions: they are necessary for modelling purposes, and do not represent an actual activity, and hence can be considered as instantaneous. As mentioned earlier, the real event log need to be translated into a model-aligned event log that can be directly replayed on the Petri net of the process model, and alignment techniques are used for this purpose. In the scenario in which events refer to either the starting or the completion of the activities, the two transitions that indicates the starting or completion of any activity *x* need both to be mapped to events for *x*, but the first to events related to the starting of *x* and the second to events related to the completion of *x*. For the model in Fig. 4(a), transition *a_s* is mapped to events related to the starting of *Enter Loan Application*, and *a_c* to events related to the completion of *Enter Loan Application*.

After computing the alignments with this mapping, it is possible to synthesize the model-aligned event log in Fig. 4(b). Gray rows refer to the firing of invisible transitions: in that case, the timestamp of the associated events is assigned to be equal to the earliest moment in which the transition could fired. Consider transition $\tau_1$ and the first trace: $\tau_1$ can fire when both transitions $b\_c$ and $c\_c$ have fired, $b\_c$ fires at time 11 and $c\_c$ at time 10 for the first trace, and consequently the earliest moment in which $\tau_1$ can fire is at time 11.

Each trace of the model-aligned event log can be replayed on the Petri net. This allows computing the amount of time in which a token resides in a given place, i.e. the difference between the timestamp in which the token was consumed and the timestamp when it was produced. For example, consider the place $a\_r$: tokens are produced in that place when transition $a\_s$ fires and are consumed when transition $a\_c$ fires. For trace



(a)

| Case Identifier | Activity | Timestamp |
|---|---|---|
| 1 | a_s | 1 |
| 1 | a_c | 4 |
| 1 | b_s | 7 |
| 1 | b_c | 11 |
| 1 | c_s | 8 |
| 1 | c_c | 10 |
| 1 | $\tau_1$ | 11 |
| 1 | $\tau_2$ | 11 |
| 1 | f_s | 15 |
| 1 | f_c | 20 |
| 2 | a_s | 2 |
| 2 | a_c | 9 |
| 2 | c_s | 13 |
| 2 | c_c | 20 |
| 2 | b_s | 13 |
| 2 | b_c | 14 |
| 2 | $\tau_1$ | 20 |
| 2 | d_s | 20 |
| 2 | d_c | 23 |
| 3 | a_s | 2 |
| 3 | a_c | 3 |
| 3 | c_s | 3 |
| 3 | c_c | 4 |
| 3 | b_s | 7 |
| 3 | b_c | 11 |
| 3 | $\tau_1$ | 11 |
| 3 | d_s | 16 |
| 3 | d_c | 20 |
| 4 | a_s | 20 |
| 4 | a_c | 24 |
| 4 | b_s | 26 |
| 4 | b_c | 28 |
| 4 | c_s | 30 |
| 4 | c_c | 33 |
| 4 | $\tau_1$ | 33 |
| 4 | d_s | 34 |
| 4 | d_c | 39 |
| 5 | a_s | 18 |
| 5 | a_c | 20 |
| 5 | c_s | 21 |
| 5 | c_c | 28 |
| 5 | b_s | 22 |
| 5 | b_c | 29 |
| 5 | $\tau_1$ | 29 |
| 5 | $\tau_2$ | 29 |
| 5 | e_s | 33 |
| 5 | e_c | 38 |

(b)

**Fig. 4.** An example of extending the process model with the time perspective. The left-hand side picture shows how the process model in Fig. 1 can be annotated with temporal information wrt. the model-aligned log shown in the right-hand side table. The gray lines in the table are the events related to invisible transitions.

with case identifier 1, $a\_c$ and $a\_s$ respectively fired at time 4 and 1, thus the difference is 3. The residence of each token in each place can be computed by replaying the model-aligned event logs: these timestamp differences are shown within the clouds associated to the different places in Fig. 4(a). The average per place can subsequently be computed, which is shown next to the respective cloud. One can red color each place with a color intensity that is proportional to the mean value of time: white is associated to an average of zero, and the color becomes closer and closer to dark red as the average is closer and closer to the largest observed value.

Considering place $a\_r$ again, the average time is 3.4 for the event log in Fig. 4(b). This indicates that the average duration of instances of activity *Enter Loan Application* is 3.4 time units (e.g., hours). Consider place $c1$: tokens are produced in the place after the completion of the same activity and consumed when transition $b\_s$ fires, namely when activity *Retrieve Applicant Data* starts. For the first trace, the amount of time a token is $c1$ is 3 time units, namely the timestamp of the event for $b\_s$, which is seven for first trace, minus the timestamp of the event for $a\_c$, i.e. 4. After collecting the times for each token in $c1$ for all traces (see the cloud connected to the place) and computing the average, one can conclude that the average time between the starting of activity instances of *Retrieve Application Data* and the completion of the corresponding instance of the preceding activity *Enter Loan Application*.

**Dealing with Non-compliant Traces and Missing Timestamps**
So far, we have assumed that *(i)* activity executions leave trails in log through both start and completion events, and *(ii)* every trace is compliant with the model. In particular, assumption *(ii)* means that the model-aligned traces only include additional events related to firing of invisible transitions. These assumptions do not always hold in reality: event logs often only contain the events related to the completions of activity instances, and some traces are not fully compliant with the model (cf. the Conformance Checking field discussed in [5]).

*Assumption* (i) *is not met.* In this case, one can employ a naïve approach that assumes that the next activity in the process starts as soon as the previous completes: in this case, the timestamp of the starting event is the same as the timestamp of the completion event of the activity that precedes. This is often unrealistic, as pictorially depicted in Fig. 5. In the timeline, *Completion of a* indicates the moment in which activity instance *a* completes and *Real start of a* is the actual moment in which *a* started, which has left no trail in the event log. Moment *Completion of the activity instance preceding a* is when the previous activity concluded. The time difference between *Completion of the activity instance preceding a* and *Real Start of a* corresponds to the waiting time of *a*. If this time difference is set to 0, no waiting time is assumed. A better estimation can be obtained if the event log contains information about the resource perspective: one can look at the completion event of a given activity instance $a$ and consider the resource $r$ that performed the instance: the starting timestamp of the activity instance is equal to the earliest moment after the completion of the activity instance that precedes $a$ in which $r$ has completed any activity instance and has become available [20].

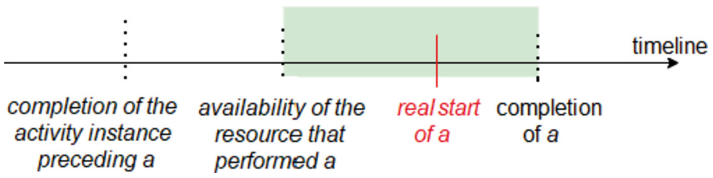

timeline
completion of the activity instance preceding a
availability of the resource that performed a
real start of a
completion of a

**Fig. 5.** Representation of the scenario when the timestamp of the start event of an activity instance $a$ is not present in the event log and needs to be estimated. This timestamp is located between the timestamp when $a$ completes and the earliest timestamp when the resource $r$ that is going to perform $a$ is available to start $a$. This time interval is represented through a green area, and the real start of $a$, which is unknown, is located within the area. In case the resource information is missing, we do not even have the earliest timestamp of availability of $r$: this introduces further uncertainty, because we can only rely on the timestamp of completion of the activity instance that precedes $a$ in the trace.

This corresponds to the moment in figure labelled as *Availability of the resource that performed a*: this introduces some waiting time, namely the time difference between *Completion of the activity instance preceding a* and *Availability of the resource that performed a*, thus being more realistic. The latter case is still often unrealistic in practice [11]: *(a)* resources work on multiple processes and continuously switch from one to the other while event logs refer to one process, *(b)* take breaks during the working days (e.g., when tired), *(c)* carry on additional duties that lead no trail in the event logs (e.g., when answering the phone). Let us consider Fig. 5 again: the actual start is in a moment between when the resource has become available and when the activity instance has been completed. The choice of estimating different start moments leads to estimating different activity-instance durations. In [14], Fracca et al. proposes a technique to estimate the starting event where different activity-duration configurations are simulated, and the resulting simulated event log is compared with the real event logs to assess the similarity with respect to time-related aspects (activity-instance waiting times and process-instance durations): the more similar are the real and simulated event log is, the more realistic are the estimation of activity instance durations. The simulation of different activity-duration configurations requires a simulation model, which consists of a process model that is extended with additional information related to the simulation aspects, such as the inter-arrival time, the routing probabilities at the XOR gateways, the roles and the resource-activity allocation, potential work calendar, and more. The simulation model can be constructed by combining different process mining techniques, as also discussed in [14].

*Assumption* (ii) *is not met.* This can be clearly caused by not meeting the assumption *(i)*: the starting events are missing, yielding model moves for every Petri-net transition linked to the starting of activity instances. We consider the situation hereafter in which assumption *(i)* is met. In this case, the deviations are related to the activities that have not been performed in accordance to the process model. In this case, both the starting and completion events are missing. If the number of non-compliant traces is limited, these can be excluded from the analysis. Otherwise, the log traces are aligned to create model-aligned traces, without adding the timestamps to the events that come from model moves for visible transitions: in this case, statistics are computed for reliability by only considering pairs of subsequent events that have a timestamp associated.

## 2 Process Extension: Advanced Techniques

This section introduces some advanced techniques to overtake the limitations of the basic algorithms for decision mining and for role discovery: In particular, the basic algorithm for decision mining introduced in Sect. 1.2 is only able to discover with atoms of form *var-op-const* where *var* is a variable, *op* is a comparison operator and *const* is a constant (e.g. $Age \leq 60$ or $Verification = FALSE$), while the basic algorithm for role discovery in Sect. 1.3 assumes a resource to be able to play one single role, only. Sections 2.1 and 2.2 discussed some advanced techniques that aim to overcome these limitations.

### 2.1 Data-Perspective Discovery of Guards with Variable Comparison

Let us consider a variant of the event log in Table 1 where *InstalmentAmount* is present but attribute *InstalmentDivIncome* is missing. As mentioned above, the basic guard-discovery algorithm will be unable to discover guards that include an atom ${}^{InstalmentAmount}/_{Income} > 0.5$, or its negation. The work by de Leoni et al. [7] reports on an extension to the basic algorithm that can discover atoms of form *var-op-var*, such as $InstalmentAmount > 0.5 \cdot Income$.

The algorithm builds on some oracle that discovers invariants in a set of observation instances, such as the Daikon system [6]. Analogously to the basic algorithm, the algorithm is applied for each place $p$ of the Petri Net modelling a process, and consists of five steps:

1. The basic algorithm in Sect. 1.2 observation instances $\Psi$ for $p$ (e.g., those in Table 2).
2. For each transition $t_i$ with outgoing arcs to $p$, we extract the observation instances $\Psi_{t_i} \subset \Psi$ with class $t_i$ (e.g., those in Table 2 with transition\class $\tau_2$). In fact, $\Psi_{t_1}, \ldots, \Psi_{t_n}$ is a clustering of $\Psi$ where $t_1, \ldots, t_n$ are the transitions with outgoing arcs to $p$.
3. A set $V_{t_i}$ of invariants that hold in $\Psi_{t_i}$ is computed (for instance, ${}^{InstalmentAmount}/_{Income} > 0.5$).
4. For each invariant $v \in V_{t_1} \cup \ldots \cup V_{t_n}$, one boolean feature $f_v$ is added to every observation instance $\psi \in \Psi$, and it takes on a true or false value depending whether invariant $v$ holds with the variable-to-value assignment defined in $\psi$. For instance, invariant ${}^{InstalmentAmount}/_{Income} > 0.5$ holds in the fourth instance in Table 1, and does not for the others (recall that attribute *InstalmentDivIncome* is assumed to not be present).

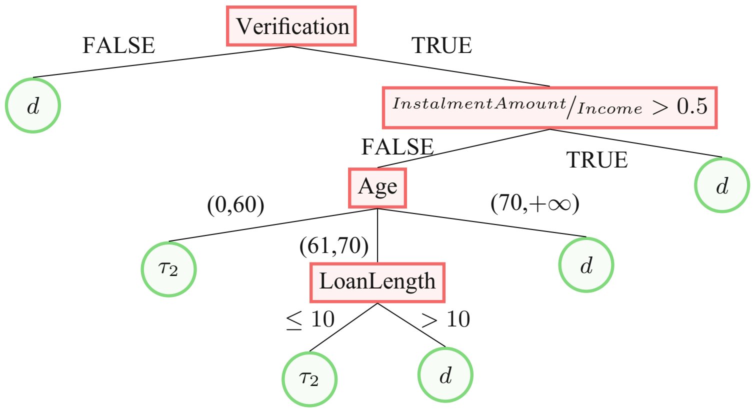


**Fig. 6.** The possible decision tree that is learned from the observation instances in Table 2 augmented with boolean features related to discovered invariants, such as $^{InstalmentAmount}/_{Income} > 0.5$.

5. A decision tree is trained using the set of augmented observation instances.

For the working example, such a decision tree as in Fig. 6 is learnt: the invariant is now able to discriminate between the instances of $d$ and of $\tau_2$.

## 2.2 Discovery Roles with Overlapping Resources

The basic organization-mining technique discussed in Sect. 1.3 relies on clustering, and thus assumes each resource to play exactly one role. In many settings, this assumption does not hold: resources can associated with multiple roles. Burattin et al. [4] lift this assumption, by clustering activities instead of resources: the clustering puts together the activities that require to be executed by resources playing the same role.[1] The starting point is a process model and its dependencies of form $a \rightarrow b$, i.e. activity $b$ can follow $a$ but $a$ cannot follow $b$. Clustering is obtained by removing all the dependencies $a \rightarrow b$ for which the handover is larger than a given threshold $\tau^w$:[2]

**Definition 2 (Resource Handover for a Model Dependency).** *Let $a \rightarrow b$ be the dependency between two activities $a$ and $b$. Let $L$ be an event log and $\mathcal{R}_{a\rightarrow b} = \biguplus_{\sigma \in L} \biguplus_{\langle e_i, e_j \rangle \in \sigma . \#_{act}(e_i)=a \wedge \#_{act}(e_j)=b} (\#_{res}(e_i), \#_{res}(e_j))$ be the multiset of pairs of resources in $L$ where the first resource executes $a$ and is immediately followed by the second resource executing $b$. Let $\mathcal{R}^a_{a\rightarrow b}$ and $\mathcal{R}^b_{a\rightarrow b}$ be the projection over the first and*

[1] The terminology and formalization used hereafter slightly different those in [4], to harmonize with the rest of the chapter.

[2] Given two multisets $X$ and $Y$, the interection $X \cap Y$ returns a multiset that contains every element $z$ present in $X$ <u>and</u> $Y$ with the lowest cardinality for $z$ between that of $X$ and of $Y$. Symbol $\uplus$ indicates the union of multisets: the cardinality of each element in the union of two multisets $X$ and $Y$ is equal to the sum of the cardinalities of the element in $X$ and in $Y$. Given a sequence $\sigma$, a second sequence $\sigma' \in \sigma$ if $\sigma'$ is a sub-sequence of $\sigma$.

*second component of $\mathcal{R}^a_{a\to b}$, respectively. Let $\mathcal{R}^=_{a\to b}$ be the pairs with the same resource value on both components. The resource handover for dependency $a \to b$ for $L$ is defined as follows:*

$$w_{ab}(L) = 1 - \frac{|\mathcal{R}^a_{a\to b} \cap \mathcal{R}^b_{a\to b}| + |\mathcal{R}^=_{a\to b}|}{|\mathcal{R}^a_{a\to b}| + |\mathcal{R}^b_{a\to b}|}$$

The definition states that $w_{ab}(L)$ is closer and closer to zero if it is more and more frequent that two activities $a$ and $b$ are performed by the same resources. If activities belong to the same cluster, the resources that perform them can play the same role.

As an example, let us consider the dependency $a \to c$ for the model in Fig. 1 and the log in Table 1. It follows $\mathcal{R}_{a\to c} = [(Mark, Max)^2, (Sue, Anne)^1]$ where the superscript indicates the cardinality; hence, $\mathcal{R}^a_{a\to c} = [Mark^2, Sue^1]$ and $\mathcal{R}^c_{a\to c} = [Max^2, Anne^1]$. Therefore, the resource handover for the dependency is $w_{ac}(L) = 1$. Value 1 is obtained when the set of resources are totally disjoint, as the case is for $a$ and $c$: the dependency is hence removed, making $a$ and $c$ belong to different clusters. Repeating the reasoning on dependency $a \to$, we obtain $w_{ab} = 1$, thus causing $a$ and $b$ to belong to different clusters.

Ultimately, this means that activity *a* is a cluster with only itself. However, Fig. 3(b) shows that activities $a$ and $d$ should belong to the same cluster, so as to add the performing resources to the same role. However, this cannot happen if we only look at the dependencies because there is no dependency $a \to d$, or vice versa. Therefore, after partitioning the activities, some clusters can be merged. This occurs if the so-called merging degree is larger than a given threshold $\tau^\rho$:

**Definition 3 (Merging Degree).** *Let $A_1 = \{a_{1,1}, \ldots, a_{1,n}\}$ and $A_2 = \{a_{2,1}, \ldots, a_{2,n}\}$ be two activity clusters. Let $L$ be an event log. For any set $A$ of activities, let us denote the multiset of resource executing activities in $A$ with $\mathcal{R}_A = \biguplus_{\sigma\in L} \biguplus_{e\in\sigma:\#_{act}(e)\in A} \#_{res}(e_i)$. The merging degree of $A_1$ and $A_2$ is defined as:*

$$\rho_{A_1,A_2}(L) = 2\frac{|\mathcal{R}_{A_1} \cap \mathcal{R}_{A_2}|}{|\mathcal{R}_{A_1}| + |\mathcal{R}_{A_2}|}$$

Similarly to Definition 2, this measures the amount of shared resources between those that execute two sets $A_1$ and $A_2$ of activities. If $\rho_{A_1,A_2}(L) > \tau^\rho$, $A_1$ and $A_2$ are merged.

In conclusion, the algorithm to discover roles where resources belong to multiple is as follows:

1. Select the resource-handover threshold $\tau^w$ and the merging threshold $\tau^\rho$.
2. For each model dependency $x \to y$, compute the handover $w_{xy}$.
3. Remove every dependency $x \to y$, such as $w_{xy} < \tau^w$.
4. Cluster the activities according to the retained dependencies.
5. Merge two activity clusters $A_1$ and $A_2$, if the merging degree $\rho_{A_1,A_2} > \tau^\rho$. Note this step can be applied recursively to merged clusters until no further merging.
6. For each final cluster $\overline{A}$, a role is created that contains every resource that has performed an instance of any activity in $\overline{A}$. A further threshold can be defined to discard resources that seldom perform activities in $\overline{A}$.

It is worthwhile reflecting that the actual relevant values for the resource-handover threshold $\tau^w$ are limited to the set of handover values $w_{xy}$ computed for each dependency $x \rightarrow y$. Given that the number of dependencies is finite and usually small, it is possible to extensively apply the role discovery setting $\tau^w$ iteratively to every value $w_{xy}$ where $x \rightarrow y$. This enables process analysts to evaluate the different configuration and determine the most realistic role set, using business knowledge. Also, once a value is set for $\tau^w$ and the clusters are created, one can similarly reason for $\tau^\rho$: the number of values to test is finite, i.e. the values $\rho_{A',A''}(L)$ for each pairs $(A', A'')$ of clusters at step 4.

## 3 Process Improvement

Process analysts and certain stakeholders (e.g., CEOs) may oftentimes have partial or helicopter-like view on the organizations in which such process are executed. As a consequence, the process models that they have in mind (also known as "to-be" models) may not summarize how processes are *really* executed by resources. In these cases, such "to-be" models are of limited use. Improvement can be regarded as altering the model so that it reflects reality (i.e., improvement on fitness) while ensuring the other quality criteria remain within a certain reasonable range (i.e., precision, generalization and simplicity). The result is an "as-is" model that show how the process is *really* executed. Section 3.1 details how models can be improved on fitness.

However, if models are used to prescribe how processes ought to be executed, they should only represent the behavior with which the organization is satisfied. If the model contains portions of behavior that lead to unsatisfactory outcomes (high costs, low customer satisfactions, etc.) or that violate norms and regulations, one would like those portions to be disallowed by the model. Section 3.2 details how models can be improved to ensure no regulation violations and to incorporate behavior that has proven to yield good performance levels.

The classical problem of process discovery discussed in [2,3] and that of process improvement share some commonalities in that they both aim to come up with "as-is" models. The difference lays on the fact that the problem of process discovery is largely unsupervised (little or no knowledge is fed in), while process improvement is supervised: an original model is provided, which constitutes the initial "backbone" that is later altered to obtain a "as-is" model. It follows naturally that process improvement is generally able to produce better models because the original model encodes behavior that is deemed appropriate from a business viewpoint. This reasoning is especially valid when the original model is hand-designed by or in concert with process owners.

The remainder of this section will use the same working example that was used in Sects. 1 and 2, namely the process modelled in Fig. 1. However, hereafter we will differently assume that the activity names in the real event log coincides with the transition names $a, \ldots, f$, to keep the discussion simple. Also, since we discuss techniques for process improvement that only consider the control flow, traces will be considered as sequences of activities, which coincide with transition names for the considerations above (i.e., a simple formulation)

## 3.1 Model Repair to Reflect Reality

The problem of repairing a process model $M$ to reflect the reality recorded in a log $L$ can be formulated as finding a process model $M'$ that is able to replay each trace in $L$ and is the closest possible to $M$ (i.e. with the minimum number of changes). Note that, if $M$ can replay $L$, $M' = M$. This section focuses on the case in which $M$ and $M'$ are accepting Petri nets, and the goal is to repair models wrt. the control-flow perspective, thereby ignoring the other perspectives. This formulation suggests that model repair primarily aims at perfect fitness, generating a set of models with optimal fitness. Within this set, the final choice refers to any model that best balances simplicity, precision and generalization (cf. the conformance-checking problem discussed in [5]).

The assumption here is that the repaired model must be able to replay every trace in event log $L$. However, event logs may record executions that are outliers or refer to process instances that were still running at the moment of the extraction of the event log. Those traces should not be allowed by the repaired model $M'$. Hereafter, we however assume that every trace that should not be replayed by $M'$ is already filtered out from $L$ before applying the model-repair algorithm on $L$.

This section reports on the repair technique discussed in [13], whose basic intuition can be given via the following example. Let us consider again the model in Fig. 1 and the following event log $\overline{L} = [\langle a, g, w, b, c, d\rangle, \langle a, w, g, b, c, e\rangle]$ where $g$ and $w$ are the shortcut names of two new activities: e.g. *fix application* and *add witnesses* respectively. These two activities are not part of the model, and thus cannot be replayed on the model. It follows that the model needs to be executed to add some transitions labelled $g$ and $w$ to make the model compliant with $\overline{L}$. The model-repair algorithm needs to determine the point in which the two transitions should be included, namely which places are in the presets and postsets of these transitions. The technique discussed in [13] aims to address this question by aligning the original model $M$ and each of the traces in $\overline{L}$. Optimal alignments for the traces in $\overline{L}$ wrt. the model in Fig. 1 are:

$$\gamma_1 = \begin{array}{|c|c|c|c|c|c|c|}
a & g & w & b & c & \gg & d \\ \hline
a & \gg & \gg & b & c & \tau_1 & d \\ \hline
[c1, c2] & & & [c3, c2] & [c3, c4] & [c5] & [end]
\end{array}$$

$$\gamma_2 = \begin{array}{|c|c|c|c|c|c|c|c|}
a & w & g & b & c & \gg & \gg & e \\ \hline
a & \gg & \gg & b & c & \tau_1 & \tau_2 & e \\ \hline
[c1, c2] & & & [c3, c2] & [c3, c4] & [c5] & [c6] & [end]
\end{array}$$

Here, the third alignment rows indicate the marking of the Petri net after each synchronous or model move. As usual, the model moves for $\tau_1$ and $\tau_2$ are not considered deviations, and hence do not need to be taken into account when repairing the model. In both of alignments, the actual deviations consist in a sequence of two log moves for activities $g$ and $w$, namely related to log sub-traces $\langle g, w\rangle$ and $\langle w, g\rangle$. These sequences of two log moves (and the corresponding sub-traces) both occurred when the Petri-net model was at marking $[c1, c2]$. The model needs to be repaired so that the log-move sequences would be replaced in the respective alignments by sequences of synchronous moves.

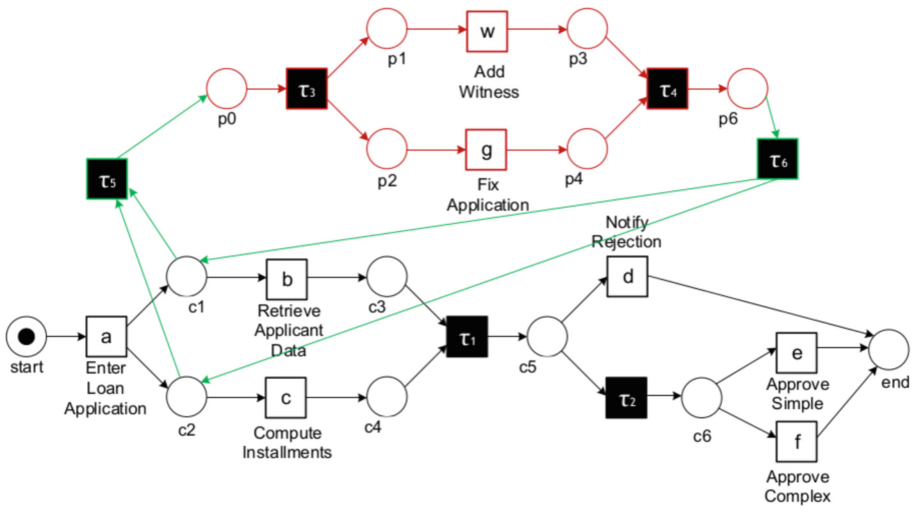


**Fig. 7.** The model in Fig. 1 repaired by adding the parts in red and green to allow for executions $\langle a, g, w, b, c, d\rangle$ and $\langle a, w, g, b, c, e\rangle$.

The two sub-traces $\langle g, w\rangle$ and $\langle w, g\rangle$ can in fact be regarded as an event log, which can be given as input to some process-discovery techniques to mine a model. If we employed the Inductive Miner, the model would be similar to the Petri net marked through a red border in Fig. 7: transitions *w* and *g* are modelled in a parallelism. Markings $[p0]$ and $[p6]$ are respectively the initial and final marking. Since $\langle g, w\rangle$ and $\langle w, g\rangle$ were observed at marking $[c1, c2]$, marking $[p0]$ needs to be reachable from $[c1, c2]$ without firing any transition: this is modelled via the invisible transition $\tau_5$. Furthermore, when reaching marking $p6$, the execution should be able to reach back marking $[c1, c2]$, motivating the introduction of invisible transition $\tau_6$.

The example above helps introduce the algorithm to repair an accepting Petri net $AN$ with respect to a log $L$:

1. The traces in $L$ are aligned, thus discovering sequences of log moves.
2. For each maximal sequence $\overline{\gamma}$ of log moves (e.g., the sequence of log moves for $g$ and $w$ in the above example), we determine the marking $m_{\overline{\gamma}}$ of $AN$ when $\overline{\gamma}$ was observed (e.g., $[c1, c2]$ in the above example), and we synthesize the sequence $\sigma_{\overline{\gamma}}$ of activities involved in $\overline{\gamma}$ (e.g., $\langle g, w, \rangle$). The pair $(m_{\overline{\gamma}}, \sigma_{\overline{\gamma}})$ is added to a multiset $P$ of pairs that consist of markings, and sequences of events observed in the log but not allowed by $AN$ at those markings.
3. The pairs $P$ discovered at previous point are grouped by marking so as to obtain a set $P_L$ of pairs where each pair consists of *(i)* a markings $m$ and a log $L = \biguplus_{((m',\sigma)\in P\ :\ m'=m)} \sigma$ that contains the sequences of events observed at $m$.
4. For each $(m, L_m) \in P_L$, an accepting Petri net $\overline{AN}$ is discovered using $L_m$ as input event log. Petri net $\overline{AN}$ is merged with $AN$ at marking $m$ as discussed in the example above (cf. the green part in Fig. 7).

The algorithm above only considers the log moves, which are linked to sequences of activities that need to be allowed by the repaired model. Of course, the alignment may also point out sequences of model moves, namely sequences of activities that were expected in an observed process instance but not observed. The repaired model should make these expected, but unobserved sequences as optional. As an example, let us consider the model in Fig. 7, which has already been repaired wrt. the sequences of log moves in the alignments. Let us suppose the event log to contain traces related to applications that are desk-rejected because of their clear incorrectness: these correspond to traces consisting of two events $\langle a, d \rangle$. The corresponding alignment would then be as follows:

$$\gamma_1 = \begin{array}{|c|c|c|c|c|} a & \gg & \gg & \gg & d \\ \hline a & b & c & \tau_1 & d \\ \hline [c1, c2] & [c3, c2] & [c3, c4] & [c5] & [end] \end{array}$$

It contains a sequence of three model moves. The repaired model should be such that those three model moves are no more necessary. This can be easily tackled, and one can insert an invisible transition that consumes one token in $c1$ and one in $c2$, i.e. the places containing tokens before the first model move (i.e., before $b$), and produces a token in $c5$, the place containing one token after the last model move (i.e. after $\tau_1$).

**Advanced Repair for Higher Precision and Simplicity**

The repairing algorithm discussed above is largely focusing on fitness, thereby overlooking the other dimensions. In fact, the procedure above can have a negative influence on precision and simplicity, because it may allow additional behavior, and increase the size of the model.

**Higher model precision** can be obtained by removing transitions that seldom appear. In a nutshell, the event-log traces are aligned with the model. For each transition $t$ in the model, we count the number of occurrence of synchronous or model move for $t$ in all the alignments. If this is smaller than a user-defined threshold, $t$ is removed, along with every arc that goes in or comes out from $t$. The procedure can cause some places to have no more incoming or outgoing arcs: these places are removed, as well.

**Model simplification** can be achieved as an a-posteriori step, e.g., using the technique proposed by Fahland et al. [12], which aims to simply the model, while preserving the same behavior and well balancing generalization and precision [12]. However, simplification can partly be achieved during the repair, e.g. in case of structured loops [13]. Let us consider the model in Fig. 1 and an event log consisting of two traces $\sigma_1 = \langle a, b, c, r, b, c, d \rangle$ and $\sigma_2 = \langle a, b, c, r, c, b, e \rangle$ where $r$ is the shortcut for a new activity *Ask for Additional Documents* to, e.g., enable a more thorough assessment. The alignments of the two traces are as follows:

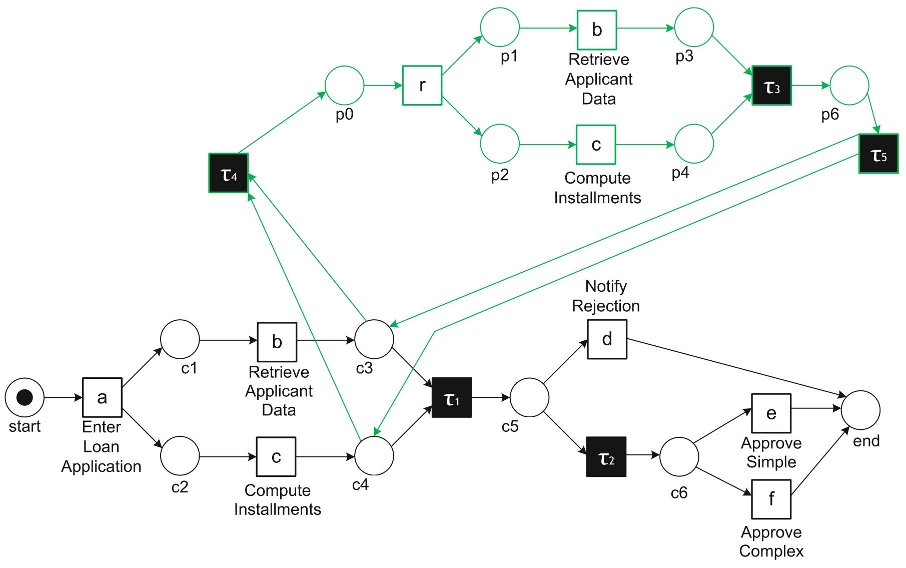


**Fig. 8.** Repair of the model in Fig. 1 to allow for $\langle a, b, c, r, b, c, d\rangle$ and $\langle a, b, c, r, c, b, e\rangle$, using the basic model-repair technique.

$\gamma_1 =$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $a$ | $b$ | $c$ | $r$ | $c$ | $b$ | $\gg$ | $d$ |
| $a$ | $b$ | $c$ | $\gg$ | $\gg$ | $\gg$ | $\tau_1$ | $d$ |
| $[c1, c2]$ | $[c3, c2]$ | $[c3, c4]$ | | | | $[c5]$ | $[end]$ |

$\gamma_2 =$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $a$ | $b$ | $c$ | $r$ | $c$ | $b$ | $\gg$ | $\gg$ | $e$ |
| $a$ | $b$ | $c$ | $\gg$ | $\gg$ | $\gg$ | $\tau_1$ | $\tau_2$ | $e$ |
| $[c1, c2]$ | $[c3, c2]$ | $[c3, c4]$ | | | | $[c5]$ | $[c6]$ | $[end]$ |

Using the repair technique discussed so far, we would obtain the model in Fig. 8, where the newly included part is shown in green. The model has multiple transitions of the same label (see $b$ and $c$), which would be actually unnecessary if the technique could discover that the green part aims to model a structured loops of repeating $b$ and $c$.

The basic repair algorithm can be extended to implement such structured loops as in the example above. We give an intuition on how the algorithm is extended via the above example: let us take $\sigma_1 = \langle a, b, c, r, b, c, d\rangle$), with the alignment $\gamma_1$ shown at page 22. The marking before the first log move is $[c3, c4]$, and the sequence of events that are associated with the maximal sequence of log moves is $\sigma_{\gamma_1} = \langle r, c, b\rangle$.

We search in the model to be repaired, namely the model in Fig. 1, for the smallest connected subnet that *(i)* ends with places $c3$ and $c4$, namely the places with a token at the marking before the first log move, and *(ii)* contains each transition $t$ in $\sigma_{\gamma_1} = \langle r, c, b\rangle$, excluding $r$, which is not in the model to be repaired. This subnet corresponds to the gray area in Fig. 9.The trace fragment $\sigma_{\gamma_1}$ is then projected on this subnet: the events related to transition of the fragment are the only retained, yielding a subtrace $\overline{\sigma_1} = \langle c, b\rangle$. We create an accepting Petri net $\overline{AN}$ from the fragment, using the marking

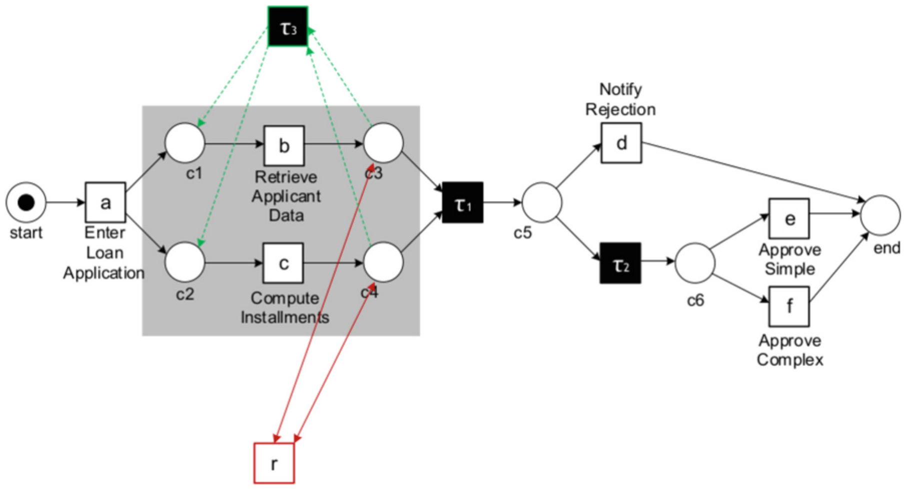


**Fig. 9.** Repair of the model in Fig. 1 to allow for $\langle a, b, c, r, b, c, d\rangle$ and $\langle a, b, c, r, c, b, e\rangle$, using the advanced model-repair algorithm that increases simplicity (cf. result of the basic algorithm in Fig. 8).

$[c3, c4]$ before the first log move as final marking, and the marking with one token in each place with no incoming arcs as initial marking. Since $\overline{\sigma_1}$ is replayable on $\overline{AN}$, the transition $\tau_3$ can be introduced, which constructs a structured loop where transitions $b$ and $c$ can be repeated. Note that the algorithm above is applied on single log sequences of individual traces, and transition $r$ in Fig. 9 has not been introduced yet. The algorithm needs to be iteratively applied to each sequence of events that come from the projection of the log component of each alignment.

In our example, after repairing the model by adding transition $\tau_3$, the algorithm is applied on $\sigma_2 = \langle a, b, c, r, c, b, e\rangle$, but yields no changes. Indeed, after the first repair, the alignment of the model in Fig. 9 with $\sigma_2$ is now as follows:

$$\gamma_2 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} a & b & c & r & \gg & c & b & \gg & \gg & e \\ \hline a & b & c & \gg & \tau_3 & c & b & \tau_1 & \tau_2 & e \\ [c1, c2] & [c3, c2] & [c3, c4] & & [c1, c2] & [c3, c2] & [c3, c4] & [c5] & [c6] & [end] \end{array}$$

Recall that transition $r$ is not yet part of the model. This will be added as final step, which consists in reapplying the basic repair algorithm on the same traces $\sigma_1 = \langle a, b, c, r, b, c, d\rangle$ and $\sigma_2 = \langle a, b, c, r, c, b, e\rangle$ and on the model in which invisible transition $\tau_3$ is included.

This section has focused on repairing the model to reflect the reality observed in the event log. However, repairing the model can also be regarded as to ensure that the model is sound. In the domain of process model, model soundness implies several properties of which the most important is the absence of deadlocks or livelocks that prevent executions from being completed. Interesting approaches that focus on model repair for soundness are provided by Gambini et al. [15] and by Lohmann et al. [16, 17], which are not discussed here due to space limitations.

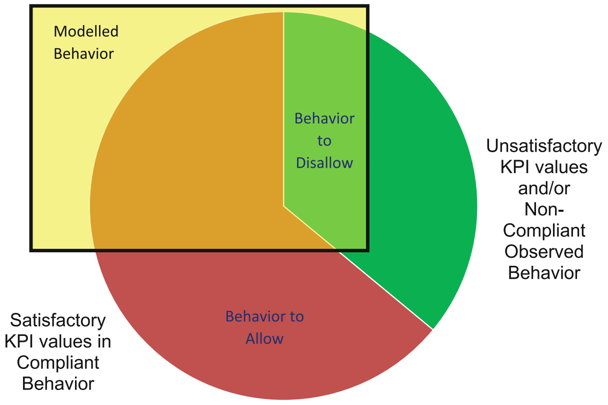

Modelled Behavior
Behavior to Disallow
Unsatisfactory KPI values and/or Non-Compliant Observed Behavior
Satisfactory KPI values in Compliant Behavior
Behavior to Allow

**Fig. 10.** The basic idea of KPI-driven Model Improvement: the observed behavior (i.e., in the event log) that is satisfactory and compliant with rules should be incorporated in the model, while the observed behavior that is not satisfactory or not compliant, should be not incorporated or disallowed in the model.

## 3.2 KPI-Driven Model Improvement

If the model is used to prescribe how the corresponding process should be carried on, one does not want to incorporate the whole behavior observed in the event log, but only that portion that has shown to usually lead to satisfactory values of a certain Key Performance Indicator (KPI) of interest. Furthermore, behavior can only be incorporated if it does not violate the protocols, regulations, and norms. The definition KPI of interest varies depending on the domain, needs to be customized, and may be numerical or defined over an enumeration of values, including boolean. Examples are execution costs, customer satisfaction, execution time, or whether or not the corresponding loan was eventually approved. Similarly, one wants to disallow the behavior allowed by the model that, unfortunately, typically yield unsatisfactory KPI values. Figure 10 graphically illustrates the idea. The rectangle shows the amount of behavior allowed by the model, while the pie shows the amount observed in the event log. The green pie slide is the portion of observed behavior that is associated with unsatisfactory KPI values or with violations of norms or protocols: the part in light green that intersects the modelled behavior should be disallowed from the model after repair. The red and the orange pie portions show the portion with satisfactory KPI values: the part in dark red is not allowed by the model but that should be incorporated because of being associated with executions characterized by satisfactory KPI values.

The remainder of this section focuses on a methodology to extend the model to allow the portion in dark red, which has been introduced by Dees et al. [9]. The starting point is an existing process model, here represented as an accepting Petri net, an event log, and the definition of a Key Performance Indicator (KPI). A KPI is a pair consisting of *(i)* a function that, given a trace, returns the KPI value, and *(ii)* the set of satisfactory KPI values:

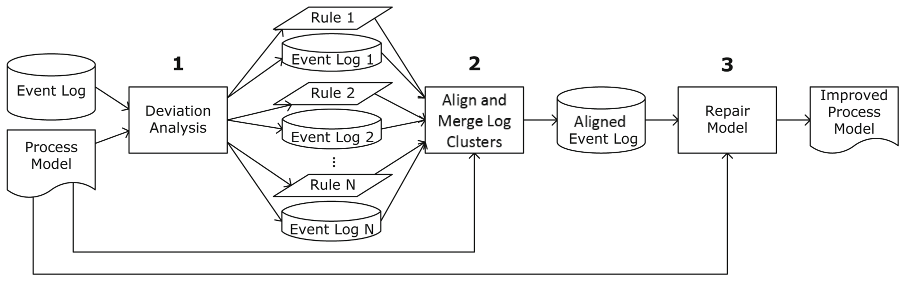


**Fig. 11.** The main steps of the methodology for KPI-driven Model Improvement (adapter from [9])

**Definition 4 (Key Performance Indicator).** *Let $L$ be a simplified multi-perspective event log. Let $\mathcal{V}$ be the set of possible values for a key performance indicator. A* key performance indicator *is a pair $(\kappa, K)$ consisting of a function $\kappa : L \rightarrow \mathcal{V}$ that assigns a KPI value $\kappa(\sigma)$ to each trace $\sigma$ and of a set $K \subset \mathcal{V}$ that contains the KPI values that are satisfactory from a business viewpoint.*

Typically, the function $\kappa$ in a KPI definition depends on the attributes present in the event log. However, this section remains general on how the KPI values of process executions (i.e., traces) are computed.

**Partially Model-Aligned Traces**
The technique described hereafter also relies on the concept of model-aligned event logs that has been introduced in Sect. 1.1. However, we extend the concept to allow for traces that are partially model-aligned. It is indeed possible to ignore individual moves: ignoring a model move means that the corresponding event is not added to the trace, and ignoring a log move means that the corresponding event is not removed. To clarify, let us consider a trace $\langle a, b, b, d\rangle$ and the model in Fig. 1. The alignment is as follows:

$$\gamma = \begin{array}{|c|c|c|c|c|c|} a & b & b & \gg & \gg & d \\ \hline a & b & \gg & c & \tau_1 & d \end{array}$$

A full model-alignment trace is $\langle a, b, c, \tau_1, d\rangle$. Ignoring log moves for $b$ would generate $\langle a, b, b, c, \tau_1, d\rangle$, namely the new alignment would still generate the log move for $b$; ignoring model moves for $c$ would generate $\langle a, b, \tau_1, d\rangle$, i.e. the model move for $c$ is still present. It is possible to ignore multiple moves at the same time: in our example, repairing neither the model move for $c$ nor the log move for $b$ would produce $\langle a, b, b, \tau_1, d\rangle$. Note that, hereafter, we always to ignore all model moves for invisible transitions when model-aligning a trace, and we consider to fully model-align a trace even when we ignore model moves for invisible transitions.

**The Methodology in a Nutshell**
The methodology takes an event log and the original process model as input and returns an improved process model. It is composed by three main steps (cf. Fig. 11):

*Step 1. Deviation Analysis.* Deviations are detected and a set of rules is discovered that correlate deviations to a selected KPI. Rules are mutually exclusive, which enables to split the event-log traces into groups of traces, such that a trace belongs to at most one cluster (in fact, outlier traces are filtered out).

*Step 2. Align and Merge Log Clusters.* Traces in the different sublogs are partially model-aligned to only keep the deviations in the original trace that have a positive impact on the value of the KPI. All sublogs are then merged to obtain a single partially aligned-model event log.

*Step 3. Repair Model.* Finally the partially aligned-model event log is used as input to repair the model: the process model is modified in such a way that it can replay all the behavior of the partially aligned-model event log. In the partially aligned-model event log we have repaired all deviations corresponding to behavior that should not be incorporated in the model. In this way the repair-model technique will only modify the model to make the desired deviating behavior possible.

The remainder will elaborate on the sub-sets within steps 1 and 2, using the same case study as in Fig. 1. Step 3 does not require further details since it consists in applying any technique for model repair to reflect reality, such as the technique by Fahland et al. [13] discussed in Sect. 3.1.

**Step 1. Deviation Analysis**

The deviation-analysis step takes an event log $L$, an accepting Petri net $AN$, and a KPI definition $(\kappa, K)$. The result is a decision tree that allows splitting $L$ in so many sub-logs as the tree leaves. Each sub-log is associated with a different KPI value. Note that certain traces are considered outliers and filtered out, namely the union of the sub-logs does not necessarily coincide with $L$. To achieve this, the following sub-steps can be identified:

*Step 1.1: Conformance Checking.* The first step is checking conformance of the event log and the process model. This is done to determine all deviations that are observed between the log and the model. The result of conformance checking is an alignment for each log trace.

*Example:* *let us consider the model in Fig. 1 and three non-compliant traces:* $\sigma_1 = \langle a, b, c, w_1, f \rangle$, $\sigma_2 = \langle a, b, c, w_2, f \rangle$ *and* $\sigma_3 = \langle a, b, c, w_3, f \rangle$ *where* $w1$, $w2$ *and* $w3$ *is a shortcut for the activities to ask for one, two or three witnesses, respectively. The alignments are of the following form where* $w_X$ *respectively stands for* $w_1$, $w_2$ *and* $w_3$*:*

$$\gamma_2 = \begin{array}{|c|c|c|c|c|c|c|} a & b & c & w_X & \gg & \gg & f \\ \hline a & b & c & \gg & \tau_1 & \tau_2 & f \end{array}$$

The KPI is here boolean: $\mathtt{true}$ and **false** respectively indicate whether the approval process has finally led to a loan that is eventually repaid in full or only in part. The latter case is undesired because it requires the involvement of a credit-collection agency. For the three executions in the example, $\sigma_1$ refers to a loan paid back in full whereas $\sigma_2$ and $\sigma_3$ to loans paid back in part.

*Step 1.2: Moves' Correlation to KPI Values.* The number of model moves and log moves of activities is correlated with the chosen KPI. To model that the improved model should comply rules and regulations, the concepts of *disallowed activities* and *mandatory activities* has been introduced. The set $G_D$ of disallowed activities include those that should never become part of the process model, whereas the set $G_M$ of mandatory activities are those that cannot become optional or be removed from the model. In this step, we build a set of so-called *observation instances*, which are used to train a classification tree. Let $T$ and $l$ be the set of transitions and the labelling function of the labelled Petri net of $AN$. Let $A$ be the activities of $N$, i.e. the Petri-net labels: $A = \cup_{t \in T} l(t)$ . To keep it simple, we assume without losing generality that the log activities coincide with $A$, too. We build one observation instance for each trace $\sigma \in L$ with the following features:

– The number of model moves in the optimal alignment of $\sigma$ for each allowed activity $a \in A \setminus G_D$.
– The number of log moves in the optimal alignment of $\sigma$ for each non-mandatory activity $a \in A \setminus G_M$.
– The KPI value for $\sigma$, namely $\kappa(\sigma)$.

From the set of observation instance, we learn a decision tree, using the KPI value as target feature, and the number of log and model moves as independent features. If the domain of the KPI values is finite (e.g., satisfactory vs unsatisfactory), a classification tree is used; otherwise, we employ a regression tree.

*Example (cont.): log moves for $w_2$ and $w_3$ are correlated with full repay, where log moves for $w_1$ are correlated with part repay. However, let us assume $w_3$ be within the set of disallowed activities (e.g., three witnesses require too much additional work). Thus, log moves for $w_3$ are not allowed as independent feature. The result could be such a decision tree as in Fig. 12: when there are log moves for $w_2$, the KPI is fulfilled: the loan is eventually repaid in full.*

*Step 1.3: Splitting of the Event Log into Groups and Outlier Filtering.* The classification tree can be seen as a clustering of the traces of an event log. Each leaf is a different cluster and the path from the root to the leaf provides a rule that characterizes the traces that belong to a certain group. For reliability, the wrongly-classified traces are removed from the groups, namely the traces classified to have KPI values that differ from the actual values. The wrongly-classified traces might potentially affect the repair-model phase, and allow behavior in the model that would not be linked to actual, satisfactory KPI values.

*Example (cont.): The trace cluster associated with leaf* Part Repay *(left leaf) is $L_1 = [\langle a, b, c, w_1, f \rangle]$, whereas the cluster for leaf* Full Repay *is $L_2 = [\langle a, b, c, w_2, f \rangle]$. Note that trace $\sigma_3 = \langle a, b, c, w_1, f \rangle$ would also be in $L_2$, but would be wrongly classified and consequently filtered out. In fact, $\sigma_3$ is associated with a loan that is repaid in full but the decision tree in Fig. 12 would classify it as partly repaid: it does not indeed contain log moves for $w_2$.*

**Step 2. Model-Align and Merge Log Clusters**
Step 1 concluded with splitting L in $n$ sublogs and filtering out those traces that are wrongly classified. Let $\{L_1, \ldots, L_n\}$ be the sublogs obtained via splitting. Each $L_i$ refers to a different decision-tree leaf $v_i$, associated with a KPI value $\mathcal{C}(v_i)$.

*Step 2.1: Conformance Checking of the Sublogs.* Conformance Checking is done with the original process model and each log cluster. Note that Step 2.1 is a conceptual step: in practice, one does not need to recompute the alignments for the cluster logs as one can simply reuse the alignments obtained as result of Step 1.1.

*Step 2.2: Model-Align of the Sublogs.* This step is repeated for each cluster $L_i$, associated with a leaf $v_i$. If $L_i$ is associated with an unsatisfactory KPI value (i.e. $\mathcal{C}(v_i) \notin K$), every deviation is repaired. Note that, even if the traces are fully model-aligned, they are kept in the log that is used for repairing the model at step 3. Those traces provide support to not remove behavior that is not observed: see discussion on achieving higher model precision in subsection *Advanced Repair for Higher Precision and Simplicity* within Sect. 3.1.

If $L_i$ is associated with a satisfactory KPI value, every deviation is repaired, except those in the conditions in the path from the decision-tree root to the leaf $v_i$.

*Example (cont.): Trace $\sigma_1$ is model-aligned in full because related to an unsatisfactory KPI value, yielding a partial model-aligned trace $\sigma_1^{\tau} = \langle a, b, c, f \rangle$. Trace $\sigma_2$ is related to satisfactory KPI values (see leaf Full Repay in the decision tree in Fig. 12), and associated to a tree path that indicates that the number of log moves for $w_2$ is larger*

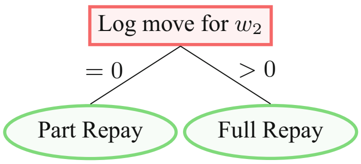


**Fig. 12.** A decision tree that correlates alignment moves to KPI values.

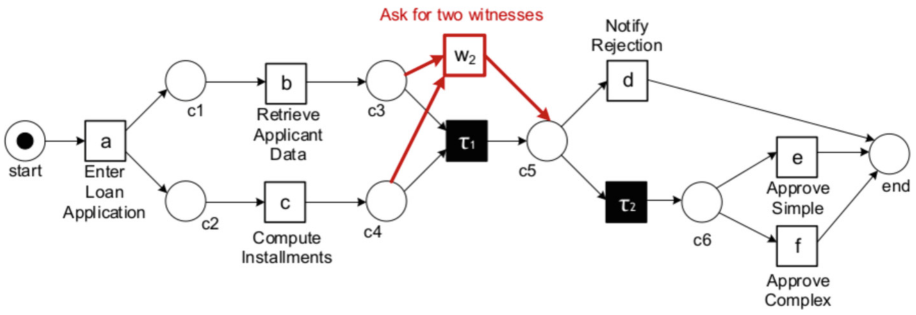


**Fig. 13.** The model repaired to increase the changes for loan to be repaid in full (the KPI). The change consists in introducing the activity *Ask for two witnesses*, which are shown to be beneficial for a better risk assessment.

*than zero. This means that the log move for $w_2$ is ignored when model-aligning $\sigma_2$: thus, the partial model-aligned trace $\sigma_2^\tau$ coincides with the original trace $\sigma_2$.*

*Step 2.3: Merge the Sublogs.* We merge all model-aligned sublogs into a single event log. This is a requirement to apply the next step, namely repairing the process model.

*Example (cont.): This step generates the event log* $\overline{L} = [\langle a, b, c, f\rangle, \langle a, b, c, w_2, f\rangle]$, *which is used for model repair.*

When the log is used with a model-repair technique (e.g., that in Sect. 3.1), the model in Fig. 2 is repaired as shown in Fig. 13: the transition $w_2$ is introduced.

**Acknowledgement.** Some of the ideas and techniques reported in this chapter is the result of author's collaborations with various researchers. While it is not possible to name them all, the author would like to give a special mention to Wil van der Aalst, Marcus Dees, Marlon Dumas, Felix Mannhardt, and Hajo Reijers (in strict alphabetical order).

## References

1. van der Aalst, W.M.P.: Process mining: a 360 degrees overview. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
2. van der Aalst, W.M.P.: Foundations of process discovery. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
3. Augusto, A., Carmona, J., Verbeek, E.: Advanced process discovery techniques. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
4. Burattin, A., Sperduti, A., Veluscek, M.: Business models enhancement through discovery of roles. In: 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), pp. 103–110 (2013)
5. Carmona, J., van Dongen, B., Weidlich, M.: Conformance checking: foundations, milestones and challenges. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. xx–yy. Springer, Cham (2022)
6. Ernst, M.D., et al.: The daikon system for dynamic detection of likely invariants. Sci. Comput. Program. **69**(1), 35–45 (2007). Special issue on Experimental Software and Toolkits
7. de Leoni, M., Dumas, M., García-Bañuelos, L.: Discovering branching conditions from business process execution logs. In: Cortellessa, V., Varró, D. (eds.) FASE 2013. LNCS, vol. 7793, pp. 114–129. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37057-1_9
8. de Leoni, M., van der Aalst, W.M.P.: Data-aware process mining: discovering decisions in processes using alignments. In: SAC 2013, pp. 1454–1461. ACM (2013)
9. Dees, M., de Leoni, M., Mannhardt, F.: Enhancing process models to improve business performance: a methodology and case studies. In: Panetto, H., et al. (eds.) OTM 2017. LNCS, vol. 10573, pp. 232–251. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69462-7_15
10. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer (2018). https://doi.org/10.1007/978-3-662-56509-4

11. Estrada-Torres, B., Camargo, M., Dumas, M., García-Bañuelos, L., Mahdy, I., Yerokhin, M.: Discovering business process simulation models in the presence of multitasking and availability constraints. Data Knowl. Eng. **134**, 101897 (2021)
12. Fahland, D., van der Aalst, W.M.P.: Simplifying discovered process models in a controlled manner. Inf. Syst. **38**(4), 585–605 (2013)
13. Fahland, D., van der Aalst, W.M.P.: Model repair—aligning process models to reality. Inf. Syst. **47**, 220–243 (2015)
14. Fracca, C., de Leoni, M., Asnicar, F., Turco, A.: Estimating activity start timestamps in the presence of waiting times via process simulation. In: Proceedings of the 34th International Conference on Advanced Information Systems Engineering (CAiSE 2022), LNCS. Springer (2022)
15. Gambini, M., La Rosa, M., Migliorini, S., Ter Hofstede, A.H.M.: Automated error correction of business process models. In: Proceedings of the 9th International Conference on Business Process Management, BPM 2011, pp. 148–165, Springer, Heidelberg (2011)
16. Lohmann, N.: Correcting deadlocking service choreographies using a simulation-based graph edit distance. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 132–147. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85758-7_12
17. Lohmann, N., Fahland, D.: Where did i go wrong? In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 283–300. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_18
18. Mannhardt, F., Blinde, D.: Analyzing the trajectories of patients with sepsis using process mining. In: RADAR+EMISA 2017, volume 1859 of CEUR Workshop Proceedings, pp. 72–80. CEUR-WS.org (2017)
19. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
20. Nakatumba, J.: Resource-aware business process management: Analysis and Support. PhD thesis, Technische Universiteit Eindhoven (2013)
21. Object Management Group (OMG): Decision model and notation (DMN) v1.1 (2016)
22. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
23. Rozinat, A., van der Aalst, W.M.P.: Decision mining in ProM. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 420–425. Springer, Heidelberg (2006). https://doi.org/10.1007/11841760_33
24. Senderovich, A.: Queue mining. In: Sakr, S., Zomaya, A.Y. (eds.) Encyclopedia of Big Data Technologies. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-77525-8
25. van der Aalst, W.M.P.: Process Mining - Data Science in Action, 2nd Ed. Springer (2016). https://doi.org/10.1007/978-3-662-49851-4
26. van der Aalst, W.M.P., Reijers, H.A., Song, M.: Discovering social networks from event logs. Comput. Supp. Coop. Wor. **14**(6), 549–593 (2005)