Supporting Risk-Informed Decisions during Business Process Execution

Raffaele Conforti[1], Massimiliano de Leoni[2], Marcello La Rosa[1], and Wil M. P. van der Aalst[2]

[1] Queensland University of Technology, Australia
[2] Eindhoven University of Technology, The Netherlands
{raffaele.conforti,m.larosa}@qut.edu.au,
{m.d.leoni,w.m.p.v.d.aalst}@tue.nl

Abstract. This paper proposes a technique that supports process participants in making risk-informed decisions, with the aim to reduce the process risks. Risk reduction involves decreasing the likelihood and severity of a process fault from occurring. Given a process exposed to risks, e.g. a financial process exposed to a risk of reputation loss, we enact this process and whenever a process participant needs to provide input to the process, e.g. by selecting the next task to execute or by filling out a form, we prompt the participant with the expected risk that a given fault will occur given the particular input. These risks are predicted by traversing decision trees generated from the logs of past process executions and considering process data, involved resources, task durations and contextual information like task frequencies. The approach has been implemented in the YAWL system and its effectiveness evaluated. The results show that the process instances executed in the tests complete with substantially fewer faults and with lower fault severities, when taking into account the recommendations provided by our technique.

1 Introduction

A *process-related risk* measures the likelihood and the severity that a negative outcome, also called *fault*, will impact on the process objectives [15]. Failing to address process-related risks can result in substantial financial and reputational consequences, potentially threatening an organization's existence. Take for example the case of Société Générale, which went bankrupt after a €4.9B loss due to a fraud.

Legislative initiatives like Basel II [3] and the Sarbanes-Oxley Act[3] reflect the need to better manage business process risks. In line with these initiatives, organizations have started to incorporate process risks as a distinct view in their operational management, with the aim to effectively *control* such risks. However, to date there is little guidance as to how this can be concretely achieved.

As part of an end-to-end approach for risk-aware Business Process Management (BPM) [5, 6], we proposed a technique to model risks in executable business process models, detect them as early as possible during process execution, and support process administrators in mitigating these risks by applying changes to the running process instances. However, the limitation of these efforts is that risks are not *prevented*, but rather *acted upon when their likelihood exceeds a tolerance threshold*. For example, a

[3] www.gpo.gov/fdsys/pkg/PLAW-107publ204

mitigation action may entail skipping some tasks when the process instance is going to exceed the defined maximum cycle time. While effective, mitigation comes at the cost of modifying the process instance, often by skipping tasks or rolling back previously-executed tasks, which may not always be acceptable. Moreover, we have shown that it is not always possible to mitigate all process risks. For example, rolling back a task may not allow the full recovery of the costs incurred in the execution of the task, for the sake of mitigating a risk of cost overrun.

In light of this, in this paper we present a technique that supports process participants in making risk-informed decisions, with the aim to *reduce* process risks preemptively. A process participant makes a decision whenever they have to choose the next task to execute out of those assigned to them at a given process state, or via the data they enter in a user form. This input from the participant may influence the risk of a process fault to occur. For each such input, the technique returns a risk prediction in terms of the likelihood and severity that a fault will occur if the process instance is carried out using that input. This prediction is obtained via a *function estimator* which is trained using historical process data such as process variables, resources, task durations and frequencies as extracted from the process log. This way the participant can make a risk-informed decision as to which task to execute next, or can learn the predicted risk of submitting a form with particular data. If the instance is subjected to multiple faults, the predictor can return the weighted sum of all fault likelihoods and severities, as well as the individual figures for each fault. The weight of each fault can be determined based on the severity of the fault's impact on the process objectives.

We implemented the function estimator via decision trees and embedded this into a custom service for the YAWL workflow management system. Our service interacts with the worklist handler of the YAWL system to prompt the process participant with risk predictions upon filling out a form or when choosing the next task to execute. We then evaluated the effectiveness of our technique by conducting experiments on a simulated process log of 2,000 traces and using different fault distributions. The results show that the technique was always able to substantially reduce the number and severity of faults upon instance completion.

The remainder of this paper is organized as follows. Section 2 introduces our approach for managing process-related risks and describes a running example. Section 3 defines the notions of event logs and faults which are required to explain our technique. Section 4 describes the proposed technique to reduce process risks which is then evaluated in Section 5. Section 6 discusses related work and Section 7 concludes the paper.

2 Background and Running Example

The technique proposed in this paper belongs to a wider approach for the management of process-related risks. This approach aims to enrich the four phases of the BPM lifecycle (Process Design, Implementation, Enactment and Analysis) [9] with elements of risks management (see Fig. 1).

Before the *Process Design* phase, we add an initial phase, namely *Risk Identification*, where existing techniques for risk analysis such as Fault Tree Analysis [4] or Root Cause Analysis [11] can be used to identify possible risks of faults that may eventuate during the execution of a business process. Faults and their risks identified in this phase are mapped onto specific aspects of the process model during the *Process Design*

phase, obtaining a risk-annotated process model. In the *Process Implementation* phase, a more detailed mapping is conducted linking each risk and fault to specific aspects of the process model, such as content of data variables and resource states. In the *Process Enactment* phase such a risk-annotated process model is executed.

Finally, information produced during the *Process Enactment* phase is used in combination with historical data during the *Process Diagnosis* phase, to monitor the occurrence of risks and faults during the execution of a process instance. This monitoring may trigger some form of mitigation in order to (partially) recover the process instance from a fault. The technique presented this paper fits in this latter phase, since it aims to provide run-time support in terms of risk prediction, by combining information on risks and faults with historical data.

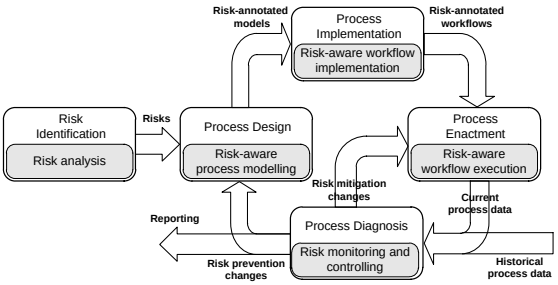


Fig. 1. Risk-aware BPM lifecycle.

To illustrate how this technique works, we use the example model shown in Figure 2. The process captured by this model may be subjected to several risks during its execution. The model is defined using the YAWL language. Thus, before explaining this example, we introduce the basic ingredients of YAWL.

We will not repeat the full definition of a YAWL specification as defined in [18]. Rather, we will only describe those parts that are relevant to this paper. Each YAWL specification is made up of one or more nets organized hierarchically in a root net and zero or more subnets (each modeling a subprocess). Each net is defined as a set of conditions C (represented as circles), an input condition $i \in C$, an output condition $o \in C$, and a set of tasks T (represented as boxes). Tasks are connected to conditions via a flow relation $F \subseteq (C \setminus \{o\} \times T) \cup (T \times C \setminus \{i\}) \cup (T \times T)$ (represented as a set of arcs). We write T_N and C_N to access the tasks and conditions of a net N.

Tasks model units of work that are performed either by process participants (*user tasks*) or by software services (*automated tasks*). An example of an automated task is Receive Confirmation Order in Fig. 2, while an example of user task is Estimate Trailer Usage. Conditions denote states of execution, for example the state before executing a task or that resulting from its execution. Conditions can also be used for routing purposes when they have more than one incoming and/or outgoing flow relation. In particular, a condition followed by multiple tasks, like condition FLT in Fig. 2, represents a *deferred choice*, i.e. a choice which is not determined by some process data, but rather by the first process participant that is going to start one of the outgoing tasks of this condition. In the example, the deferred choice is between tasks Arrange Delivery Appointment, Arrange Pickup Appointment and Create Shipment Information Document, each assigned to a different process participant. When the choice is based on data, this is captured in YAWL by an XOR-split if only one outgoing flow can be taken, or by an OR-split if one or more outgoing flows can be taken. XOR-join and OR-join capture the merging behavior of their respective splits. Finally, an AND-split captures two or more flows that have to be executed in parallel while the AND-join is used to synchronize parallel flows. Splits and joins are represented as decorators on the task's box.

In YAWL trivial conditions, i.e. those having a single incoming flow and a single outgoing flow, can be hidden. To simplify the discussion in the paper, without loss of generality, we assume a strict alternation between tasks and conditions. Under this assumption, the preset of a task t is the set of its input conditions: ${}^\bullet t = \{c \in C_N \mid (c,t) \in F\}$. Similarly, the postset of a task t is the set of its output conditions: $t^\bullet = \{c \in C_N \mid (t,c) \in F\}$. The preset and postset of a condition can be defined analogously.

Placing a token in the input condition of a YAWL net initiates a new process instance. The token corresponds to the thread of control and it flows through the net as tasks are executed. Each task execution consumes one token from some of its input conditions (depending on the type of join preceding the task) and produces one token in some of its output conditions (depending on the type of split following the task).

Example 1 *The example in Fig. 2 shows the Carrier Appointment subprocess of an Order Fulfillment process. This process is inspired by the VICS industry standard for logistics [20]. This standard is endorsed by 100+ companies worldwide, with a total sales volume of $2.3 Trillion annually [34]. The Carrier Appointment subprocess starts when a Purchase Order Confirmation is received. In this case a Shipment Planner makes an estimation of the trailer usage and prepares a route guide. Once they are ready a Supply Officer prepares a quote for the transportation which indicates the cost of the shipment, the number of packages and the total freight volume.*

If the total volume is over 10,000 lbs a full track is required. In this case two different Client Liaisons will try to arrange a pickup appointment and a delivery appointment. Before these two tasks are performed, a Senior Supply Officer may create a Shipment Information document. In case the Shipment Information document is prepared before the appointments are arranged, a Warehouse Officer will arrange a pickup appointment and a Supply Officer will arrange a delivery appointment, with the possibility of modifying these appointments until a Warehouse Admin Officer produces a Shipment Notice after which the freight will be picked up from the Warehouse.

If the total volume is below 10,000 lbs and there is more than one package, a Warehouse Officer arranges the pickup appointment and a Client Liaison tries to arrange the delivery appointment. Afterwards, a Senior Supply Officer creates a Bill of Lading, which is similar to the Shipment Information document. If a delivery appointment is missing a Supply Officer takes care of it. After this point the rest of the process is the same as for the full track option.

If the customer ordered a single package, a Supply Officer has to arrange a pickup appointment, a delivery appointment, and has to create a Carrier Manifest, after which a Warehouse Admin Officer produces a Shipment Notice. □

3 Event Logs And Fault Severity

The execution of completed and running process instances can be stored in an event log:

Definition 1 (Event Log). *Let T and V be a set of tasks and variables, respectively. Let U be the set of values that can be assigned to variables. Let R be the set of resources that are potentially involved during the execution. Let D be the universe of timestamps. Let Φ be the set of all partial functions $V \nrightarrow U$ that define an assignment of values to a sub set of variables in V. An **event log** $\mathcal{L}$ is a multiset of traces where each trace is a sequence of events of the form (t, r, d, ϕ), where $t \in T$ is a task, $r \in R$ is the resource performing t, $d \in D$ is the event's timestamp, $\phi \in \Phi$ is an assignment of values to a sub set of variables in V. In other words, $\mathcal{L} \in \mathcal{B}((T \times R \times D \times \Phi)^*)$.*[4]

[4] $\mathcal{B}(X)$ the set of all multisets over X.

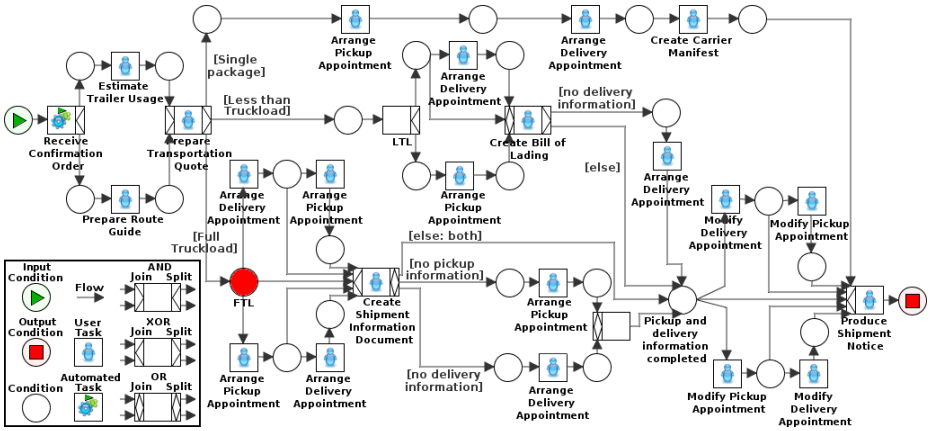


Fig. 2. Order-Fulfillment: Carrier Appointment subprocess.

Each completed trace of the event log is assigned a fault's severity between 0 and 1, where 0 identifies an execution with no fault and 1 identifies a fault with the highest severity. To model this, a risk analyst needs to provide a fault function f. The set of all such functions is:

$$\mathcal{F} = (T \times R \times D \times \Phi)^* \rightarrow [0, 1]$$

In many settings, processes are associated with different faults. These faults can be combined together by assigning different weights. Let us suppose to have n faults $\{f_1, \ldots, f_n\}$ with $f_i \in \mathcal{F}$ for every $i \in [1, n]$, we can have a *composite fault*:

$$\widehat{f} = \frac{\sum_{1 \leq i \leq n} w_i f_i}{\sum_{1 \leq i \leq n} w_i} \in \mathcal{F}$$

where w_i is the weight of the fault f_i, with $1 \leq i \leq n$.

Example 2 *Three faults can naturally be thought for a complete trace σ relative to a process instance of our running example of Carrier Appointment:*

Over-time fault. *It is linked to a Service Level Agreement (SLA) which establishes that the process must terminate within a certain Maximum Cycle Time d_{mct} (e.g. 21 hours), in order to avoid pecuniary penalties that will incur as consequence of a violation of the SLA. The severity of the fault grows with the amount of time that the process execution exceeds d_{mct}. Let d_σ be the duration of the process instance, i.e. difference between the timestamps of the last and first event of σ. Let $d_{\max}$ be the maximum duration among all process instances already completed (including σ). The severity of an overtime fault is measured as follows:*

$$f_{time}(\sigma) = \max\left(\frac{d_\sigma - d_{mct}}{d_{\max} - d_{mct}}, 0\right)$$

Reputation-loss fault. *During the execution of the process when a "pickup appointment" or a "delivery appointment" is arranged, errors with location or time of the appointment may be committed due to the misunderstanding between the company's employee and the customer. In order to keep the reputation high, the company wants to avoid these misunderstandings*

and from having to call the customer again, which may affect the reputation. The severity of this fault is:

$$f_{rep}(\sigma) = \begin{cases} 0 & \textit{if tasks} \text{ Modify Delivery Appointment } \textit{and} \text{ Modify Pick-up Appointment } \textit{do not appear in } \sigma \\ 1 & \textit{if both} \text{ Modify Delivery Appointment } \textit{and} \text{ Modify Pick-up Appointment } \textit{appear in } \sigma \\ 0.5 & \textit{otherwise} \end{cases}$$

Cost Overrun fault. *During the execution of this process, several activities need to be executed, and each of these has an execution cost associated with it. Since the profit of the company decreases with a higher shipping cost of a good (or goods), the company wants to reduce them. Of course, there is a minimum cost under which it is actually impossible to go. The severity increases as the cost goes beyond the minimum. Let $c_{\max}$ be the greatest cost associated with any process instance that has already been completed (including σ). Let c_σ be the cost of σ and $c_{\min}$ be the minimum cost that any process instance can undergo. The severity of a cost fault is:*

$$f_{cost}(\sigma) = \min\left(\frac{c_\sigma - c_{\min}}{c_{\max} - c_{\min}}, 1\right)$$

Moreover, we assume that the company reputes Reputation-loss Fault to be less significant than the others. Therefore, e.g., we can also define a composite fault where the reputation weights half:

$$f_{car}(\sigma) = \big(f_{cost}(\sigma) + f_{time}(\sigma) + 0.5 \cdot f_{rep}(\sigma)\big)/2.5$$

□

We distinguish between a fault's severity and a fault's likelihood. The risk is the product of the estimation of the fault's severity at the end of the process-instance execution and the likelihood of such an estimation.

When a process instance is being executed, many factors that may influence the risk and, ultimately, the severity of a possible fault. For instance, a specific order with which a certain set of tasks is performed may increase or decrease the risk, with respect to other orders. Nonetheless, it is opportune to leave freedom to resources to decide the order of their preference. Indeed, there may be factors outside the system that let resources opt for a specific order. For similar reasons, when there are alternative tasks that are all enabled for execution, a risk-aware decision support may highlight those tasks whose execution yields less risk, anyway leaving the final decision up to the resource.

4 Decision Support for Risk Reduction

In order to provide decision support for risk reduction, it is necessary to predict the most likely fault severity associated with continuing the execution of a process instance with each task enabled for execution. The problem of providing such a prediction can be translated into the problem of finding the best estimator of a function.

Definition 2 (Function estimator). *Let $X_1, \ldots, X_n$ be n finite or infinite domains. Let Y be a finite domain. Let $f : X_1 \times X_2 \times \ldots \times X_n \to Y$. An estimator of function f is a function $\psi_f : Y \to 2^{X_1 \times X_2 \times \ldots \times X_n \times [0,1]}$, such that, for each $y \in Y$, $\psi_f(y)$ returns a set of tuples $(x_1, \ldots, x_n, l)$ where $(x_1, \ldots, x_n) \in (X_1 \times X_2 \times \ldots \times X_n)$ is an input domain tuple for which the expected output is y and l is the likelihood of such an estimation. Moreover, for each $y \in Y$, $\psi_f(y)$ cannot contain identical domain tuples with different likelihood: $(x_1, \ldots, x_n, l_1) \in \psi_f(y) \wedge (x_1, \ldots, x_n, l_2) \in \psi_f(y) \Rightarrow l_1 = l_2$.*

The function estimator is trained through a set of observations. An observation instance is a pair $(\overrightarrow{x}, y)$ where $\overrightarrow{x} \in X_1 \times X_2 \times \ldots \times X_n$ is the observed input and $y \in Y$ is the observed output. Given a set I of observation instances, the construction of a function estimator is abstracted as a function buildFunctionEstimator(I), which returns a function ψ_f.

The function estimator can be easily built using many machine learning techniques. In this paper, we employ decision-tree building algorithms, specifically the C4.5 algorithm [14]. Decision trees classify instances by sorting them down in a tree from the root to some leaf node. Each non-leaf node specifies a test of some attribute $x_1, \ldots, x_n$ and each branch descending from that node corresponds to a range of possible values for this attribute. In general, a decision tree represents a disjunction of conjunctions of expressions: each path from the tree root to a leaf corresponds to an expression that is, in fact, a conjunction of attribute tests. Each leaf node is assigned one of the possible output values: if an expression e is associated with a path to a leaf node $\overline{y}$, every input domain tuple $\overrightarrow{x} \in X_1 \times X_2 \times \ldots \times X_n$ for which e evaluates to true is expected to return $\overline{y}$ as output.

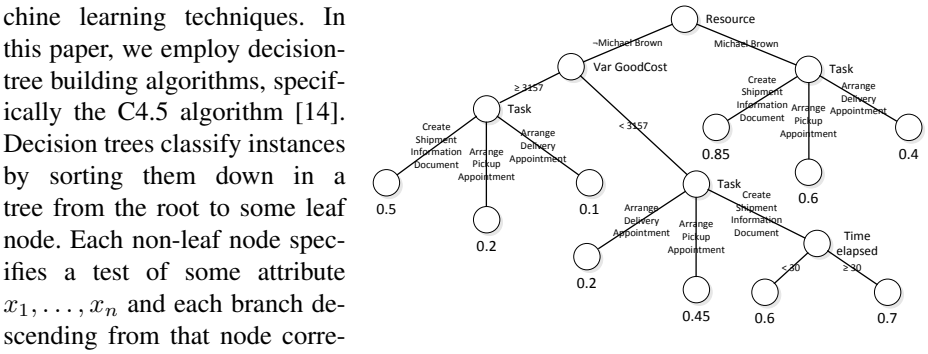


Fig. 3. An example of Decision Tree used to build a function estimator.

We link the likelihood of a prediction for $\psi_f(\overline{y})$ to the quality of e as classifying expression. Let $\#_n$ be number of observation instances $(\overrightarrow{x}, y)$ such that e evaluates to true with respect to $\overrightarrow{x} = (x_1, \ldots, x_n)$. A subset of these instances are correctly classified (i.e., $y = \overline{y}$). If $\#_c$ is the number of those correctly classified, for all $(x_1, \ldots, x_n, l) \in \psi_f(\overline{y})$, likelihood $l = \#_c / \#_n$.

Example 3 *Figure 3 shows an example of a possible decision tree obtained through a set of observation instances to build the estimator $\psi_{f_{\hat{c}}}$ of a function $f_{\hat{c}}(Resource, Task, GoodCost, TimeElapsed) = y \in [0, 1]$. For instance, let us consider the value $y = 0.6$. Analyzing the tree, the value is associated with two expressions:* $e_1 \leftarrow (Resource =$ Michael Brown $\wedge\ Task =$ Arrange Pickup Appointment*) and* $e_2 \leftarrow (Resource \neq$ Michael Brown $\wedge\ GoodCost < 3157 \wedge TimeElapsed < 30 \wedge Task =$ Create Shipment Information Document*). Let us suppose that, among observation instances $(Resource, Task, GoodCost, TimeElapsed, y)$ s.t. e_1 or e_2 evaluates to true, $y = 0.6$ occurs 60% or 80% of times, respectively. With this tree, e.g., $\psi_{f_{\hat{c}}}(0.6)$ contains the tuples $(Resource, Task, GoodCost, TimeElapsed, 0.6)$ for each e_1 evaluates to true, along with the tuples $(Resource, Task, GoodCost, TimeElapsed, 0.8)$ for each e_2 evaluates to true.* □

Regarding computational complexity, if decision trees are used, training ψ_f with m observation instances is computed in quadratic time [14] with respect to n, specifically $O(n^2 \cdot m)$.

As mentioned before, it is necessary to predict the most likely fault severity associated with continuing the execution of a process instance with each task enabled

for execution. Function estimators are used for such a prediction. Since tasks consume tokens from their own input conditions, we associate a function estimator with each condition of the YAWL specification. Given a condition c, the function estimator ψ_c for c predicts the risk associated with consuming a token from c by each task t in the postset of c.

Example 4 *The function estimator $\psi_{f_{\hat{c}}}$ can be associated with the condition FTL, which is red-colored in Figure 2. Here, for simplicity, let us suppose the likelihood is 1 for all estimations. If the execution is such that there is a token in FTL, $GoodCost < 3157$, executing tasks* Arrange Pickup Appointment*,* Arrange Delivery Appointment *are associated with a risk of 0.2 and 0.45, respectively. Conversely, executing task* Create Shipment Information Document *is given a risk of either 0.6 or 0.7, depending on the time elapsed since the instance has started. Therefore, it is evident that it is less "risky" to execute* Arrange Pickup Appointment. □

Given a concluded process instance identified by a log trace $\sigma \in (T \times R \times D \times \Phi)^*$, an observation instance $(\overrightarrow{x}, y)$ to train ψ_c is relative to each event (t, r, d, ϕ) such that t is in the postset of c. More specifically, the input $\overrightarrow{x}$ contains the process variable's value before the event has occurred, task t, the time elapsed since the execution has started, and the contextual information; the output y is the fault's severity observed for σ, i.e. $y = f(\sigma)$ for some fault function f. The contextual information of the event is relative to the prefix σ' of the trace σ before the occurrence of that event. In particular, it contains, for each task in the process specification, the number of times that the task has been performed and the last resource that has executed it. This information is clearly relevant to guarantee predictions with higher likelihood. Indeed, the risk is generally linked to the resources that perform the activities, since some resources may be more prone to be mistaken. Concerning the number of executions of tasks, let us consider the overtime fault in Example 2: the risk reasonably increases with the number of repetitions of certain tasks. In the remainder, the retrieval of the contextual information is abstracted as function $C = \mathsf{getContextInformation}(\sigma')$ which returns a tuple C containing this information.

In the remainder, we use $\odot$ to denote the operator to concatenate tuples: given two tuples $\overrightarrow{x} = (x_1, \ldots, x_n)$ and $\overrightarrow{y} = (y_1, \ldots, y_m)$, $\overrightarrow{x} \odot \overrightarrow{y} = (x_1, \ldots, x_n, y_1, \ldots, y_m)$. Operator $\odot$ can also be overridden to deal with functions defined on a finite and ordered domain. Let $f : W \rightarrow Z$ be a function defined on an ordered domain $W = \langle w_1, \ldots, w_o \rangle$. If we denote $z_i = f(w_i)$ with $1 \leq i \leq o$, $f \odot \overrightarrow{x} = (z_1, \ldots, z_o, x_1, \ldots, x_n)$.

Algorithm 1 details how the function estimators ψ_c mentioned above can be constructed. This algorithm is periodically executed, e.g., every week or every k process instances are completed. In this way, the predictions are updated according to the recent process executions. The input parameters of the algorithm are a YAWL net N, an event log with traces referring to past execution of instances of this process, and a fault function. The output is a function Ψ that associates each condition with the opportune function estimator. Initially, in line 1, we initialize function I which is going to associate each condition c with the set of observation instances relative to execution of tasks in the postset of p. From line 2 to line 12, we iteratively replay all traces σ to build the observation instances. While replaying, a function A keeps the current value's assignment to variables (line 3). For each trace's event (t_i, r_i, d_i, ϕ_i), first we build the tuple C of the contextual information (line 5) and compute the elapsed time $\overline{d}$ (line 6). Then, we build an observation instance J where tuple $A \odot (t_i, r_i, \overline{d}) \odot C$ is the observed input

Algorithm 1: GENERATEFUNCTIONESTIMATORSFORRISKPREDICTION

Data: $\mathbf{N}$ – A YAWL net, $\mathcal{L}$ – An event log, $f \in \mathcal{F}$ – A fault function
Result: A Function Ψ that associates each condition $c \in C_N$ with a function estimator ψ_c

1 **Let** I be a function whose domain is the set of conditions $c \in C_N$, and initially $\forall c \in C.\ I(C) = \emptyset$.
2 **foreach** trace $\sigma = \langle (t_1, r_1, d_1, \phi_1), \ldots, (t_n, r_1, d_n, \phi_n) \rangle \in \mathcal{L}$ **do**
3 **Set** function A such that $\mathsf{dom}(A) = \emptyset$
4 **for** $i \leftarrow 1$ **to** n **do**
5 $C \leftarrow \mathsf{getContextInformation}(\sigma)$
6 Time elapsed $\overline{d} \leftarrow (d_i - d_1)$
7 $J \leftarrow (A \odot (t_i, r_i, \overline{d}) \odot C, f(\sigma))$
8 **foreach** $c \in {}^\bullet t_i$ **do**
9 $I(c) \leftarrow I(c) \cup \{J\}$
10 **end**
11 **foreach** *variable* $v \in \mathsf{dom}(\phi_i)$ **do**
12 $A(v) \leftarrow \phi_i(v)$
13 **end**
14 **end**
15 **end**
16 **Set** function Ψ such that $\mathsf{dom}(\Psi) = \emptyset$
17 **foreach** *condition* $c \in C_N$ **do**
18 $\Psi(c) \leftarrow \mathsf{buildFunctionEstimator}\big(I(c)\big)$
19 **end**
20 **return** Ψ

Algorithm 2: GENERATERECOMMENDATIONS

Data: Ψ – A Function that associates each condition with a function estimator, σ – a sequence of events, $\mathcal{T}$ - a set of tasks, A – the value's assignment to variables after the occurrence of the events in σ
Result: A Function R that associates each task $t \in \mathcal{T}$ with a risk.

1 **Let** $\sigma = \langle (t_1, r_1, d_1, \phi_1), \ldots, (t_n, r_1, d_i, \phi_i) \rangle \in (T \times R \times D \times \Phi)^*$
2 $C \leftarrow \mathsf{getContextInformation}(\sigma)$
3 $d_e = d_{now} - d_1$
4 **foreach** *task* $t \in \mathcal{T}$ **do**
5 $R(t) \leftarrow 0$
6 $(x_1, \ldots, x_n) \leftarrow A \odot (t, r, d_e) \odot C$
7 **foreach** *condition* $c \in {}^\bullet t$ **do**
8 $\psi_c \leftarrow \Psi(c)$
9 **if** $\exists y, l$ **such that** $(x_1, \ldots, x_n, l) \in \psi_c(y)$ **then**
10 $R(t) \leftarrow \max\big(R(t), y \cdot l\big)$
11 **end**
12 **end**
13 **end**
14 **return** R

and the fault severity $f(\sigma)$ is the observed output. This observation instance is put into the set of observation instances relative to each condition $c \in {}^\bullet t_i$. In lines 11-13, we update the current value's assignment during the replay, i.e. we rewrite function A. Finally, in lines 16-19, we build each function estimator ψ_c for condition c by the relative observation instances and rewrite Ψ s.t. $\Psi(c) = \psi_c$.

At run-time, function Ψ is used to predict the risk and, hence, to provide recommendations. In fact, function Ψ is input for Algorithm 2, which produces the recommendations for a set of task $\mathcal{T}$ relative to a given process instance, in which a sequence σ of events has occurred. The input of the algorithm contains also a function A that associates each instance's variable v with the value $A(v)$ after the events in σ have occurred. The algorithm's output is a function that associates each task $t \in \mathcal{T}$ with the relative risk. For each task $t \in \mathcal{T}$, the algorithm computes the tuple $(x_1, \ldots, x_n)$ that

contains the instance's state, task t, the time elapsed d_e since the execution has started, and the contextual information C (line 6). Then, for each function estimator ψ_c associated with each condition in the preset of t, we find the expected fault's severity y and the expectation's likelihood l such that $\psi_c(x_1, \ldots, x_n, l) = y$. The value $y \cdot l$ is the risk associated with the instance if t is performed and consumes a token from c. The risk associated with continuing the execution by performing t is the maximum with respect to all conditions from which tokens are consumed when t is executed.

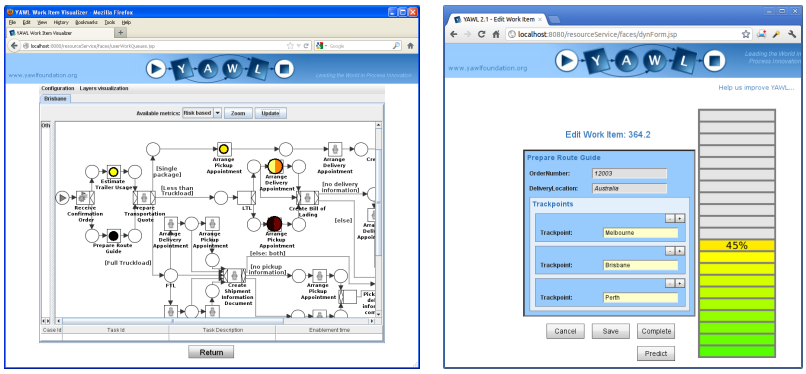
5 Implementation and Evaluation

We operationalized our technique for the YAWL system by extending a visual plug-in for the YAWL worklist handler and by implementing a new custom YAWL service. The YAWL system [18] is an open source workflow managemet system which is based on the workflow patterns[5] and uses a service-oriented architecture.

The intent of our technique is to "drive" participants during the execution of a process instance. This goal can be achieved if participants can easily understand a proposed suggestion. In order to do this, we extended a previous visual plug-in for YAWL [7] for decision support, named *Map Visualizer*. This plug-in provides a graphical user interface to suggest the tasks to execute, along with assisting during their execution. The tool is based on two orthogonal concepts: maps and metrics. A map may can be a geographical map, a process model, an organizational diagram, etc. For each map, tasks can be visualized by dots which are located in a meaningful position (e.g., for a geographic map, tasks are projected onto the locations where they need to be executed, or for a process-model map onto the corresponding tasks in the model). Dots can also be colored according to certain metrics, which determine the suggested level of priority of a task to be executed. However very basic metrics are only defined in [7]. With this paper, we designed a new metric which is computed by employing the technique described in Section 4. The validity of the metaphors of maps and metrics has been confirmed through a set of experiments, as reported in [7].

Figure 4(a) shows a screenshot of the Map Visualizer where a risk-based metric is employed. The map shows the process model using the YAWL notation and dots are projected onto the corresponding element of the model. Each dot corresponds to a different task and is colored according to the risks for the three faults defined before. When multiple dots are positioned at the same coordinates, they are merged into a single larger dot whose diameter grows with the number of dots being amalgamated. According to the analysis reported in [7], the possible colors go from white to black, passing through intermediate shades of yellow, orange, red, purple and brown. The white and black colors identify tasks associated with a risk of 0 and 1, respectively. The screenshot in Fig. 4(a) refers to a configuration where multiple process instances are being carried on at the same time and, hence, the tasks refer to different process instances. The configuration of dots highlights that the risk is lower if the process participant performs the task *Estimate Trailer Usage*, *Arrange Pickup Appointment* or *Arrange Delivery Appointment* for a certain instance. When clicking on the dot, the participant is shown the process instance of the relative task(s). As mentioned in Sections 1 and 4, the activity of compiling a form is also supported. Figure 4(b) shows a screenshot where, while filling

[5] www.workflowpatterns.com



(a) The UI to support participants in choosing the next task to perform based on risks.

(b) The UI to support participants in filling out a form based on risks.

Fig. 4. Screenshots of the Map Visualizer extension for risk-aware prediction in YAWL.

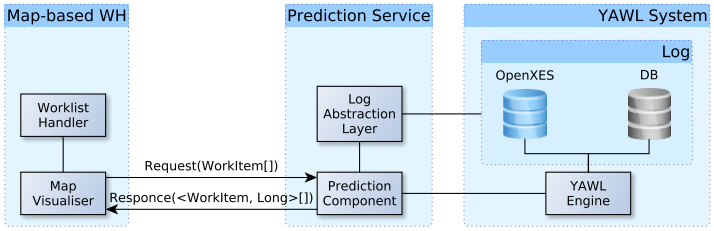


Fig. 5. The integration of the implemented tools with the YAWL system.

in a form, participants are shown the risk associated with that specific input for that form via a vertical bar (showing a value of 45% in the example). While a participant changes the data in the form, the risk value is recomputed accordingly.

Besides the extension to the Map Visualizer, we implemented a new custom service for YAWL, namely the *Prediction Service*. This service provides risk-aware prediction and recommendation. It employs the technique described in Section 4 using the J48 algorithm to generate decision trees from the Weka toolkit for data mining.[6] The prediction service communicates with the *Log Abstraction Layer* described in [5], to be able to retrieve event logs from textual files, such as from OpenXES event logs, or from the database that is used by YAWL, storing both historical information and the current system's state. The *Prediction Service* is invoked by the *Map Visualizer* to obtain the risk predictions and recommendations. The map visualizer works together with the standard *Worklist Handler* provided by YAWL to obtain the up-to-date distribution of work to resources. Figure 5 shows the diagram of these connections.

We evaluated the technique using the Carrier Appointment example described in Section 2. We used CPN Tools [7] to simulate the process model and the resource behavior. We performed three sets of experiments with different faults. First, we only used

[6] Available at `www.cs.waikato.ac.nz/ml/weka/`

[7] Available at `www.cpntools.org`

a composite fault that includes Overtime and Reputation; then, we included all of the three faults; finally, we relaxed the Overtime fault by increasing the maximum cycle time d_{mct} from 21 hours to 25, and by increasing the minimum cost that any process instance can undergo c_{min} from 70% to 85% of the value of the good sold.

For each set of experiments, we randomly generated 2,000 log traces with CPN Tools, which we used to train the function estimators. In fact, these traces are relative to process instances that do not follow any suggestion, for which we also computed the severity of the different composite faults as mentioned in the previous paragraph. Finally, we generate 200 new log traces where the executions have always followed the recommendations proposed by our tool. Figure 5 shows the results comparing the fault's severity when recommendations were and were not followed. It is worth highlighting how the results are given in terms of severity measured for completed instances. Risks are relative to running instances and estimate the expected fault's severity and likelihood when they will eventually complete.

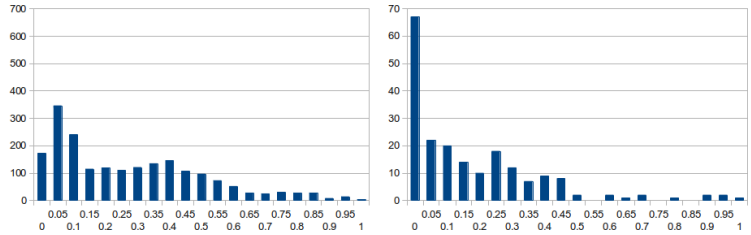
Despite the number of instances in which recommendations were followed is a tenth of the number of instances in which the execution was random, a comparison among the distributions of fault's severity is possible. In all three sets of experiments, our tool significantly reduced the overall severity and number of faults and changed the shape of the distribution increasing the number of instances terminating without faults (i.e., severity equal to 0).

6 Related Work

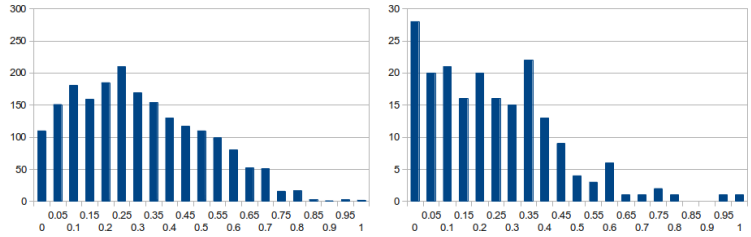
Various risk analysis methods have been defined which provide elements of risk-aware process management. Meantime, academics have recognized the importance of managing process-related risks. However, risk analysis methods only provide guidelines for the identification of risks and their mitigations, while academic efforts mostly focus on risk-aware BPM methodologies in general, rather than on concrete approaches for risk prevention [17]. For a comprehensive review of approaches and methods for managing and analyzing process risks, we refer to the survey in [17].

An exception is made by the works of Pika et al. [13] and Suriadi et al. [16]. Pika et al. propose an approach for predicting overtime risks based on statistical analysis. They identify five process risk indicators whereby the occurrence of these indicators in a trace indicates the possibility of a delay. Suriadi et al. propose an approach for Root Cause Analysis based on classification algorithms. After enriching a log with information like workload, occurrence of delay and involvement of resources, they use decision trees to identify the causes of overtime faults. The cause of a fault is obtained as a disjunction of conjunctions of the enriching information. Despite looking at the same problem from different prospectives, these two approaches result to be quite similar. The main difference between them and our technique is that we use risk prediction as a tool for providing suggestions in order to prevent the eventuation of faults, while they limit their scope to the identification of indicators of risks or of causes of faults. Moreover, both approaches do not consider the data prospective and have only been designed for overtime risks.

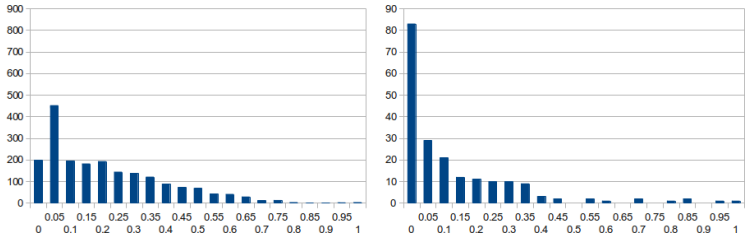
Our work shares commonalities with recommendation and Decision Support Systems (DSSs), since it provides recommendations to process participants to make risk-informed decisions. Our technique fully embraces the aim of these systems to improve



(a) Results using a composite fault that includes Overtime and Reputation risks. Following the suggestions, the fault's severity is reduced from an average of 5.5 to 3.47 (-37%)



(b) Results using a composite fault that includes all the three faults defined in the running example with maximum cycle time $d_{mct} = 21$ and minimum cost $c_{min} = 70\%$. Following the suggestions, the fault's severity is reduced from an average of 6.1 to 4.7 (-21%)



(c) Results using a composite fault that includes all three faults with maximum cycle time $d_{mct} = 25$ and minimum cost $c_{min} = 85\%$. Following the suggestions, the fault's severity is reduced from an average of 4.24 to 2.59 (-38.9%)

Fig. 6. The outcome of the experiments for different composite faults. The x-axis represents the severity of the composite fault and the y-axis represents the number of instances that terminated with such a severity. The left-hand side figures show the distribution of the fault's severity when recommendations were not followed, the right-hand side figures show the distribution when recommendations were always followed.

decision making within work systems [2], by providing an extension to existing process-aware information systems. In this area, Dey [8] describes how DSS can be used for risk management. He also uses decision trees despite those trees are manually generated as a final step of brainstorming sessions and used only as a reference when risks occur.

Operational support is an emerging field of research in process mining, which shares commonalities with DSSs. Operational support concerns three dimensions: detection, prediction and recommendation [1]. In this paper, we focus on the latter two dimensions. Prediction is about forecasting which faults are likely to be reached at the

end of the process instance and with what severity. Recommendation concerns enacting the appropriate actions to prevent faults from occurring. We dealt with run-time risk detection in previous work [5]. Regarding prediction, van der Aalst et al. [19] propose an approach to predict the remaining cycle time till the end of the execution of a process instance on the basis of the process control-flow, while Folino et al. [10] use decision trees to improve the remaining cycle time estimation by also taking process data into account. Unfortunately, both approaches only focus on time, which is, in fact, linked to one possible cause of process fault (i.e. the overtime fault). Our technique is more generic since we aim to predict faults that are related to different dimensions such as cost and reputation. Westergaard et al. propose protocols and infrastructures for providing recommendation during process execution [12]. However, concrete recommendation algorithms are out of scope.

7 Conclusion

We proposed a technique that allows process participants to make risk-informed decisions when taking part in a business process. The technique relies on a risk estimator trained using historical information extracted from the process log. For each state of a process instance where input is required from a process participant, the estimator determines the severity and likelihood that a fault (or set of faults) will occur if the participant's input is going to be used to carry on the process instance.

We designed the technique in a language-independent manner and implemented it as a Web service. This service can train risk estimators by importing process logs in the standard OpenXES format or directly from the database of a workflow management system like YAWL. The service was linked to the YAWL system as a proof-of-concept. Specifically, we extended the Map Visualizer plug-in of YAWL so that risk predictions can be visualized as colored circles on top of tasks, indicating the likelihood and severity of faults. We also extended the YAWL user form visualizer. Based on the data inserted by the participant, a risk profile is shown. This way, participants are offered risk-based recommendations when selecting the next task to execute or filling out a form.

We simulated a large process model based on an industry standard for logistics and generated event logs for it. We then executed three experiments with different faults and fault conditions in order to obtain different fault distributions and used this log to train our risk estimator. We simulated new process instances according to the recommendations provided by our risk estimator and measured the number and severity of the faults upon instance completion. In all experiments we were able to substantially reduce the number of faults and their severities provided that the simulated users followed the recommendations provided by our technique. And while this shows that the technique is effective, it has to be considered as an upper-bound result, since in reality it might not always be feasible to follow the recommendations provided.

The technique suffers from some limitations that will be addressed in future work. First, it lacks an empirical evaluation of its usefulness with domain experts. We are planning to overcome this problem by performing experiments with risk analysts and process participants of a large Australian insurance company. Second, the technique does not support user decisions involving inter-instance or inter-process data, but only looks at single business processes. For example, the technique can be extended to support process administrators in taking risk-informed decisions when (re)allocating resources

to tasks, potentially belonging to instances of different processes. Finally, we plan to also investigate different machine-learning techniques to build function estimators, e.g. Bayesian networks or the k-means algorithm, to evaluate if other techniques outperform decision-tree building algorithms.

References

1. W. M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
2. S. Alter. A work system view of DSS in its fourth decade. *Decision Support Systems*, 38(3):319–327, 2004.
3. Basel Committee on Bankin Supervision. *Basel II - International Convergence of Capital Measurement and Capital Standards*, 2006.
4. International Electrotechnical Commission. *IEC 61025 Fault Tree Analysis (FTA)*, 1990.
5. R. Conforti, G. Fortino, M. La Rosa, and A. H. M. ter Hofstede. History-aware, real-time risk detection in business processes. In *Proc. of CoopIS*, volume 7044 of *LNCS*. Springer, 2011.
6. R. Conforti, A. H. M. ter Hofstede, M. La Rosa, and M. Adams. Automated risk mitigation in business processes. In *Proc. of CoopIS*, volume 7565 of *LNCS*. Springer, 2012.
7. M. de Leoni, M. Adams, W. M. P. van der Aalst, and A. H. M. ter Hofstede. Visual support for work assignment in process-aware information systems: Framework formalisation and implementation. *Decision Support Systems*, 54(1):345–361, 2012.
8. P. K. Dey. Decision support system for risk management: a case study. *Management Decision*, 39(8):634–649, 2001.
9. M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software through Process Technology*. Wiley & Sons, 2005.
10. F. Folino, M. Guarascio, and L. Pontieri. Discovering context-aware models for predicting business process performances. In *Proc. of CoopIS*. Springer, 2012. To appear.
11. W.G. Johnson. *MORT - The Management Oversight and Risk Tree*. U.S. Atomic Energy Commission, 1973.
12. J. Nakatumba, M. Westergaard, and W.M.P. van der Aalst. An infrastructure for cost-effective testing of operational support algorithms based on colored petri nets. In *Proc. of ATPN*, volume 7347 of *LNCS*. Springer, 2012.
13. A. Pika, W.M.P. van der Aalst, C. Fidge, A.H.M. ter Hofstede, and M. Wynn. Predicting deadline transgressions using event logs. In *Proc. of BPM Workshop 2012*. Springer, 2012. To Appear.
14. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.
15. Standards Australia and Standards New Zealand. *Standard AS/NZS ISO 31000*, 2009.
16. S. Suriadi, Chun Ouyang, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Root cause analysis with enriched process logs. In *Proc. of BPM Workshop 2012*. Springer, 2012. To Appear.
17. S. Suriadi, B. Weiß, A. Winkelmann, A. ter Hofstede, M. Wynn, C. Ouyang, M.J. Adams, R. Conforti, C. Fidge, M. La Rosa, and A. Pika. Current research in risk-aware business process management - overview, comparison, and gap analysis. BPM Center Report BPM-12-13, BPMcenter.org, 2012.
18. A. H. M. ter Hofstede, W. M. P. van der Aalst, M. Adams, and N. Russell, editors. *Modern Business Process Automation: YAWL and its Support Environment*. Springer, 2010.
19. W. M. P. van der Aalst, M. H. Schonenberg, and M. Song. Time prediction based on process mining. *Information Systems*, 36(2):450–475, 2011.
20. Voluntary Interindustry Commerce Solutions Association. *Voluntary Inter-industry Commerce Standard (VICS)*. http://www.vics.org. Accessed: June 2011.