Automated Risk Mitigation in Business Processes

Raffaele Conforti[1], Arthur H.M. ter Hofstede[1,2,3],
Marcello La Rosa[1,2], and Michael Adams[1]

[1] Queensland University of Technology, Australia
{raffaele.conforti,a.terhofstede,m.larosa,mj.adams}@qut.edu.au
[2] NICTA Queensland Lab, Australia
[3] Eindhoven University of Technology, The Netherlands

Abstract. This paper proposes a concrete approach for the automatic mitigation of risks that are detected during process enactment. Given a process model exposed to risks, e.g. a financial process exposed to the risk of approval fraud, we enact this process and as soon as the likelihood of the associated risk(s) is no longer tolerable, we generate a set of possible mitigation actions to reduce the risks' likelihood, ideally annulling the risks altogether. A mitigation action is a sequence of controlled changes applied to the running process instance, taking into account a snapshot of the process resources and data, and the current status of the system in which the process is executed. These actions are proposed as recommendations to help process administrators mitigate process-related risks as soon as they arise. The approach has been implemented in the YAWL environment and its performance evaluated. The results show that it is possible to mitigate process-related risks within a few minutes.

1 Introduction

Business processes in various sectors such as financial, healthcare and oil&gas, are constantly exposed to a wide range of risks. Take for example the BP oil spill in 2010 which resulted in an environmental disaster, or the fraud at Société Générale in 2008, which led to a €4.9B loss.

A *process-related risk* measures the likelihood and the consequence that something happening will impact on the process objectives [29]. Failing to address process-related risks can result in substantial financial and reputational consequences, potentially threatening an organization's existence, like in the case of Société Générale. There is thus an increasing need to better manage business process risks, as also highlighted by legislative initiatives like Basel II [7] and the Sarbanes-Oxley Act.[1] Organizations are attempting to incorporate process-related risks as a distinct view in their operational management, seeking effective ways to *control* such risks. However, whilst conceptually appealing, to date there is little guidance as to how this can be concretely done.

In previous work [12], we presented a mechanism to model risks in executable business process models and detect them as early as possible during process execution. Unfortunately, detecting a risk in time is often not enough to avoid the negative outcome associated. A *prompt* risk mitigation should be taken to restore the process instance to

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

R. Meersman et al. (Eds.): OTM 2012, Part I, LNCS 7565, pp. 212–231, 2012.

a safe state, before the instance progresses any further. Moreover, taking the *right* mitigation at the right time may make the difference between success and failure. In fact, the number of possible ways a process-related risk may be mitigated is potentially very large that it is difficult for a process administrator to take the right decision at the right time, without any support. One has to consider all mitigations that are possible, given the current state of the process instance (including a snapshot of the associated data and resources), and the context in which the instance is running, i.e. the state of other running instances, to make such a decision. For example, in order to mitigate the risk of a process instance A to run overtime, a mitigation may entail to reallocate resources from a process instance B (potentially of another process) to A.

In light of the above, in this paper we propose a technique for automatically mitigating process-related risks. Since a process instance may be affected by multiple risks at the same time, we treat this problem as a *multi-objective optimization problem*. A solution to this problem is a variant of the risky process instance obtained by applying a sequence of mitigation actions, in order to reduce the risks' probability down to a tolerable level, or in the best case, to annul the risks altogether. Mitigation actions include control-flow aspects (e.g. skipping a task to be executed), process resources (e.g. reallocating a resource to a different task), and data (e.g. rolling back an executed task to restore its input data). To explore the potentially large solution space, we use dominance-based Multi-Objective Simulated Annealing (MOSA) [28]. At each run, the algorithm generates a small set of solutions similar to the original process instance but with less risks. It stops when either a maximum number of *non-redundant* solutions (i.e. solutions proposing different mitigations) is found or a given timeframe elapses. This approach is not meant to replace human judgement. Instead, it aims to support process administrators in deciding what mitigations to take, by reducing the number of feasible options, and consequently the time needed to take a decision.

We defined the mitigation actions in collaboration with an Australian risk consultant. To prove the feasibility of this approach, we implemented these actions and the MOSA algorithm on top of the YAWL system. We instantiated a set of process models from the logistics [34] and screen business [25] domains that are affected by one or more risks, and executed a series of tests to mitigate such risks. The tests show that the technique can find a set of possible solutions within a few minutes of computation, and that in all cases the associated risks are mitigated.

The rest of this paper is organized as follows. Section 2 introduces the required background concepts in the context of an example. Section 3 describes the proposed technique to mitigate process risks which is then evaluated in Section 4. Section 5 covers related work and Section 6 concludes the paper.

2 Background and Running Example

Our technique for risk mitigation is part of a holistic approach for managing process-related risks throughout the process lifecycle. Accordingly, the four phases of the traditional BPM lifecycle (Design, Implementation, Enactment and Analysis) [16] are each extended to incorporate elements of risk, as shown in Fig. 1. First, in a *Risk Identification* phase, the process model to be designed is analyzed for potential risks. Established risk analysis methods such as Fault Tree Analysis [11] or Root Cause Analysis [21]

can be employed in this phase. The output of this phase is a set of risks, each expressed as a *risk condition* that describes the set of events that lead to a potential fault occurrence. Then, in the Design phase, these high-level risk conditions are mapped to process model-specific aspects. The result of this phase is a risk-annotated process model. Next, in the Implementation phase, these conditions are linked to workflow-specific aspects, such as the content of data variables and resource allocation states. The model is then executed by a risk-aware process engine in the Enactment phase. To evaluate risk conditions in this phase, we need to consider the current state of all running instances of any process (and not only the instance for which we are computing the risk condition), the resources that are busy and available, and the values of the data variables being created and consumed. Moreover, we need to consider historical data, i.e. the archived execution data of all previous instances of the process. Finally, the Diagnosis phase involves risk monitoring and controlling, which can trigger changes in the current process instance, to *mitigate* the likelihood of a fault occurring, or in the underlying process model, to *prevent* a given risk from occurring in future instances. This risk mitigation phase is the focus of this paper.

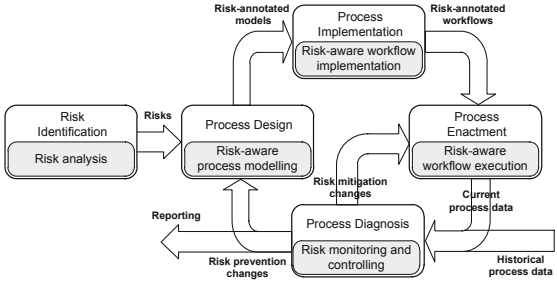

Fig. 1. Risk-aware BPM lifecycle

Let us now consider an example process for which we have defined several risks, to understand how risk conditions can be formulated in terms of process model elements. These conditions will provide input for the risk mitigation technique presented in the next section. The example process, shown in Figure 2, describes a *Payment* subprocess of an order fulfillment process, inspired by the VICS industry standard for logistics [34]. This standard is endorsed by 100+ companies worldwide, with a total sales volume of $2.3 Trillion annually [34]. The example process begins after freight has been picked up by a carrier and deals with the payment of shipment costs. First, a Shipment Invoice is produced for costs related to a specific order. If payment has been made in advance, a Finance Officer simply issues a Shipment Remittance Advice to the customer specifying the amount paid. Otherwise, the Finance Officer issues a Shipment Payment Order, which requires approval by a Senior Finance Officer (a superior of the Finance Officer) who may request amendments be made by the Finance Officer that issued the Order. After the document is finalized and the customer has paid, an Account Manager can process the payment. If the customer underpays, the Account Manager issues a Debit Adjustment, the customer makes a further payment and the payment is reprocessed. If a customer overpays, the Account Manager issues a Credit Adjustment. In the latter case and in the case of correct payment, the Payment subprocess completes.

In this process, we can identify various faults that may occur during execution. For example, a Service Level Agreement (SLA) may establish that the process (or one of its tasks) may not last longer than a Maximum Cycle Time MCT (e.g. 5 days), otherwise

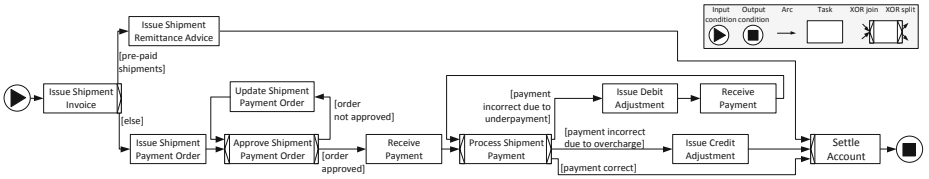

Fig. 2. Order-Fulfillment: Payment subprocess (using the YAWL [19] notation)

a pecuniary penalty may be incurred. To detect the risk of *overtime fault* at run-time, we should check the likelihood that the running instance does not exceed the MCT based on the amount of time T_c expired to that point. Let us consider T_e as the remaining cycle time, i.e. the amount of time estimated to complete the current instance given T_c based on past executions, which can be computed using the approach in [2]. Then the probability of exceeding MCT can be computed as $1 - MCT/(T_e + T_c)$ if $T_e + T_c > MCT$ and is equal to 0 if $T_e + T_c \leq MCT$. If this probability is greater than a tolerance value (e.g. 60%), we notify the risk to the user.

A second fault is related to the resources participating in the process. The Senior Finance Officer who has approved a Shipment Payment Order for a given customer must have not approved another order by the same customer in the last d days, otherwise there is a potential for *approval fraud*, a violation of a four-eyes principle across different instances of the Payment subprocess. To detect this risk we first have to check that there is an order, say order o of customer c, to be approved. Moreover, we need to check that either of the following conditions holds: i) o has been allocated to a Senior Finance Officer who has already approved another order for customer c in the last d days; or ii) at least one Senior Finance Officer is available who approved an order for customer c in the last d days and all other Senior Finance Officers who did not approve an order for c during the last d days are unavailable.

Finally, a third fault relates to a situation where a process instance executes a given task too many times, typically via a loop. Not only could this lead to a process slow-down, but also to a "livelock" if the task is in a loop whose exit condition is deliberately never met. In general, given a task t, a maximum number of allowable executions of t per process instance, $MAE(t)$, can be fixed as part of the service-level agreement for t. In our example, this fault may occur if task "Update Shipment Payment Order" is re-executed five times within the same process instance. We call this an *order unfulfillment* fault. To detect the risk at run-time, we need to check if: i) the Update task is currently being performed for order o; and ii) it is likely that the task will be repeated within the same process instance. The probability that the number of times a task will be repeated within the same instance is computed by dividing the number of instances where the MAE for the task has been reached by the number of instances that have executed this task at least as many times as it has been executed by the current instance, and have completed. If the probability to exceed $MAE(t)$ is greater than a tolerance value for t, e.g. 60%, we notify the risk to the user.

In the next section we propose a set of mitigation actions that can be performed "on-the-fly" on a running process instance in order to mitigate its risks.

3 Approach

In this paper we deal with the problem of automatically mitigating one or more business process risks for a specific running process instance (*case* for short), without raising other business process risks for the same case. This problem belongs to the family of multi-objective optimization problems, and we propose the use of simulated annealing for finding a Pareto-optimal solution, or a set of such solutions.

The Process Risk Simulated Annealing (PRSA) algorithm is an application of the DBMOSA [28] algorithm where at each iteration a new solution is discovered through the use of one or more random mitigation actions. The algorithm proposes a solution, or mitigation, as a sequence of elementary mitigation actions. A "behavioral cost" (cost for short) is associated with each mitigation action, and measures how deeply an action affects the process instance to which it is applied. For example, allocating a different resource to a work item has a lower cost than skipping a task that has to be executed. The total cost of a solution is the sum of the costs of each mitigation action used in that solution. A good solution to the PRSA algorithm is one that reduces the likelihood of a risk under its threshold, keeping the total cost as low as possible.

When comparing solutions that have the same cost, a solution that fully mitigates a risk is better than one that mitigates that risk because its risk condition is no longer evaluable. And in turn, this solution is better than one that does not mitigate the risk at all. Finally, if two solutions mitigate the same risk, we privilege the one that yields the lowest risk probability. Given two solutions a, b we say that a dominates b if it mitigates the same risks mitigated by b with a lower total cost. As result, we define them as mutually non-dominating if neither one dominates the other.

Below we describe the more elementary mitigation actions that can be used to create a solution, and how they affect a process case. We use the YAWL language as a reference language to define the mitigation actions, since this language has a formal foundation on which we can build our definitions and algorithms. However, the notions presented in this section can easily be generalized to other languages. Before introducing them, we introduce a number of preliminary concepts and notations.

YAWL Specification. We will not repeat the full definition of a YAWL specification as defined in [19], we will only use selected parts. The set of net identifiers is given by *NetID* and the process identifier is the net identifier of the root net, $ProcessID \in$ *NetID*. Furthermore, each net has, among others, a set of conditions C, an input condition $i \in C$, an output condition $o \in C$, and a set of tasks T and there is a flow relation $F \subseteq (C \setminus \{o\} \times T) \cup (T \times C \setminus \{i\}) \cup (T \times T)$.

We use the following auxiliary functions from [19]. The pre-set of x is defined as $\bullet x = \{y \in C \cup T \mid (y, x) \in F\}$ and the post-set of x is defined as $x\bullet = \{y \in C \cup T \mid (x, y) \in F\}$. We also introduced other auxiliary functions. The set of tasks that directly or through a place precedes a task ts is referred to as the *task pre-set* of t and is defined as $\circ t = \{x \in T \mid x \in \bullet t \vee \exists y \in C[y \in \bullet t \wedge x \in \bullet y]\}$. Similarly, the *task post-set* of t is defined as $t\circ = \{x \in T \mid x \in t \bullet \vee \exists y \in C[y \in t \bullet \wedge x \in y\bullet]\}$. Finally, to detect all the successors of a task, it is defined as $t\circ^*$ and it is the transitive closure of $t\circ$.

Following the convention in [19], we write e.g. T_n to access the tasks of net n. Moreover, for a YAWL specification y, T_y is the set of tasks that occur in any of its

nets, i.e. $T_y = \cup_{n \in NetID} T_n$, and for a set of YAWL specifications Y, T_Y is the set of tasks that occur in any of the nets of any of the specifications, i.e. $T_Y = \cup_{y \in Y} T_y$.

In our context we have only one Organizational model [19] and what is relevant for us is the set of resources, $UserID$, to whom work items can be assigned. Finally, we defined the set of *skippable* tasks as $\{t \in T_Y \mid \exists r \in UserID[skip \in UserTaskPriv(r,t)]\}$.

The set $StatusType$ contains the various statuses that a work item may go through during its lifecycle. These are: $offered$, $allocated$, $started$, $completed$, $forceCompleted$, $cancelled$, $failed$, $deadlocked$ used by the YAWL system and additionally $deoffered$, $deallocated$, $destarted$, $rollback$, $skipped$ used for mitigation purposes. Many of these statuses are self-explanatory. The status $rollback$ is the status of a work item which was completed but then enabled again though not $offered$. The status $skipped$ is the status of a work item that was skipped, which is similar to the status $completed$ but the work item was not actually performed. For convenience, we provide certain groupings of event types. In particular, $Rel \triangleq StatusType \setminus \{cancelled, failed, rollback\}$ is the set of event types that identify a work item as subject to mitigation. $Active \triangleq \{offered, allocated, started\}$ is the set of event types that mark a work item as in progress, $Completed \triangleq \{completed, forceCompleted\}$ is the set of event types that mark a work item as completed, and $ActiveC : Active \cup Completed$ is their union.

Given set $ActiveC$ we define a partial order $\subseteq_{\preccurlyeq} ActiveC \times ActiveC$ such that it preserves the partial ordering $deoffered < offered < allocated < started < completed = forceCompleted$.

Definition 1 (Log). *In the context of a set of YAWL specifications Y, with associated set of tasks T_Y and a set of root nets $\mathcal{R}$, a log is defined as $L = (\mathcal{E}, \mathcal{W}, \mathcal{C}, Model, WI, Case, Task, EvType, Time, Res, Inp, Outp)$ where:*

- *$\mathcal{E}$ is a set of events,*
- *$\mathcal{W}$ is a set of work items,*
- *$\mathcal{C}$ is a set of case identifiers,*
- *$Model : \mathcal{C} \to \mathcal{R}$ is a function relating cases to the root nets of the associated YAWL specification,*
- *$WI : \mathcal{E} \to \mathcal{W}$ is a surjective function relating events to work items,*
- *$Case : \mathcal{E} \to \mathcal{C}$ is a surjective function relating events to cases,*
- *$Task : \mathcal{W} \to T_Y$ is a function relating work items to tasks,*
- *$EvType : \mathcal{E} \to StatusType$ is a function relating events to work item statuses,*
- *$Time : \mathcal{E} \to \mathcal{T}$ is an injective function relating events to timestamps, hence no two events in the log can have identical timestamps,*
- *$Res : \mathcal{E} \to 2^{UserID}$ is a function relating events to sets of resources, as some events may concern multiple resources (e.g. a work item being offered),*
- *$Inp : \mathcal{E} \times Var \rightarrowtail \Omega$ is a partial function relating events and variables to (input) values,*
- *$Outp : \mathcal{E} \times Var \rightarrowtail \Omega$ is a partial function relating events and variables to (output) values.*

Definition 2 (Event Comparison). *Let L be a log, given $\mathcal{E}' \subseteq \mathcal{E}$, $\mathcal{E}' \neq \varnothing$, we define the operators $e_1 < e_2$ iff $Time(e_1) < Time(e_2)$ and $e_1 \leqslant e_2$ iff $Time(e_1) \leq Time(e_2)$,*

which reflect the temporal ordering on events, and the operators $min\ \mathcal{E}' = e_1$ *iff* $e_1 \in \mathcal{E}'$ *and for all* $e_2 \in \mathcal{E}', e_1 \leqslant e_2$*, which determines the earliest event of an event set, and* $max\ \mathcal{E}' = e_1$ *iff* $e_1 \in \mathcal{E}'$ *and for all* $e_2 \in \mathcal{E}', e_2 \leqslant e_1$*, which determines the latest event of an event set.*

Useful is the possibility of identifying events belonging to the same work item.

Definition 3 (Work Item Event Grouping). *Let* L *be a log,* e *an event in this log,* $e \in \mathcal{E}$*, and* w *a work item in this log,* $w \in \mathcal{W}$*, we define the set of events that belong to work item* w *as* $\overline{WI}(w) \triangleq \{e \in E \mid WI(e) = w\}$*. Similarly, we define the set of events that belong to the same work item of* e *as* $\overline{WI}(e) \triangleq \overline{WI}(WI(e))$*. Finally, the latest event for work item* w *is defined as* $\omega_w \triangleq max\ \overline{WI}(w)$*.*

As for events we are interested in being able to compare work items.

Definition 4 (Work Item Comparison). *Let* L *be a log, with* $w_1, w_2 \in \mathcal{W}$*, we define* $w_1 < w_2$ *as* $max\ \overline{WI}(w_1) < min\ \overline{WI}(w_2)$*. This operator reflects the partial temporal order between work items, i.e. work item* w_1 *precedes work item* w_2 *if its latest event is earlier than the earliest event of* w_2*.*

An *execution graph* for a process case provides a view of its execution and is defined on the basis of a log and its corresponding process model.

Definition 5 (Execution Graph). *Let* L *be a process log with case* c*,* Y *its YAWL specification, and* $UserID$ *the set of resources, we define the execution graph of* c *as* $G(c) = (Node, NodeTask, Status, \rightsquigarrow, NodeRes, TimeNode, VarNode)$ *where:*

- $Node = \{w \in \mathcal{W} \mid EvType(\omega_w) \in Rel \wedge Case(w) = c\}$ *is the set of nodes, where each node represents a work item that is not modifiable,*
- $NodeTask = Task_{|Node}$ *is the restriction of the function* $Task$ *to the set of nodes,*
- $Status = \{(\omega_w, s) \in Node \times Rel \mid s = EvType(\omega_w)\}$ *is a function relating a node with its status of execution,*
- $\rightsquigarrow = \{(w_1, w_2) \in Node \times Node \mid Status(w_1) \in \{completed, skip\} \wedge NodeTask(w_1) \in \circ NodeTask(w_2) \wedge \nexists w_3 \in Node[(NodeTask(w_1) = NodeTask(w_3) \vee NodeTask(w_2) = NodeTask(w_3)) \wedge w_1 < w_3 \wedge w_3 < w_2])\}$ *is the flow relation between work items. Its reflexive transitive closure is defined as* $\rightsquigarrow^*$*,*
- $NodeRes = \{((w, s), r) \in (Node \times Active) \times 2^{UserID} \mid \exists e_1 \in \overline{WI}(w)[EvType(e_1) = s \wedge r = Res(e_1) \wedge \nexists e_2 \in \overline{WI}(w)[e_1 < e_2 \wedge EvType(e_2) \preccurlyeq s]]\}$ *is a function that yields the resources that are involved in the latest changing* w *to status* s*,*
- $TimeNode = \{((w, s), t) \in (Node \times ActiveC) \times \mathcal{T}) \mid \exists e_1 \in \overline{WI}(w)[EvType(e_1) = s \wedge t = Time(e_1) \wedge \nexists e_2 \in \overline{WI}(w)[e_1 < e_2 \wedge EvType(e_2) \preccurlyeq s]]\}$ *is a partial function that yields the timestamp when* w *latest moved to status* s*,*
- $VarNode = \{((w, x), v) \in (Node \times Var) \times \Omega \mid EvType(\omega_w) \notin \{skip, deoffered\} \wedge v = Inp(max\ \{e_2 \in \overline{WI}(w) \mid EvType(e_2) = offered\}, x)\} \oplus \{((w, x), v) \in (Node \times Var) \times \Omega \mid EvType(\omega_w) \in Completed \wedge v = Outp(\omega_w, x)\}$ *is a partial function relating nodes and variables to values.*

As we explore mitigation options the execution graph should evolve along with it, and the initial execution graph becomes a dynamic data structure from which we can modify nodes. We will refer to this modified execution graph as *mitigation graph*.

The concept of *border* identifies work items that can be modified. Such work items are currently in execution, or they are completed work items for which there are no successor work items that are completed or being executed.

Definition 6 (Border). *Let G be a mitigation graph. We define the border of G, $\circlearrowright_G$, as* $\{n_1 \in Node \mid \forall n_2 \in Node[n_1 \rightsquigarrow^* n_2 \Rightarrow Status(n_2) \in \{deoffered, skipped, rollback\}]\}$.

Definition 7 (Mitigations). *Let Y be a set of YAWL specifications, with associated set of tasks T_Y, a set of resources $UserID$, and a log L. A mitigation is represented as $M = (\mathcal{A}, AcType, AcTask, AcRes, AcCase, \succ)$ where:*

- *$\mathcal{A}$ is a set of mitigation actions,*
- *$AcType : \mathcal{A} \rightarrow \{deoffer, deallocate, destart, offer, allocate, start, rollback, skip\}$, is a function relating actions to types of mitigation,*
- *$AcTask : \mathcal{A} \rightarrow T_Y$ is a function relating actions to tasks,*
- *$AcRes : \mathcal{A} \rightarrowtail UserID$ is a partial function relating actions to resources,*
- *$AcCase : \mathcal{A} \rightarrow \mathcal{C}$ is a function relating actions to cases,*
- *$\succ \subseteq \mathcal{A} \times \mathcal{A}$ is a total ordering on mitigation actions indicating the order in which they need to be performed. We refer to this total ordering as the* mitigation sequence.

The insertion of a new mitigation action $a \notin \mathcal{A}$ into mitigation M, can be expressed as $addMit(M, a, et, t, r, c) \triangleq (\mathcal{A} \cup \{a\}, AcType \cup \{(a, et)\}, AcTask \cup \{(a, t)\}, AcRes \cup \{(a, r)\}, AcCase \cup \{(a, c)\}, \succ \cup \{(x, a) \mid x \in \mathcal{A}\})$.

Now we are in a position to introduce the mitigation actions. For each action we will provide a short description of its behavior; we will quantify its cost and specify the precondition(s) required for its application. All these actions are executed in the context of a mitigation M. As soon as a risk is detected we collect the log L containing all process cases. This log is used to generate an execution graph G', that we refer to as the original execution graph. It is used as a reference for comparison with the original status of the system. The effects of mitigations actions are explored, though not yet applied, during execution of the mitigation algorithm, and hence they are performed on a clone of the original execution graph which we will refer to as G.

Throughout the remainder of this section G' is the original execution graph, G the mitigation graph in use, and $c \in C$ is a case. Moreover, whenever a node is modified, we need to store the time this modification occurred. In order to capture the time, we use function $curr()$.

A mitigation is a sequence of mitigation actions. Below we describe the mitigation actions supported by the PRSA algorithm, and the effects that each action yields. Due to space issues, only some actions are fully furnished in the form of an algorithm in this paper. We refer the reader to the technical report for the algorithms of all mitigation actions [13].

Deoffer. This action deoffers a task from a resource to whom the task was offered. We can execute $deOff(c, G, M)$ as described in Algorithm 1 if there is a work item

$x \in \circlearrowleft_G$ such that x is an *offered* work item. The cost of this action was set to 1 and this action serves as a reference for the cost of the other actions. With reference to our working example, let us assume that for certain process instances an order cannot be updated (e.g. when an order's line items have already gone into production their quantity can no longer be reduced). To prevent such update, we can set a risk condition that is satisfied as soon as a work item of task "Update Shipment Payment Order" is offered to a resource in a specific instance. This risk can be annulled by deoffering this work item and then skipping the work item altogether to prevent it from being reoffered.

Deallocate. This action deallocates a task from the resource to whom the task was allocated. If there is a work item $x \in \circlearrowleft_G$ such that x is an *allocated* work item, we can execute $deAll(c, G, M)$. We set the cost of this action to 2, since considering the progress status of a work item, deallocating a work item should be more "expensive" than deoffering it. In the Payment subprocess this action could be used to mitigate the approval fraud risk. The work item of "Approve Shipment Payment Order" can be deallocated from the resource to whom this work item is allocated when the risk is detected, since this resource approved another order for the same customer in the past.

Destart. This action brings an already started work item back to the state *allocated* and allocates it to the resource who started it. We can execute $deSta(c, G, M)$ if there is a work item $x \in \circlearrowleft_G$ such that x is a *started* work item. For this action we set the cost to 3 as destarting a work item requires more effort than deallocating a work item. The *destart* action may also be used to mitigate an approval fraud risk. For example, it may be used to "free up" a resource who has never approved an order for the current customer, reducing this way the probability of allocating the work item of "Approve Shipment Payment Order" to a resource who has approved another order for the same customer in the past.

Offer. This action offers a work item to a resource to whom the task is not currently offered, either because it is not yet part of the set of resources to whom the task is currently offered, or because the task is currently *deoffered*. Given a function D that relates tasks to the set of resources to whom their work items can be offered, we can execute $off(D, c, G, G', M)$ if there is a work item $x \in \circlearrowleft_G$ such that x is an *offered* or *deoffered* work item, and this work item is an *offered*, *allocated* or *started* work item in the execution graph G'. This action has a cost of 1, the same as *deoffer*. Since this action can only be executed if we previously executed a *deoffer*, these two actions can be combined to "reoffer" a work item to another resource (with a total cost of 2). For example, to reduce the risk of approval fraud in the Payment subprocess, we can deoffer the work item of "Approve Shipment Payment Order" from all Senior Financial Officers that have already approved another order for the same customer in the past, and offer that work item to a Senior Financial Officer that does not satisfy this condition.

Allocate. This action reallocates a work item that was deallocated before (and still has not been allocated) to a resource to whom the task was not allocated when the deallocation took place. We can execute $all(c, G, G', M)$ if there is a work item $x \in \circlearrowleft_G$ such that x is an offered work item, and x is originally an *allocated* or *started* work item. This action has a cost of -1. This action can only be executed if we previously executed a *deallocate*, with the result of changing the resource involved in a work item (so the total cost is $2 - 1 = 1$). We can use the combination *deallocate* + *allocate* as an

Algorithm 1: Deoffer Task

function *deOff(Case c, Mitigation Graph G, Mitigation M)*;
Output: Execution Graph G, Mitigation M
begin
  $n \Leftarrow Any(\{x \in \circlearrowleft_G \mid Status(x) = offered\})$;
  if $n \neq \perp$ **then**
    $r \Leftarrow Any(NodeRes(n, offered))$;
    if $|NodeRes(n, offered)| > 1$ **then**
      $et \Leftarrow offered$;
      $TimeNode \Leftarrow TimeNode \oplus \{((n, offered), curr())\}$;
      $NodeRes \Leftarrow NodeRes \oplus \{((n, offered), NodeRes(n, offered) \setminus \{r\})\}$;
    else
      $et \Leftarrow deoffered$;
      $TimeNode \Leftarrow \{(n, offered)\} \triangleleft TimeNode$;
      $NodeRes \Leftarrow NodeRes(n, offered) \triangleleft NodeRes$;
      $VarNode \Leftarrow \{(n, v) \mid VarNode(n, v) \in \Omega\} \triangleleft TimeNode$;
    $Status \Leftarrow Status \oplus \{(n, et)\}$;
    $M \Leftarrow addMit(M, NewAction(), deoffer, NodeTask(n), r, c)$;
  return (G, M)
end

alternative to mitigate the risk of approval fraud. With a total cost of 1, this combination would be preferred to using *deoffer* + *offer*.

Start. This action restarts a work item that was previously destarted (and has not yet been restarted) and associates it with a different resource from the one who started the task. We can execute $sta(c, G, G', M)$ if there is a work item $x \in \circlearrowleft_G$ such that x is an *allocated* work item, and x is originally a *started* work item. The cost of this action is -1, and the reasoning is similar to that used for the *allocate* action. This mitigation action can be used to reduce the negative impact of a *deoffer* or *deallocate* previously performed on a process instance.

Rollback. This action returns a completed work item to the status of unoffered. We can execute $rollbackTask(c, G)$ if there is a work item $x \in \circlearrowleft_G$ such that x is a *completed* work item. Its operationalization is described in Algorithm 2. The rollback action restores the case to a consistent status where the execution of a given work item never happened. A compensation routine can be associated with a task, so that it is triggered when the task is rolled back. The idea of this compensation routine is to deal with elements outside the control of the workflow engine (e.g. returning the money to a client after their payment has been rolled back). The rollback action is our most powerful action and has a cost of 9, obtained by adding the absolute values of all the actions introduced until now. In our Payment subprocess, we can use this action when we execute a large number of updates on the same Payment Order.

Skip. This action marks an unoffered and skippable task as 'to be skipped'. If there exists a task $t \in skippable$ which does not have any work item active or completed, and there not exists a mitigation action $a \in \mathcal{A}$ which skipped task t for case c, then we define $skipTask(c, G)$. To limit the use of this action, since this action may produce

Algorithm 2: Rollback Task

function *rollbackTask(Case c, Execution Graph G, Mitigation M)*;
Output: Execution Graph G, Mitigation M
begin
  $n_1 \Leftarrow Any(\{x \in \circlearrowleft_G \mid Status_G(x) \in Completed\})$;
  if $n_1 \neq \perp$ **then**
    foreach $n_2 \in Node_G$ **do**
      if $n_1 \rightsquigarrow^*_G n_2$ **then** $Status_G \Leftarrow Status_G \oplus \{(n_2, rollback)\}$;
    $Status_G \Leftarrow Status_G \oplus \{(n_1, rollback)\}$;
    $M \Leftarrow addMit(M, NewAction(), rollback, NodeTask_G(n_1), \perp, c)$;
  return (G, M)
end

inconsistency in the data, we decided to assign a cost of 9. The utility of this action can be seen in two situations when we consider our running example. The first situation is the order unfulfillment. In this case, to prevent the reiterated execution of an update, we may decide to skip the "Update Shipment Payment Order" task. The second situation is the overtime process risk. In this case we may decide to skip some tasks in order to complete the process in time.

Relocate Resource. This action looks for a resource that is only involved in the execution of a work item belonging to a case for which no risk was defined. If once such a resource is found, it deallocates (and destarts if necessary) the work item associated with this resource and allocates the resource to a work item of the process case that we want to mitigate. The cost of this action is 7 since this action performs a (partial) sequence of destart and deallocate on two work items, and another allocate and a start action on one work item. Let x be an active border work item $x \in \circlearrowleft_G$ in case c, r be a resource involved only in case c_2, and c_2 be process which is not risky. If resource r only started or allocated one work item (of any active border events), then we can execute $relRes(c, G, M, SRC)$.

4 Evaluation

We implemented the PRSA algorithm as a custom service in the YAWL system.[2] We extended the YAWL system as it is built on a service-oriented architecture, which facilitates the addition of new services; it is open-source, which facilitates its distribution among academics and practitioners; and as the underlying YAWL language provides comprehensive supports for the workflow patterns [19].

The risk mitigation service interacts with the *risk detection service*[3] that we developed previously [12], for the sake of identifying risks and computing their probabilities. It uses as input a reference to the process instance whose risks need to be mitigated, the complete YAWL specification for this instance, a log of the process (as extracted from the YAWL system), and a copy of the risk sensors associated with the process instance, as provided by the risk detection service. Modifications that a mitigation may introduce

[2] `http://www.yawlfoundation.org`
[3] `http://www.yawlfoundation.org/prsa`

are communicated to the risk detection service, which recomputes the risk probabilities. The final solutions are returned to the user as recommendations. The one chosen by the user is then applied to the process instance under exam using the APIs provided by the YAWL engine. We implemented compensation actions associated with rolled back work items via the YAWL Worklet mechanism [19]. Accordingly, we equipped the YAWL Editor with an interface to allow users to associate a Worklet containing a compensation action to a task. When an instance of this task is rolled back, the associated Worklet is run as a separate process instance in the YAWL engine, so that from an engine perspective, the Worklet and its invoking processes are two distinct cases.

To prove the feasibility of our approach, we ran three experiments. First, we tested the required time to mitigate the same set of risks on different process models. Second, we checked the dependency of the mitigation time on different variables. Third, we checked the quality of the mitigations proposed on a specific process model.

For the first experiment, we used four real-life process models available to the research team, for which we could identify risk conditions. The sizes of these models range from 5 to 20 tasks. The first model (Process A) describes a film production process, carried out on a daily basis. This process is taken from a case study we conducted in collaboration with the Australian Film, Television and Radio School [25]. The other three models are subprocesses of an order fulfillment process inspired by the VICS industry standard for logistics [34]. The first one (Process B) deals with the ordering, the second (Process C) deals with the payment for the goods and is the process we showed in Section 2, the third model (Process D) deals with the delivery of the goods.

Next, we defined seven generic risk conditions that are applicable to all these processes, where with "generic" we mean risks that are not linked to a specific context, such as a financial frauds, but that are linked to control-flow aspects. These conditions represent possible undesirable situations that may arise in a process, and relate to different process aspects such as data, resources and control-flow elements. They are domain-independent so that we could define them on all four process models. The first condition detects a situation where two concurrent work items may not complete in a desired order. The second one is used to detect a violation of the four-eyes principle between parallel work items. The third one detects whether a time limit is exceed when executing a loop. The fourth condition detects a possible delay with the execution of a work item. The fifth one detects the possibility that two concurrent work items that should be executed by the same resource are actually allocated to two different resources (a situation that is not possible to enforce with many workflow management systems). The sixth one detects a delay with the execution of a portion of the process while the seventh one detects a data error, specifically if the data values produced by two concurrent work items are not the same.

For each process model we generated a variant with a specific combination of the above risk conditions. This led to a total of 180 process models (not every risk identified could be applied to every process model, since some of these risk conditions require parallelism and/or loops that are not present in every process model). These process models are as follows: 19 models with 1 risk condition, 40 models with 2 risk conditions, 50 with 3 risk conditions, 41 models with 4 risk conditions,

22 process models with five risk conditions, seven process models with six risk conditions, and one process model with all seven risk conditions.

Table 1. Time and number of candidate solutions explored to find the first solution

Process	Size	Variants	Risks	Mitigation time [sec]			Candidates		
			avg/max	min	max	avg	min	max	avg
Process A	20	127	3.53 / 7	0.003	178.891	26.415	2	20,181	3,456
Process B	5	7	1.71 / 3	0.001	0.033	0.015	3	54	32
Process C	15	31	3.05 / 5	0.001	0.117	0.030	2	256	60.93
Process D	5	15	2.13 / 4	0.004	0.929	0.170	2	553	78.2
Total	45	180	3.18 / 7	0.001	178.891	18.657	2	20,181	2,457

For each process model we ran ten tests and averaged the results. Each test was executed on the first state of a process instance where all the risk conditions evaluated to true. For each group of tests on the same process model we measured the time required to obtain the first solution that mitigates all risks, and the number of candidate solutions generated by the algorithm in order to obtain this solution. We performed the tests on an Intel Core I5 M560 2.67GHz processor with 4GB RAM, running Linux Lubuntu v11.10.

Table 1 shows the results of this experiment. The second, third and fourth columns show the size (as number of tasks), the number of variants and the number of risk conditions for each of the four process models. The fifth and sixth columns show the mitigation time required to find the first solution, and the number of candidate solutions explored to find such a solution. From this table we can observe that the algorithm takes at most 3 mins (179 secs) to mitigate multiple risks in a variant of Process A (this timing refers to a combination of 5 risks for this process), though the average time is much lower (19 secs across all models). It seems reasonable to assume that in most business scenarios mitigation times in the order of a few minutes are acceptable, compared to the average time required to perform a task, and thus the average duration of a process instance. For example, let us assume an average duration of 24 hours for the Payment subprocess, with a new task being executed every 30 mins. Let us also assume that we sample the risk conditions every 5 mins. This means we have up to 6 mins to mitigate all identified risks before a new task is executed which may change the risk conditions.

Table 1 also shows that the algorithm needs to explore a very large number of candidate solutions to find the first solution (2,456 solutions on average across all models). While it is not fair to compare the computation power of a machine to that of humans, this result highlights the complexity of finding a solution. It is reasonable to think that many of these candidate solutions explored by the algorithm would also need be evaluated by a human in order to find the right solution.

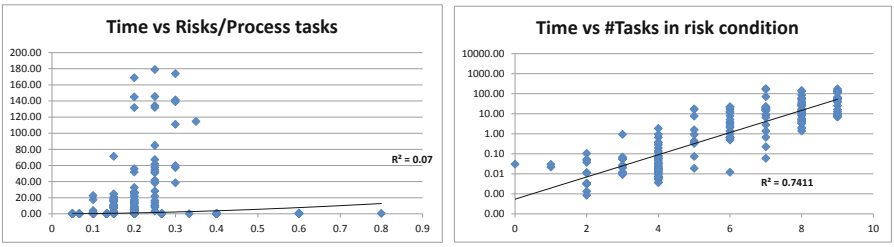

Fig. 3. Correlation between time and a) risks/tasks ratio, b) tasks in risk conditions

In the second experiment, we investigated the factors affecting the performance of the algorithm. One would think that the mitigation time is proportional to the number of risks defined in a process model, and to the model size itself. The larger the number of risks and/or the model size, the longer it should take to mitigate such risks. However the data we extrapolated from Table 1 does not confirm this hypothesis. For example, the 21 variants of Process A with 5 risks have mitigation times ranging from 3.3 to 179 secs, despite their sizes and number of risks being the same. To verify that the mitigation time is not sensitive to the number of risks, nor to the process size, we plotted the correlation between the mitigation time and the ratio risks/process size in Figure 3a (the solid line is the linear regression of the points). The low value of the coefficient of determination R^2 (0.07) confirms this intuition. We then checked the correlation between the mitigation time and the number of tasks used in risk conditions. The intuition is that the more work items of these tasks are pending in a given state of the process instance, the larger the number of possible mitigation actions. The corresponding scatter plot is shown in Figure 3b, which indeed confirms this intuition ($R^2 = 0.74$).

Finally, we checked the feasibility of the solutions proposed by the algorithm, when mitigating the domain-specific risks associated with the Payment subprocess (cf. Section 2). We recall that two of these risks (overtime process and order unfulfillment) are detected when the associated probability, obtained by analyzing historical data, exceeds a tolerance threshold, whereas the third risk (approval fraud) involves a complex risk condition. We considered the first state of an instance of the Payment subprocess when all three risks are active. This occurs after executing "Update shipment payment order" for the third time, once task "Approve shipment payment order" has been allocated to a resource who has already executed this task in the past.

To obtain a small number of solutions, we stopped the algorithm after one min of execution. In this timeframe, five solutions were retrieved. For each solution, Table 2 reports whether the solution mitigates each of the three risks, and the cost of the solution in terms of mitigation actions performed on the initial process instance. In particular, a "$-$" indicates a risk not mitigated, a "$+$" indicates a risk mitigated (with risk probability lower than the specific threshold if the condition depends on the risk probability), and a "$\pm$" indicates a risk mitigated whose condition cannot be computed for lack of information, i.e. some of the variables used in the risk condition are null. We recall that the algorithm prioritizes a solution whose risk is mitigated by computing the risk condition, than a solution whose risk is mitigated because the respective condition cannot be computed.

Table 2. Payment subprocess mitigation

Solutions [at 1 min]	**1**	**2**	**3**	**4**	**5**
Overtime Process	$+$	$+$	$+$	$+$	$+$
Approval Fraud	$+$	$+$	$+$	$+$	$+$
Order Unfulfillment	$+$	$+$	$\pm$	$\pm$	$-$
Cost	50	50	40	40	19

The five solutions identified are pairwise mutually non-dominating. Solutions 1 and 2 are dominated by solutions 3, 4 and 5 cost-wise, but dominate these solutions w.r.t. the mitigation of the order unfulfillment risk. Solution 5 dominates solutions 3 and 4 cost-wise but is dominated by these two solutions w.r.t. the mitigation of the order unfulfillment risk.

Let us briefly examine the mitigations performed by the five solutions. The first four solutions mitigate the approval fraud by deallocating the resource that was allocated

"Approve shipment payment order", while solution 5 additionally allocates the work item to a resource who did not execute this task for the same customer in the past. All these mitigations are feasible, though the one provided by solution 5 is more robust, since there is no risk that the task gets allocated to a resource who has already executed it. The order unfulfillment risk is mitigated by solutions 1 and 2 through rolling back the work item of task "Update shipment payment order" (which leads to a deoffer of the work item of task "Approve shipment payment order" that comes afterwards). Solutions 3 and 4 do this too but also mark this task 'to be skipped' preventing a possible re-execution of it. This action sets to null the risk variables associated with this task that retrieve the number of executions and its estimated remaining time making the risk mitigated but not computable. Thus, while all four solutions are feasible, we would prioritise the first two since these ensure that the risk probability has actually dropped below the threshold. Finally, all solutions differ in the way they mitigate the overtime process risk. Each of them skips a different task among those not yet executed (for simplicity, all of them have the same estimated duration). Despite the fact that all these solutions are feasible, only the mitigation proposed by solution 3 is interesting since it proposes to skip tasks "Update Shipment Payment Order" and "Approve Shipment Payment Order" avoiding this way that the loop is taken again. In other words, it prevents the order to undergo further updates, and subsequent approvals.

5 Related Work

Risk mitigation is an essential step in the risk management process [29]. Several risk analysis methods such as OCTAVE [4], CRAMM [6] and CORAS [22] describe guidelines for identifying risk mitigations. For example, these guidelines provide instructions on how to conduct structured brainstorming sessions with experts in order to identify viable mitigation procedures. Although helpful, these guidelines are too generic and no support is offered on how mitigation procedures could be operationalized. Similarly, the academic literature recognizes the importance of mitigating process-related risks, though it focuses on risk-aware BPM methodologies in general, rather than on concrete algorithms for automating risk mitigation [24,20,14,30,27,8,33]. For example, the ROPE (Risk-Oriented Process Evaluation) methodology [33] is concerned with threats to the resources required for process execution. If a required resource becomes unavailable, pre-planned countermeasures and recovery procedures are *manually* enacted to handle the fault. These procedures are defined and validated via a simulator at design-time; enactment at runtime is designated to a 'responsible person', that is, the mitigation and recovery operations are not automated. For a comprehensive survey of these risk-aware BPM methodologies, we refer to [31].

The mitigation actions proposed in this paper share commonalities with the workflow exception patterns [26]. For example, the *reoffer* pattern at the work item level can be obtained by combining our mitigation actions *deoffer* + *offer*, which represent atomic operations in this respect. Another similarity is between our *rollback* action and the recovery patterns *rollback* and *compensate*, since our rollback can also trigger a compensation action if this is available for the task being rolled back. Given that we need to operationalize these actions, we do provide a precise characterization of all actions and their effects, whereas the work in [26] limits itself to a textual description

of these exception patterns, as they are observed from an analysis of process modeling languages and tools. That said, the main contribution of our paper is not the proposed set of mitigation actions per se (which could be replaced or modified) but rather an automated mechanism for combining these actions in an optimal way in order to mitigate a set of process-related risks as far as possible.

Risks like those we illustrated in this paper can also be encoded as *constraints* on top of a process model. Approaches exist that can check whether there exists at least an instance of a constrained process model that can be executed without violating the constraints. For example, Combi and Posenato [10] propose a general method for checking the satisfiability of temporal process constraints (like our overtime process risk) at design-time. Other approaches can also enforce these constraints at run-time, so that any process instance will satisfy all the constraints by construction. For example, Tan et al. [32] propose a model for constrained workflow execution that addresses cardinality constraints (e.g. to control how many times a task can be executed), binding of duty constraints (i.e. the ability of a resource to retain a familiar work item) and separation of duty constraints (i.e. different resources should execute different tasks). These constraints can be enforced only within a given process instance. The approach proposed by Warner and Atluri [35] overcomes this limitation by proposing a constraint specification language for resource allocation that also addresses inter-instance constraints. Compared to our work, constrained workflow execution provides a rigid approach according to which if the constraints cannot be satisfied, a process model will simply not be instantiated, or if instantiated, the instance violating the constraints will throw an exception. Commercial systems such as IBM WebSphere and AristaFlow also support constrained workflow execution, and handle these violations (e.g. violations of temporal constraints) by escalating control to a process administrator.[4] Instead, our approach allows a process instance to be automatically *adapted* on-the-fly in order to reduce the risk of violating such constraints.

Various research frameworks have been proposed for the *dynamic adaptation* of process instances. For example, ADEPT [15] supports adding, deleting and changing the sequence of tasks at both the model and instance levels, however such changes must be achieved via manual intervention by an administrator. AgentWork [23] provides the ability to modify process instances by dropping and adding individual tasks based on events and rules. CBRFlow [36] uses case-based reasoning to support runtime adaptation by allowing users to annotate rules during process execution. CEVICHE [18] is a service-based framework that uses the AO4BPEL (Aspect-Oriented for BPEL) language [9] to provide an option for skipping or reallocating tasks to other services in an ad-hoc manner. While these approaches could be used for risk mitigation purposes, they do not provide any help for the identification of which particular mitigation actions should be used. The YAWL Worklet Service [3] provides each task of a process instance with the ability to be associated with an extensible repertoire of actions ('drop-in' processes), one of which is contextually and dynamically bound to the task at runtime. It also supports capabilities for dynamically detecting and handling runtime exceptions.

[4] `http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/index.jsp?topic=/com.ibm.wbit.612.help.tel.ui.doc/topics/tescal8.html`

However the approach is generic and not specifically designed for risk detection and mitigation. Also a new situation cannot automatically be dealt with but requires a workflow administrator to intervene.

Our work is also related to *operational support* in process mining [1]. Operational support deals with the analysis of current and historical execution data, with the aim to predict future states of a running process instance, and provide recommendations to guide the user in selecting the next activity to execute based on certain objectives. For example, the approach for cycle time prediction in [2] could be, with the opportune modifications, adapted for risk prediction. Using this approach it would be possible to estimate the probability of an overtime risk and suggest the next steps the current instance should take in order to keep this risk under control. The application of this approach unfortunately requires that the process model captures all the possible mitigation actions as normal activities, i.e. as control-flow alternatives. For instance, if a task can be skipped, there should be a path without that task that leads to the end node of the process model. This may drastically increase the complexity of the process model. Moreover, this approach would not be applicable to capture mitigation operations on resources (i.e. deallocating a resource) or on task states (e.g. suspending a task). That said, more in general, our approach could be seen as a possible provider for operation support, and could thus be integrated in process mining environments like ProM.[5]

Our work provides recommendations to users as to which mitigation actions can be applied to the specific context at hand. As such, it shares commonalities with recommendation and decision support systems. Alter [5] states that the focus of such systems should be towards improving decision making within work systems, rather than externalizing support. This view is shared by our technique, which provides an extension to existing process-aware information systems, rather than a separate standalone tool. As such, it may be considered a member of the domain known as *Group Decision Support Systems*, which facilitate task support in group environments.

In previous work [17] we explored the use of dominance-based MOSA for automatically fixing behavioral errors in process models, at design-time. Our work on risk mitigation can thus be seen as an adaptation of that idea to run-time aspects, since we aim to improve running process instances. Besides their distinct aims, the main difference between the two approaches is that for correcting behavioral errors we defined three objective functions capturing the structural and behavioral similarity of a solution to the incorrect model, whereas in risk mitigation the number and type of objective functions depends on which risks are active in a given state of a process instance.

6 Conclusion

This paper contributes a concrete technique for the automatic mitigation of process-related risks at run-time. The technique requires as input an executable process model and a set of associated risk conditions. At run-time, when one or more risk conditions evaluate to true, a process administrator can launch our technique to mitigate the identified risks and bring the process instance back to a safe state. This is achieved by generating a set of possible mitigations that change the current instance in order to bring

[5] `http://processmining.org`

the likelihood of the identified risks below a tolerance level. These mitigation actions are not performed directly on the instance under consideration. Rather, their effects are simulated and those solutions that mitigate the most risks in a given timeframe, are proposed as recommendations to the process administrator.

The mitigation actions are determined via a dominance-based MOSA algorithm. This choice allows us to explore the solution space as widely as possible, avoiding local optima. In essence, each risk is treated as an objective function whose likelihood needs be minimized. The objective is reached as soon as the likelihood goes below the tolerance value for that particular risk. Mitigation actions affect various aspects of a process, such as task execution and resources utilization. To the best of our knowledge, this is the first time that process-related risks can be mitigated automatically.

The technique was implemented in the YAWL system and its performance evaluated with real-life process models. The tests show that on the analyzed process models a set of possible solutions can be found in a matter of seconds, or within a few minutes in the worst case, and that in all cases the associated risks are mitigated. We expect this technique to reduce the effort and time required by process administrators to understand what mitigation actions are feasible based on a particular state of the system. That said, we still need to validate the feasibility and appropriateness of the proposed mitigation actions with domain experts. We plan to do so by comparing the solutions obtained with our algorithm with those proposed by them. We also plan to improve the exploration of the solution space by prioritizing the mitigation of those risks that have the highest impact on the process objectives. In fact, currently all risks are treated alike whereas in reality this might not be the case. Finally, the algorithm could also be extended to prioritize certain mitigation actions based on how these have been ranked by the users in previously mitigated instances.

Acknowledgments We thank Peter Hughes and Wil van der Aalst for their valuable comments and suggestions. This research is funded by the ARC Discovery Project "Risk-aware Business Process Management" (DP110100091) and by the NICTA Queensland Lab.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Information Systems 36(2), 450–475 (2011)
3. Adams, M., ter Hofstede, A.H.M., van der Aalst, W.M.P., Edmond, D.: Dynamic, Extensible and Context-Aware Exception Handling for Workflows. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 95–112. Springer, Heidelberg (2007)
4. Alberts, C.J., Dorofee, A.J.: OCTAVE criteria, version 2.0. Technical Report CMU/SEI-2001-TR-016, Carnegie Mellon University (2001)
5. Alter, S.: A work system view of DSS in its fourth decade. In: DSS, vol. 38 (December 2004)
6. Barber, B., Davey, J.: The use of the CCTA Risk Analysis and Management Methodology CRAMM in health information systems. In: MEDINFO. North Holland Publishing (1992)
7. Basel Committee on Bankin Supervision. Basel II: International Convergence of Capital Measurement and Capital Standards (2006)

8. Betz, S., Hickl, S., Oberweis, A.: Risk-aware business process modeling and simulation using xml nets. In: IEEE CEC, pp. 349–356 (September 2011)
9. Charfi, A., Mezini, M.: AO4BPEL: An aspect-oriented extension to BPEL. In: WWW (2007)
10. Combi, C., Posenato, R.: Controllability in Temporal Conceptual Workflow Schemata. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 64–79. Springer, Heidelberg (2009)
11. International Electrotechnical Commission. IEC 61025 Fault Tree Analysis, FTA (1990)
12. Conforti, R., Fortino, G., La Rosa, M., ter Hofstede, A.H.M.: History-Aware, Real-Time Risk Detection in Business Processes. In: Meersman, R., Dillon, T., Herrero, P., Kumar, A., Reichert, M., Qing, L., Ooi, B.-C., Damiani, E., Schmidt, D.C., White, J., Hauswirth, M., Hitzler, P., Mohania, M. (eds.) OTM 2011, Part I. LNCS, vol. 7044, pp. 100–118. Springer, Heidelberg (2011)
13. Conforti, R., ter Hofstede, A.H.M., La Rosa, M., Adams, M.J.: Automated risk mitigation in business processes (extended version). QUT ePrints 49331 (2012)
14. Cope, E.W., Kuster, J.M., Etzweiler, D., Deleris, L.A., Ray, B.: Incorporating risk into business process models. IBM Journal of Research and Development 54(3), 4:1–4:13 (2010)
15. Dadam, P., Reichert, M.: The ADEPT project: a decade of research and development for robust and flexible process support. CSRD 23, 81–97 (2009)
16. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Process-Aware Information Systems: Bridging People and Software through Process Technology. Wiley & Sons (2005)
17. Gambini, M., La Rosa, M., Migliorini, S., Ter Hofstede, A.H.M.: Automated Error Correction of Business Process Models. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 148–165. Springer, Heidelberg (2011)
18. Hermosillo, G., Seinturier, L., Duchien, L.: Using complex event processing for dynamic business process adaptation. In: SCC, pp. 466–473. IEEE (2010)
19. ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M., Russell, N. (eds.): Modern Business Process Automation: YAWL and its Support Environment. Springer (2010)
20. Jallow, A.K., Majeed, B., Vergidis, K., Tiwari, A., Roy, R.: Operational risk analysis in business processes. BTTJ 25(1), 168–177 (2007)
21. Johnson, W.G.: MORT: The Management Oversight and Risk Tree. U.S. Atomic Energy Commission (1973)
22. Lund, M.S., Solhaug, B., Stølen, K.: Model-Driven Risk Analysis: The CORAS Approach. Springer (2011)
23. Muller, R., Greiner, U., Rahm, E.: AgentWork: a workflow system supporting rule-based workflow adaptation. Data & Knowledge Engineering 51(2), 223–256 (2004)
24. Neiger, D., Churilov, L., zur Muehlen, M., Rosemann, M.: Integrating risks in business process models with value focused process engineering. In: ECIS. AISeL (2006)
25. Ouyang, C., La Rosa, M., ter Hofstede, A.H.M., Dumas, M., Shortland, K.: Toward web-scale workflows for film production. IEEE, Internet Computing 12(5), 53–61 (2008)
26. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Workflow Exception Patterns. In: Martinez, F.H., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 288–302. Springer, Heidelberg (2006)
27. Sienou, A., Lamine, E., Pingaud, H., Karduck, A.P.: Risk driven process engineering in digital ecosystems: Modelling risk. In: Proc. of IEEE DEST, pp. 647–650 (2010)
28. Smith, K.I., Everson, R.M., Fieldsend, J.E., Murphy, C., Misra, R.: Dominance-based multiobjective simulated annealing. IEEE TEC 12(3), 323–342 (2008)
29. Standards Australia and Standards New Zealand. Standard AS/NZS ISO 31000 (2009)
30. Strecker, S., Heise, D., Frank, U.: RiskM: A multi-perspective modeling method for IT risk assessment. Information Systems Frontiers, 1–17 (2010)

31. Suriadi, S., Weiß, B., Winkelmann, A., ter Hofstede, A., Wynn, M., Ouyang, C., Adams, M.J., Conforti, R., Fidge, C., La Rosa, M., Pika, A.: Current research in risk-aware business process management - overview, comparison, and gap analysis. QUT ePrints 50606 (2012)
32. Tan, K., Crampton, J., Gunter, C.A.: The consistency of task-based authorization constraints in workflow. In: Proc. of IEEE CSFW, pp. 155–169 (June 2004)
33. Tjoa, S., Jakoubi, S., Goluch, G., Kitzler, G., Goluch, S., Quirchmayr, G.: A formal approach enabling risk-aware business process modeling and simulation. IEEE TSC 4(2) (2011)
34. Voluntary Interindustry Commerce Solutions Association. Voluntary Inter-industry Commerce Standard (VICS), `http://www.vics.org` (accessed: June 2011)
35. Warner, J., Atluri, V.: Inter-instance authorization constraints for secure workflow management. In: Proc. of SACMAT, pp. 190–199. ACM, New York (2006)
36. Weber, B., Wild, W., Feige, U.: CBRFlow: Enabling Adaptive Workflow Management Through Conversational Case-Based Reasoning. In: Funk, P., González Calero, P.A. (eds.) ECCBR 2004. LNCS (LNAI), vol. 3155, pp. 434–448. Springer, Heidelberg (2004)