A Recommendation System for Predicting Risks across Multiple Business Process Instances

Raffaele Conforti[a], Massimiliano de Leoni[c,b], Marcello La Rosa[a,d],
Wil M. P. van der Aalst[b,a], Arthur H. M. ter Hofstede[a,b]

[a] *Queensland University of Technology, Australia*
{*raffaele.conforti,m.larosa,a.terhofstede*}*@qut.edu.au*
[b] *Eindhoven University of Technology, The Netherlands, Australia*
{*m.d.leoni,w.m.p.v.d.aalst*}*@tue.nl*
[c] *University of Padua, Italy*
[d] *NICTA Queensland Lab, Brisbane, Australia*

Abstract

This paper proposes a recommendation system that supports process participants in taking risk-informed decisions, with the goal of reducing risks that may arise during process execution. Risk reduction involves decreasing the likelihood and severity of a process fault from occurring. Given a business 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 suggest the participant the action to perform which minimizes the predicted process risk. Risks are predicted by traversing decision trees generated from the logs of past process executions, which consider process data, involved resources, task durations and other information elements like task frequencies. When applied in the context of multiple process instances running concurrently, a second technique is employed that uses integer linear programming to compute the optimal assignment of resources to tasks to be performed, in order to deal with the interplay between risks relative to different instances. The recommendation system has been implemented as a set of components on top of the YAWL BPM system and its effectiveness has been evaluated using a real-life scenario, in collaboration with risk analysts of a large insurance company. The results, based on a simulation of the real-life scenario and its comparison with the event data provided by the company, show that the process instances executed concurrently complete with significantly fewer faults and with lower fault severities, when the recommendations provided by our recommendation system are taken into account.

Keywords: business process management, risk management, risk prediction, job scheduling, work distribution, YAWL.

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 [1]. Failing to address process-related risks can result in substan-

tial 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 fraud.

Legislative initiatives like Basel II [2] and the Sarbanes-Oxley Act[1] 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), in [3, 4, 5] we proposed several techniques 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 mitigation action may entail skipping some tasks when the process instance is very likely 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 [4]. For example, rolling back a task for the sake of mitigating a risk of cost overrun, may not allow the full recovery of the costs incurred in the execution of that task.

To address these limitations we propose a recommendation system that supports process participants in taking risk-informed decisions, with the aim to *reduce* process risks preemptively. A process participant takes 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 decision trees which are trained using historical process data such as process variables, resources, task durations and frequencies. The historical data of a process is observed using decision trees which are built from the execution logs of the process, as recorded by the IT systems of an organization.

This way, the participant can take 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 potential 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.

The above technique only provides "local" risk predictions, i.e. predictions relative to a specific process

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

instance. In reality, however, multiple instances of (different) business processes may be executed at any time. Thus, we need to find a risk prediction for a specific process instance that does not affect the prediction for other instances. The interplay between risks relative to different instances can be caused by the sharing of the same pool of process participants: two instances may require the same scarce resource. In this setting, a sub-optimal distribution of process participants to the set of tasks to be executed, may result in a risk increase (e.g. overtime or cost overrun risk). To solve this problem, we equipped our recommendation system with a second technique, based on integer linear programming, which takes input from the risk prediction technique, to find an *optimal distribution* of process participants to tasks. By optimal distribution we mean one that minimizes the overall execution time (i.e. the time taken to complete *all* running instances) while minimizing the overall level of risk. This distribution is used by the recommendation system to suggest process participants the next task to perform.

We operationalized our recommendation system on top of the YAWL BPM system by extending an existing YAWL plug-in and by implementing two new custom YAWL services. This implementation prompts process participants with risk predictions upon filling out a form or for each task that can be executed. We then evaluated the effectiveness of our recommendation system by conducting experiments using a claims handling process in use at a large insurance company. With input from a team of risk analysts from the company, this process has been extensively simulated on the basis of a log recording one year of completed instances of this process. The recommendations provided by our recommendation system significantly reduced the number and severity of faults in a simulation of a real life scenario, compared to the process executed by the company as reflected by the event data. Further, the results show that it is feasible to predict risks across multiple process instances without impacting on the execution performance of the BPM system.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 contextualizes the recommendation system within our approach for managing process-related risks, while Section 4 presents the YAWL language as part of a running example. Next, Section 5 defines the notions of event logs and faults which are required to explain our techniques. Section 6 describes the technique for predicting risks in a single process instance while Section 7 extends this technique to the realm of multiple process instances running concurrently. Section 8 and Section 9 discuss the implementation and evaluation of the recommendation system, respectively. Finally, Section 10 concludes the paper. The Appendix provides the formal definition of a YAWL specification, the algorithms to generate a prediction function, and technical proofs of two lemmas presented in Section 7.

2. Related Work

The approach presented in this paper is related to work on risk prediction, job scheduling, operational support and work-item distribution for business processes. In this section we review the state of the art in these fields to motivate the need for our approach.

2.1. Risk Prediction

Various risk analysis methods such as OCTAVE [6], CRAMM [7] and CORAS [8] 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 mitigation, while academic efforts mostly focus on risk-aware BPM methodologies in general, rather than on concrete approaches for risk prediction [9].

An exception is made by the works of Pika et al. [10] and Suriadi et al. [11]. 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. These two approaches suffer from the limitation of not considering the data prospective. Further, they limit their scope to the identification of indicators of risks or of causes of faults to support overtime risks only.

In previous work, we presented a wider approach which aims to bridge the gap between risk and process management. This approach consists of two techniques. The first one [3, 5] allows process modelers to specify process-related faults and related risks on top of (executable) process models, and to detect them at run-time when their risk likelihood exceeds a tolerance threshold. Risks are specified as conditions over control-flow, resources and data aspects of the process model. The second technique [4] builds on top of the first one to cover risk mitigation. As soon as one or more risks are detected which are no longer tolerable, the technique proposes a set of alternative mitigation actions that can be applied by process administrators. A mitigation action is a sequence of controlled changes on a process instance affected by risks, which takes into account a snapshot of the process resources and data, and the current status of the system in which the process is executed.

For a comprehensive review and comparative analysis of work at the intersection of risk management and BPM, we refer to [9].

2.2. Job Scheduling

The problem of distributing work items to resources in business process execution shares several similarities with the job-shop scheduling [12, 13, 14, 15]. Job-shop scheduling concerns M jobs that needs to be assigned to a N machines, with $N < M$, while trying to minimize the make-span, i.e. the total length of the schedule. Jobs may have constraints, e.g. job i needs to finish before job j can be started, certain jobs can only be performed by given machines.

Unfortunately, these approaches are intended for different settings and cannot be specialized for risk-informed work-item assignment. To our knowledge, techniques of job-shop scheduling are unaware of the concept of cases or process instances, since typically jobs are not associated with a case.

The concept of case is crucial when dealing with process-aware information systems. Work items are executed within process instances and many process instances can be running at the same time, like so many work items may be enabled for execution at the same time. Different instances may be worked on by the same resources and, hence, the allocation within a instances may affect the performance of other instances. Without considering the instances in which work items are executed, an important aspect is not considered and, hence, the overall allocation is not really optimized. Moreover, applying job-scheduling for work-item distribution, such work items will be distributed with a push method, i.e. a work item is pushed to a single qualifying resource. This is also related to the fact the jobs are usually assumed to be executed by machines, whereas, in process-aware information systems, work items are normally being executed by human resources. Work items may also be executed by automatic software services, but this is not the situation in the majority of setting. In [16], it is shown that push strategies already perform very poorly when the resource work-load is moderately high. Therefore, work items ought to be distributed with a pull mechanism, i.e. enabled work items are put in a common pool and offered to qualifying resources, which can freely pick any of them. As a matter of fact, a pull metho is far the most common used in current-day process-aware information systems.

2.3. Operational Support

The work proposed in this paper is also related to body of work that is concerned with devising frameworks and architectures to provide operational support for business processes as a service. For instance, Nakatumba et al. [17] propose a service for operational support which generalizes what is proposed in [18]. This service is implemented in ProM, a pluggable framework to implement process-aware techniques in a standardized environment. On its own, the service does not implement recommendation algorithms but provides an architecture where such algorithms can be easily plugged in. For instance, the prediction technique in [19] is an example of algorithm plugged into this architecture (more details on this work are provided in the next subsection). Another example is the work in [20], which concerns a recommendation algorithm

Table 1: Comparison of different approaches for operational support in Process-aware Information Systems

Approach	Weight	Process Perspectives Computation	Optimal Distribution	Objective	Assignment Method
Kim et al. [21]	Dynamic	Control-flow, Resource	-	Time, Cost	-
Yang [22]	Static	-	Instance level	Customizable	PUSH
Kumar et al. [23]	Dynamic	Control-flow, Resource	Instance level	Cooperation[a]	PUSH
Kumar et al. [16]	Static	-	Instance level	Suitability, Urgency, Workload	PUSH/PULL[b]
Huang et al. [24]	Dynamic	Control-flow, Resource, Data, Time	Instance level	Customizable	PUSH
van der Aalst et al. [25]	Dynamic	Control-flow	-	Time	-
Folino et al. [26]	Dynamic	Control-flow, Resource, Data, Time	-	Time	-
van der Spoel et al. [27]	Dynamic	Control-flow	-	Cost	-
Cabanillas et al. [28]	Static	Control-flow, Resource	Process level	User preference[c]	PUSH
Barba et al. [29]	Static	Control-flow, Resource	Instance level	Time	PULL
Maggi et al. [19]	Dynamic	Control-flow, Resource, Data	-	Customizable LTL formulas[d]	-

[a] Work items are distributed to maximize the quality of the cooperation among resources. This approach assumes that some resources can cooperate better than others when working on a process instance.

[b] Resources declare their interest in picking some work items for performance. The approach assigns each work item to the interested resource that guarantees the better distribution.

[c] At design time, users provide preferences for work items. At run time, the system allocates work items to resources to maximize such preferences.

[d] The expressiveness power of business goals in the form of a single LTL formula is lower than what our approach allows for. In principle, multiple LTL formulas can be provided though one has to balance contrasting recommendations for the satisfiability of such formulas.

based on monitoring the satisfaction of business constraints. This work does not make any form of prediction nor automatic optimal work-items' distribution.

As a matter of fact, there is no conceptual or technical limitation that would prevent our approach from being implemented as a plug-in for an operational-support service.

2.4. Work-item distribution

Our work on work-item distribution to minimize risks shares commonalities with Operational support and Decision Support Systems (DSSs). We aim to provide recommendations to process participants to take risk-informed decisions. Our work fully embraces the aim of these systems to improve decision making within work systems [30], by providing an extension to existing process-aware information systems.

Mainstream commercial and open-source BPM systems do not feature work-item prioritization. They only allow one to indicate a static priority for tasks (e.g. low, medium or high priority), independently of the characteristics of the process instance and of the qualified resources. Similarly, the YAWL system, which is the one we extended, does not provide means for operational support, besides the extension proposed by de Leoni et al. [31], which, however, defines very basic metrics only.

Several approaches have been proposed in the literature. Table 1 summarizes and compares the most significant ones, using different criteria:

Weight Computation. In order to perform an optimal distribution, every work item needs to be assigned a weight, which may also depend on the resources that is going to perform it or on the moment in time when such work item is performed. These weights can be defined either statically by analysts or dynamically computed on the basis of the past history recorded in an event log.

Process Perspective. When weights are dynamically defined, they may be computed considering different perspectives: control-flow, resources, data and time.

Optimal Assignment. The optimization of work-item distribution can be computed by considering single instances in isolation or trying to optimize the overall performances of all running instances.

Objective. The work-item distribution can be optimized with respect to several factors, such as minimizing the cost, time or maximizing the cooperation. Only few approaches allow one to customize the objective function to minimize/maximize.

Assignment Method. Once an optimal distribution is computed, each work item can be pushed to a single qualified resource (push method), or it can be put in a common pool and simply recommended to a given resource within this pool, that can then pull the work item (pull method). Note that in this last method different resources within the same pool other than the one the work item was recommended to, may still execute the work item. method

Among the available approaches only the one by Cabanillas et al. [28] computes the optimal allocation of resources at the process level. Specifically, this work proposes a priority-based resource allocation, where resources are ranked according to preferences defined using the Semantic Ontology of User Preferences [32]. Once a work item needs to be executed it is pushed to the resource ranking the highest on the basis of the expressed preferences.

Among the approaches providing optimal distribution only two approaches support a pull assignment. The approach of Barba et al. [29] optimizes process performances, using constraint programming (planning and scheduling problem) where constrains are defined considering control-flow and resources only. On the other hand, the approach of Kumar et al. [16] aims to obtain the right balance between execution time and quality. This approach uses work allocation metrics and various quality attributes to find the optimal allocation strategy keeping into consideration the preference of resources for certain work items.

The approach of Yang [22], similar to all the approaches discussed so far, assigns a static weight to each work item. This approach optimizes process execution time and total execution cost according to user preferences. Preferences are defined using a multi-attribute utility function that is optimized using the particle swarm optimization algorithm. A second approach by Kumar et al. [23], and the approach of Huang et al. [24], conclude the list of approaches providing optimal distribution of work items. Kumar et al. [23] propose an approach for optimal resource cooperation using integer linear programming to identify the group of resources with the best synergy to perform a process instance while Huang et al. [24] propose to use task operation models.

There are also approaches that focus on prediction only. Van der Aalst et al. [25] propose an approach to predict total execution time and remaining execution time. The approach uses logs to generate transition

systems annotated with timing information. Transition systems are employed to provide predictions using similarly completed executions as a reference. Folino et al. [26] use a combination of clustering techniques and transition systems. Using clustering they identify process variants in a log and for each cluster they generate a transition system. When a prediction is required, using decision trees the authors identify which cluster the current instance belongs to, and then use the associated transition system to provide a prediction.

Van der Spoel et al. [27] propose an approach to predict the cash flow of a process. This approach uses a combination of process flow prediction, i.e. predicting how the process execution will proceed, and cost prediction, i.e. predicting how much the execution of a predicted activity will cost. Kim et al. [21] propose the use of decision trees to minimize completion time or total labor cost, where the resource with the lowest predicted completion time or total labor cost is suggested.

Finally, Maggi et al. [19] propose a predictive approach to prevent process constraints violation. Users can define linear temporal logic constraints at any point in time during the execution of a process. Then, when a prediction is required, the approach retrieves all traces having a similar prefix of the current instance. These instances are then used to generate a decision tree that is used to predict how the process execution should proceed to satisfy the predefined constraints.

There are also approaches (e.g.,[33, 34, 35]) that mine association rules from event logs to define the preferable distribution of work items. However, in the end a resource manager needs to manually assign work items to resources. Manual distributions are clearly inefficient because they are both unlikely to be optimal and some work items probably remain unassigned for a certain amount of time until the manager takes charge of their assignment. Moreover, the mined rules consider process instances in isolation.

On the basis of the insights emerging from Table 1, we propose a technique that satisfies the following requirements: it should i) use information form different process perspectives to provide predictions; ii) use such predictions to compute an optimal distribution that is not local to individual process instances (instance level) but global across all running instances, which can be from different processes (process level); iii) use user-defined faults as objective functions; iv) leave process participants the final choice of whether to execute a recommended work item (pull assignment method).

This paper is an extended version of the conference paper in [36]. With respect to the conference paper, the main extension relates to the provision of support for multi-instance risk prediction. This is achieved by combining our existing technique for risk estimation [36], with a technique for identifying the best distribution of resources to work items of concurrent process instances, using integer linear programming. This technique has been implemented via a new YAWL custom service, the Multi Instance Prediction Service. Further, the evaluation has been completely redone using a real-life business process in use at a large insurance company. With input from a team of risk analysts from the company, this process has been extensively simulated on the basis of an event log recording one year of completed instances of this process, to show that it is feasible to predict risks across multiple process instances without impacting on performance, and that the

recommendations provided by our recommendation system significantly reduce the number and severity of faults, for all instances simulated.

3. Risk Framework

In this section we elaborate on the type of process-related risks that we can address and on the basis of this, we illustrate an overarching approach for managing process-related risks within which the contribution of this paper fits.

3.1. Process-related Risk

In this paper we focus on process-related risks that can be identified within the boundaries of a business process. In particular, we only consider process-related risks which depend on information available during process execution, e.g. task input and output data, allocated resources, time performance. This implies that process-related risks depending on information outside the process boundaries, i.e. the process context (e.g. market fluctuations or weather forecast), cannot be detected. For this reason organizational risks in general are not addressed, such as those related to partners going bankrupt, or price of the fuel going up. Moreover, since we require process execution information we only consider *executable* business processes. These processes should either be executed by a BPMS on the basis of a process model or be supported by an information system that produces event logs [37], i.e. logs of process-related information which we can use to reconstruct the process instances being executed by aggregating events, such that each instance can be unequivocally identified.

3.2. Risk Approach

The technique proposed in this paper can be seen as part of a wider approach for the management of process-related risks. This approach aims to enrich the four phases of the traditional BPM lifecycle (Process Design, Implementation, Enactment and Diagnosis) [38] with elements of risk management (cf. Figure 1).

Before the Process Design phase, we define an initial phase, namely *Risk Identification*, where existing techniques for risk analysis such as Fault Tree Analysis [39] or Root Cause Analysis [40] 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

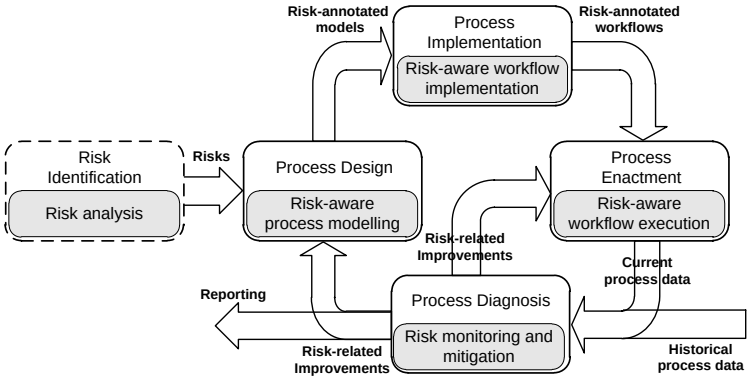


Figure 1: Risk-aware BPM lifecycle.

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 the content of data variables and resource states. In the *Process Enactment* phase such a risk-annotated process model can be executed to ensure risk-aware process execution. Finally, in the *Process Diagnosis* phase, information produced during Process Enactment is used in combination with historical data to monitor the occurrence of risks and faults as process instances are executed. This monitoring may trigger mitigation actions in order to (partially) recover the process instance from a fault.

The technique presented in 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. The techniques developed to support the other phases of our risk-aware BPM approach fall outside the scope of this paper, but have beed addressed in our earlier work [3, 5, 4].

4. YAWL Specification and Running Example

We developed our technique on top of the YAWL language [41] for several reasons. First, this language is very expressive as it provides comprehensive support for the workflow patterns[2], patterns covering all main process prospective such as control-flow, data-flow, resources, and exceptions. Further, it is an executable language supported by an open-source BPM system, namely the YAWL System. This system is based on a service-oriented architecture, which facilitates the seamless addition of new services, like the ones developed as part of this work. Further, the open-source license facilitates its distribution among academics and practitioners (the system has been downloaded over 100,000 times since its first inception in the open-source community). However the elements of the YAWL language used by our technique are common to all process modeling languages, so our technique can in principle be applied to other executable process modeling languages such as BPMN 2.0.

In this section we introduce the basic ingredients of the YAWL language and present them in the context of a running example. This example, whose YAWL model is shown in Figure 2, captures the Carrier Appointment subprocess of an Order Fulfillment process, which is subjected to several risks. This process is inspired by the VICS industry standard for logistics [42], a standard endorsed by 100+ companies worldwide.

The Carrier Appointment subprocess (see Figure 2) starts when a Purchase Order Confirmation is received. A Shipment Planner then estimates the trailer usage and prepares a route guide. Once ready, a Supply Officer prepares a quote for the transportation indicating the cost of the shipment, the number of packages and the total freight volume.

[2] `www.workflowpatterns.com`

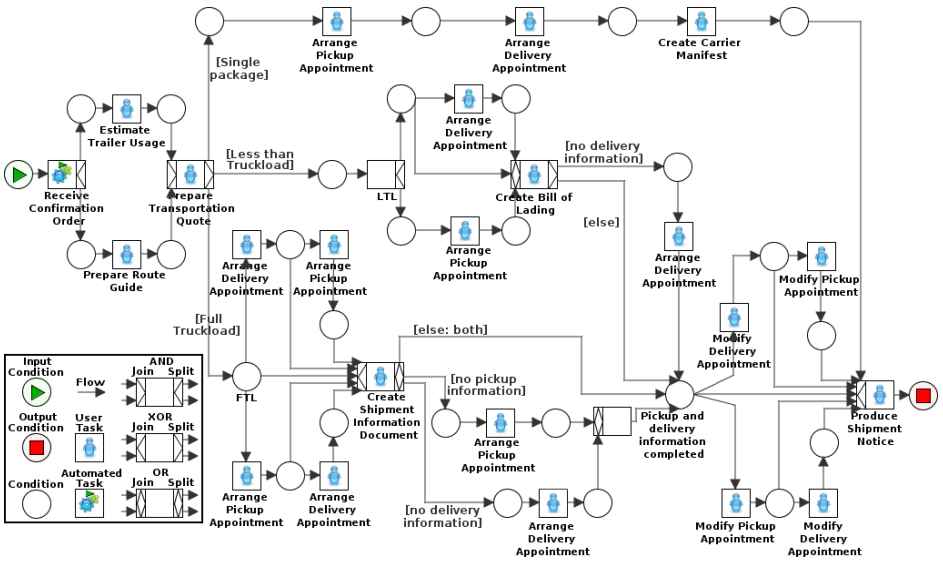


Figure 2: The carrier appointment subprocess of an order fulfillment process, shown in YAWL.

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

Finally, if a single package is to be shipped, a Supply Officer has to arrange a pickup appointment, a delivery appointment, and create a Carrier Manifest, after which a Warehouse Admin Officer can produce a Shipment Notice.

In YAWL, a process model is encoded via a YAWL specification. A specification is made up of one or more nets (each modeling a subprocess), organized hierarchically in a root net and zero or more subnets. Each net is defined as a set of conditions (represented as circles), an input condition, an output condition, and a set of tasks (represented as boxes). Tasks are connected to conditions via flow relations (represented as arcs). 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.

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 FTL in Figure 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 arc can be taken like after executing Prepare Transportation Quote. If one or more outgoing arcs can be taken it is captured by an OR-split like after executing Create Shipment Information Document. Similarly, we have XOR-joins and OR-joing that merge multiple incoming arcs in to one. If among all the incoming arcs only one is active we use a XOR-join like before executing Produce Shipment Notice, while if among all incoming arcs one or more arcs are active we use a OR-join like before executing task Create Bill of Lading. Finally, an AND-split is used when all outgoing arcs need to be taken, like after Receive Confirmation Order, while an AND-join is used to synchronize parallel arcs like before executing Prepare Transportation Quote. Splits and joins are represented as decorators on the task's box.

Tasks are considered to be descriptions of a piece of work that forms part of the overall process. Thus, control-flow, data, and resourcing specifications are all defined with reference to tasks at design time. At runtime, each task acts as a template for the instantiation of one or more work items. A work item $w = (ta, id)$ is the run-time instantiation of a task ta for a process instance id.

A new process instance id is started and initialized by placing a token in the input condition of a YAWL net. The token represents the thread of control and flows through the net as work items are executed. The execution of a work item (ta, id) consumes one token from some of ta's input conditions (depending on the task's type of join) and produces one token in some of ta's output conditions (depending on the task's type of split). In YAWL, work items are performed by either process participants (*user tasks*) or software services (*automated tasks*). An example of an automated task is Receive Confirmation Order in Figure 2, while an example of user task is Estimate Trailer Usage.

Finally, the preset ${}^\bullet t$ of a task t is the set of its input conditions. Similarly, the postset $t^\bullet$ of a task t is the set of its output conditions. The preset and postset of a condition can be defined analogously.

The notions presented above are formalized in the Appendix.

5. 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 (a.k.a. process instance) 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, $d \in \mathbb{N}$ is the event's timestamp, $\phi \in \Phi$ is an assignment of values to a sub set of variables in V. In other words, $\mathcal{L} \in \mathcal{B}((T \times R \times \mathbb{N} \times \Phi)^*)$.*[3]

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:

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

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 $\{f_1, \ldots, f_n\} \subset \mathcal{F}$, we can have a *composite fault*:

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

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

A complete trace σ of our Carrier Appointment process, can be affected by three faults:

Over-time fault. This fault is linked to a Service Level Agreement (SLA) which establishes that the process must terminate within a predefined 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:

$$f_{time}(\sigma) = \max\left(\frac{d_\sigma - d_{mct}}{\max(d_{\max} - d_{mct}, 1)}, 0\right)$$

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 occur due to a misunderstanding between the company's employee and the customer. In order to keep the reputation high, the company wants to avoid these misunderstandings and having to call the customer again. The severity of this fault is:

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

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

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 profit cost beyond which the company will not make any profit. The severity increases as the cost goes beyond the profit cost. 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 profit cost. The severity of a cost fault is:

$$f_{cost}(\sigma) = \min\left(\frac{\max(c_\sigma - c_{\min}, 0)}{\max(c_{\max} - c_{\min}, 1)}, 1\right)$$

Moreover, we assume that the company considers Reputation-loss Fault to be less significant than the other faults. The company could decide to define a composite fault where the reputation weights half:

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

The risk is the product of the estimation of the fault's severity at the end of the process-instance execution and the accuracy of such an estimation.

When a process instance is being executed, many factors may influence the risk and, ultimately, the severity of a possible fault. For instance, a specific order in which a certain set of tasks is performed may increase or decrease the risk, compared to any other. 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.

6. Risk Estimation

We aim to provide work-items' recommendation to minimize the risk corresponding to the highest product of fault severity and likelihood. For this purpose, it is necessary to predict the most likely fault severity associated with continuing the execution of a process instance for each enabled task. 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 accuracy of such an estimation. Moreover, $(x_1, \ldots, x_n, l_1) \in \psi_f(y_1) \wedge (x_1, \ldots, x_n, l_2) \in \psi_f(y_2) \Rightarrow l_1 = l_2 \wedge y_1 = y_2$.*

The function estimator is trained through a set of observation instances. 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.

The function estimator can easily be built using a number of machine learning techniques. In this paper, we employ the C4.5 algorithm to build decision trees. We decided to use decision tree classification, and specifically the C4.5 algorithm, for the following reasons: i) it can handle both continuous and discrete (categorical) attributes; ii) it can handle training data with missing attribute values; iii) it can build models that can be easily interpreted; iv) it can deal with noise; v) it automatically finds a subset of the features that are relevant to the classification (i.e. no need for feature selection); and vi) it automatically discretizes continuous features. This last function helps us significantly simplify the problem of finding an optimal distribution of work items to resources, as we will discuss in Section 7.

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 tuple $\overrightarrow{x} \in X_1 \times X_2 \times \ldots \times X_n$ satisfying e is expected to return $\overline{y}$ as output.

We link the accuracy of a prediction for $\psi_f(\overline{y})$ to the quality of e as classifying expression. Let I be the set of observation instances used to construct the decision tree. Let $I_e = \{(\overrightarrow{x}, y) \in I \mid \overrightarrow{x} \text{ satisfies } e\}$ and $I_{e,\overline{y}} = \{(\overrightarrow{x}, y) \in I_e \mid y = \overline{y}\}$. The accuracy is $l = |I_{e,\overline{y}}|/|I_e|$; therefore, for all $((x_1, \ldots, x_n), y) \in I_e, (x_1, \ldots, x_n, l) \in \psi_f(\overline{y})$.

Figure 3 shows an example of a possible decision tree. It is the estimator $\psi_{f_{\hat{c}}}$ of a function that returns a value belonging to the set H containing the numbers between 0 and 1 with no more than 2 decimals. It is obtained through a set of observation instances based on all data attributes generated during the execution of the process. For example, having as data attributes a resource, a task, the cost of a good, and a process instance's elapsed time, we obtained the following function $f_{\hat{c}} : \mathit{Resource} \times \mathit{Task} \times \mathit{GoodCost} \times \mathit{TimeElapsed} \rightarrow H$. For instance, let us consider the value $y = 0.6$. Analyzing the tree, the value is associated with two expressions: e_1 is ($\mathit{Resource} = \mathit{MichaelBrown} \wedge \mathit{Task} = \mathit{ArrangePickupAppointment}$) and e_2 is ($\mathit{Resource} \neq \mathit{MichaelBrown} \wedge \mathit{GoodCost} < 3157 \wedge \mathit{TimeElapsed} < 30 \wedge \mathit{Task} = \mathit{CreateShipmentInformationDocument}$). Let us suppose that, among observation instances ($\mathit{Resource}$, Task, $\mathit{GoodCost}$, $\mathit{TimeElapsed}, y$) s.t. e_1 or e_2 evaluates to true, $y = 0.6$ occurs 60% or 80% of times, respectively. Therefore, $\psi_{f_{\hat{c}}}(0.6)$ contains the tuples ($\mathit{Resource}$, Task, $\mathit{GoodCost}$, $\mathit{TimeElapsed}, 0.6$) satisfying e_1, along with tuples ($\mathit{Resource}, \mathit{Task}$, $\mathit{GoodCost}$, $\mathit{TimeElapsed}$, 0.8) satisfying e_2. Regarding computational complexity, if decision trees are used, training ψ_f with m observation instances is computed in quadratic time with respect to the dimension n (i.e. the number of attributes) of the input tuple, specifically $O(n^2 \cdot m)$ [43].

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.

Let $N = (T_N, C_N, R_N, V_N, U_N, can_N)$ be a YAWL net. In order to provide accurate risks associated with performing work items of a certain process instance, it is important to incorporate the execution history of that process instance into the analysis. In order to avoid overfitting predictive functions the history needs to be abstracted. Specifically, we abstract the execution history as two functions: $C_r : T_N \rightarrow R$ denoting the last executor of each task and $C_t : T_N \rightarrow \mathbb{N}$ denoting the number of times that each task has been performed in the past. Pairs $(c_r, c_t) \in C_r \times C_t$ are called *contextual information*. Given the execution trace of a (running) instance $\sigma' \in (T_N \times R_N \times \mathbb{N} \times \Phi)$, we introduce function $\mathsf{getContextInformation}(\sigma')$ that returns the contextual information (c_r, c_t) that can be constructed from σ'.

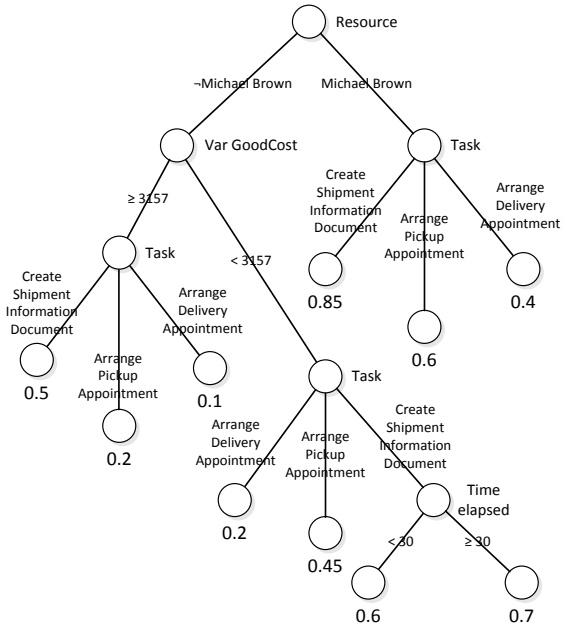


Figure 3: An example of decision tree used to build a function estimator.

Let Φ be the set of all possible assignments of values to variables, i.e. the set of all partial functions $V_N \not\rightarrow U_N$. Each condition $c \in C_N$ can be associated with a function $f_c : \Phi \times c^\bullet \times R_N \times \mathbb{N} \times C_r \times C_t \rightarrow H$. If $f_c(\phi, t, r, n, c_r, c_t) = y$, at the end of the execution of the process instance, the fault's severity is going to be y if the instance continues with resource $r \in R_N$ that performs task $t \in c^\bullet$ at time n with contextual information (c_r, c_t) when variables are assigned values as for function ϕ. Of course, this function is not known but it needs to be estimated, based on the behavior observed in an event log $\mathcal{L}$. Therefore, we need to build am estimator ψ_{f_c} for f_c. Let us consider condition c_{FTL} (see Figure 2), and the associated function estimator $\psi_{f_{c_{FTL}}}$. Let us suppose that the accuracy is 1, i.e. for each $t \in {c_{FTL}}^\bullet$, $\psi_{f_{c_{FTL}}}(t)$ always returns 1.

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 moment in which task *Create Shipment Information Document* is started. Therefore, it is evident that it is less "risky" to execute *Arrange Pickup Appointment*.

The generation of function estimators is obtained as follows. For each process instance in the log and for each event generated during the execution of each process instance, we retrieve context information, time elapsed, and data variables produced. These three elements together constitute an observation instance.

This observation instance is assigned to the decision point which precedes the activity generating the event. Once all observation instances are generated, the observation instances associated with each decision point are used to build the function estimator associated with the decision point, using, for example, decision trees. In the Appendix we formalize this algorithm (see Algorithm 1).

In this section, we presented a technique to generate prediction functions. It is important to observe that the number of risks that may eventuate during the execution of a process does not affect the prediction algorithm, since we consider the combined risk level of all risks. Specifically, we do so by assigning a relative weight to each risk. This weight system allows process administrators to fine tune the predictive function on the basis of the relative importance of each risk.

7. Multi-Instance Work-Item Distribution

With the technique presented so far, each resource is given *local* risk advice as to what work item to perform next, i.e. a resource is suggested to perform the work item with the lowest overall risk for that combination of process instance and resource, without looking at other resources that may be assigned work items within the same instance or in other instances running concurrently. Clearly, such a local work-item distribution is not optimal, since work items have to compete for resources and this may not guarantee the best allocation from a risk viewpoint. For example, let us consider two resources r_1 and r_2 and two work items w_a and w_b such that the risk of r_1 performing w_a is 0.2, and the risk of r_1 performing w_b is 0.6, while the risk of r_2 performing w_a is 0.1 and the risk of r_2 performing w_a is 0.4. Moreover for the company executing these work items, it is equally important to minimize the eventuation of risks as well as the overall execution time. If w_a is assigned to r_2 because locally this resource has the lowest risk, r_1 will be forced to perform w_b leading to an overall risk of 0.7. Another option is to assign both work items to r_2, yielding an overall risk of 0.5. Both these solutions are non-optimal distributions: the former because the overall risk is too high, the latter, despite the lower risk, because the workload between the two resources is unbalanced, with the result of increasing the overall execution time.

In this section we combine our technique for risk prediction with a technique for computing an *optimal distribution* of work items to resources (available or busy). By optimal distribution we mean a distribution that minimizes the weighted sum of overall execution time and overall risk across all running instances. In other words, the algorithm aims to balance the distribution of work items across resources while keeping the risk low. This distribution can then be used to provide work item recommendations to resources, such that these can be aided in selecting the best work item to perform. In the example above, the optimal distribution is r_1-w_a and r_2-w_b with an overall risk of 0.6. While this is higher than 0.5 obtained with the second solution, r_1 and r_2 will work in parallel thus reducing the overall execution time.

7.1. Optimal Work-Item Distribution

Let $\mathfrak{f}$ be a certain (composite) fault function and assuming we at time τ. Let $I = \{id_1, \ldots, id_n\}$ be the set of running instances of N. Given an instance $id \in I$, $timeElapsed_\tau(id) \in \mathbb{N}$ denotes the time elapsed since instance id has started and $varAssign_\tau(id) \in (V_N \to U_N)$ is the current assignment of values to variables. Moreover, let us denote a function $use_N : R_N \to 2^{T_N \times I}$ that associates each resource with the work items that he/she is executing within the set I of running process instances. Let WE be the set of work items being executed, i.e. $WE = \sum_{r \in R_N} use_N(r)$. Let $W \subseteq T_N \times I$ be the set of work items that are enabled but not started yet. Section 4 has discussed the concept of deferred choice, highlighting that some of the enabled work items are mutually exclusive. Therefore, we introduce an equivalence relation $\sim$ between elements of W, such that $w_a \sim w_b$ if, picking $w_a \in W$ for execution disables $w_b \in W$ or vice versa. Let $W_\sim$ be the partition of W according to relation $\sim$.

For each enabled work item $w \in W$, we perform an estimation $time(w)$ of the expected duration of work item w. For each started work item $w \in WE$, we also perform an estimation $time(w)$ of the amount of time needed by w to be completed. To compute such estimations, we employ the technique proposed in [25] using event log $\mathcal{L}$ as input.

Let Ψ be the set of function estimators that are computed through Algorithm 1, using net N, event log $\mathcal{L}$ and given fault function $\mathfrak{f}$ as input. For each work item $w \in W$, let us denote with $risk_{r,w,t}$ the risk of starting a work item w at time t. For example, given a work item w, this can be computed by retrieving the estimation function associated with each decision point preceding w and taking the maximum value of the predicted risk: $risk_{r,w,t} = \mathsf{calcRisk}(N, \mathfrak{f}, r, t, w, \Psi)$. See Algorithm 2 in the Appendix for a formal definition of this algorithm.

Let $maxTime = \sum_{w \in W \cup WE} time(w)$ be the maximum duration of executing all work items that are currently enabled and started. This corresponds to the situation in which work items are just executed sequentially, i.e. a new work item starts only when no other work item is being executed. Given a resource $r \in R_N$ and a work item $(ta, id) \in W$ such that $ta \in can_N(r)$, we compute the set of moments in time in which the risk of r performing (ta, id): $start_{r,w} = \{t \in [\tau, \tau + maxTime] \mid risk_{r,w,t} \neq risk_{r,w,t-1}\} \cup \{\tau\}$.

Certainly, this can be naively computed by computing the risk for all moments in time between τ and $\tau + maxTime$. Nonetheless, it can be done more efficiently by observing the occurrences of splits on the time variable that are present in the decision trees. For instance, let us consider the decision tree in Figure 3: the only time reference is 30. This reference occurs in a root-to-leaf path in which resource $r \neq$ *Michael Brown* and $Task =$ *Create Shipment Information*. Therefore, for each resource $r \in R \setminus \{$*Michael Brown*$\}$ and work-item $w = ($*Create Shipment Information*$, id) \in W$, $start_{r,w} = \{\tau, elapsed(id) + 30\}$. Moreover, for each work item $w = (ta, id) \in W$ with $ta \neq$ *Create Shipment Information* and for each resource $r \in R$, $start_{r,w} = \{\tau\}$. Similarly, for each work item $w = (ta, id) \in W$, $start_{r',w} = \{\tau\}$ with $r' =$ *Michael Brown*.

Given a work item w, a resource r and a time t, $\Delta_{r,w}(t)$ denotes the first moment t' in time after t in which the risk changes, i.e. $t' > t$, $t' \in start_{r,w}$ and there exists no $t'' \in start_{r,w}$ such that $t' > t'' > t$. If such a moment t' does not exist, $\Delta_{r,w}(t) = \tau + maxTime$.

We formulate the problem of distributing work items as a Mixed-Integer Linear Programming (MILP) problem. The following two sets of variables are introduced:

- for each resource $r \in R_N$ and work-item $w = (ta, id) \in W$ such that $ta \in can_N(r)$, there exists a variable $x_{r,w,t}$. If the solution of the MILP problem is such that $x_{r,w,t} = 1$, r is expected to start performing w in interval between t and $\Delta_{r,w}(t)$, $x_{r,w,t} = 1$; otherwise, $x_{r,w,t} = 0$;
- for each work item $w \in W \cup WE$ (i.e., running or enabled), we introduce a variable $wa_{r,w}$. If work item w is not being executed at time τ and is eventually distributed to resource r, the MILP solution assigns to $wa_{r,w}$ a value that is equal to the moment in time when resource r is expected to start work item w. If w is not expected to be started by r, $wa_{r,w} = 0$; if w is already being executed by r at time τ (i.e. $w \in WE$), $wa_{r,w}$ is statically assigned value τ.

The MILP problem aims to minimize the weighted sum of the expected total execution time and the overall risk:

$$\min \left(\frac{\alpha}{maxTime} \sum_{r \in R_N} \sum_{w \in W \cup WE} wa_{r,w} + (1-\alpha) \sum_{r \in R_N} \sum_{w \in W \cap can_N(r)} \sum_{t \in start_{r,w}} risk_{r,w,t} \cdot x_{r,w,t} \right)$$

where $\alpha \in [0, 1]$ is the weight of the expected total execution time w.r.t. the overall risk.

This MILP problem is subject to a number of constraints:

- for each $r \in R_N$ and $w = (ta, id) \in W$ such that $ta \in can_N(r)$, if r starts performing w in the interval between t and $\Delta_{r,w}(t)$, $x_{r,w,t}$ must be equal to 1 (and vice versa):
$$x_{r,w,t} = 1 \Leftrightarrow \Delta_{r,w}(t) > wa_{r,w} \wedge wa_{r,w} \geq t; \tag{1}$$
- For each partition $D \in W_\sim$, only one work item in D can be executed and it can only be executed by one resource and can only start within one interval:
$$\sum_{r \in R_N} \sum_{w \in D \cap can_N(r)} \sum_{t \in start_{r,w}} x_{r,w,t} = 1 \tag{2}$$
- Every resource $r \in R_N$ cannot execute more than one work item at any time. Therefore, for each $r \in R_N$ and for each pairs of partitions $D_1, D_2 \in W_\sim$:
$$\Big(\sum_{w_a \in D_1} wa_{r,w_a} - \sum_{w_b \in D_2} wa_{r,w_b} \geq \sum_{w_b \in D_2} \sum_{t \in start_{r,w_b}} time(w_b) \cdot x_{r,w_b,t}\Big) \vee \Big(\sum_{w_b \in D_2} wa_{r,w_b} - \sum_{w_a \in D_1} wa_{r,w_a} \geq \sum_{w_a \in D_1} \sum_{t \in start_{r,w_a}} time(w_a) \cdot x_{r,w_a,t}\Big) \tag{3}$$

In the Appendix, we show how constraints in Equation 1 and in Equation 3 can be translated into an equivalent set of linear constraints.

We observe that we can compute $\Delta_{r,w}(t)$ only if we use a machine-learning method, such as decision trees, that can automatically discretize continuous features such as the time feature in this case. By automatically

identifying those time moments that *discriminate* over risk values, we can split the time feature in time intervals and thus base our predictions on such intervals (e.g., "if elapsed time $< t$ OR elapsed time $\geq t$") instead of working with individual time moments ("if elapsed time $= t_1$ OR elapsed time $= t_2$ OR elapsed time $= t_3$..."). If such automatic discretization of continuous features was not available, we could not compute $\Delta_{r,w}(t)$ and consequently we would need to introduce a different variable $x_{r,w,t}$ for each moment t in time. This would lead to an increase of the complexity of finding a solution to the MILP problem, which is exponential on the number of variables.

As an example of an instance of the class of MILP problems, let us consider a case where at time τ we want to schedule three work items w_a, w_b and w_c, and we have two resources, r_1 and r_2, who can perform them. We know that w_a and w_b are mutually exclusive generating the following partitions $D_1 = \{w_a, w_b\}$, and $D_2 = \{w_c\}$. Moreover, we know that the expected duration of each work item is $time(w_a) = 30$ mins, $time(w_b) = 10$ mins, and $time(w_c) = 40$ mins. We also know that the risk associated with each work item does not change over time. Finally, we know that when performed by resource r_1 the work items have the following expected risk levels: $risk_{r_1,w_a,\tau} = 0.2$, $risk_{r_1,w_b,\tau} = 0.7$, and $risk_{r_1,w_c,\tau} = 0.6$ while when performed by resource r_2 the work items have the following expected risk levels: $risk_{r_2,w_a,\tau} = 0.1$, $risk_{r_2,w_b,\tau} = 0.7$, and $risk_{r_2,w_c,\tau} = 0.4$.

The MILP problem for distributing work items will take the following form (assuming $\alpha = 0.5$):

$$\text{minimize } \frac{0.5}{\tau + 80} \cdot (wa_{r_1,w_a} + wa_{r_1,w_b} + wa_{r_1,w_c} + wa_{r_2,w_a} + wa_{r_2,w_b} + wa_{r_2,w_c})$$
$$+ 0.5 \cdot (0.2 \cdot x_{r_1,w_a,\tau} + 0.7 \cdot x_{r_1,w_b,\tau} + 0.6 \cdot x_{r_1,w_c,\tau} + 0.1 \cdot x_{r_2,w_a,\tau} + 0.7 \cdot x_{r_2,w_b,\tau} + 0.4 \cdot x_{r_2,w_c,\tau})$$

subject to the following constraints:

either work item w_a or w_b is executed, whereas w_c has to (instantiation of Equation 2):

$$x_{r_1,w_a,\tau} + x_{r_1,w_b,\tau} + x_{r_2,w_a,\tau} + x_{r_2,w_b,\tau} = 1$$

$$x_{r_1,w_c,\tau} + x_{r_2,w_c,\tau} = 1$$

at any time, all resources, i.e. r_1 and r_2, can only perform one work item (Equation 3):

$$\big(wa_{r_1,w_c} - wa_{r_1,w_a} - wa_{r_1,w_b} \geq 30 \cdot x_{r_1,w_a,\tau} + 10 \cdot x_{r_1,w_b,\tau}\big) \vee \big(wa_{r_1,w_a} + wa_{r_1,w_b} - wa_{r_1,w_c} \geq 40 \cdot x_{r_1,w_c,\tau}\big)$$

$$\big(wa_{r_2,w_c} - wa_{r_2,w_a} - wa_{r_2,w_b} \leq 30 \cdot x_{r_2,w_a,\tau} + 10 \cdot x_{r_2,w_b,\tau}\big) \vee \big(wa_{r_2,w_a} + wa_{r_2,w_b} - wa_{r_2,w_c} \leq 40 \cdot x_{r_2,w_c,\tau}\big)$$

instantiation of Equation 1 for resources r_1 and r_2 and work items w_a, w_b and w_c:

$$x_{r_1,w_a,\tau} = 1 \Leftrightarrow wa_{r_1,w_a} \geq \tau \wedge wa_{r_1,w_a} < \tau + 80 \qquad x_{r_1,w_a,\tau} = 1 \Leftrightarrow wa_{r_2,w_a} \geq \tau \wedge wa_{r_2,w_a} < \tau + 80$$

$$x_{r_1,w_b,\tau} = 1 \Leftrightarrow wa_{r_1,w_b} \geq \tau \wedge wa_{r_1,w_b} < \tau + 80 \qquad x_{r_1,w_b,\tau} = 1 \Leftrightarrow wa_{r_2,w_b} \geq \tau \wedge wa_{r_2,w_b} < \tau + 80$$

$$x_{r_1,w_c,\tau} = 1 \Leftrightarrow wa_{r_1,w_c} \geq \tau \wedge wa_{r_1,w_c} < \tau + 80 \qquad x_{r_1,w_c,\tau} = 1 \Leftrightarrow wa_{r_2,w_c} \geq \tau \wedge wa_{r_2,w_c} < \tau + 80$$

The optimal solution to this problem is $wa_{r_1,w_a} = 1$, $wa_{r_1,w_b} = 0$, $wa_{r_1,w_c} = 0$, $wa_{r_2,w_a} = 0$, $wa_{r_2,w_b} = 0$, $wa_{r_2,w_c} = 1$, $x_{r_1,w_a,\tau} = 1$, $x_{r_1,w_b,\tau} = 0$, $x_{r_1,w_c,\tau} = 0$, $x_{r_2,w_a,\tau} = 0$, $x_{r_2,w_b,\tau} = 0$, $x_{r_2,w_c,\tau} = 1$, that is a schedule where resource r_1 performs work item w_a and resource r_2 performs work item w_c.

7.2. Recommendations for Work Items Execution

After the optimal distribution is computed, we need to provide a recommendation to r for executing any $w \in W \cap can_N(r)$. For any work item w, the recommendation $rec(w, r)$ is a value between 0 and 1, where 0 is assigned to the work item with the highest recommendation and 1 to the work item with the least one. Let us consider an optimal solution s of the MILP problem to distribute work items while minimizing risks. The work-item recommendations for each resource r are given as follows:

- If there exists a work item $w \in W \cap can_N(r)$ such that $x_{r,w,\tau} = 1$ for solution s, the optimal distribution suggests w to be performed by r at the current time. Therefore, $rec(w, r) = 0$. For any other work item w', the value $rec(w', r)$ is strictly greater than 0 and lower than or equal to 1:

$$rec(w', r) = \frac{risk_{r,w',\tau} + risk_{r,w,\tau}}{risk_{r,w,\tau} + 1}$$

  $rec(w', r)$ grows proportionally to $risk_{r,w',\tau}$, with $rec(w', r) = 1$ if $risk_{r,w',\tau} = 1$.

- Otherwise, r is supposed to start no work item at the current time. However, since recommendations need to be provided also to resources that are not supposed to execute any work item, for each $w \in W \cap can_N(r)$, we set $rec(w, r) = risk_{r,w,\tau}$.

It is possible that the optimal distribution assigns no work item to a resource r at the current time. This is the case when r is already performing a work item (i.e., no additional work item should suggested) or there are more resources available than work items to assign.

Let us consider the problem illustrated at the end of Section 7.1. In this problem we have two resources r_1 and r_2 and three work items w_a, w_b, and w_c. We recall that the expected risk levels associated with a resource performing a given work item were: $risk_{r_1,w_a,\tau} = 0.2$, $risk_{r_1,w_b,\tau} = 0.7$, and $risk_{r_1,w_c,\tau} = 0.6$ for resource r_1, and $risk_{r_2,w_a,\tau} = 0.1$, $risk_{r_2,w_b,\tau} = 0.7$, and $risk_{r_2,w_c,\tau} = 0.4$ for resource r_2. We can then derive that the best allocation requires that resource r_1 performs work item w_a and resource r_2 performs work item w_c. Finally, when recommendations about which work item should be performed and by whom will they be required, the recommendation system will return the following values: $rec(r_1, w_a) = 0$, $rec(r_1, w_b) = 0.75$ and $rec(r_1, w_c) = 0.67$ for resource r_1, and $rec(r_2, w_a) = 0.36$, $rec(r_2, w_b) = 0.79$ and $rec(r_2, w_c) = 0$ for resource r_2.

7.3. Recommendations for Filling Out Forms

In addition to providing risk-informed decision support when picking work items for execution, we provide support during the execution of the work items themselves. Human resources usually perform work items

by filling out a form with the required data. The data that are provided may also influence a process risk. Therefore, we want to highlight the expected risk whenever a piece of data is inserted by the resource into the form.

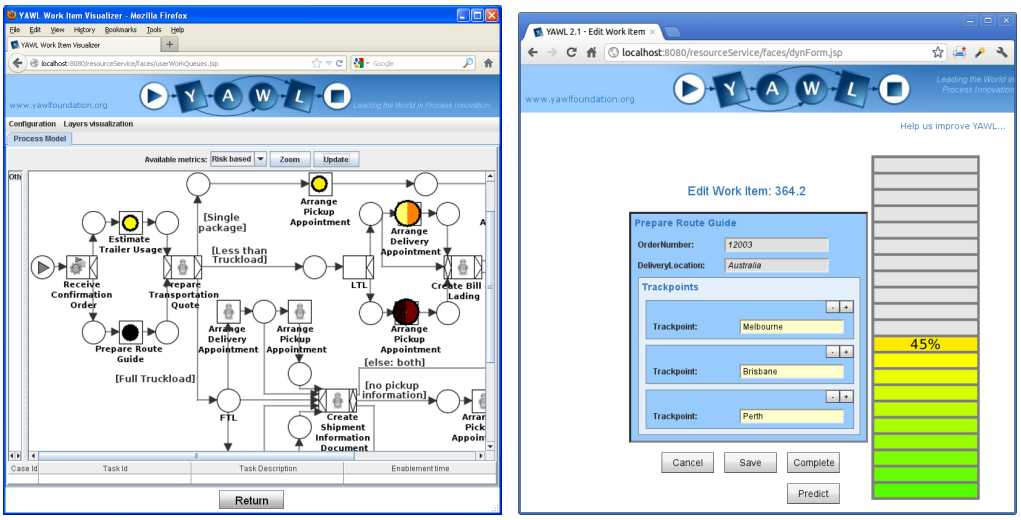
The risk associated with filling a form with particular data is also computed using Algorithm 2. When used to compute the risk associated with filling a form to perform a work item (ta, id), $varAssign(id)$ is the variable assignment that would result by submitting a form using the data the resource has inserted so far.

8. Implementation

We operationalized our recommendation system on top of the YAWL BPM system, by extending an existing YAWL plug-in and by implementing two new custom YAWL services. This way we realized a risk-aware BPM system supporting multi-instance work distribution and forms filling-out.

The intent of our recommendation system is to "drive" participants during the execution of process instances. This goal can be achieved if participants can easily understand the suggestions proposed by our tool. For this we decided to extend a previous plug-in for the YAWL Worklist Handler, named *Map Visualizer* [31]. This plug-in provides a graphical user interface to suggest process participants the work items to execute, along with assisting them during the execution of such work items. The tool is based on two orthogonal concepts: maps and metrics. A *map* can be a geographical map, a process model, an organizational diagram, etc. For each map, work items can be visualized by dots which are located in a meaningful position (e.g., for a geographic map, work items are projected onto the locations where they need to be executed, or for a process-model map onto the boxes of the corresponding tasks in the model). Dots can also be colored according to certain *metrics*, which determine the suggested level of priority of a work item. This approach offers advantages over traditional BPM systems, which are only equipped with basic client applications where work items available for execution are simply enlisted, and sorted according to given criteria. When users are confronted with hundreds of items, this visualization does not scale well. The validity of the metaphors of maps and metrics used for decision support in process execution was confirmed through a set of experiments reported in [31]. De Leoni et al. [31] only define very basic metrics. We have extended the repertoire of these metrics with a new metric that is computed by employing the technique described in Section 7.

Figure 4a 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 elements of the model. Each dot corresponds to a different work item and is colored according to the risks for the three faults defined before. When multiple dots are positioned on the same coordinates, they are merged into a single larger dot whose diameter grows with the number of dots being amalgamated. Colors go from white to black, passing through intermediate shades of yellow, orange, red, purple and brown. The white



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

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

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

and black colors identify work items associated with a risk of 0 and 1, respectively. The screenshot in Figure 4a refers to a configuration where multiple process instances are being carried out at the same time and, hence, the work items refer to different process instances. The configuration of dots highlights that the risk is lower if the process participant performs a work item of 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 work item(s).

As discussed in Section 7.3, the activity of compiling a form is also supported. Figure 4b shows a screenshot where, while filling 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, which means a risk of 0.45). While a participant changes the data in the form, the risk value is recomputed accordingly.

Besides the extension to the Map Visualizer, we implemented two new custom services for YAWL, namely the *Prediction Service* and *Multi Instance Prediction Service*. The *Prediction Service* provides risk prediction and recommendation. It implements the technique described in Section 6 and constructs decision trees through J48, which is the implementation of the C4.5 algorithm in the Weka toolkit for data mining.[4] Since the algorithm is not capable of predicting continuous values, in order to provide a risk prediction we grouped risk levels that are close to each other in intervals of 0.05 (e.g. all risk likelihoods from 0 to 0.04 are considered as 0, from 0.05 to 0.09 as 0.1 and so on).

The Prediction Service communicates with the *Log Abstraction Layer* described in [3], to be able to

[4]The Weka toolkit is available at `www.cs.waikato.ac.nz/ml/weka/`

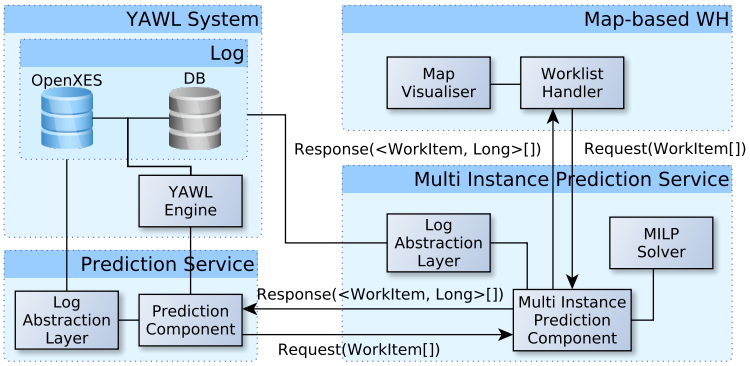


Figure 5: The integration of the implemented tools with the YAWL system.

retrieve event logs from textual files, such as from OpenXES event logs, or directly from the YAWL database, which stores both historical information and the current system's state.

The *Multi Instance Prediction Service*, similarly to the *Prediction Service*, provides risk prediction and recommendation. The difference between these two services is that in the former a recommendation takes into account *all* process instances currently running in the system. The Multi Instance Prediction Service interacts with the Prediction Service to obtain "local" predictions that, in combination with other information derived from the log (e.g. expected task duration, other running instances), are used to find the optimal resource allocation using the technique described in Section 7. To this purpose, the Multi Instance Prediction Service also interacts with the MILP Solver. The *MILP Solver* provides an interface for the interaction with different integer linear programming solvers. So far we support Gurobi[5], SCIP[6] and LPSolve[7]. Finally, the Multi Instance Prediction Service is invoked by the *Map Visualizer* to obtain the risk predictions and recommendations and show these to process participants in the form of maps. The map visualizer works 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.

9. Evaluation

We evaluated our recommendation system using the claims handling process and related event data, of a large insurance company kept under condition of anonymity. The event data recording about one year of completed instances (total: 1,065 traces) was used as a benchmark for our evaluation. The claims handling process, modeled in Figure 6, starts when a new claim is received from a customer. Upon receipt of a claim, a file review is conducted in order to assess the claim, then the customer is contacted and informed

[5]Available at `www.gurobi.com`
[6]Available at `scip.zib.de`
[7]Available at `lpsolve.sourceforge.net`

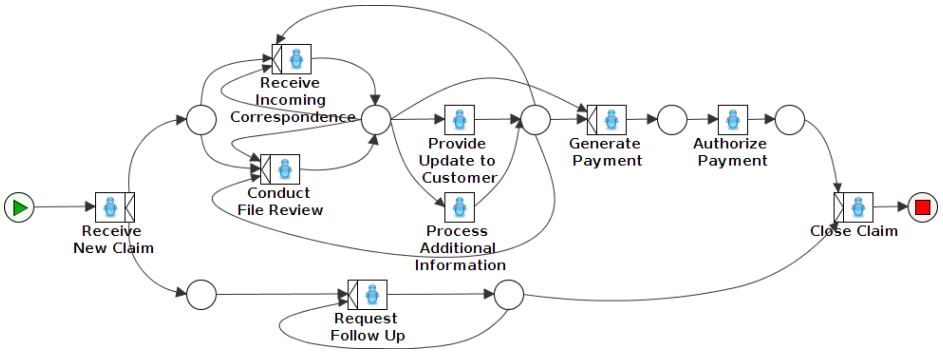


Figure 6: The Claims Handling process used for the evaluation.

about the result of the assessment. The customer may provide additional documents ("Receive Incoming Correspondence"), which need to be processed ("Process Additional Information") and the claim may need to be reassessed. After the customer has been contacted, a payment order is generated and authorized in order to process the payment. During the execution of the process model, several updates about the status of the claim may need to be provided to the customer as follow-ups. The claim is closed once the payment has been authorized.

As one can see from the model, this process contains several loops, each of which is executed multiple times, in general.

Four risk analysts working in this insurance company were consulted through an iterative interview process, to identify the risks this process is exposed to.[8] They reported about three equally-important faults related to complete traces σ of the claim handling process:

Over-time fault. This fault is the same as the over-time fault described in Section 5. For this risk we set the Maximum Cycle Time $d_{mct} = 30$ (i.e. 30 days) and the maximum duration $d_{\max} = 300$ (i.e. 300 days). The severity of an overtime fault is measured as follows:

$$f_{time}(\sigma) = \max\left(\frac{d_\sigma - d_{mct}}{\max(d_{\max} - d_{mct}, 1)}, 0\right)$$

Customer-dissatisfaction fault. During the execution of the process, if a customer is not updated regularly on their claim, they may feel "unheeded". A customer dissatisfied may generate negative consequences such as negative publicity for the insurance company, leading to bad reputation. In order to avoid this kind of situations, the company's policy is to contact their customers at least once every 15 days. Given the set $\Lambda = \{(t, r, d, \phi) \in \sigma | t = \text{Request Follow Up} \vee t = \text{Receive New Claim} \vee t = \text{Close Claim}\}$ of events belonging to task *Request Follow Up*, to task *Receive New Claim*, or to task

[8] Three interviews were conducted for a total of four hours of audio recording

Close Claim, ordered by timestamp, the severity of this fault is:

$$f_{dissatisfaction}(\sigma) = \sum_{1 \leq i \leq \|\Lambda\|} \max(0, d_{i+1} - d_i - 15days)$$

where d_i is the time stamp of i^{th} event $\in \Lambda$.

Cost Overrun fault. Each task has an execution cost associated with it, e.g. the cost of utilizing a resource to perform a task. Since the profit of the company decreases with a higher number of tasks executed, the company clearly aims to minimize the number of tasks required to process a claim, for example by reducing the number of follow-ups with the claimant or the need for processing additional documents, and reassessing the claim, once the process has started. The severity of the cost overrun fault increases as the cost goes beyond the minimum. Let c_σ be the number of work items executed in σ, $c_{\max}$ be the maximum number of work items (e.g. 30) that should be executed in any process instance that has already been completed (including σ), and $c_{\min}$ be the number of work items with unique label executed in σ. The severity of a cost overrun fault is:

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

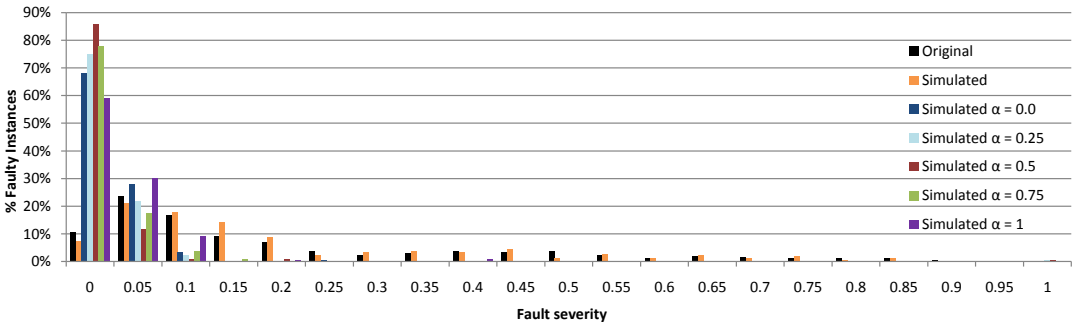
The occurrence of these three faults in the logs is checked using the technique that we proposed in [5], which was originally designed for run-time detection of process-related risks.

Trialling our recommendation system within the company was not possible, as the claims handling process concerns thousands of dollars, which cannot be put in danger with experiments. So we had to simulate the execution of this process and the resource behavior using CPN Tools.[9] We mined the control-flow of our simulation model from the original log and refined it with the help of business analysts of the company, and added the data, resource utilization (i.e. who does what), and tasks duration, which we also obtained from the log. We then add the frequency of occurrence of each of these elements, on the basis on that observed from the log. This log was also used to train the function estimators.

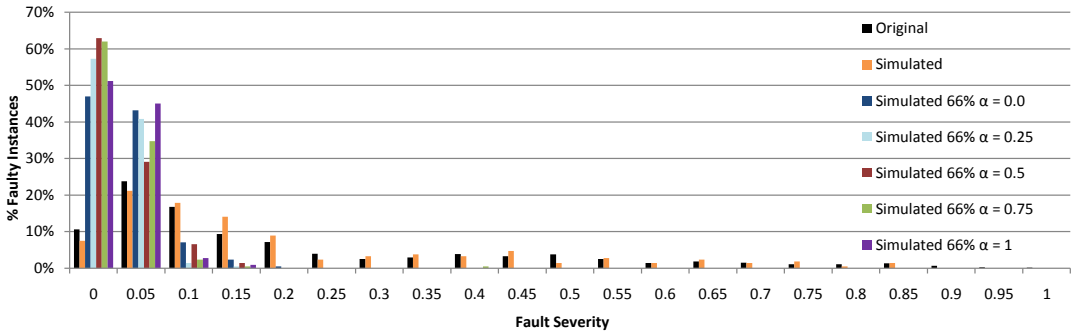
The CPN Tools model we created is a hierarchical model composed of ten nets that all together count 65 transitions and 62 places. The main net is based on the model showed in Figure 6, with additional places and transitions in order to guarantee the interaction with our recommendation system. The remaining nine nets define the behavior of each one of the nine tasks showed in Figure 6.

We used this model to simulate a constant workload of 50 active instances, in order to maintain a similar ratio to the original log (in the original log we had 271 active instances on average). In order to maintain the ratio between active instances and resources, we reduced the number of resources utilized to one-sixth of the original number observed in the log. Finally, we analyzed the fault distribution of the generated log using the technique presented in [5].

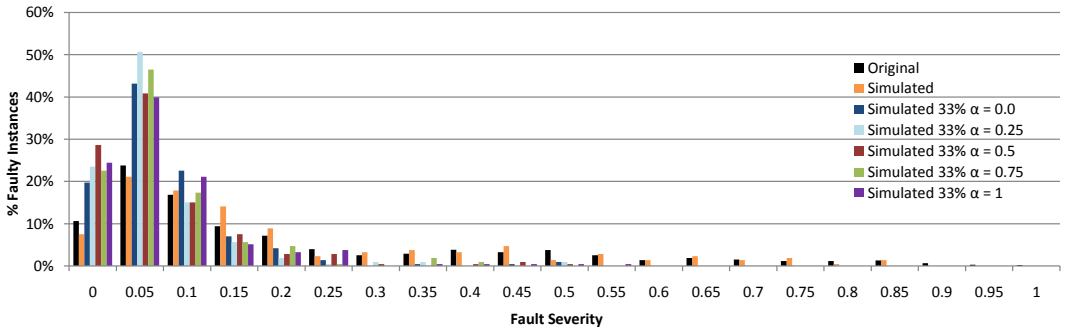
[9] Available at `www.cpntools.org`



(a) Results following 100% of the suggestions provided.



(b) Results following 66% of the suggestions provided.



(c) Results following 33% of the suggestions provided.

Figure 7: Comparison of the fault severity when recommendations are and are not followed, with 0 denoting absence of faults. The x-axis represents the severity of the composite fault and the y-axis represents the percentage of instances that completed with a certain severity.

The model created with CPN Tools was able to reproduce the behavior of the original log. The *Kolmogorov-Smirnov Z* two-samples test ($Kolmogorov - SmirnovZ = 0.763$, p $= 0.605 > 0.05$) shows no significant difference between the distribution of the composite fault in the original log and that in the simulated log. This result is confirmed by the *Mann-Whitney* test ($U = 109,163.0$, $z = -0.875$, $p = 0.381 > 0.05$).

We performed three sets of experiments. In the first set, all the suggestions provided by the recommendation system were followed. In the second set, only 66% of the times the suggestions were followed, and executing the process as the company would have done for the remaining 33% of the times. Finally, in the

Reference Logs	# Traces	% Faulty Instances	Average	Median
Original	1065	89.4%	0.22	0.10
Simulation model	1065	92.5%	0.22	0.15

		Suggestions 100%			Suggestions 66%			Suggestions 33%		
Test Logs	# Traces	% Faulty Instances	Avg	Mdn	% Faulty Instances	Avg	Mdn	% Faulty Instances	Avg	Mdn
Simulated aggregated	1065	26.8%	0.02	0.00	43.9%	0.03	0.00	76.3%	0.07	0.05
- Simulated $\alpha = 0.0$	213	31.9%	0.02	0.00	53.1%	0.03	0.05	80.3%	0.08	0.05
- Simulated $\alpha = 0.25$	213	24.9%	0.02	0.00	42.7%	0.02	0.05	76.5%	0.07	0.05
- Simulated $\alpha = 0.5$	213	14.1%	0.01	0.00	37.1%	0.02	0.05	71.4%	0.07	0.05
- Simulated $\alpha = 0.75$	213	22.1%	0.01	0.00	38.0%	0.02	0.05	77.5%	0.07	0.05
- Simulated $\alpha = 1.0$	213	40.8%	0.03	0.00	48.8%	0.03	0.05	75.6%	0.08	0.05

Table 2: Percentage of faulty instances, mean and median fault severity occurring in the reference logs, i.e. original log and simulation model log. Percentage of faulty instances, mean and median fault severity occurring in the test logs aggregated into a unique log, i.e. simulated aggregated, and for each value of α, reported for each of the three sets of experiments (33%, 66% and 100% suggestions used).

third set of experiments, only 33% of the times the suggestions provided by our recommendation system were followed. Moreover, for each set of experiments we tested several values of α (i.e. 0.0, 0.25, 0.5, 0.75 and 1.0), where α equal to 0 will shift focus on reducing risks, while α equal to 1 on reducing the overall execution time (see Section 7).

All experiments were executed simulating the execution of the process by means of the CPN Tools model. For each experiment we generated a new log containing 213 fresh log traces (a fifth of the traces contained in the original log). We used a computer with an Intel Core i7 CPU (2.2 GHz), 4GB of RAM, running Lubuntu v13.10 (64bit). We used Gurobi 5.6 as MILP solver as this is the most efficient solver among the three that we support [44][10] and imposed a time limit of 60 seconds, within which a solution needs to be provided for each problem. For mission-critical processes, the time limit can also be reduced. If a time limit is set and Gurobi cannot find a solution within the limit, a sub-optimal solution is returned, i.e. the best solution found so far. The experiments have shown that, practically, the returned solution is always so close to the optimal that it does not influence the final fault's magnitude.

Figure 7 shows the results of each of the three sets of experiments, comparing the fault severity of the original log with that obtained when recommendations are 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 severity and likelihood when such instances complete.

Table 2 shows the results of the experiments. In this table we show percentage of faulty instances, mean and median fault severity obtained during our tests. The values are shown for the original log and the log obtained by our simulation model without using our recommendation system (Simulation model). Same values are also reported for each log obtained using our recommendation system, both in an aggregated log

[10]Gurobi is free of use for academic purposes but is not open-source. This is the reason why we also support other two implementations: SCIP and LPSolve.

(Simulated aggregated) and for each value of α, over the three sets of experiments (33%, 66% and 100% suggestions used). In the best case (Simulated log with $\alpha = 0.5$), our recommendation system was able to reduce the percentage of instances terminating with a fault from 89.4% to 14.1% and the average fault severity from 0.216 to 0.01. In particular, the use of our recommendation system significantly reduced the number of instances terminating with faults, as evidenced by the result of the *Person's* χ^2 test ($\chi^2(1) = 857.848$, $p < 0.001$ for the first set of experiments, $\chi^2(1) = 494.907$, $p < 0.001$ for the second set, and $\chi^2(1) = 64.663$, $p < 0.001$ for the third one, computed over the original log and the simulated aggregated log). Based on the *odds ratio*, the odds of an instance completing without a fault are respectively 23.06, 10.75, and 2.62 times higher if our suggestions are followed. Moreover, we tested if the number of suggestions followed influences the effectiveness of our recommendation system. The *Kruskal-Wallis* test ($H(3) = 1,603.61$, $p < 0.001$) shows that the overall fault severity among the three sets of experiments (using the Simulated overall dataset, i.e. independently of the value of the parameter α) and the original log is significantly different, and as revealed by *Jonkheere's* test ($J = 1,658,630.5$, $z = -41.034$, $r = -0.63$, $p < 0.001$), the median fault severity decreases as more suggestions are followed (see Figure 8). These two tests indicate that our recommendation system is capable of preventing the occurrence of faults and of reducing their severity. Clearly, it is preferable to follow as many suggestions as possible in order to obtain the best results though this may not always be possible.

We tested how the value of the parameter α influences the effectiveness of our recommendation system. We compared the performances obtained with each value of α for each set of experiment. The *Kruskal-Wallis* test ($H(4) = 46.176$, $p < 0.001$ for the first set of experiments, $H(4) = 17.191$, $p = 0.002 < 0.05$ for the second one, $H(4) = 5.558$, $p = 0.235 > 0.05$ for the third one) shows how the value of the parameter α significantly influences the median fault severity if the suggestions proposed are followed in at least 66% of the instances. *Jonkheere's* test ($J = 251,305$, $z = 5.577$, $r = 0.17$, $p < 0.001$ for the first set of experiments, $J = 246,322.5$, $z = 3.918$, $r = 0.12$, $p < 0.001$ for the second one) revealed that the median fault severity increases when the value of α diverges from 0.5 moving either toward 0 or 1.

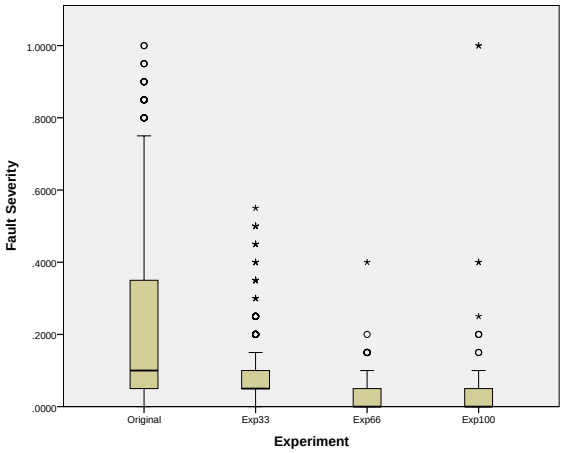


Figure 8: BoxPlot showing the fault severity occurring in instances of each of the three experiments and of the original log.

In the case study taken in exam, the duration of an instance has an influence over the over-time fault and the cost overrun fault. A short execution time will directly minimize the duration of an instance (thus preventing the over-time fault) but also reduce the number of activities that are executed inside such an

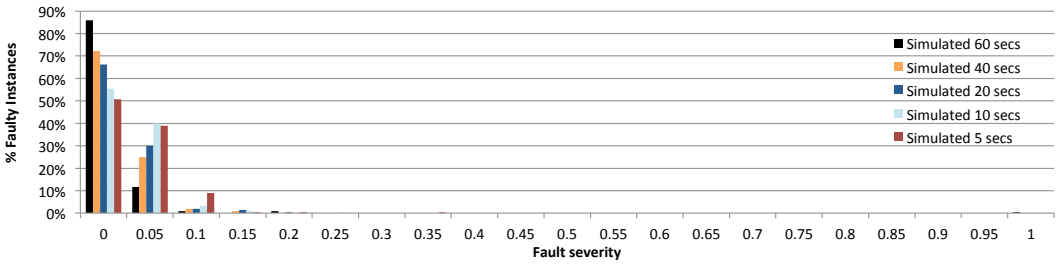


Figure 9: Fault severity distribution using different time limits.

instance (thus preventing the cost overrun fault). In light of so, it is not strange that the best results are obtained with $\alpha = 0.5$ which strikes a good balance between minimizing risks and overall execution time.

Finally, we performed a sensitivity test over the time limit granted to the MILP solver. We tested our recommendation system with five different time limits, while keeping the value of α equal to 0.5 and following all suggestions (best configuration for risk prevention). The time limits used were: 5, 10, 20, 40 and 60 seconds. Figure 9 shows the distribution of fault severities obtained using these different time limits. We can observe that changing the time limit yields statistically different distributions, as revealed by the *Kruskal-Wallis* test ($H(4) = 74.738$, $p < 0.001$). Moreover, the *Jonkheere's* test ($J = 186,238$, $z = -8.631$, $r = -0.264$, $p < 0.001$) reveals that the median fault severity decreases when more time is granted to the MILP solver. From a practical point of view though, it is interesting to observe that even with a time limit of 5 seconds the approach can still notably reduce the faults severity, with 90% of the instances terminating with a fault severity up to 0.05 out of 1. This suggests that users may set the time limit to be granted to the MILP solver on the basis of the number of process activities that are critical, i.e. using a low time limit if the number of critical activities is low and a high time limit if that number is high.

Based on the results of our experiments we can conclude that the approach produces a significant reduction in the number of faults and their severity. Specifically, for the case study in question we achieved the best results with α equal to 0.5, with a time limits of 60 seconds. We observe that this parameter can be customized based on the priorities of the company where our approach would be deployed, e.g. an organization may use lower values of α if risk reduction is prioritized over reduction of process duration.

10. Conclusion

This paper proposes a recommendation system that allows users to take risk-informed decisions when partaking in multiple process instances running concurrently. Using historical information extracted from process execution logs, for each state of a process instance where input is required from a process participant, the recommendation system determines the risk 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. This input can be in the form of data used to fill

out a user form, or in terms of the next work item chosen to be executed.

The recommendation system relies on two techniques: one for predicting risks, the other for identifying the best assignment of participants to the work items currently on offer. The objective is to minimize both the overall risk of each process instance (i.e. the combined risk for all faults) and the execution time of all running process instances.

We designed the recommendation system in a language-independent manner, using common notions of executable process models such as tasks and work items borrowed from the YAWL language. We then implemented the recommendation system as a set of components for the YAWL system. For each user decision, the recommendation system provides recommendations to participants in the form of visual aids on top of YAWL models. We also extended the YAWL user form visualizer, to show a risk profile based on the data inserted by the participant for a given form. Although we implemented our ideas in the context of the YAWL system, our recommendation system can easily be integrated with other BPM systems by implementing an interface that allows the communication through the "log abstraction layer" (in [5] we showed how it can be integrated with the Oracle BPEL 10g database), and by extending the Map-Based Worklist Handler in order to list work items belonging to a different BPM system than the YAWL system.

We simulated a real-life process model based on one year of execution logs extracted from a large insurance company, and in collaboration with risk analysts from the company we identified the risks affecting this process. We used these logs to train our recommendation system. Then we performed various statistical tests while simulating new process instances following the recommendations provided by our recommendation system, and measured the number and severity of the faults upon instance completion. Since in reality it might not always be feasible to follow the recommendations provided, we varied the percentage of recommendations to be followed by the simulated instances. Even when following one recommendation out of three, the recommendation system was able to significantly reduce the number and severity of faults. Further, results show that risks can be predicted online, i.e. while business processes are being executed, without impacting on execution performance.

The proposed recommendation system can only address process-related risks in so far as these depend on information available during process execution, i.e. task input and output data, allocated resources and time performance. This implies that risks depending on information outside the boundaries of a process, i.e. its context (e.g. market fluctuations or weather forecast) cannot be detected.

While our approach is independent of any specific machine-learning method, in this paper we leveraged on decision-tree classification. Decision trees have, among others, the advantage of automatically discretizing continuous features. We used this information to drastically simplify the MILP problem in order to find an optimal work-item allocation to resources. However, decision trees cannot deal with class attributes that are defined over a continuous domain, such as the fault severity. To overcome this issue, we had to discretize the range of fault severity values (between 0 to 1) into intervals of 0.05. This limitation could be lifted by

using methods that combine classification and regression trees, such as CART methods. This is certainly a direction for future work.

Another limitation is that we cannot guarantee to find the optimal solution, because of the time bound that we impose on the MILP solver for efficiency reasons. However, our experiments show that this time limit can be as short as 5 seconds (i.e. near real-time), to obtain a significant reduction in the number of faults.

The recommendation system we propose relies on a couple of assumptions. While we deal with multiple process instances sharing the same pool of participants, we assume no sharing of data between instances. Further, we only assume that one participant can perform a single task at a time. These assumptions offer opportunities for future work. For example, for the sharing of data between instances we need to reformulate the MILP problem in order to consider that the risk estimation of a work item may change as a consequence of the modification of data by work items that have been scheduled to be performed first. For allowing participants to perform multiple tasks at a time we need to assign a capacity to each resource as the maximum number of work items that resource can perform in parallel. Our MILP problem needs to be reformulated in order to take this capacity into account.

Acknowledgments This research is partly funded by the ARC Discovery Project "Risk-aware Business Process Management" (DP110100091). NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] Standards Australia and Standards New Zealand, Standard AS/NZS ISO 31000, 2009.

[2] Basel Committee on Bankin Supervision, Basel II - International Convergence of Capital Measurement and Capital Standards, 2006.

[3] R. Conforti, G. Fortino, M. La Rosa, A. ter Hofstede, History-aware, real-time risk detection in business processes, in: Proc. of CoopIS, volume 7044 of *LNCS*, Springer, 2011.

[4] R. Conforti, A. H. M. ter Hofstede, M. La Rosa, M. Adams, Automated risk mitigation in business processes, in: Proc. of CoopIS, volume 7565 of *LNCS*, Springer, 2012.

[5] R. Conforti, M. La Rosa, G. Fortino, A. ter Hofstede, J. Recker, M. Adams, Real-Time Risk Monitoring in Business Processes: A Sensor-based Approach, Journal of Systems and Software 86 (2013) 29392965.

[6] C. Alberts, A. Dorofee, OCTAVE Criteria, Version 2.0, Technical Report CMU/SEI-2001-TR-016, Carnegie Mellon University, 2001.

[7] B. Barber, J. Davey, The use of the CCTA Risk Analysis and Management Methodology CRAMM in health information systems, in: MEDINFO, North Holland Publishing, 1992.

[8] M. Lund, B. Solhaug, K. Stølen, Model-Driven Risk Analysis - The CORAS Approach, Springer, 2011.

[9] S. Suriadi, B. Weiß, A. Winkelmann, A. ter Hofstede, M. Adams, R. Conforti, C. Fidge, M. La Rosa, C. Ouyang,

M. Rosemann, A. Pika, M. Wynn, Current Research in Risk-Aware Business Process Management - Overview, Comparison, and Gap Analysis, Communications of the Association for Information Systems (2014 (forthcoming)).

[10] A. Pika, W. van der Aalst, C. Fidge, A. ter Hofstede, M. Wynn, Predicting deadline transgressions using event logs, in: Proc. of BPM Workshop 2012, volume 132 of *LNBIP*, Springer, 2013.

[11] S. Suriadi, C. Ouyang, W. van der Aalst, A. ter Hofstede, Root cause analysis with enriched process logs, in: Proc. of BPM Workshop 2012, volume 132 of *LNBIP*, Springer, 2013.

[12] D. Vengerov, A reinforcement learning approach to dynamic resource allocation, Engineering Applications of Artificial Intelligence 20 (2007) 383–390.

[13] S. E. Elmaghraby, Resource allocation via dynamic programming in activity networks, European Journal of Operational Research 64 (1993) 199 – 215.

[14] K. Baker, Introduction to Sequencing and Scheduling, Wiley, 1974.

[15] W. Zhang, T. G. Dietterich, A reinforcement learning approach to job-shop scheduling, in: Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2, IJCAI'95, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995, pp. 1114–1120.

[16] A. Kumar, W. M. P. van der Aalst, E. M. W. Verbeek, Dynamic work distribution in workflow management systems: How to balance quality and performance, Journal of Management Information Systems 18 (2002) 157–193.

[17] J. Nakatumba, M. Westergaard, W. M. P. van der Aalst, A Meta-model for Operational Support, BPM Center Report BPM-12-05, BPMcenter.org, 2012.

[18] H. Schonenberg, B. Weber, B. F. Dongen, W. M. P. Aalst, Supporting flexible processes through recommendations based on history, in: Proceedings of the 6th Conference Business Process Management (BPM 2008), volume 5240 of *LNCS*, Springer Berlin Heidelberg, 2008.

[19] F. M. Maggi, C. Di Francescomarino, M. Dumas, C. Ghidini, Predictive monitoring of business processes, in: M. Jarke, J. Mylopoulos, C. Quix, C. Rolland, Y. Manolopoulos, H. Mouratidis, J. Horkoff (Eds.), Proceedings of the 26th International Conference on Advanced Information Systems Engineering (CAiSE'14), volume 8484 of *Lecture Notes in Computer Science*, Springer International Publishing, 2014, pp. 457–472.

[20] M. Montali, F. M. Maggi, F. Chesani, P. Mello, W. M. P. v. d. Aalst, Monitoring business constraints with the event calculus, ACM Transactions on Intelligent Systems and Technology 5 (2013) 17:1–17:30.

[21] A. Kim, J. Obregon, J.-Y. Jung, Constructing decision trees from process logs for performer recommendation, in: Proceedings of 2013 Business Process Management Workshops, LNBIP, Springer, 2014. To appear.

[22] I.-T. Yang, Utility-based decision support system for schedule optimization, Decision Support Systems 44 (2008) 595 – 605.

[23] A. Kumar, R. Dijkman, M. Song, Optimal resource assignment in workflows for maximizing cooperation, in: Business Process Management, volume 8094 of *LNCS*, Springer Berlin Heidelberg, 2013, pp. 235–250.

[24] Z. Huang, X. Lu, H. Duan, A task operation model for resource allocation optimization in business process management, IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 42 (2012) 1256 – 1270.

[25] W. M. P. van der Aalst, M. H. Schonenberg, M. Song, Time prediction based on process mining, Information Systems 36 (2011) 450–475.

[26] F. Folino, M. Guarascio, L. Pontieri, Discovering context-aware models for predicting business process performances, in: Proceedings of Cooperative Information Systems (CoopIS 2012), volume 7565 of *LNCS*, Springer, 2012.

[27] S. van der Spoel, M. van Keulen, C. Amrit, Process prediction in noisy data sets: a case study in a dutch hospital, in: Proceedings of the Second International Symposium on Data-Driven Process Discovery and Analysis, SIMPDA 2012, volume 162 of *LNBIP*, Springer Verlag, 2013, pp. 60–83.

[28] C. Cabanillas, J. M. García, M. Resinas, D. Ruiz, J. Mendling, A. Ruiz-Cortés, Priority-based human resource allocation

in business processes, in: Proceedings of 11th International Conference on Service Oriented Computing (ICSOC 2013), volume 8274 of *LNCS*, 2013, pp. 374–388.

[29] I. Barba, B. Weber, C. Valle, Supporting the optimized execution of business processes through recommendations, in: Business Process Management Workshops, volume 99 of *LNBIP*, Springer Berlin Heidelberg, 2012, pp. 135–140.

[30] S. Alter, A work system view of DSS in its fourth decade, Decision Support Systems 38 (2004) 319–327.

[31] M. de Leoni, M. Adams, W. M. P. van der Aalst, A. H. M. ter Hofstede, Visual support for work assignment in process-aware information systems: Framework formalisation and implementation, Decision Support Systems 54 (2012) 345–361.

[32] J. Garca, D. Ruiz, A. Ruiz-Corts, A model of user preferences for semantic services discovery and ranking, in: L. Aroyo, G. Antoniou, E. Hyvnen, A. ten Teije, H. Stuckenschmidt, L. Cabral, T. Tudorache (Eds.), The Semantic Web: Research and Applications, volume 6089 of *LNCS*, Springer, 2010, pp. 1–14.

[33] S. Rinderle-Ma, W. M. van der Aalst, Life-Cycle Support for Staff Assignment Rules in Process-Aware Information Systems, Technical Report WP 213, Eindhoven University of Technology, BETA Working Paper Series, 2007.

[34] Z. Huang, X. Lu, H. Duan, Mining association rules to support resource allocation in business process management, Expert Systems with Applications 38 (2011) 9483–9490.

[35] Y. Liu, J. Wang, Y. Yang, J. Sun, A semi-automatic approach for workflow staff assignment, Computers in Industry 59 (2008) 463 – 476.

[36] R. Conforti, M. de Leoni, M. La Rosa, W. M. van der Aalst, Supporting risk-informed decisions during business process execution, in: Proceedings of CAiSE, volume 7908 of *LNCS*, Springer, 2013, pp. 116–132.

[37] W. Aalst, Process Mining - Discovery, Conformance and Enhancement of Business Processes, Springer, 2011.

[38] M. Dumas, W. M. P. van der Aalst, A. H. M. ter Hofstede, Process-Aware Information Systems: Bridging People and Software through Process Technology, Wiley & Sons, 2005.

[39] I. E. Commission, IEC 61025 Fault Tree Analysis (FTA), 1990.

[40] W. Johnson, MORT - The Management Oversight and Risk Tree, U.S. Atomic Energy Commission, 1973.

[41] A. H. M. ter Hofstede, W. M. P. van der Aalst, M. Adams, N. Russell (Eds.), Modern Business Process Automation: YAWL and its Support Environment, Springer, 2010.

[42] Voluntary Interindustry Commerce Solutions Association, Voluntary Inter-industry Commerce Standard (VICS), http://www.vics.org. Accessed: June 2011.

[43] J. R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc., 1993.

[44] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. Bixby, E. Danna, G. Gamrath, A. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. Steffy, K. Wolter, Miplib 2010, Mathematical Programming Computation 3 (2011) 103–163.

[45] N. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, D. Edmond, Workflow resource patterns: Identification, representation and tool support, in: Proceedings of CAiSE, volume 3520 of *LNCS*, Springer, 2005, pp. 216–232.

Appendix A.

This appendix provides the formal definition of YAWL, the algorithms discussed in Section 6, and the mathematical proofs of Lemma 1 and Lemma 2 discussed in Section 7.1.

YAWL Definition

Definition 3. *A YAWL net $N \in \mathcal{N}$ is a tuple $N = (T_N, C_N, i, o, F_N, R_N, V_N, U_N, can_N)$ where:*

- *T_N is the set of tasks of N;*
- *C_N is the set of conditions of N;*
- *$i \in C_N$ is the input condition;*
- *$o \in C_N$ is the output condition;*
- *A flow relation $F_N \subseteq (C_N \setminus \{o\} \times T_N) \cup (T_N \times C_N \setminus \{i\})$;*
- *R_N is the set of resources authorized to perform any tasks in T_N;*
- *V_N is the set of variables that are defined in the net;*
- *U_N is the set of values that can be assigned to variables;*
- *$can_N : R_N \to 2^{T_N}$ is a function that associates resources with the tasks that are authorized to perform.*

Compared to [41] we use a simplified definition of YAWL nets, which describes those parts that are relevant for the article. YAWL supports sophisticated authorization mechanisms as described in the resource patterns [45]. The above definition describes a simplified version where authorizations are specified at task level and applies to all work items of a certain task. As such, this definition is generalizable to other executable process modeling languages.

We use the following auxiliary functions from [41]. The preset of a task t is the set of its input conditions: ${}^\bullet t = \{c \in C_N \mid (c, t) \in F_N\}$. Similarly, the postset of a task t is the set of its output conditions: $t^\bullet = \{c \in C_N \mid (t, c) \in F_N\}$. The preset and postset of a condition can be defined analogously.

Algorithms

Algorithm 1: generateFunctionEstimatorsForRiskPrediction

Data: $\mathbf{N} = (T_N, C_N, R_N, V_N, U_N, can_N)$ – A YAWL net, $\mathcal{L}$ – An event log, $\mathfrak{f} \in \mathcal{F}$ – 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 for all $c \in C_N$, $I(c) = \emptyset$.
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 $(c_r, c_t) \leftarrow \mathsf{getContextInformation}(\langle (t_1, r_1, d_1, \phi_1), \ldots, (t_i, r_i, d_i, \phi_i) \rangle)$
6 Time elapsed $\overline{d} \leftarrow (d_i - d_1)$
7 $J \leftarrow (A \odot (t_i, r_i, \overline{d}) \odot c_r \odot c_t), \mathfrak{f}(\sigma))$
8 **foreach** $c \in {}^\bullet t_i$ **do** $I(c) \leftarrow I(c) \cup \{J\}$;
9 **foreach** *variable* $v \in \mathsf{dom}(\phi_i)$ **do** $A(v) \leftarrow \phi_i(v)$;
10 **end**
11 **end**
12 **Set** function Ψ such that $\mathsf{dom}(\Psi) = \emptyset$
13 **foreach** *condition* $c \in C_N$ **do** $\Psi(c) \leftarrow \mathsf{buildFunctionEstimator}\big(I(c)\big)$;
14 **return** Ψ

Algorithm 1 details how function estimators ψ_{f_c} can be constructed. In the algorithm, we use $\odot$ 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 overloaded to deal with functions defined on a finite and ordered domain. Let $f : W \to Z$ be a function defined on an ordered domain $W = \{w_1, \ldots, w_o\}$. 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 is periodically executed, e.g., every week or after 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 executions of instances of the process modelled by N, and a fault function. The output is a function Ψ that associates each condition c with function estimator ψ_{f_c}. Initially, in line 1, we initialize function I which is going to associate each condition c with the set of observation instances associated with the executions 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 $(A \odot (t_i, r_i, \overline{d}) \odot c_r \odot c_t)$ is the observed input 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_{f_c} for condition f_c by the relative observation instances and rewrite Ψ s.t. $\Psi(c) = \psi_c$.

Algorithm 2: calcRisk

Data: $\mathbf{N} = (T_N, C_N, R_N, V_N, U_N, can_N)$ – A YAWL net, $f \in \mathcal{F}$ – A fault function, r – resource, t – time, (ta, id) – work item
Result: A risk value

```
risk ← 0
φ ← varAssign(id)
d ← timeElapsed(id)
(c_r, c_t) ← getContextInformation(history(id))
foreach condition c ∈ •t do
    ψ ← Ψ(c)
    Pick (severity, l) such that (φ, t_a, r, d, c_r, c_t, l) ∈ ψ(severity)
    risk ← max(severity · l, risk)
end
```

Algorithm 2 details how to calculate the risk associated with the execution of a work item. When a prediction for the execution of work item w by resource r is required, the algorithm retrieves the values of all variables during the execution, the elapsed time (i.e. the time passed from the start of the process), and contextual information about the process instance to which the work item belongs to.

The algorithm then retrieves the function estimator associated with each decision point in the pre-set of w. From each of these function estimators, an prediction of the risk resulting from r executing w is obtained. The variable assignments, the elapsed time, the contextual information, w, and r are used as input for the function estimator. Finally, the prediction having the highest product between predicted fault severity and likelihood of the prediction is returned.

Lemmas

Lemma 1. *Constraints of the form as in Equation 1 can be rewritten into sets of equivalent constraints of the form as in Equations A.1.*

$$\begin{aligned} &-wa_{r,w} - M \cdot (1 - x_{r,w,t}) \leq -t \\ &wa_{r,w} - M \cdot (1 - x_{r,w,t}) < \Delta_{r,w}(t) \\ &wa_{r,w} - M \cdot x_{r,w,t} - M \cdot o'_{r,w,t} < t \\ &-wa_{r,w} - M \cdot x_{r,w,t} - M \cdot (1 - o'_{r,w,t}) \leq -\Delta_{r,w}(t) \end{aligned} \tag{A.1}$$

where M is a sufficiently large number (e.g., the largest machine-representable number) and $o_{r,w,t}$ is a boolean variable that needs to be introduced in the MILP problem.

Proof of Lemma 1 Let us consider $x_{r,w,t}$ and its possible values 1 and 0. If $x_{r,w,t} = 1$ then the last two constraints will be satisfied by $-M \cdot x_{r,w,t} \ll t - wa_{r,w}$ and $-M \cdot x_{r,w,t} \ll -\Delta_{r,w}(t) - wa_{r,w}$. In order to satisfy the first two constraints, since $M \cdot (1 - x_{r,w,t}) = 0$, $wa_{r,w}$ must be $wa_{r,w} \geq t \wedge wa_{r,w} < \Delta_{r,w}(t)$, that is exactly the second part of the constraint defined in Equations 1.

If $x_{r,w,t} = 0$ then $M \cdot (1 - x_{r,w,t}) = M$. This satisfies the first two constraints since $-M \cdot (1 - x_{r,w,t}) \ll -t + wa_{r,w}$ and $-M \cdot (1 - x_{r,w,t}) \ll \Delta_{r,w}(t) - wa_{r,w}$. The third constraint can be satisfied only if $wa_{r,w} < t$ or if $o'_{r,w,t} = 1$, similar thing can be said for the fourth constraint that will be satisfied only if $wa_{r,w} \geq \Delta_{r,w}(t)$ or if $o'_{r,w,t} = 0$. We can derive that in order to satisfy the last two constraints we either have $wa_{r,w} < t$ and $o'_{r,w,t} = 0$, or we have $wa_{r,w} \geq \Delta_{r,w}(t)$ and $o'_{r,w,t} = 1$. As we can see for $x_{r,w,t} = 0$ the only way to satisfy the constraints of Equations A.1 is to violate the second part of the constraint defined in Equations 1. □

Lemma 2. *Constraints of the form as in Equations 3 can be rewritten into sets of equivalent constraints of the form as in Equations A.2.*

Similarly, the constraints in Equation 3 can be transformed into a set of linear constraints as follows:

$$\begin{aligned} \sum_{w_b \in D_2} wa_{r,w_b} - \sum_{w_a \in D_1} wa_{r,w_a} + \sum_{w_b \in D_2} \sum_{t \in start_{r,w_b}} time(w_b) \cdot x_{r,w_b,t} - M \cdot o_{r,D_1,D_2,t} \leq 0 \\ \sum_{w_a \in D_1} wa_{r,w_a} - \sum_{w_b \in D_2} wa_{r,w_b} + \sum_{w_a \in D_1} \sum_{t \in start_{r,w_a}} time(w_a) \cdot x_{r,w_a,t} - M \cdot (1 - o_{r,D_1,D_2,t}) \leq 0 \end{aligned} \tag{A.2}$$

where M is a sufficiently large number and $o_{r,D_1,D_2,t}$ is a boolean variable that needs to be introduced in the MILP problem.

Proof of Lemma 2 Let us consider the constraints in Equations A.2, and let introduce for readability purposes the following equality:

$$\sum_{w_b \in D_2} wa_{r,w_b} - \sum_{w_a \in D_1} wa_{r,w_a} + \sum_{w_b \in D_2} \sum_{t \in start_{r,w_b}} time(w_b) \cdot x_{r,w_b,t} = a$$

$$\sum_{w_a \in D_1} wa_{r,w_a} - \sum_{w_b \in D_2} wa_{r,w_b} + \sum_{w_a \in D_1} \sum_{t \in start_{r,w_a}} time(w_a) \cdot x_{r,w_a,t} = b.$$

we can then rewrite Equations A.2 as:

$$a - M \cdot o_{r,D_1,D_2,t} \leq 0$$

$$b - M \cdot (1 - o_{r,D_1,D_2,t}) \leq 0$$

The first constraint in Equations A.2 can only be satisfied if either $a \leq 0$ or if $-M \cdot o_{r,D_1,D_2,t} \leq 0$. Similarly, the second constraint can only be satisfied if either $b \leq 0$ or if $-M \cdot (1 - o_{r,D_1,D_2,t}) \leq 0$. Since $o_{r,D_1,D_2,t}$ can only be 0 or 1, we can see that in order to satisfy both constraints either $a \leq 0$ or $b \leq 0$ must be satisfied that is exactly the constraint defined in Equations 3. □